КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Дипломна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Алгоритм розпізнавання рукописного математичного тексту в реальному часі

Виконав студент 4-го курсу
Гаврилюк Мирослав Максимович _____
(підпис)

Науковий керівник:
професор, доктор фізико-математичних наук
Терещенко Василь Миколайович _____
(підпис)

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

Київ – 2021

РЕФЕРАТ

Обсяг роботи 50 сторінок, 12 ілюстрацій, 3 таблиці, 18 джерел посилання.

ПЕРЕТВОРЕННЯ РУКОПИСНОГО ТЕКСТУ, МАТРИЦЯ, ЛОГІСТИЧНА РЕГРЕСІЯ, ДІАГРАМА ВОРОНОГО, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, НЕЙРОННА МЕРЕЖА, АЛГОРИТМ ПЕРЕТВОРЕННЯ.

Об’єктом роботи є алгоритм перетворення рукописного математичного тексту в математичний об’єкт.

Метою роботи є розробка алгоритму для розпізнавання рукописного математичного тексту, а саме рукописних матриць в реальному часі.

Методи розроблення: аналіз результатів навчання нейронної мережі, розробка алгоритму на основі еволюційної моделі. Інструменти розроблення: вільно поширюване інтегроване середовище розробки PyCharm IDE 2021.1.1, мова програмування Python 3.9.

Результати роботи: вивчено існуючі алгоритми та підходи в обраній галузі; розроблено та реалізовано алгоритм для виділення рядків рукописної матриці з використанням онлайн даних; створено логістичну регресію для об’єднання цифр у числа для рядка матриці; підібрано архітектуру згорткової нейронної мережі для визначення цифри; реалізовано простий графічний інтерфейс для тестування роботи алгоритму.

Розроблений алгоритм може перетворювати рукописні матриці для їх подальшого використання в обчисленнях. Також він може бути успішно застосований у процесі навчання в загальноосвітніх навчальних закладах та закладах вищої освіти.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Задача розпізнавання матриць є досить відомою та добре вивченою. Існує три класи алгоритмів для її вирішення: офлайн перетворення друкованих матриць, офлайн перетворення рукописних матриць та онлайн перетворення рукописних матриць. Розроблений нами алгоритм належить до останнього класу, так як він використовує дані, отримані під час користувацького вводу, а також може робити перетворення під час введення нових даних.

Серед підходів до вирішення поставленої задачі можна назвати: кластеризація проекцій центроїдів на осі для виділення рядків та стовпчиків [1], побудова мінімального кістякового дерева для рядків і сегментація в рядках по відступам [2].

Актуальність теми. Проблема перенесення рукописного математичного тексту в цифровий варіант виникала у кожного, і того добре відома. Як правило, математичні вирази доводиться відцифровувати вручну, друкуючи їх у тому чи іншому редакторі, що є не надто зручним процесом та займає немало часу.

Також у наш час кількість приладів, які можуть сприймати рукописний текст, стрімко зростає, як і їх зручність. Мультимедійні дошки, планшети, графічні планшети - всі ці прилади призначені для введення рукописного тексту. Однак додатків та алгоритмів, що можуть його опрацьовувати все ще не так багато і їх точність не завжди є задовільною.

В цілому проблема перетворення рукописного математичного тексту стоїть досить гостро, адже вводити текст від руки значно зручніше та швидше, але для подальших математичних операцій необхідно мати вже оброблений ввід у вигляді математичних об’єктів для їх подальшого використання.

Саме тому розробка даного алгоритму є важливим для перетворення рукописного математичного тексту в математичний об’єкт для взаємодій із ним.

Мета й завдання роботи. Метою роботи була розробка алгоритму для розпізнавання рукописного математичного тексту, а саме рукописних матриць, в реальному часі.

Для досягнення поставленої мети вирішували такі завдання:

- обрати архітектуру алгоритму, виділити його основні підчастини;
- реалізувати виокремлення рядків рукописної матриці, використовуючи дані, записані під час користувацького вводу;
- реалізувати алгоритм для об’єднання цифр у числа в рядках матриці з використанням логістичної регресії;
- спроектувати та навчити нейронну мережу для розпізнавання цифр у комірках діаграми Вороного, згенерувати датасет для оптимального навчання;
- оцінити отримані результати та порівняти з існуючими алгоритмами.

Об’єкт і методи дослідження або розроблення.

Об’єктом наших досліджень був алгоритм перетворення рукописного математичного тексту в математичний об’єкт.

Методами дослідження були: алгоритмізація, аналіз результатів навчання нейронної мережі, розробка алгоритму на основі еволюційної моделі.

Можливі сфери застосування. Розроблений алгоритм може бути використаний у різних галузях. Зокрема, алгоритм може перетворювати рукописні матриці для їх подальшого використання в обчисленнях. Також він може бути успішно застосований у процесі навчання в загальноосвітніх навчальних закладах та закладах вищої освіти.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

В цілому для перетворення матриць можна виділити три основних типи алгоритмів: офлайн перетворення друкованих матриць, офлайн перетворення рукописних матриць, та онлайн перетворення рукописних матриць. В нашій роботі ми будемо описувати алгоритм, що відноситься до останнього класу, так як він використовуватиме дані отримані під час користувацького вводу, а також зможе робити перетворення по мірі вводу нових даних.

Для початку розглянемо одне з найкращих уже існуючих рішень даної задачі [1]. Даний алгоритм може розпізнавати набагато більше математичного тексту ніж матриці. Однією з головних переваг даного алгоритму є можливість опрацьовувати математичні вирази, що складаються з декількох рядків.

Алгоритм працює в декілька етапів. Спершу вони визначають положення матриці через дужки, так як алгоритм може працювати не тільки з матрицями. Саме за допомогою ідентифікації дужок, відбувається виділення області з матрицею для подальшого опрацювання. Для сегментації рядків використовується підхід проектування матриці на вісі OX та OY. Через те, що матриці мають структуру решітки, на осях будуть з'являтись піки, на основі яких будуть виділятись рядки та стовпчики матриці.

Також дуже важливою функцією даного алгоритму є те, що він може працювати з матрицями що складаються не тільки з чисел, а й з інших математичних виразів. Ця можливість значно розширює область застосування алгоритму з точки зору роботи з матрицями.

Розглянемо й інші існуючі підходи до розпізнаванню рукописних матриць.

У [3] відбувається використання автоматичної сегментації та розпізнавання декількох виразів забезпечуються на основі алгоритму інтервалів, який використовує розпізнані ідентичності символів, розміри, та відносне розташування окремих символів. Спочатку формуються елементи матриці, роздуваючи обмежувальні рамки та об’єднуючи пересічені символи. Алгоритм розпізнає як добре, так і не добре сформовані матриці.

У [4] пропонується новий підхід розпізнавання матриць, включаючи символи повторення та символи площі. Спосіб складається з 4 частин: виявлення матриць, сегментація елементів, побудова мереж, аналіз матричної структури. При побудові мереж матриця розглядається як мережа елементів, пов'язаних між собою за допомогою зв'язків, що представляють їх відносини. Використовується підхід вивчення матриці через її горизонтальне та вертикальне проектування.

Використання мінімального кістякового дерева можна побачити в [2], сегментація відбувається по рядкам, потім рядки діляться на елементи через проміжки між ними.

Щодо офлайн перетворень матриць, та рукописних математичних виразів в загальному, можна виділити наступні підходи:

- послідовні рішення;
- інтегровані рішення;
- повне використання нейронних мереж.

Послідовні та інтегровані рішення характеризуються наявністю двох стадій перетворення: сегментація та класифікація символів, структурний аналіз отриманого результату [5].

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Нейронна мережа – (англ. *neural network, NN*) – це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин [6].

В наш час знаходяться все нові області використання нейронних мереж. Основні задачі які вони розв’язують:

- класифікація – поділ даних по класам;
- передбачення – можливість передбачити характеристики нових даних, інформації про які ще нема, наприклад необхідно передбачити вартість будинку за його характеристиками [7].

У цілому зрозуміти принцип роботи нейронної мережі досить просто проводячи аналогію якраз з мозком. Основною структурною одиницею є нейрон – обчислювальна одиниця, яка отримує певну інформацію на вхід, проводить над нею певні прості обчислення та передає далі.

Нейрони об’єднуються в шари, які є основною структурною одиницею мережі. Є три основні типи шарів – вхідний, скритий, вихідний. Вхідний шар це представлення вхідних даних, після певної обробки. Наприклад це можуть бути значення пікселів зображення. Скритих шарів зазвичай більше одного, на відміну від східного та вихідного. Саме в цих шарах відбувається перетворення вхідних даний у вихідні. Вихідний шар несе в собі результат роботи мережі, який можна використовувати далі [8].

У кожного нейрона є вхідні дані та вихідні. Для нейрону з вхідного шару вхідні дані рівні вихідним. Для нейронів усіх інших шарів його вихідне значення формується з сумарної інформації усіх нейронів попереднього шару, після чого воно нормується за допомогою функції активації.

Нейрони зв’язані між собою так званими синапсами, у яких є єдиний параметр – його вага. Чим більша ця вага, тим більшу роль відіграє нейрон з попереднього шару у формуванні значення нейрону з наступного. При ініціалізації значення ваг задаються випадковим чином. Визначення ваг синапсів і є основною задачею навчання нейронної мережі.

Функція активації відіграє досить важливу роль в навчанні, і їх правильний підбір є дуже важливим етапом проектування нейронної мережі. Основна задача функції – нормалізувати дані отримані з попереднього шару. Ідейно функція активації буде відображати на скільки значення з попереднього шару активували нейрон – чим більше значення тим більше активація [7].

Процес навчання зазвичай включає в себе аналіз вхідний даних, аналізуючи які визначаються необхідні значення параметрів моделі. Для навчання зазвичай використовують три датасети:

- трейн – дані що використовуються для навчання, за допомогою нього будуть визначатися ваги;
- тест – незалежний від трейну, допомагає не перенавчатись.

Навчання нейронної мережі відбувається в два етапи – пряме та зворотне розповсюдження похибки.

Пряме розповсюдження полягає в тому, що в нейронну мережу завантажується якесь вхідне значення, для якого відбувається прогнозування за описаним вище алгоритмом. Отримані вихідні значення використовуються для зворотного розповсюдження. На цьому етапі обраховується вектор різниці між отриманим значення з мережі та правильною відповіддю. За допомогою цього значення відбувається перерахунок ваг для мінімізації функції втрат [9].

Ще одним поняттям в навчанні є перенавчання. Це явище відбувається коли нейронна мережа замість навчання просто

запам’ятовує вхідні дані. Для того щоб уникнути цього використовують більші датасети, тест датасети. Також є можливість використовувати так званий шар DropOut який обнуляє частину вхідних даних [10].

Окремим видом нейронних мереж є згорткова нейронна мережа. Задачі, які зазвичай вирішують з допомогою цієї технології:

- пошук границь – знаходження границь об’єктів на зображенні;
- визначення об’єктів уваги – визначення об’єктів, на які би звернула увагу людина в першу чергу;
- семантична сегментація – поділ об’єктів на класи по їх структурі ще до їх розпізнавання;
- виділення частин людини на зображенні;
- розпізнавання об’єктів [11].

У цілому архітектуру зготкової нейронної мережі можна поділити на дві складові:

1 шари згортки – виділення ознак на зображенні;

2 повнозв’язні шари – шар звичайної нейронної мережі описаної вище.

Розглянемо принцип роботи шару згортки. В цілому цей шар представляє з себе набір ознак і у кожної є своє ядро.

Ядро це по суті фільтр, який проходиться по всій області попередньої ознаки і визначає нові ознаки. Розмір ядра зазвичай беруть 3х3. Ознака ж представляє з себе результат проходження ядра по ознаці з попереднього слою. Таким чином поєднавши декілька таких слоїв і обравши правильні ваги для ядра і його активації ми зможемо виділяти певні ознаки на зображенні. Наприклад після навчання по розпізнаванню на обличчі людини то одне з ядр могло б видавати найбільший сигнал на оці або роті [12].

Діаграма Вороного - це спеціальний поділ метричного простору, який визначається відстанями до заданої дискретної множини ізольованих точок цього простору [13].

Розглянемо принцип роботи рекурсивного алгоритму побудови діаграми Вороного. Нехай маємо множину точок S відсортованих по X координаты, для якої необхідно побудувати діаграму. Тоді етапи алгоритму будуть виглядати наступним чином:

- ділимо множину S на дві приблизно однакові частини S_1та S_2;
- продовжуємо ділити підмножини, доки не дійдемо до дна рекурсії, тобто в кожній підмножині не залишиться одна або дві точки;
- в множинах де залишилось дві точки будуємо діаграму Вороного по означенню;
- починаємо рухатись по дереву розбиття множини вгору, об’єднуючи побудовані діаграми Вороного, доки не дійдемо до кореня дерева [14].

Алгоритм об’єднання діаграм Вороного описувати в нашій роботі ми не будемо, однак ознайомитись з тонкощами його виконання можна за посиланням в літературі [15].

Регресія (лат. *regressio* - зворотній хід) в теорії ймовірностей та математичній статистиці - одностороння стохастична залежність між випадковими змінними [16]. Тобто, математичний вираз, відображаючий зв’язок між залежною і незалежною змінною при умові, що вираз буде мати статистичну значимість. На відміну від функціональної залежності, коли кожному значення незалежної змінної ставиться у відповідність одне значення залежної, при регресійному зв’язку одному і тому ж значенню *x* можуть відповідати різні значення *y*.

В нашій роботі ми будемо використовувати логістичну регресію. Логістична регресія це статистична модель, що використовується для

прогнозування ймовірності виникнення деякої події шляхом його порівняння з логістичною кривою [17]. На відміну від звичайної регресії, в методі логістичної регресії не відбувається передбачення значення числової змінної на основі вибірки вхідних значень. Замість цього, значення функції є ймовірність того, що дане значення належить до певного класу.

Основна ідея логістичної регресії полягає в тому, що простір вхідних значень може бути розділений прямою на дві області, відповідаючи вхідним класам.

РОЗДІЛ 3. ЗАПРОПОНОВАНИЙ МЕТОД РОЗВ'ЯЗАННЯ ЗАДАЧІ

Для розв'язання поставленої задачі нами було розроблено оригінальний алгоритм для онлайн перетворення рукописної матриці в математичний об'єкт.

З початку буде розглянуто загальну ідею алгоритму, не вдаючись в тонкощі реалізації окремих його частин. Це допоможе зрозуміти загальну суть розробленого підходу до розв'язання поставленої задачі.

В загальному наш алгоритм можна поділити на п'ять основних частини, які по суті теж є алгоритмами (Рис. 3.1):

1. виділити рядки рукописної матриці, використовуючи дані, записані під час користувацького вводу;
2. в кожному рядку виділити області цифр, щоб у кожній знаходилась рівно одна цифра, і не було жодної без цифри;
3. в кожному рядку визначити, які цифри мають бути об'єднаними і сформувати число, а які мають залишитись окремо;
4. визначити яке саме число написано в кожній області;
5. поєднати всі дані, отримані на попередніх етапах алгоритму, та відновити рукописну матрицю як математичний об'єкт.

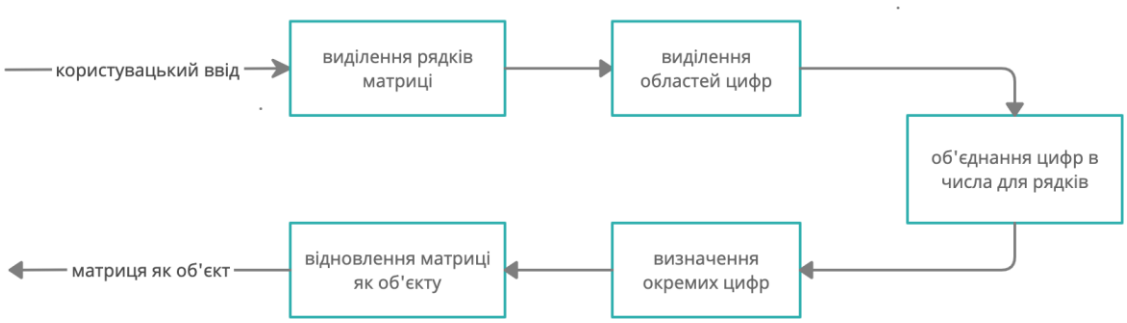


Рисунок 3.1 – Загальна схема роботи алгоритму

Для перших двох кроків відбувається запис дотиків користувача до екрану, штрихів які були відображені. Враховуючи висоти штрихів та

їх центри, на їх основі будуть формуватися рядки матриці. Тут може бути два випадки. Якщо центр даного штриха знаходиться у середині вже виділеного рядка, його буде додано до нього та опрацьовано. Якщо ж рядка знайдено не було, буде створено новий. Таким чином використовуючи дані користувацького вводу ми зможемо виділити які цифри формують рядки, і при цьому не залежати від самого формату вводу. Тобто даний підхід працює однаково добре незалежно від того що вводиться спочатку: рядки, стовпчики, або ж взагалі випадковий порядок.

Наступним етапом є виділення областей з цифрами. Тут так само використовуються записані дані по діям користувача, на основі яких кожній цифрі ставиться у відповідність її центр - точка, яка має знаходитись всередині цифри. Далі для виділених центрів будується діаграма Вороного. Саме комірки цієї діаграми будуть містити цифри матриці, і будуть інтерпретуватися як область цифри. Завдяки тому, що на попередньому етапі було виділено рядки матриці, ми можемо побудувати ланцюжки комірок діаграми Вороного для кожного рядка матриці.

Далі необхідно об’єднати деякі цифри, що знаходяться поруч в одне число, якщо вони йдуть досить близько. Для цього було побудовано логістичну регресію, яка по нормалізованим відстаням між цифрами може визначити чи необхідно їх об’єднувати в одне число.

Наступний етап - це визначення, яка саме цифра знаходиться в кожній комірці діаграми Вороного. Для цього було навчено згорткову нейронну мережу, яка може визначати цифри на зображенні. Для більшої точності моделі було згенеровано власний датасет.

Останнім етапом є відновлення матриці. Відбувається ітерація по кожному рядку матриці та по кожній комірці діаграми Вороного. Цифри

в комірках об’єднуються в числа, якщо створеною логістичною моделлю визначено, що вони мають йти разом.

Також слід зазначити, що розроблений алгоритм не використовує дужки матриці для її розпізнавання, адже навіть без них структура матриці є визначеною і для перетворення ними можна знехтувати. Тому в нашій роботі використовуються лише записи самих рядків з числами. Таким чином, в усіх прикладах що наведено в роботі матриці будуть зображуватися без дужок.

На рисунку 3.2 можна побачити приклад виконання перших двох описаних етапів на рукописній матриці. Червоні лінії сполучають центри цифр у рядках, які були виділені з користувацького вводу. Зелені відображають верхню та нижню границі рядків, саме по ним відбулося виділення рядків. Сині лінії відображають поділ площини на області, по яким далі буде відбуватися визначення написаних цифр. Цифри в кожній комірці позначають її положення: перша цифра індекс рядку, друга порядок в рядку.

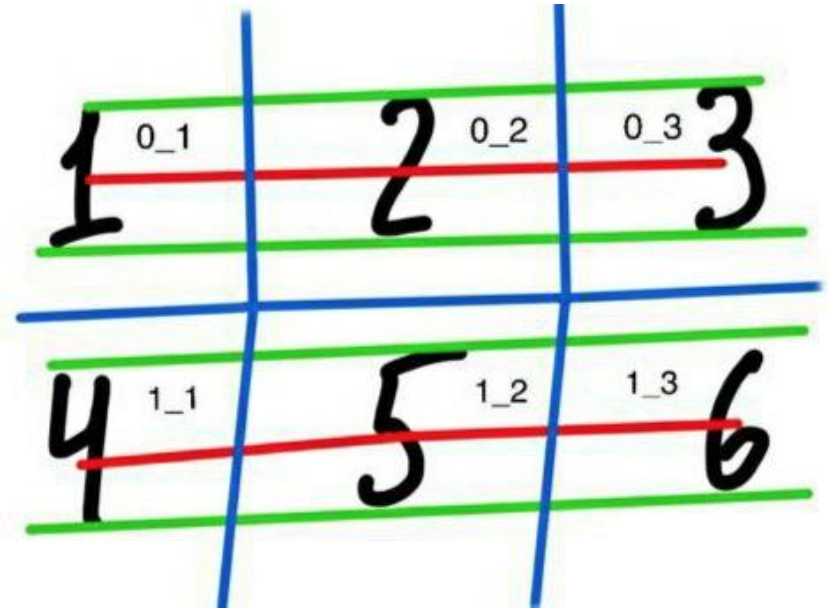


Рисунок 3.2 – Результат виокремлення рядків та виділення областей цифр рукописної матриці

РОЗДІЛ 4. ВИОКРЕМЛЕННЯ РЯДКІВ У РУКОПИСНІЙ МАТРИЦІ

Як уже зазначено в попередньому розділі, першим кроком для перетворення рукописної матриці в математичний об’єкт було вирішено зробити виокремлення окремих рядків матриці. Виокремив рядки, а в подальшому і числа в них, ми можемо визначити і стовпчики матриці відповідно. Сегментувати стовпчики можна буде обравши числа, що стоять під певним індексом в кожному рядку. Наприклад, якщо ми захочемо виокремити стовпчик під номером 2 ми можемо взяти числа, що стоять другими у всіх рядках. Така проста маніпуляція дозволить нам отримати стовпчики, попередньо виокремивши та сегментувавши рядки.

Під виокремленням рядка будемо вважати алгоритм, який кожному рядку рукописної матриці поставить у відповідність:

1. вектор, що проходить через увесь рядок матриці, та знаходиться у напрямку введення рядка;
2. висоту рядка, що буде відображати відхилення по осі OY від його вектора.

4.1 Виокремлення рядків матриці на основі онлайн записів

В нашій роботі ми вирішили реалізувати виокремлення рядків у матриці, використовуючи дані, записані під час користувацького введення. На нашу думку, з користувацького вводу можна записати достатньо даних для відокремлення рядків у рукописній матриці з високою точністю.

Під час вводу матриці користувачем, програма буде відслідковувати, обробляти і зберігати всі зроблені штрихи. Штрихом у

нашій роботі будемо називати фігуру, намальовану користувачем одним дотиком до екрану, не відриваючись від нього під час проведення.

Ключовою характеристикою штриха буде його так званий центр. Центри штрихів будуть відігравати ключову роль в подальших перетвореннях матриці, так як на їх основі будуть виділятись рядки та відокремлюватись цифри. Центр має виконувати наступні умови:

- знаходитись максимально близько до умовного поняття центру штриха, тобто бути всередині фігури;
- кожна цифра в матриці має мати тільки один центр, не може бути центру без цифри що йому відповідає.

У цілому для кожного штриха будемо розглядати та зберігати наступні характеристики:

- максимальна та мінімальна висота - максимальне та мінімальне значення по осі Y відповідно, яке здобувалось в процесі проведення штриха;
- максимальне ліве та праве відхилення - мінімальне та максимальне значення по осі X відповідно, яке здобувалось в процесі проведення штриха;
- центр - точка, що знаходиться по центру від максимальних відхилень по вертикалі та по горизонталі, тобто рівновіддалена від країв.

Головна властивість центру штриха визначеного таким чином, що він завжди буде знаходитись всередині цифри, або ж досить близко до цього. На рисунку 4.1 можна побачити приклади центрів цифр в нашому розумінні, і переконатись що вони задовольняють поставленій умові.

Рисунок 4.1 – Центри штрихів для цифр 2, 3, 6 відповідно; червона точка - центр

Проблема з якою ми зіштовхнулись, що не всі цифри пишуться одним дотиком до екрану, а можуть складатись з кількох штрихів. Це може бути справжньою проблемою, адже це порушить умову, яку ми наклали на центр штриху - те, що він має бути один на цифру, тобто кількість центрів у рядку має бути рівна кількості цифр у відповідному рядку.

Вирішення проблеми полягає в більш детальному аналізі цифр, з якими виникає така проблема, а саме: 1, 2, 4, 5, 7. Для всіх цифр, окрім 4, додаткові штрихи є горизонтальними прямими. Для 1 та 2 це може бути ще один штрих знизу, для 5 це може бути окремий штрих згори, для 7 це може бути окремий штрих по центру. Також, нескладно помітити, що в арабському запису немає цифр, для яких горизонтальна пряма є ключовою. Таким чином, ми можемо просто ігнорувати горизонтальні штрихи на етапі запису центрів та відокремленню рядків. Як приклад, на рисунку 4.2 можна побачити центри для цифри 7 в обох варіаціях та переконатись, що покладені на нього умови виконуються в обох випадках.

а) б)

Рисунок 4.2 – для можливі написання цифри 7: а - без штриху, б - з штрихом

Щодо цифри чотири, то поки що у випадку її написання за допомогою двох штрихів, будемо розглядати їх як два дотики і відповідно два центри. Вони нам не будуть заважати для виокремлення рядків, а додаткову обробку застосуємо на наступних кроках.

Основна ідея по виокремленню рядків використовуючи уже записані та оброблені дані полягає у виділенні рядків по висоті. Обробляючи штрихи одни за одним, ми будемо відносити їх або до уже сегментованих рядків, або ініціалізувати нові.

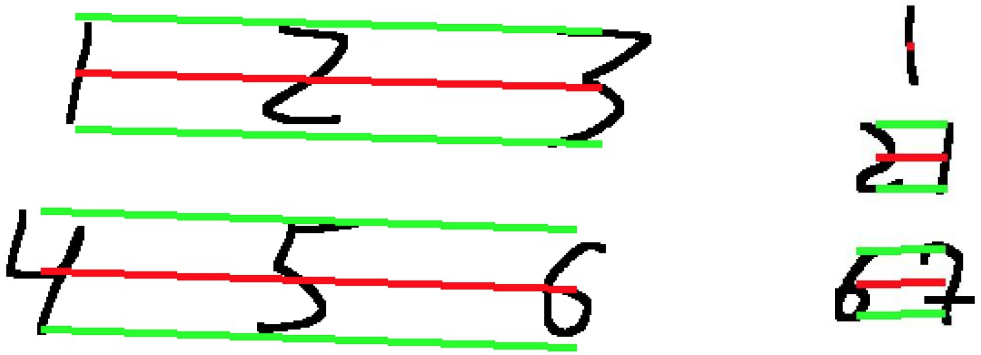
Ітерація відбувається по відображеним штрихам. Для кожного з них перевіряється, чи знаходиться даний центр в уже відокремленому рядку. Якщо такий рядок було знайдено, новий центр додається до нього та обробляється. Під обробкою маються на увазі наступні перерахунки:

- перевірка чи не змінить новий штрих висоту рядка та оновити її при необхідності;
- перерахунок вектора рядка, враховуючи новий центр.

Якщо ж підходящого рядка знайдено не було, то такий буде створений. Відповідно його висота буде ініціалізована висотою штриха, а вектор буде починатися і закінчуватися в центрі штриха.

З псевдокодом методу, який виокремлює рядки описаним алгоритмом можна ознайомитись в додатку А.

На рисунку 4.3 можемо побачити результат роботи алгоритму на різних матрицях. Червона лінія відображає фінальни вектор рядка, який сполучає центри першої та останньої цифри. Зелені ліній відображають верхні та нижні межі рядків. У випадку, що зображено на рисунку 3.3б, цифра 1 позначена просто червоною точкою (без прямої), так як вона єдина в рядку і це її вектор.



а) б)

Рисунок 4.3 – приклади виділення рядків у рукописній матриці: а - матриця з двома рядками та трьома стовпчиками, б - вектор

4.2 Програмна реалізація виокремлення рядків рукописної матриці

Обробка рядків матриці, як і вся наша робота, була написана на мові програмування Python 3.9. Розглянемо основні аспекти алгоритму по виокремленню рядків матриці.

Для програмного відображення рядків матриці, їх обробки та збереження було створено два класи - MatrixRow та RowsManager. Клас MatrixRow є програмним відображенням рядка матриці, з усією необхідною інформацією про нього та необхідним інтерфейсом. Клас RowsManager відповідає за рядки на вищому рівні, зберігаючи їх та

маючи змогу працювати з ними. Також даний клас було обгорнуто в декоратор singleton, адже він має існувати єдиний у програмі, а декоратор допоможе не продукувати копії і завжди мати швидкий доступ до єдиного об’єкту класу.

Розглянемо більше детально клас MatrixRow. Кожен екземпляр цього класу відповідає розпізнаному рядку в рукописній матриці, та зберігається в класі RowsManager. Поля класу:

- index - унікальний ідентифікатор рядка;
- vectors_list - список векторів, що сполучають центри штрихів, що утворили даний рядок;
- cos_list - список косинусів кутів між центрами штрихів, що утворили даний рядок;
- top - максимальне відхилення вверх від центру штриха, по суті буде формувати верхню межу рядка;
- bottom - максимальне відхилення вниз від центру штриха, по суті буде формувати нижню межу рядка;
- _top_dot - точка, що утворила нинішню верхню межу, використовується для перерахунку нової межі при додаванні нового штриху до рядка;
- _bottom_dot - точка, що утворила нинішню нижню межу, використовується для перерахунку нової межі при додаванні нового штриху до рядка.

Методи класу MatrixRow:

- property final_vector - вектор, що сполучає центри першого та останнього штрихів, що утворюють даний рядок;
- add_hatch - додавання нового штриха до рядка, додасть новий вектор с писок та обновить верхню та нижні межі рядка;

- is_dot_inside_row - поверне булеве значення, чи передана методу точка знаходиться всередині даного рядка, використовується при перевірці чи вже існує рядок для даного центру;
- _update_top - перераховує верхню межу рядка при додаванні нового штриха;
- _update_bottom - перераховує нижню межу рядка при додаванні нового штриха.

Щодо класу RowsManager то як уже зазначено, його головна роль це зберігати рядки та розподіляти дані між ними. Його поля:

- vectors - список усіх векторів між центрами штрихів, не поділено між рядками;
- dots - список усіх центрів штрихів, що були образовані на матриці;
- cos_list - список усіх косинусів між векторами, що сполучають центри штрихів;
- top_list - список всіх крайніх верхніх точок для кожного штриха;
- bottom_list - список всіх крайніх нижніх точок для кожного штриха;
- rows - список об’єктів класу MatrixRow, всі виділені рядки.

Методи класу RowsManager:

- add_new_hatch - зберегти новий штрих в RowsManager, ніякої обробки;
- process - обробити всі збережені штрихи, виокремивши рядки матриці зберігши їх у rows полі;
- clear - очистити всі збережені та оброблені дані;
- _add_hatch_to_current_row - метод який збереже штрих в поточному рядку;

- _process_hatch_not_current_row - метод який буде шукати інсуючий рядок для штриху, або ж створить новий;
- is_hatch_horizontal - поверне булеве значення чи є даний штрих горизонтальним.

Клас MatrixRow на пряму майже не використовується, всі взаємодії відбувають через MatrixRow клас. Так наприклад у вікні програми відбувається збереження штрихів через виклик MatrixRow.add_new_hatch, коли користувач відпускає натиск. Коли ж треба виконати обробку збережених даних та виділити рядки, то викликається MatrixRow.process.

РОЗДІЛ 5. ГРУПУВАННЯ ЦИФР У ЧИСЛА ДЛЯ РЯДКА РУКОПИСНОЇ МАТРИЦІ

Після виокремлення рядків в матриці необхідно обробити кожен рядок окремо, та визначити числа що в ньому знаходяться. Обробивши кожен рядок та отримавши числа в цьому ми з легкістю можемо відновити всю матрицю, що і є нашою кінцевою метою.

Далі будемо розглядати роботу алгоритмів на прикладі одного конкретного сегментованого рядка матриці. Описавши алгоритм таким чином, ми можемо запустити його на всіх рядках матриці та отримати необхідні дані.

Таким чином однією з задач на шляху перетворення рукописної матриці в математичний об’єкт є групування цифр в рядку в числа. Іншими словами необхідно описати правила, по яким алгоритм може при необхідності групувати цифри в числа об’єднуючи їх, якщо вони стоять досить близько і формуть одне ціле. Так, на приклад, алгоритм має розрізняти рядки 1 2 3 4 та 12 34.

Одним з найважливіших властивостей створеного алгоритму має бути не чутливість до масштабу рядка і відповідно цифр у ньому. Оскільки наша програма працює з користувацьким вводом, масштаб матриці не буде постійною величиною. Саме тому, найочевидніший підхід знайти константу відстані між цифрами меньше якої вони можуть вважатись разом, а більше окремо не підходить.

Алгоритмів які можуть розв'язати поставлену задачу знайдено не було. Існують схожі алгоритми для букв [18], однак там активно використовується специфіка написання літер та не мало евристики навколо цього, тому дана робота нам не може сильно допомогти.

5.1 Об’єднання цифр в числа з використанням діаграми Вороного

Наша початкова ідея полягала у використанні діаграми Вороного для групування цифр в числа для рядка рукописної матриці. Задум був у тому, щоб визначити середню ширину цифри для рядка і потім міряти відстані між цифрами в цьому рядку і дивитись скільки цифир могло б поміститись у проміжку. Таким чином можна спробувати отримати константу ширини яка буде відображати межу відстані між тим що цифри формують окремі числа, та тим що вони йдуть разом.

Головна властивість такої константи буде в тому, що вона має стосуватись конкретного рядка і буде під нього обрахована. Саме такий підхід задовольняє початкову умову по не чутливості до масштабу матриці.

Якщо говорити більш детально про алгоритм, то розглянемо його по частинам. Будемо вважати що частина розпізнавання матриці по виокремленню рядків уже виконана, а отже ми маємо центри цифр та знаємо які з них утворюють рядки, а також їх порядок відносно матриці.

Першим кроком нам необхідно побудувати діаграму Вороного для знайдених центрів. Оскільки ми ще раніше наклали умови того, що центр має бути єдиний для цифри і що він має знаходитись всередині, то дана діаграма Вороного буде побудована не тільки для центрів, а і для самих цифр. Приклад такої побудови наведено на рисунку 5.1.

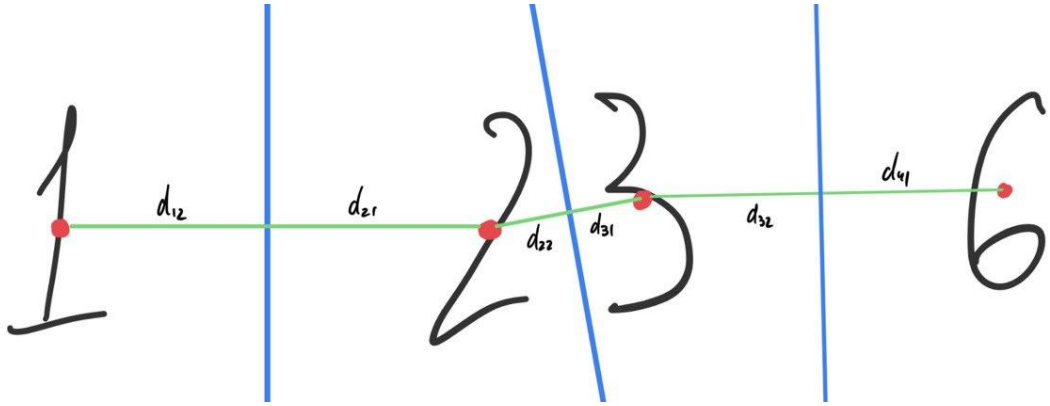


Рисунок 5.1 – приклад діаграми Вороного для одного рядка матриці

На рисунку сині лінії відображають сторони комірок діаграми Вороного побудованої навколо центрів цифр позначених червоним кольором.

Наступним ключовим значенням для алгоритму є так звана ширина цифри. Це значення було вирішено визначити як мінімальна з відстаней від центру цифри до комірок діаграми помножена на два. Якщо цифра є крайньою в рядку, то береться єдина відстань та множиться на два. Так на приклад для рисунка 5.1 ширина цифри один буде $d_{12} * 2$, а цифри три $d_{31} * 2$. Далі вираховується середнє значення ширини цифри в рядку. Наступним кроком визначаються відстані між центрами цифр, і діляться на середнє значення ширини цифри.

Таким чином для кожного пропуску між цифрами ми обрахували певне значення, що відображає його відносний розмір. Далі необхідно було знайти таке значення відносної відстані, відстані менше якого відстані будуть відповідати цифрам що мають бути об’єднаними. Але подальші перевірки алгоритму показали, що він не буде працювати у всіх випадках. Даний алгоритм не може коректно опрацьовувати рядки, в яких всі цифри йдуть разом або окремо. В такому випадку алгоритм буде поводити себе не стабільно і не давати правильних результатів незалежно від початково обраної константи.

Так наприклад якщо в матриці перший рядок складає число 123, алгоритм не завжди буде його як одне число, незалежно від того наскільки щільно вони знаходяться.

Дану проблему можна помітити більш пильно подивившись на алгоритм. Якщо відстані між цифрами однакові, то і відстані в діаграмі Вороного будуть близько рівними до відстаней між центрами. Таким чином відносні відстані будуть варіюватись біля значення 1, не сильно від нього відхиляючись.

Коли ж в рядку є принаймні дів цифри які треба об’єднати, то їх комірки діаграми Вороного починають стягувати середню ширину цифри вниз, та інші відстані на контрасті починають визначитися правильно.

На жаль підходу до вирішення даної проблеми знайдено не було, того даний підхід до об’єднання не був впроваджений в кінцевий варіант алгоритму.

Не зважаючи на це, частина коду та логіки яка була реалізована для цього підходу все ж потрапила у фінальну версію алгоритму. Було вирішено залишити частину з побудовою діаграми Вороного для центрів цифр, та їх розбиття на рядки. Використовуючи побудовані комірки діаграми в яких знаходяться цифри по означенню, ми зможемо визначати яка саме цифра написала всередині.

5.2 Об’єднання цифр в числа з використанням логістичної регресії

Оскільки описаний вище алгоритм не зміг виправдати очікування, було вирішено розробити та протестувати інший підхід до об’єднання цифр в числа.

Як і в попередньому алгоритмі, ідея заключалась в тому, щоб якимось чином поділити множину можливих відстані між цифрами на

дві підмножини - відстані які знаходяться між цифрами що мають бути об’єднаними, та ті що мають йти окремо. Умова на те, що алгоритм має масштабуватись на різні розміри рядка також справедлива.

Для початку, було вирішено позбутись проблеми з масштабом рядка нормалізувавши його розміри. Оскільки з записів під час користувацького вводу матриці ми знаємо висоту рядка, ми можемо нормалізувати по ній. Зробити ми це можемо поділивши всі відомі розмірності рядка на його висоту, відповідно звівши висоту до одиниці.

Після таких перетворень в новій системі розмірностей рядка ми можемо вводити константи, які матимуть однаковий сенс для всіх нормалізованих таким чином рядків.

Нехай висота рядка була рівна h, і введемо нове значення $k = \frac{1}{h}$. Це і буде наш коефіцієнт масштабування. Саме на це значення будуть помножені розмірності рядка: висота, та відстані між центрами цифр.

Під час нормалізації розмірностей рядка, також було нормалізовано відстані між центрами штрихів. Було висунуто припущення, що якщо відобразити залежність нормалізованих відстаней від коефіцієнта масштабування k, ми зможемо розділити дві підмножини відстаней логістичною регресією з досить високою точністю.

Для перевірки даного припущення, нам був необхідний датасет. Він має містити наступні значення для кожного рядка:

- висота - відстань між верхніми пікселями чисел в рядку та нижніми;
- відстані між цифрами, бажано в приблизно в тому ж розумінні центру цифри, що ми означили вище.

На жаль, такого датасету знайдено не було, тому було вирішено написати певне розширення до вже існуючого графічного інтерфейсу та

алгоритму по виділенню рядків, щоб згенерувати такий датасет самостійно.

Для цього було написаний простий скрипт, який після вводу рядка матриці знаходив в ньому центри цифр, його висоту, записував ці дані в json файл, та очищав екран для нового вводу. Таким чином можна було швидко ввести багато рядків матриці, обробити їх, та записати необхідні нам дані в файл. Для вивчення правдоподібності моделі було згенеровано за збережено дані по 25 різним рукописним рядкам матриці, виконаних в різних масштабах для більшої достовірності даних.

Оскільки алгоритму який міг би розподілити ці відстані на класи разом та окремо в нас на цей момент ще не було, ці дані необхідно було вводити в файл вручну. Таким чином кожна відстань між цифрами була промаркова і віднесена до одного в двох класів - вона знаходиться між цифрами що йдуть разом, або окремо.

Згенерувавши датасет по описанії технології, отримані дані було візуалізовано на графіку (Рис. 5.2). На ньому було відображено залежність відстані між цифрами, від k- коефіцієнту масштабування рядка. Відстані, що знаходяться між цифрами йдуть разом, позначені оранжевим, ті що знаходяться між цифрами йдуть окремо - синім.

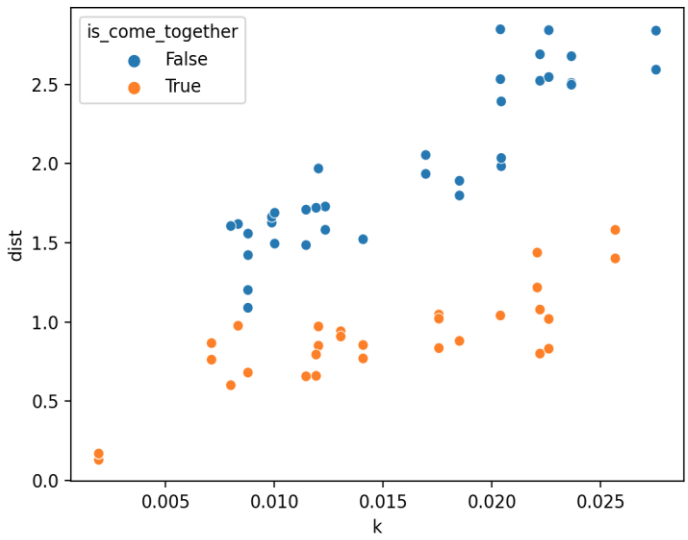


Рисунок 5.2 – розподілення відстаней між цифрами в залежності від коефіцієнту масштабування

Так, на графіку можна чітко побачити, що дві множини відстаней чітко розподіляються прямою лінією. Таким чином можемо сказати, що наше припущення про залежність є вірним, але потребує додаткової перевірки.

Далі необхідно побудувати лінійну регресію, для подальшої класифікації відстаней і використання моделі. Для побудови лінійної регресії було використано бібліотеку sklearn. Отриману регресію можна побачити на рисунку 5.3. Для більш зручного використання коефіцієнта k, його значення було помножено на 1000. На модель це ніяк не впливає, адже цей коефіцієнт відображає масштабування. Все що необхідно буде замінити, це під час перевірки моделі також використовувати значення $k * 1000$.

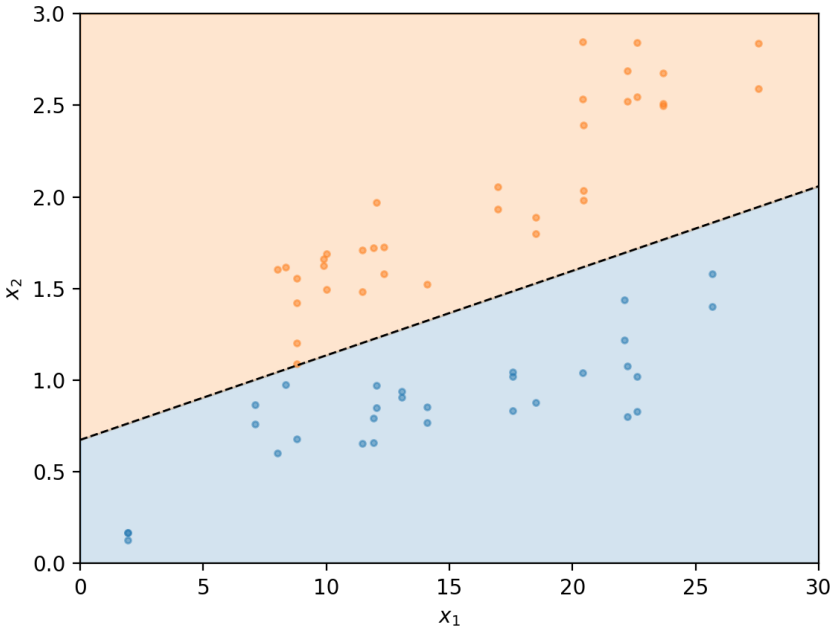


Рисунок 5.3 – візуалізація лінійної регресії для згенерованого датасету

Якщо казати про рівняння прямої, яке ділить дві множини, то воно рівне $x2 = 0.0461394 * x1 + 0.67440243$, де $x2$- відстань між центрами, $x1$- коефіцієнт масштабування рядка.

Далі, для перевірки достовірності створеної моделі, було згенеровано та збережено дані по ще 10 рядкам матриці. На цих даних було запущено нашу модель і перевірено точність результатів. З усіх переданих на тестування відстаней, 97.67% моделлю було визначено правильно.

5.3 Програмна реалізація об'єднання цифр в числа

Даний алгоритм для побудови регресії, та перевірки підходу з використанням діаграми Вороного було написано на мові програмування Python 3.9. Для початку коротко розглянемо особливості створення рішення з використанням діаграми Вороного.

Оскільки центри цифр ми вже знайшли, побудувати діаграму Вороного навколо них не є складною задачею. Оскільки даний алгоритм так і не потрапив у фінальну версію, для побудови діаграми Вороного було використано бібліотеку sciру. Вона приймає на вхід точки, і на вихід дає список даних по кожній комірці діаграми. Вихідної інформації цілком достатньо для наших цілей.

Оскільки діаграма Вороного будується по центрам цифр, які вже розподілені по рядкам матриці, ми з легкістю можемо так само розподілити і комірки діаграми співставивши їх центри. Також ми знаємо порядок центрів, так що корміки також можна упорядкувати. Так, ми будемо знати які комірки утворюють які рядки матриці, які з них є крайніми.

Ключевими класами для роботи з діаграмою в цілому та її комірками, є VoronoiTesselation, Cell, CellNeighbor. Розглянемо їх ключові поля те методи.

Класс VoronoiTesselation відповідає за перетворення та збереження комірок на високому рівні. Він отримує на вхід центри фігур, та по виклику методу process запускає обробку отриманих даних та побудову діаграми.

Клас Cell відповідає комірці діаграми вороного. Розглянемо його ключові поля:

- id - унікальний ідентифікатор комірки;
- center - центр цифри, навколо якого було побудовано комірку;
- is_closed - true, якщо комірка є крайньою в рядку, накладає інші умови на обрахунок ширини цифри;
- neighbors - список з CellNeighbor класів, зберігає сусідкі комірки з даною;
- digit_width - обрахована ширина цифри.

Щодо класу CellNeighbor, він відображає ключову інформацію про сусідні комірки для даної. Безпосередньо, він зберігаю ідентифікатор сусіда, та дані межу між ними. Границя між комірками зображується як дві точки, які лежать на ній. Оскільки нас буде цікавити не скільки сама стінка комірки, а відстань до неї, то такого відображення буде досить.

Тепер розглянемо реалізацію створення регресійної моделі. Для початку розглянемо код, який було написано для побудови моделі. Як уже було сказано раніше, для уже існуючого графічного інтерфейсу було створено додатковий скрипт який дозволяє швидко і легко малювати за зберігати дані по рядкам матриці. Отримана інформація зберігалась в json файл.

Далі було написано інший скрипт, який використовує збережені дані, візуалізую їх за допомогою бібліотеки matplotlib, будує регресію, візуалізує її, та переверія точність. Сам код простий та прямолінійний, того більш детально говорити про те, як він працює ми не будемо.

Отримана модель, а саме рівняння прямої що її описує, була вбудована в уже існуючий клас MatrixRow. До цього було додано нові методи:

- is_dots_come_together - повертає істину, якщо цифри мають бути об’єднані, саме тут захардкоджені значення отримані з логістичної моделі;
- predict_neighbor_cells - викликає метод is_dots_come_together для всі цифр рядка, зберігає отримані результати.

Таким чином, ми можемо дізнатись які центри цифр, а відповідно і самі цифри, мають бути об’єднаними в одне число, а які мають йти окремо.

РОЗДІЛ 6. НЕЙРОННА МЕРЕЖА ДЛЯ ВИЗНАЧЕННЯ ЦИФР

Наступним кроком в нашому алгоритмі є розпізнавання цифри, що знаходиться в комірці діаграми Вороного. Для рішення даної задачі було обрано зготкову нейронну мережу, так як вона може дати досить непогані результати для класифікації зображень.

Важливою частиною цього етапу алгоритму є те, що саме тут буде додана додаткова обробка необхідна для правильної ідентифікації цифри чотири, якщо вона була написана в два дотики. Ідея полягає в тому, щоб навчати модель для розпізнавання всіх 10 цифр, та ще частини цифри чотири яка буде намальована окремо (лівої частини, що складається зі штрихів вниз і вправо). Таким чином наша модель має розпізнавати 11 об’єктів.

Оскільки ми додали додаткове розпізнавання для частини цифри чотири, то таке написання буде розпізнане як додатково введений об’єкт і одиниця, адже це саме те що залишилось від написання цифри. Оскільки ці об’єкти знаходяться дуже близько один до одного, наша логістична модель об'єднає їх в одне число. Таким чином, на етапі відтворення матриці як математичного об’єкту ми можемо додати додаткову умову, що якщо було розпізнано частину цифри чотири, одиницю, та логістична модель визначила їх як об’єкти що йдуть разом, то вони будуть заміщенні на цифру чотири.

6.1 Навчання нейронної мережі для визначення цифр

Як уже зазначено, для рішення поставленої задачі було обрано згорткову нейронну мережу. Причинами такого вибору можна назвати досить високу стабільність роботи, гнучкість до вхідних даних на відміну від повнозв’язної мережі.

В цілому нейронна мережа що розпізнає цифру на зображенні є мабуть найпопулярнішою моделлю для новачків. Відповідно датасетів для навчання існує досить багато. Для початку було вирішено спробувати використати якраз існуючі датасети, щоб не витрачати час на генерацію власного. Обрано було mnist датасет з python бібліотеки keras. Датасет включає 60000 навчальних зображень цифр від 0 до 9 розміру 28 на 28 пікселей в чорно-білому варіанті (один канал). Як додаткова обробка було використано лише нормалізацію вхідних значень на проміжок [0, 1].

Сама обрана модель має наступну архітектуру.

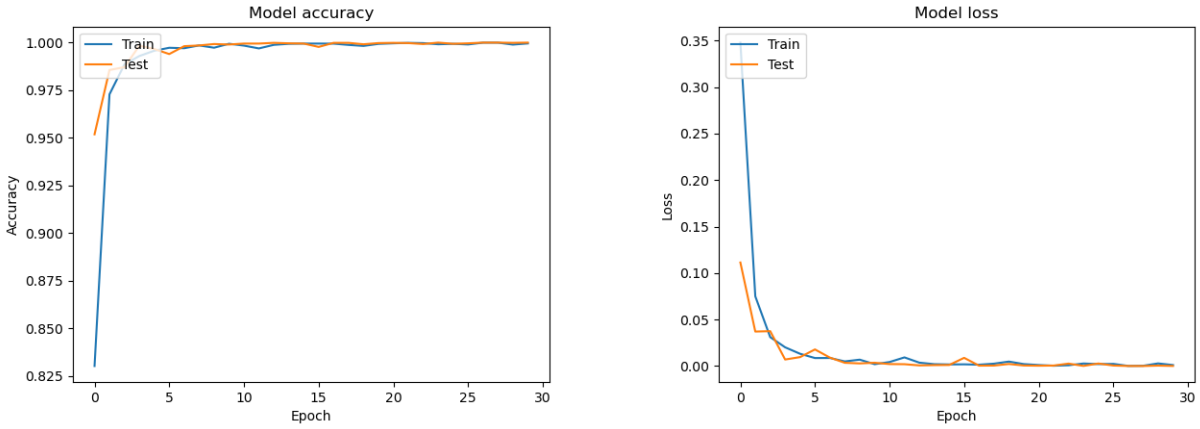
1. convolution layer – filters – 32, kernel size - (3, 3), activation – relu;
2. max-pool – pool-size – (2, 2);
3. convolution layer – filters – 64, kernel size - (3, 3), activation – relu;
4. max-pool – pool-size – (2, 2);
5. flatten
6. dropout – rate – 0.2;
7. dense – units – 128, activation – relu;
8. dropout – rate – 0.6;
9. dense – units – 10, activation – softam.

Тобто спочатку ми збільшуємо кількість ознак, та зменшуємо розмір самого зображення для кращої концентрації ознак. Далі ми згладжуємо отримані результати в один вектор і використовуємо повнозв’язну нейронну мережу зі зменшенням ширини. Додано досить багато шарів відкидання для того щоб модель не перенавчалась. З більш детальним аналізом архітектури можна ознайомитись в додатку Б.

Щодо тестового датасету, то його було згенеровано з використанням уже існуючих частин розробляємого алгоритму та

додаткового скрипту. Цей скрипт запускав розпізнавання рядків та побудову діаграми Вороного, що вже було описано. Далі алгоритм виділяє частину зображення всередині комірки вороного та зберігав його як файл. Далі ці файли були вручну промарковані цифрами, які були на них зображені. Таким чином було створено датасет розміром 125 зображень. Даний датасет точно відображає вхідні дані, які буде отримувати нейронна мережа в нашому алгоритмі, тому максимізувати точність но цьому датасеті є дуже важливо.

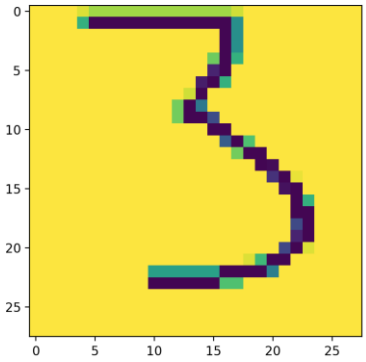
Датасет для навчання після підготовки завантажується в модель та починається процес навчання, який триває 30 епох. Були отримані наступні графіки похибки та точності (Рис 6.1).



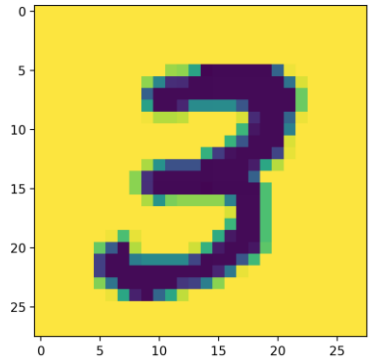
а) б)

Рисунок 6.1 – Результати навчання: а – точність, б – втрати

Однак на згенерованому тестовому датасеті була отримана точніть в 71%, що є дуже мало. Скоріше за все причиною цієї неточності є те, що датасет на якому відбулося навчання має досить суттєві відмінності від тестового, а саме: різні розміри цифр, їх ширина. Приклад різниці можна побачити на рисунку 6.2.



а)



б)

Рисунок 6.2 – Приклади зображень цифри 3: а - тестовий датасет, б - keras mnis датасет

Далі було прийнято рішення, що буде простіше створити свій датасет для навчання ніж шукати той, який буде підходити під наш тестовий. Для цього було використано вже існуючий графічний інтерфейс та додано до нього невеликий скрипт для швидкого введення та збереження даних. Суть скрипту в тому, що перед введенням треба захардкодити цифру яка зараз буде малюватись, для того щоб розкидати зображення по різним директоріям. Далі запускається вікно для малювання, яке на натискання клавіші зберігає намальоване зображення за очищає екран. Таким чином можна намалювати та зберегти до 100 унікальних зображень цифри за 4-5 хвилин.

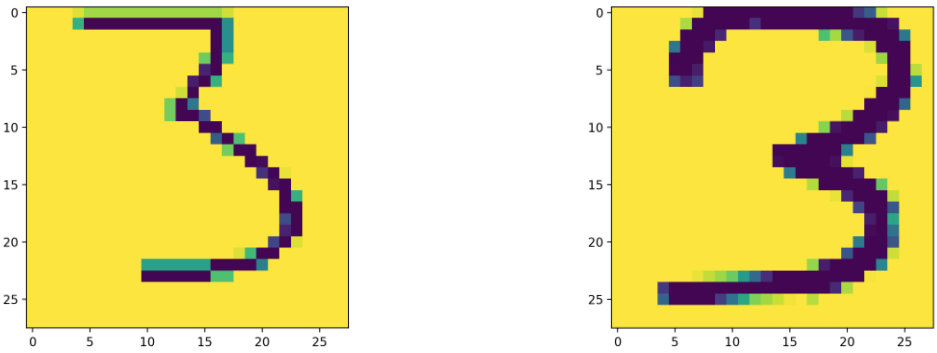
Таким чином було згенеровано датасет на 1200 зображень, тобто по 120 на цифру. Очевидно що цього буде замало для якісного навчання моделі, того було вирішено використати технологію аугментації датасету. З кожного вхідного зображення було згенеровано 11 нових, використовуючи наступні техніки:

- поворот вхідного зображення на кут що не перевищує 15˚;

- зміна масштабу вхідного зображення в межах 10% від початкового;
- стиснення та розтягнення по висоті та ширині на 15% порівняно з початковим зображенням.

Таким чином було згенеровано новий датасет, який уже містив 13200 зображень, тобто по 1320 зображень кожної цифри.

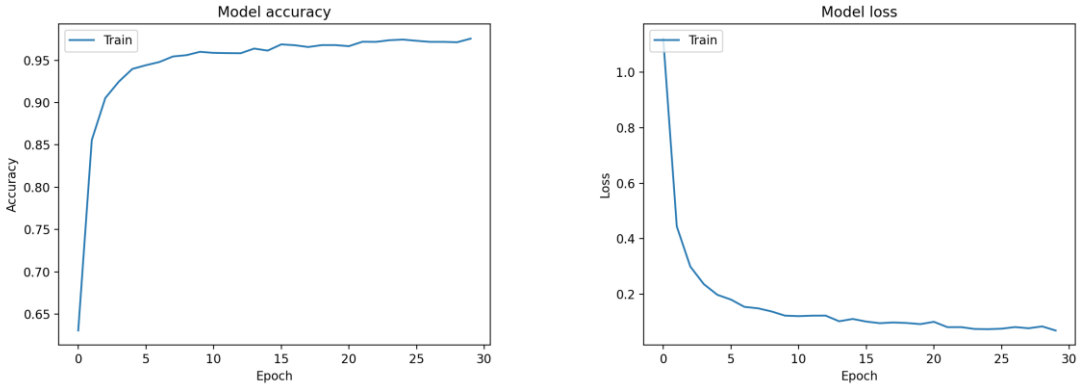
На рисунку 6.3 можна побачити приклади навчального та тестового зображення цифри 5. Тепер вони значно більш схожі, отже можемо очікувати вищих показників точності.



а) б)

Рисунок 6.3 – Приклади зображень цифри 3: а - тестовий датасет, б - згенерований датасет

Архітектуру мережі для нового датасету змінено не було, адже її має бути достатньо для ефективного навчання з високими показниками на тестовому датасеті. До препроцесінгу даних було додано зміну розміру вхідних зображень до 28 на 28 пікселей. Стискання відбувається за допомогою бібліотеки opencv-python. На рисунку 6.4 можна побачити результати навчання моделі. Кількість епох було не змінено.



а) б)

Рисунок 6.4 – Результати навчання: а – точність, б – втрати

На тестовому датасеті отримана точність 96.76%, що було прийнято як задовільний результат.

6.2 Програмна реалізація нейронної мережі для визначення цифр

Для реалізації згорткової нейронної мережі було використано мову програмування Python 3.9, бібліотеки keras та tensorflow.

Для опису логіки потрібної для нейронної мережі було реалізовано клас CNN (Convolution neuron network). Розглянемо його основні методи:

- train – навчання та збереження мережі;
- load – завантаження раніше навченої моделі для подальшого використання;
- predict – зробити передбачення для переданих в метод даних;
- test_model - перевірити точність моделі на тестовому датасеті;

- __create_model – створення архітектури мережі для подальшого навчання;
- __load_data – завантаження та попередня обробка даних з переданої як параметр директорії.

Тобто весь необхідний нам функціонал по нейронній мережі сконцентровано в одному класі, що досить зручно.

Щодо генерації датасету, то це був додатковий скрипт до вже існуючого графічного інтерфейсу. На натискання клавіші пробіл відбувалось збереження зображення, та очищення вікна вводу.

РОЗДІЛ 7. АДАПТАЦІЯ АЛГОРИТМУ ДЛЯ ПЕРЕТВОРЕННЯ У РЕАЛЬНОМУ ЧАСІ

Алгоритм, що було описано в попередніх розділах, реалізовано таким чином що він запускається один раз і вже по завершенню користувацького вводу. Тобто він розрахований на одноразове опрацювання даних, без їх доповнення і доопрацювання.

Однак, до описаного алгоритму можна внести певні поправки для того щоб він зміг опрацьовувати дані по мірі їх надходження. Таким чином, алгоритм зможе робити перетворення рукописної матриці в реальному часі. Під перетворенням у реальному часі мається на увазі таке перетворення, при якому результат роботи може змінюватись по мірі надходження нових даних і їх опрацювання. З точки зору введення матриці це може означати що матриця як математичний об’єкт буде генеруватись по мірі користувацього вводу, а не по факту його завершення.

Розглянемо які зміни необхідно внести в кожну частину нашого алгоритму для того, щоб він почав працювати описаним чином.

Першою частиною алгоритму є виділення рядків. На цьому етапі необхідно просто запустити вже описану частину по опрацюванню нового штриха для кожного нового. Таким чином всі нові штрихи просто будуть розширювати вже існуючу базу, продукуючи нові рядки, або ж доповнюючи вже існуючі.

Щодо виділення області цифр, то ця задача зводиться до розширення діаграми Вороного. Це можна зробити рекурсивним алгоритмом, і поставити у відповідність нові комірки вже існуючим центрам цифр. Цього буде досить щоб комірка стала у потрібне місце в рядок.

Наступним етапом є визначення цифр, які мають бути об’єднаними в числа. Для рішення цієї задачі можна просто розглянути всіх сусідів в рядку для нових цифр та вирахувати чи необхідно їх об’єднувати по вже існуючій моделі. Сама модель константна і змін в її роботі не потрібно.

Для визначення цифр що знаходяться всередині комірок діаграми Вороного необхідно запустити нейронну мережу для кожної з них та записати передбачені значення.

Останнім кроком є об’єднання отриманої інформації, і на цьому етапі можна використати той самий підхід, що і в попередній версії алгоритму - ітерація по рядкам і коміркам всередині рядків з об’єднанням необхідних цифр в числа.

В цілому, дану адаптацію алгоритму можна запускати спираючись на який тригер. Тригером може виступати закінчення вводу штриха, бездія користувача протягом якогось часу, або ж він може запускатись через якийсь визначений інтервал.

Таким чином, запропонована адаптація може опрацьовувати рукописну матрицю по мірі її вводу. Серед переваг такого підходу можна назвати:

- кращий користувацький досвід адже можна бачити проміжний результат перетворення;
- швидша реакція користувача по помилки під час перетворення;
- з точки зору користувача менший час роботи алгоритму, адже він опрацьовує дані по мірі їх надходження, а не всі за раз.

РОЗДІЛ 8. ПОРІВНЯННЯ ОТРИМАНОГО РЕЗУЛЬТАТУ З ІСНУЮЧИМИ РОЗВ'ЯЗКАМИ

Як уже зазначено, розроблений нами алгоритм відноситься до класу онлайн перетворення рукописних матриць, так як він використовує дані отримані під час користувацького вводу, а також може робити перетворення по мірі вводу нових даних. Таким чином і для порівняння було обрано алгоритм, який також може розпізнавати рукописні матриці в онлайн режимі [1].

Спершу слід зазначити, що цей алгоритм може розпізнавати набагато більше математичного тексту ніж розроблений нами, тому для порівняння ми будемо розглядати його тільки з точки розу розпізнавання рукописної матриці.

Для тестування роботи нашого алгоритму було обрано датасети CROHME 2014 та CROHME 2016 (Таблиця 8.1). Оскільки наш алгоритм може розпізнавати не весь математичний текст, то для більш коректного порівняння з нього були видалені всі вирази що не стосуються матриць. Для самого тестування було реалізовано додатковий скрипт для інтеграції роботи з форматом inkml, в якому знаходяться тестові дані.

	CROHME 2014	CROHME 2016
коефіцієнт визначених комірок матриці, %	80.14	82.23
коефіцієнт визначених рядків матриці, %	84.65	86.57
коефіцієнт визначених стовпчиків матриці, %	83.88	84.74
середній час роботи алгоритму, мс	294.51	301.83

Таблиця 8.1 – Результати роботи алгоритму на тестових датасетах

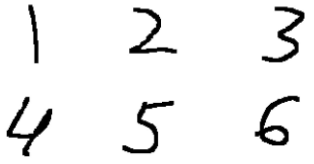
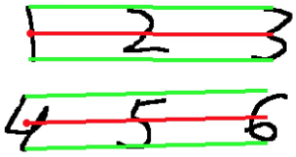
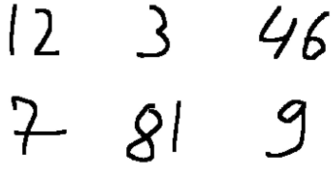
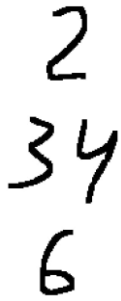
Таку точність можна пояснити недосконалістю нейронної мережі для розпізнавання цифр в комірках діаграми Вороного, та досить сильну прив’язку до методів вводу матриці користувачем.

У таблиці 8.2 наведено порівняння роботи нашого алгоритму з уже існуючими розв’язками по головним показникам.

	Наш алгоритм	Алгоритм порівняння [1]	MyScript
коефіцієнт визначених комірок матриці, %	82.23	91.42	87.49
коефіцієнт визначених рядків матриці, %	86.57	93.86	95.61
коефіцієнт визначених стовпчиків матриці, %	84.74	94.76	90.71

Таблиця 8.2 – Порівняння результатів нашого алгоритму з уже існуючими

Також хочеться розглянути результати роботи алгоритму з використанням реалізованого графічного інтерфейсу. Саме за допомогою графічного інтерфейсу можна провести найповніше тестування користувачем. Так, у таблиці 8.3 можна побачити приклади введних матриць, та результати роботи алгоритму.

Вхідна матриця	Виділення рядків	Результат алгоритму
		1 2 3 4 5 6
		12 5 46 7 81 9
		2 34 6

Таблиця 8.3 – результати роботи програми через графічний інтерфейс

Можна побачити, що алгоритм досить добре справляється з поставленою задачею, однак є невеликі неточності у визначенні цифр.

В цілому можна сказати що розроблений алгоритм працює досить стабільно та точно. Однак, йоловною його проблемою є те, що він дуже зав’язаний на форматі користувальцоко вводу. Його можна збити з пантелику якщо писати цифри не так як зазвичай, або ж зробити дотики які не мають ніякого відношення до матриці.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було проаналізовано існуючі алгоритми, що виконують задачу по онлайн перетворенню рукописної матриці та була здійснена спроба покращити отримані результати шляхом власного рішення задачі. Було розроблено та створено алгоритм, що вирішує поставлену задачу.

У процесі виконання роботи були вирішені наступні задачі:

- вивчено та проаналізовано існуючі розв'язки для поставленої задачі;
- було обрано архітектуру алгоритму, виділено його основні підчастини;
- реалізовано виокремлення рядків рукописної матриці, використовуючи дані, записані під час користувацького вводу;
- реалізовано алгоритм для об'єднання цифр у числа в рядках матриці з використанням логістичної регресії, а також перевірено роботу алгоритму з використанням діаграми Вороного;
- було спроектовано та навчено нейронну мережу для розпізнавання цифр у комірках діаграми Вороного, згенеровано датасет для оптимального навчання.

Під час виконання дипломної роботи було розглянуто та вивчено основні принципи роботи нейронної мережі, особливості використання діаграми Вороного та вивчено потенціал використання даних користувацького вводу для онлайн перетворення рукописних матриць.

Як результат виконаної роботи було створено алгоритм для онлайн перетворення рукописної матриці в математичний об'єкт. Отриманий результат було порівняно з існуючими розв'язками на адаптованих тестових датасетах CROHME 2014 та CROHME 2016.

В цілому можна зазначити, що алгоритм справляється з поставленою задачею. Від виділяє рядки матриці, групує цифри та ідентифікує їх. Головним недоліком реалізованого алгоритму можна назвати те, що від носить чутливий до нестандартних вводів матриці та написання цифр. Також помилковий штрих або додик може сильно погіршити отриманий результат.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Yakovchuk O., Cherneha A., Zhelezniakov D., Zaytsev V. Methods for lines and matrices segmentation in RNN-based online handwriting mathematical expression recognition systems: *Mater. Conference «2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)»*. Lviv, Ukraine, 2020. DOI:10.1109/DSMP47368.2020.9204273

2. Tapia E., Rojas R. Recognition of On-line Handwritten Mathematical Expressions Using a Minimum Spanning Tree Construction and Symbol Dominance. *Graphics Recognition. Recent Advances and Perspectives. 5th International Workshop, GREC 2003 (Barcelona, Spain, July 30-31, 2003)* / Lladós J., Kwon YB. (eds). Berlin, Heidelberg : Springer, 2004. P. 329–340. DOI: 10.1007/978-3-540-25977-0_30

3. Zeleznik C. Li, R., Miller T., LaViola J. J. Online recognition of handwritten mathematical expressions with support for matricesn. *Proceedings of* the *19th International Conference on Pattern Recognition* (8-11 Dec. 2008, Tampa, FL, USA). IEEE, 2008. P. 1–4. DOI: 10.1109/ICPR.2008.4761825

4. Toshihiro K., Masakazu S. A recognition method of matrices by using variable block pattern elements generating rectangular area. *Graphics Recognition. Recent Advances and Perspectives. 4th International Workshop, GREC 2001* (Kingston, Ontario, Canada, September 7–8, *2001)* / Blostein D., Kwon YB. (eds). Berlin, Heidelberg : Springer, 2001. P. 329–340. DOI: 10.1007/3-540-45868-9_28

5. Chan K.-F., Yeung D.-Y. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*. 2000. Vol. 3, No. 1. P. 3–15.

6. Yung-Yao Chen, Yu-Hsiu Lin, Chia-Ching Kung, Ming-Han Chung, I.-Hsuan Yen. Design and Implementation of Cloud Analytics-Assisted Smart

Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes // *Sensors*. 2019. Vol. 19, No. 9. P. 2047. doi:10.3390/s19092047.

7. Тарик Р. Создаем нейронную сеть. К. : Диалектика, 2020. 272 с.

8. Нейронные сети для начинающих. Часть 1. URL: https://habr.com/ru/post/312450/ (дата звернення: 20.03.2021)

9. Avinash Sh. V. Understanding Activation Functions in Neural Networks. URL: https://medium.com/the-theory-of-everything/ (дата звернення: 10.03.2021)

10. TensorFlow layers documentation. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout (дата звернення: 16.01.2021)

11. Ефремова Н. Виды нейронных сетей и их применение. HighLoad^{++}, 2016. URL: https://www.highload.ru/2016/abstracts/2417.html (дата звернення: 15.01.2021)

12. Deshpande A. A Beginner's Guide To Understanding Convolutional Neural Networks. URL: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/ (дата звернення: 25.01.2021)

13. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. М.: Мир, 1989. 295 с.

14. Александров А. Д., Вернер А. Л., Рыжик В. И. Стереометрия. Геометрия в пространстве. Висагинас, ALFA, 1998. 576 с.

15. Препарата Ф., Шеймос М. Вычислительная геометрия. Введение. М. : Мир, 1989. 478 с.

16. Фёрстер Э., Рёнц Б. Методы корреляционного и регрессионного анализа. Руководство для экономистов. М.: Финансы и статистика, 1983. 304 с.

17. Andrew Ng. Stanford CS229 Lecture Notes. URL: https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf (дата звернення: 18.01.2021)

18. Yue Lu, Zhe Wang, Chew Lim Tan. Word Grouping in Document Images Based on Voronoi Tessellation. *Proceedings of the Conference: Document Analysis Systems VI, 6th International Workshop, DAS 2004* (Florence, Italy, September 8-10, 2004) / Marinai S., Dengel A. R. (eds). Springer, 2004. P. 147-157. DOI:10.1007/978-3-540-28640-0_14

ДОДАТОК А

```
function ProcessMatrixRows(hatches)
```
Input: hatches - list of detected hatches
Output: rows - list of processed rows

```
  rows = []
  for hatch in hatches:
    if IsHatchHorizontal(hatch) then
      continue
    endif

    hatch_row = FindRowForHatch(rows, hatch)
    if hatch_row then
      AddHatchToRow(hatch_row, hatch)
    else
      rows.append(new Row(hatch))
    end if
  end for

  return rows
end function
```

ДОДАТОК Б

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
```