# ЗМІСТ

[ЗМІСТ 3](#_Toc470367092)

[ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, АБРЕВІАТУР І ТЕРМІНІВ 5](#_Toc470367093)

[ВСТУП 6](#_Toc470367094)

[РОЗДІЛ 1. ХАРАКТЕРИСТИКА ОБ’ЄКТУ ДОСЛІДЖЕННЯ 8](#_Toc470367095)

[РОЗДІЛ 2. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ 10](#_Toc470367096)

[2.1. Біологічні нейронні мережі 10](#_Toc470367097)

[2.2. Штучний нейрон 11](#_Toc470367098)

[2.3. Одношарові нейронні мережі 12](#_Toc470367099)

[2.4. Парсептрони і зародження штучних нейромереж 13](#_Toc470367100)

[2.5. Навчання нейронних мереж 14](#_Toc470367101)

[2.5.1. Навчання з учителем 14](#_Toc470367102)

[2.5.2. Навчання без вчителя 15](#_Toc470367103)

[2.6. Процес навчання нейронної мережі 15](#_Toc470367104)

[2.7. Аналоги розробки 16](#_Toc470367105)

[РОЗДІЛ 3. СИСТЕМНИЙ АНАЛІЗ 22](#_Toc470367106)

[3.1. Побудова дерева цілей 23](#_Toc470367107)

[3.2. Аналіз і вибір методів, алгоритмів та засобів розв’язання задачі 25](#_Toc470367108)

[3.2.1. GPGPU 25](#_Toc470367109)

[3.2.3 OpenCV 29](#_Toc470367111)

[РОЗДІЛ 4. РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ 32](#_Toc470367112)

[4.1 Відношення з іншими програмами 32](#_Toc470367113)

[4.2 Модель взаємодії системи з сутностями 32](#_Toc470367114)

[РОЗДІЛ 5. ВИБІР, РОЗРОБКА МЕТОДІВ І АЛГОРИТМІВ ФУНКЦІОНУВАННЯ АПАРАТНО-ПРОГРАМНИХ ЗАСОБІВ 34](#_Toc470367115)

[5.1. Опис вибраної методології проектування та життєвого циклу системи 34](#_Toc470367116)

[5.2. Побудова моделей бізнес-процесів за допомогою діаграм IDEF0 35](#_Toc470367117)

[5.3. Відображення потоків даних за допомогою DFD діаграми 37](#_Toc470367118)

[РОЗДІЛ 6. РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ 39](#_Toc470367119)

[6.1. Опис використаних сторонніх бібліотек та модулів 39](#_Toc470367120)

[6.2. Принцип роботи програми 39](#_Toc470367122)

[6.3. Установка 40](#_Toc470367136)

[6.4. Розробка та опис інтерфейсу користувача 44](#_Toc470367156)

[ВИСНОВКИ 49](#_Toc470367175)

[ПЕРЕЛІК ПОСИЛАНЬ 50](#_Toc470367180)

[ДОДАТОК А 52](#_Toc470367181)

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, АБРЕВІАТУР І ТЕРМІНІВ

НМ – нейронна мережа.

CPU – центральний процесор.

GPU – графічний процесор .

GPGPU – технологія використання графічного процесора.

API – інтерфейси програмування.

OpenCV – це файл або набір файлів, котрі містять підпрограми, функції, які використовуються для розробки програмного забезпечення.

SSADM – структурний підхід, який використовується для проектування системи.

DFD – діаграма потоків.

IDEF0 – діаграма, котра відображає бізнес-процеси системи.

UML (unified modelling language) – мова графічного опису для об’єктного моделювання в сфері розробки програмного забезпечення.

# ВСТУП

В сучасних умовах розвитку інноваційних технологій у всіх галузях і сферах діяльності людини з’явилися нові наукові  напрями.  Швидкого та інтенсивного піднесення за останні роки зазнала інформатика, що виросла з класу теоретичних фундаментальних дисциплін та значно розширила практичні сфери свого застосування. Однією з перспективних галузей сучасної інформатики на сьогодні є нейроінформатика.

Нейроінформатика – це принципово новий підрозділ інформатики, що стосується аналізу та переробки інформації, базується на використанні моделей штучного нейрона та побудові на їх основі нейронних мереж.

Важливою властивістю нейронних мереж, що свідчить про їх великий потенціал і широкі прикладні можливості -  паралельна обробка інформації одночасно великою кількістю нейронів. Завдяки цьому досягається значне пришвидшення обробки інформації. Іншою не менш важливою особливістю нейронних мереж є здатність до навчання та узагальнення інформації. Таким чином досягається деяка схожість з роботою головного мозку людини.

Останнім часом спостерігається тенденція зростання інтересу до використання нейронних мереж для вирішення різних завдань і застосування їх в різних сферах людського життя.

З використанням нейронних мереж відкрилися можливості проведення обчислень в сферах, що до цього відносилися лише до сфери людського інтелекту. З’явилися можливості створення систем, які здатні вчитися, запам'ятовувати та аналізувати інформацію, що дуже нагадує розумові здібності людини.

Типовими задачами, що можуть бути вирішеними за допомогою нейронних мереж та нейрокомп'ютерів є: задача класифікації, автоматизація прогнозування, автоматизація процесу ухвалення рішень, управління, кодування і декодування інформації, розпізнавання образів та ін.

Нейроінфоматикою та дослідженнями нейромереж у різних галузях займаються науковці з усього світу. За допомогою штучних нейронних мереж можна опрацьовувати, аналізувати та узагальнювати інформації, що аналогічно роботі головного мозку людини. Нейтронні мережі використовуються у економіці, медицині, зв'язку, безпеці та охоронних системах, введенні та обробці інформації. Безумовно, даний перелік не повний, проте він дозволяє отримати уявлення про характер застосування нейромережевих технологій.

# РОЗДІЛ 1. ХАРАКТЕРИСТИКА ОБ’ЄКТУ ДОСЛІДЖЕННЯ

Приступивши до роботи над проектом необхідно з’ясувати, що є особливістю моєї предметної області, чим вона характеризується, що буде кінцевим результатом моєї роботи предметною областю моєї курсової роботи є нейромережі для розпізнавання цифр. Тому в даній роботі планується дослідити цю предметну область та створити програму з використанням графічного процесора для виконання обчислень.

Розпізнавання образів (цифр, сигналів, ситуацій, явищ або процесів) - завдання ідентифікації об'єкта або визначення будь-яких його властивостей по його зображенню (оптичне розпізнавання) або аудіозаписи (акустичне розпізнавання) та інші характеристики [1].

Якщо ж говорити про напрямки в розпізнаванні образів, то можна виділити два основних напрямки:

1. Вивчення здібностей до розпізнавання, якими володіють живі істоти, їхнє пояснення й моделювання;
2. Розвиток теорії й методів побудови пристроїв, призначених для розв’язання окремих задач у прикладних цілях.

Одним з базових є те, що не має конкретного формулювання поняття безлічі. У комп'ютері безліч представляється набором неповторюваних однотипних елементів. Слово "неповторюваних" означає, що якийсь елемент в множині або є, або його там немає. Універсальне безліч включає всі можливі для розв'язуваної задачі елементи, порожнє не містить жодного.

Образ - класифікаційне угруповання в системі класифікації, яка об'єднує (виділяє) певну групу об'єктів за певною ознакою. Образи мають характерним властивістю, що виявляється в тому, що ознайомлення з кінцевим числом явищ з одного і того ж безлічі дає можливість дізнаватися як завгодно велике число його представників. Образи мають характерні об'єктивними властивостями в тому сенсі, що різні люди, які навчаються на різному матеріалі спостережень, здебільшого однаково і незалежно один від одного класифікують одні і ті ж об'єкти. У класичній постановці задачі розпізнавання універсальне безліч розбивається на частини-образи. Кожне відображення будь-якого об'єкта на сприймаючі органи розпізнає, незалежно від його положення щодо цих органів, прийнято називати зображенням об'єкта, а безлічі таких зображень, об'єднані певними загальними властивостями, є образи.

Отже, під час виконання даної роботи необхідно провести огляд предметної області, обрати оптимальні інструменти для виконання поставленої задачі та реалізувати прототип нейронної мережі для задачі розпізнавання цифр.

**РОЗДІЛ 2. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ**

Предметною областю моєї роботи є нейронні мережі.

Про роботу мозку в даний час відомо дуже мало, тому штучні нейронні мережі (neural networks) є надзвичайно спрощеною моделлю біологічних нейронних мереж. Особливістю нейромереж (neuronet) є те, що вони навчаються, а не програмуються.

## **2.1. Біологічні нейронні мережі**

Нервова система людини побудована з елементів (нейронів), має приголомшуючу складність. Близько 1011 нейронів беруть участь в приблизно 1015 передаючих зв'язках, що мають довжину метр і більше. Кожен нейрон володіє багатьма якостями, спільними з іншими елементами тіла, але його унікальною здатністю є прийом, обробка і передача електрохімічних сигналів по нервових шляхах, які утворюють комунікаційну систему мозку [2].



*Рис. 2.1. Біологічний нейрон*

На рис. 1 показана структура пари типових біологічних нейронів. Дендрити (входи нейрона) йдуть від тіла нервової клітини до інших нейронів, де вони приймають сигнали в точках з'єднання (синапсах). Прийняті синапсом вхідні сигнали підводяться до тіла нейрона. Тут вони підсумовуються, причому одні входи стимулюють активізацію нейрона, а інші – зниження його активності. Коли сумарна активність (збудження) нейрона перевищує деякий поріг, нейрон переходить в активний стан, посилаючи по аксону (виходу нейрона) сигнал іншим нейронам. У цієї основної функціональної схеми багато спрощень і виключень, проте більшість штучних нейронних мереж моделює лише ці прості властивості.

## **2.2. Штучний нейрон**

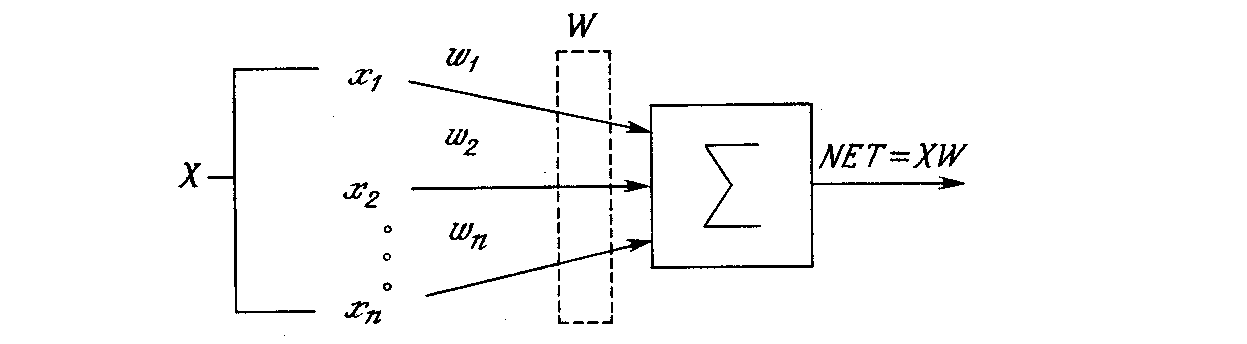
Основними компонентами нейромережі є нейрони /neurons/ (елементи, вузли), які з’єднані зв’язками. Сигнали передаються по зваженим зв’язкам (connection), з кожним з яких пов’язаний ваговий коефіцієнт (weighting coefficient) або вага [3].

Моделі НМ – програмні і апаратні, найбільш поширені – програмні.

Використання – розпізнавання образів, прогнозування, створення асоціативної пам’яті.

Штучний нейрон імітує в першому наближенні властивості біологічного нейрона. На вхід штучного нейрона поступає множина сигналів, які є виходами інших нейронів. Кожен вхід множиться на відповідну вагу, аналогічну його синаптичній силі, і всі виходи підсумовуються, визначаючи рівень активації нейрона. На рис.2 представлена модель, що реалізує цю ідею. Хоча мережеві парадигми досить різноманітні, в основі майже всіх їх лежить ця конфігурація. Тут множина вхідних сигналів, позначених x1, x2,…, xn, поступає на штучний нейрон. Ці вхідні сигнали, в сукупності позначаються вектором X, відповідають сигналам, що приходять в синапси біологічного нейрона. Кожен сигнал множиться на відповідну вагу w1, w2,…, wn, і поступає сумуючий блок, позначений Σ. Кожна вага відповідає «силі» одного біологічного синаптичного зв'язку (множина ваг в сукупності позначається вектором W). Сумуючий блок, який відповідає тілу біологічного нейрона, складає зважені входи алгебраїчно, створюючи вихід, який ми називатимемо NET. У векторних позначеннях це може бути компактно записано таким чином:

NET = XW.



*Рис. 2.2. Штучний нейрон*

## **2.3. Одношарові нейронні мережі**

Хоча один нейрон і здатний виконувати прості процедури розпізнавання, сила нейронних обчислень виникає від з'єднань нейронів в мережах. Проста мережа складається з групи нейронів, створюючий шар, як показано в правій частині рис.2.3. Відзначимо, що вершини-круги зліва служать лише для розподілу вхідних сигналів. Вони не виконують будь-яких обчислень, і тому не вважатимуться шаром. З цієї причини вони позначені кругами, щоб відрізняти їх від обчислюючих нейронів, позначених квадратами. Кожен елемент з множини входів Х окремою вагою сполучений з кожним штучним нейроном. Кожен нейрон видає зважену суму входів в мережу. У штучних і біологічних мережах багато з'єднань можуть бути відсутніми, можуть мати місце також з'єднання між виходами і входами елементів в шарі [4].



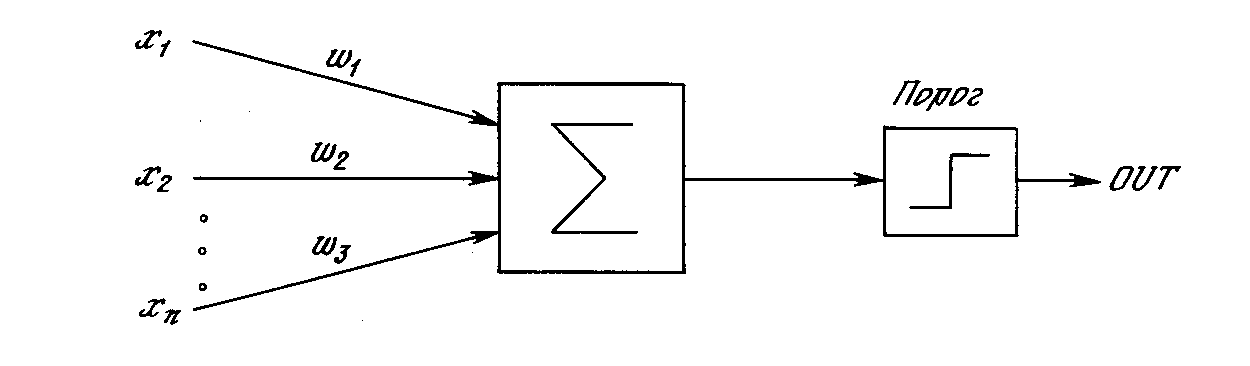
*Рис. 2.3. Одношарова НМ*

Зручно вважати вагу елементами матриці W. Матриця має m рядків і n стовпців, де m - число входів, а n - число нейронів. Наприклад, w3,2 - це вага, що пов'язує третій вхід з другим нейроном. Таким чином, обчислення вихідного вектора Y, компонентами якого є виходи OUT нейронів, зводиться до матричного множення Y = XW, де Y і Х - вектори-рядки.

## **2.4. Парсептрони і зародження штучних нейромереж**

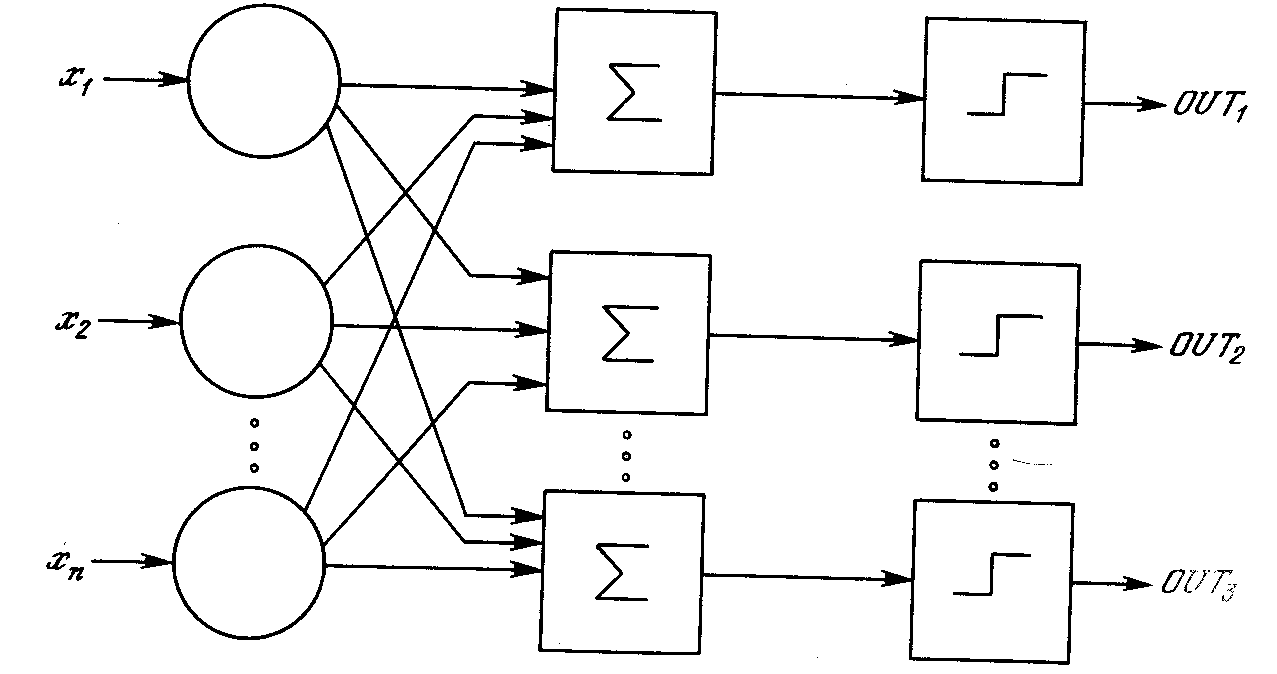
Як науковий предмет штучні нейронні мережі вперше заявили про себе в 40-ві роки. Прагнучи відтворити функції людського мозку, дослідники створили прості апаратні (а пізніше програмні) моделі біологічного нейрона і системи його з'єднань. Коли нейрофізіологи досягли глибшого розуміння нервової системи людини, ці ранні спроби стали сприйматися як досить грубі апроксимації. Проте на цьому шляху були досягнуті вражаючі результати, що стимулювали подальші дослідження, що привели до створення досконаліших мереж.

Перше систематичне вивчення штучних нейронних мереж було зроблене Мак-Каллоком і Піттсом в 1943 р.. В роботі вони досліджували мережеві парадигми для розпізнавання зображень, що піддаються зсувам і поворотам. Проста нейронна модель, показана на рис.7, використовувалася в більшій частині їх роботи. Елемент Σ множить кожен вхід х на вагу w і підсумовує зважені входи. Якщо ця сума більше заданого порогового значення, вихід рівний одиниці, інакше - нулю. Ці системи (і множина ним подібних) одержали назву персептронів. Вони складаються з одного шару штучних нейронів, сполучених за допомогою вагових коефіцієнтів з множиною входів (рис.2.4), хоча описуються і складніші системи [5] [6].



*Рис. 2.4. Нейрон персептрона*

Перші персептрони були створені Ф.Розенблатом у 60-х роках і викликали великий інтерес. Первинна ейфорія змінилася розчаруванням, коли виявилося, що персептрони не здатні навчитися рішенню ряду простих задач. М.Мінський строго проаналізував цю проблему і показав, що є жорсткі обмеження на те, що можуть виконувати одношарові персептрони, і, отже, на те, чому вони можуть навчатися. Оскільки у той час методи навчання багатошарових мереж не були відомі, дослідники перейшли в більш багатообіцяючі області, і дослідження у області нейронних мереж прийшли в занепад. Відкриття методів навчання багатошарових мереж більшою мірою, ніж який-небудь інший чинник, вплинуло на відродження інтересу і дослідницьких зусиль.



*Рис. 2.5 Персептрон*

## **2.5. Навчання нейронних мереж**

Серед властивостей штучних нейронних мереж основним є їх здатність до навчання, вони навчаються найрізноманітнішими методами. Більшість методів навчання походять від загальних передумов, і мають багато ідентичних характеристик. Їх навчання нагадує процес інтелектуального розвитку людської особистості. Можливості навчання штучних нейронних мереж обмежені. Проте, вже отримані переконливі досягнення, такі як «Мережа, котра говорить» Сейновського, і виникає багато інших практичних застосувань. Мережа навчається, щоб для деякої безлічі входів давати необхідне безліч виходів. Кожна така вхідна (або вихідна) множина розглядається як вектор. Навчання здійснюється шляхом послідовного пред’явлення вхідних векторів з одночасним підстроюванням ваг у відповідності до певної процедури. У процесі навчання ваги мережі поступово стають такими, щоб кожен вхідний вектор виробляв вихідний вектор. Навчальні алгоритми можуть бути класифіковані, як алгоритми навчання з учителем і без вчителя.

### **2.5.1. Навчання з учителем**

При навчанні з учителем існує вчитель, котрий висуває вхідні образи мережі, порівнює результуючі виходи з необхідними значеннями, а потім налаштовує ваги мережі таким чином, щоб зменшити відмінності. Навчання з учителем передбачає, що для кожного вхідного вектора існує цільовий вектор, що представляє собою необхідний вихід. Разом вони називаються навчальною парою. Зазвичай, мережа навчається на деякому числі таких навчальних пар. Висувається вихідний вектор, обчислюється вихід мережі і порівнюється з відповідним цільовим вектором, різниця (помилка) за допомогою зворотного зв’язку подається в мережу, і ваги змінюються відповідно до алгоритму, прагнуть мінімізувати помилку. Вектори навчальної множини висуваються послідовно, обчислюються помилки і ваги підлаштовуються для кожного вектора до тих пір, поки помилка по всьому навчальному масиву не досягне прийнятного низького рівня.

### **2.5.2. Навчання без вчителя**

Навчання з вчителем критикувалося за свою біологічну неправдоподібність. Важко уявити навчальний механізм в мозку, який би порівнював бажані і дійсні значення виходів, виконуючи корекцію за допомогою зворотного зв’язку. Навчання без вчителя є набагато більш правдоподібною моделлю навчання біологічної системи. Розвинена Кохоненом і іншими, вона не потребує цільового вектора для виходів і, отже, не вимагає порівняння зі зумовленими ідеальними відповідями. Навчальна множина складається лише з вхідних векторів. Навчальний алгоритм підлаштовує ваги мережі так, щоб виходили узгоджені вихідні вектори, щоб пред’явлення досить близьких вхідних векторів давало однакові виходи. Процес навчання, отже, виділяє статистичні властивості навчальної множини і групує подібні вектори в класи. Представленняя на вхід вектора з даного класу дасть певний вихідний вектор, але до навчання неможливо передбачити, який вихідний буде проводитися даним класом вхідних векторів. Отже, виходи подібної мережі повинні трансформуватися в деяку зрозумілу форму, зумовлену процесом навчання.

## **2.6. Процес навчання нейронної мережі**

Процес навчання штучної нейронної мережі нового класу задача включає наступні стадії:

1. Формулюється постановка задачі і виділяється набір ключових параметрів, характеризуються предметну область.

2. Вибирається парадигма нейронної мережі (модель, що включає в себе вид вхідних даних, порогової функції, структури мережі і алгоритмів навчання), найбільш краща для вирішення даного класу задач. Як правило, сучасні нейропакети, нейрооплати і нейрокомп’ютери дозволяють реалізувати не одну, а кілька базових парадигм.

3. Готується, можливо, більш широкий набір навчальних прикладів, організованих у вигляді наборів вхідних даних, асоційованих з відомими вихідними значеннями. Вхідні значення для навчання можуть бути неповні і частково суперечливі.

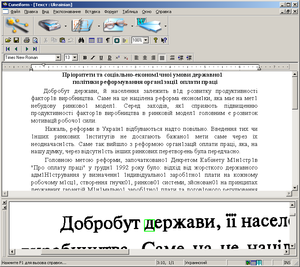
4. Вхідні дані по черзі представляються штучною нейронною мережею, а отримане значення порівнюється з еталоном. Потім проводиться підстроювання вагових коефіцієнтів міжнейронних з’єднань для мінімізації помилки між реальним і бажаним виходом мережі.

5. Навчання повторюється до тих пір, поки сумарна помилка у всій безлічі вхідних значень не досягне прийнятного рівня, або штучна нейронна мережа не прийде в стаціонарний стан. Розглянутий метод навчання нейроподібної мережі носить назву «Зворотне поширення помилки» (error backpropagation) і відноситься до числа класичних алгоритмів нейроматематики.

Налагоджена і навчена штучна нейронна мережа може використовуватись на реальних вхідних даних, не тільки підказуючи користувачеві коректне рішення, а й оцінюючи ступінь його достовірності.

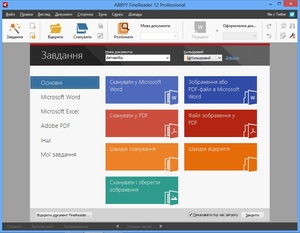
## **2.7. Аналоги розробки**

CuneiForm – це інструмент, побудований на оптичному розпізнаванні символів, котрий був розроблений російською компанією Cognitive Technologies. Дана програма має можливість перетворити файли зображення, які отримані зі сканера або іншим шляхом на текст. Програму випустили у 2007 році, яка працювала з Windows та Linux. До того ж CuneiForm є шрифтонезалежною системою. Алгоритми, котрі лежать в даній програмі, виходять з правил написання букв, з їх топології, і при цьому не вимагають завдань різних еталонів, або ж самого навчання. Визначаються будь-які друкарські шрифти, в тому числі книги, газети, журнали, друк на лазерних і матричних принетерах. Але дана програма не розпізнає рукописний текст і декоративні шрифти (готичний, стилізований почерк). Програма має спеціальні настройки для розпізнавання тексту з матричного принтера і факсу 200х100 DPI. CuneiForm має можливість зберегти розпізнаний текст у форматі RTF, HTML або текст. Крім цього, можна передати редакторові або електронної таблиці Excel [7].



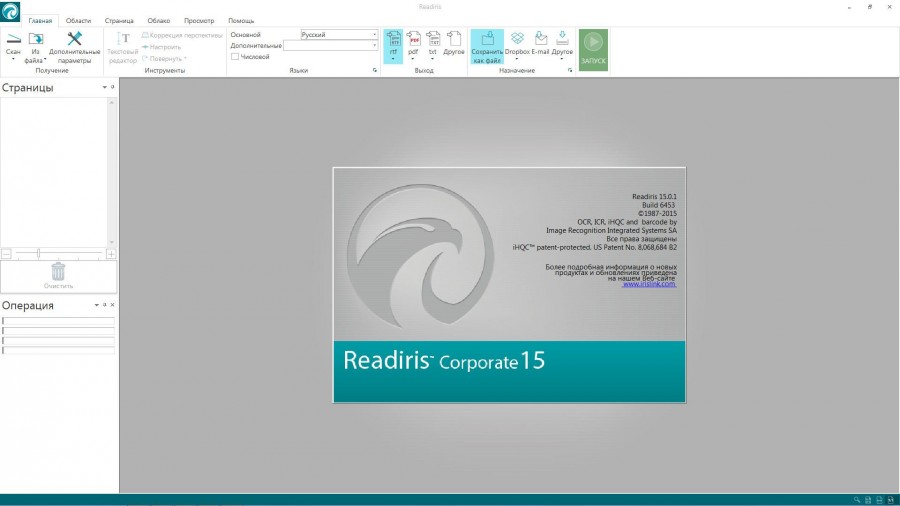
*Рис. 2.6. CuneiForm*

Ще однією системою оптичного розпізнавання символів є FineReader, розроблена російською компанією ABBYY. Ця програма досить таки швидко і точно розпізнає відскановані або сфотографовані документи, перетворюючи їх в редагований формат або ж PDF, при цьому маю можливість пошуку. Коли розпізнаються якісні документи, швидкий режим збільшить швидкість на 40% без шкоди для точності. А чорно-білі документи також можуть бути використані в режимі чорного-білого розпізнавання, котрий прискорює роботу ще на 30%. Завдяки технології Adaptive Document Recognition Technology, ABBYY FineReader 12 зберігає вихідну структуру багатосторінкових документів, включаючи розташування тексту, таблиць, колонтитулів, приміток, нумерацію сторінок, змісту, змісту та ін. Задати типи областей (Текст, Картинка, Таблиця і ін.) і вказати їх призначення, можна і вручну [8].

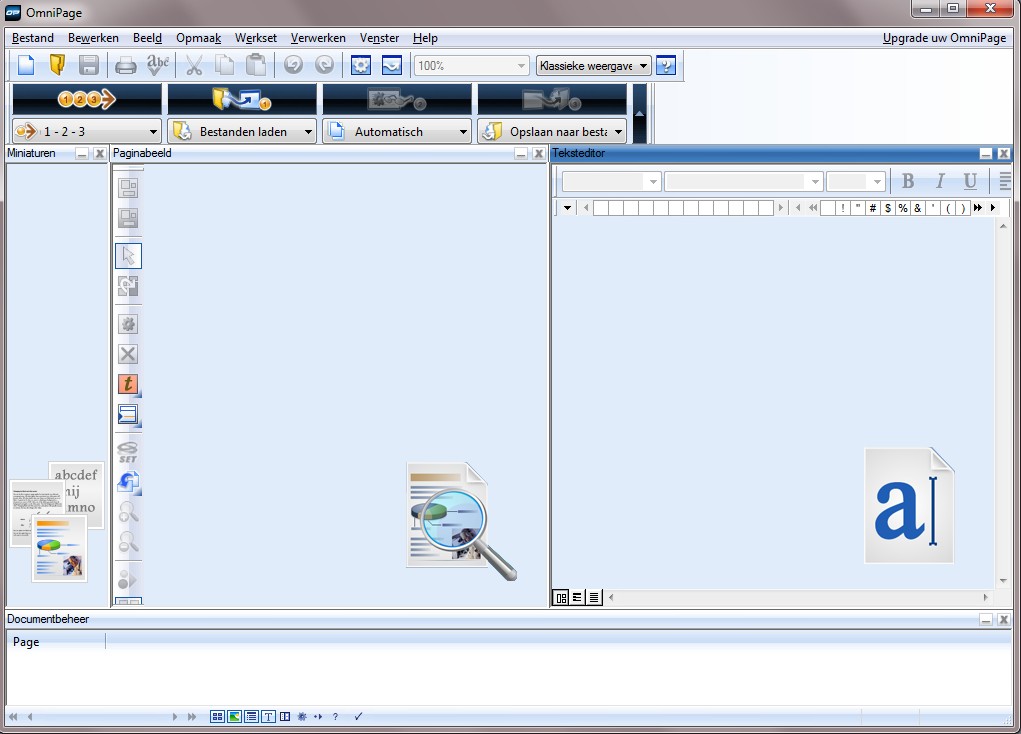


*Рис. 2.7. FineReader*

Readiris Pro – це програма, яка розпізнає текст, дозволяє перетворювати документи в будь-які формати. Швидко і без проблем конвертує формат зображення в txt, xml, pdf або docx документ. У підсумку можна легко редагувати відскановану інформацію, доповнюючи її необхідними даними або ж видаляючи зайве. Дана утиліта не вимагає додаткової установки драйверів на сканер, принтер, тощо. Тоді ця програма розпізнає ці обладнання і починає працювати з ними. Основними функціями програми є: витягування інформації з цифрових зображень; за лічені секунди розпізнає символи у відсканованому документі; зберігає текст в будь-якому зручному форматі (при цьому найкраще працює з текстовим редактором Microsoft Word); працює з різними форматами зображень (JPEG – 2000, BMP, DjVU, JPEG, TIFF та інші); переносить фоновий колір і стиль тексту [9].

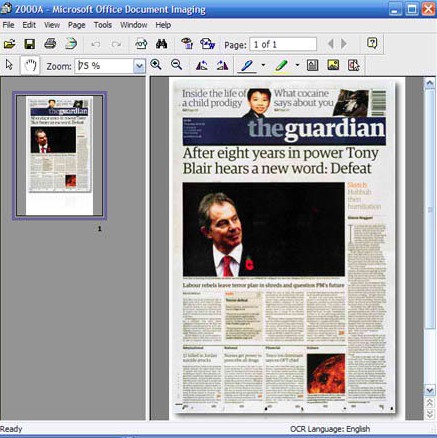


*Рис. 2.8. Readiris Pro*

OmniPage – це програма сканування і розпізнавання тексту компанії Nuance Communications. Програма має високу точність і швидкість. Визнає понад 120 мов з різними алфавітами: латинський, грецький, кирилиця, китайський, японський і інші. Як FineReader, Omnipage впевнено розпізнає документи, отримані за допомогою цифрової камери зображення корекції технології «3D-кореції». Ця програма призначена для підтримки паралельної роботи декількох документів, можна відразу відкрити, розпізнати, виправити і зберігати кілька документів одночасно. OmniPage доступний в трьох версіях: Standart, Professional, Enterprise [10]. 

*Рис. 2.9. OmniPage*

Microsoft Office Document Imaging – це програма розпізнавання тексту компанії Microsoft. Дана програма вміщає у собі інструменти для сканування і розпізнавання тексту – MS Office Document Scanning та Document Imaging. Програма може працювати лише з двома мовами англійською і мовою локалізації самого MS Office. Для підтримки інших мов необхідно додатково встановлювати пакет Multilingual User Interface. OCR налаштувань в програмі практично немає, програма в автоматичному режимі підтримує розпізнавання шрифтів, картинок і простих таблиць [12].



*Рис. 2.10. Microsoft Office Document Imaging*

Розроблювана програма буде виконувати задачу розпізнавання цифр з допомогою нейронної мережі. Для прискорення виконання обчислень, їх буде реалізовано на графічному процесорі.

# РОЗДІЛ 3. СИСТЕМНИЙ АНАЛІЗ

Перед тим як приступати до даного розділу, коротко охарактеризуємо системно-аналітичне бачення даної системи. На сьогоднішній день є безперечним значний науковий та практичний інтерес до обчислювальних структур нового типу – штучних нейронних мереж. Це спричинено низкою успішних застосувань цієї нової технології, яка дозволила розробити ефективні підходи до вирішення проблем, що вважалися складними для реалізації на традиційних комп’ютерах. На назву «нейронні мережі» зараз претендують усі обчислювальні структури, які в тій чи іншій міри модулюють роботу мозку.

Успішний розвиток теорії нейронних мереж за останнє десятиліття дозволив реалізувати ряд таких глобальних властивостей. Найвідомішими з них є навчання, узагальнення та абстрагування.

Властивість навчання проявляється у здатності нейронної мережі змінювати свою поведінку в залежності від стану навколишнього середовища. Завдяки цій властивості нейронні мережі привертають до себе значну увагу. Існує велике розмаїття алгоритмів навчання нейронних мереж, кожен з яких має свої сильні та слабкі сторони.

Вже зараз розпізнавання образів щільно увійшли у повсякденне життя і є одним з найбільш насущних знань сучасного інженера. У медицині розпізнавання образів допомагає лікарям ставити більш точні діагнози, на заводах воно використовується для прогнозу шлюбу в партіях товарів. Системи біометричної ідентифікації особистості в якості свого алгоритмічного ядра так само засновані на результатах нейронних мереж. Подальший розвиток штучного інтелекту, зокрема проектування комп’ютерів п’ятого покоління, здатних до більш безпосереднього спілкування з людиною на природних для людей мовах і за допомогою мови, немислимі без розпізнавання. Тут рукою подати і до робототехніки, штучних систем управління, що містять як життєво важливих підсистем системи розпізнавання.

Саме тому до розвитку розпізнавання образів з самого початку було прикута чимало уваги з боку фахівців самого різного профілю – кібернетиків, нейрофізіологів, психологів, математиків, економістів і т.д. Багато в чому саме з цієї причини сучасне розпізнавання образів саме харчується ідеями цих дисциплін.

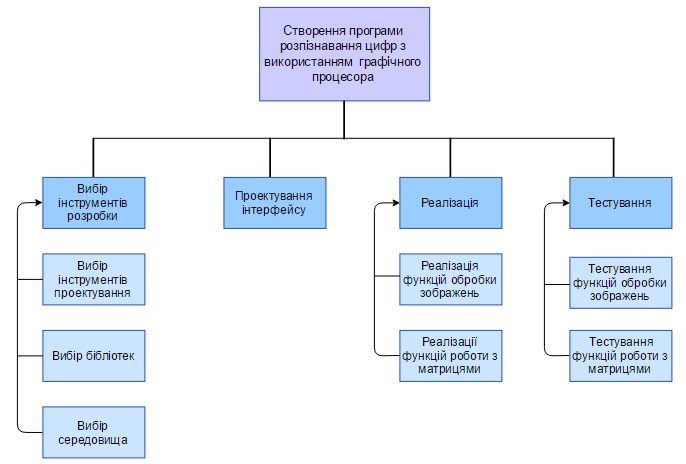
Отже, з легкістю можна сказати, що розпізнавання образів в сучасному суспільстві використовується скрізь, і ця тема є завжди актуальною.

## **3.1. Побудова дерева цілей**

Дерево цілей – це графічне зображення взаємозв’язку і підпорядкованості цілей, що відображає розподіл місії і мети на цілі, підцілі, завдання на окремі дії [13].

Отже, щоб продовжити системний аналіз, потрібно визначити і поставити перед собою цілі, які необхідно досягнути під час роботи над системою. Визначення цілей нам потрібно для планування роботи над системою.

В загальному вигляді дерево цілей можна представити таким чином:



*Рис. 3.1. Дерево цілей*

Головна ціль, яку потрібно виконати в процесі роботи – це створення програми розпізнавання цифр з використанням графічного процесора.

Щоб досягнути головну ціль потрібно виконати 4 підцілі – вибір інструментів розробки, проектування інтерфейсу, реалізація, тестування.

1. Вибір інструментів розробки поділяється:

* Вибір інструментів проектування;
* Вибір бібліотек;
* Вибір середовища.

2. Проектування інтерфейсу.

3. Реалізація поділяється на наступні підцілі:

* Реалізація функцій обробки зображень;
* Реалізація функцій роботи з матрицями.

4. Тестування передбачає наступні підцілі:

* Тестування функцій обробки зображень;
* Тестування функцій роботи з матрицями.

## **3.2. Аналіз і вибір методів, алгоритмів та засобів розв’язання задачі**

Завданням даної роботи є програма розпізнавання цифр з допомогою нейронної мережі. Як згадувалося раніше, за своєю природою дана система потребує обчислень великих об’ємів даних. Через це необхідно розглянути концепцію, пов’язану з розробкою таких систем.

### **3.2.1. GPGPU**

GPGPU – це технологія використання графічного процесора, який обробляє розрахунок тільки для комп’ютерної графіки, щоб виконати обчислення в додатках, традиційно виконуваних центральним процесором [14].

Останнім часом обчислювальні потужності графічних процесорів (GPU)

значно зросли, в зв’язку з чим можливість використання задач загального призначення стає все більш привабливим. Сьогодні ми можемо спостерігати велику активність в даному напрямку. Комп’ютерне баченння є однією з областей, в яких можна ефективно використовувати GPU для широкого кола актуальних задач, включаючи детектування осіб і пішоходів відновлення тривимірних поверхонь, розпізнавання об’єктів, тексту. Багато алгоритмів такого роду можна ефективно розпаралелити за даними і виконувати на GPU з високою швидкістю.

Говорячи про розрахунок графічних процесорів загального призначення, варто сказати, що вони стали досить таки практичним і популярним після 2001 року, з появою програмованих шейдерів і підтримки з плаваючою точкою на графічних процесорах. Зокрема, проблеми з участю матриць і векторів – особливо двох-трьох- або чотирьох вимірних векторів – були легко перенесені на GPU, яка працює з рідною швидкістю і підтримує ці типи. Експерименти наукових обчислень з новими обладнаннями почалося зі звичайного матричного множення (2001); одним з перших спільних дослідницьких програм, що працювала швидше на GPU, ніж на графічних процесорах була реалізація LU факторизації (2005).

Ці ранні спроби використовувати графічні процесори, як процесори загального призначення вимагають переформулювання обчислювальних задач з точки зору графічних примітивів, підтримуються два основних інтерфейси програмування (API) для графічних процесорів: OpenGL і DirectX. Це громіздке перетворення було усунуто з появою загального призначення мов програмування та інтерфейсів.

У зв'язку з тенденцією збільшення потужності мобільних графічних процесорів, програмування загального призначення стало доступне і на мобільних пристроях, що працюють під управлінням основних мобільних операційних систем.

Google Android 4.2 підтримує виконання Renderscript коду на графічному процесорі мобільних пристроїв.

Apple представила власний програмний інтерфейсMetal для iOS додатків, здатний виконати довільний код через обчислювальні шейдери GPU.

Комп'ютерні відеокарти виробляються різними постачальниками, такими як NVIDIA і AMD / ATI. Карти від таких постачальників відрізняються за реалізацією підтримки формату даних, наприклад, цілочисельні(integer) і формати з плаваючою точкою(floating-poing) (32-біт і 64-біт). Microsoft(силка) представила стандарт «Shader Model», щоб допомогти ранжувати різні особливості графічних карт в простому номері версій шейдерної моделі (1.0, 2.0, 3.0 і т. д.).

### *Цілочисельний формат даних*

Графічні карти до DirectX 9 підтримували тільки палітрені або цілочисельні типи кольору. Доступні різні формати, кожен з яких містить червоний, зелений і синій елементи. Іноді додають додаткові значення альфа, які будуть використовуватися для прозорості. Загальні формати: +8 Біт на піксель — Іноді режим палітри, де кожне значення індексу в таблиці з реальним значенням вказане в одному з інших форматів. Іноді три біта для червоного, три біта для зеленого і два біти для синього. 16 біт на піксель — Звичайно виділяється у вигляді п'яти біт для червоного, шість біт для зеленого і п'ять біт для синього. 24 біт на піксель — вісім бітів для кожного з червоного, зеленого і синього 32 біт на піксель — вісім біт для кожного з червоного, зеленого, синього та альфа.

*Формат даних з плаваючою комою*

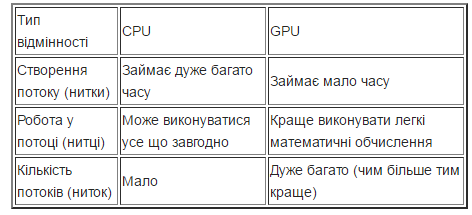
Для ранньо фіксованої функції або обмежених можливостей програмування графіки (тобто до і включаючи DirectX 8.1-сумісний GPU) цього було достатньо, тому що це також представлення яке використовується в дисплеях. Таке представлення має певні обмеження. Враховуючи достатню потужність обробки графіки, навіть програмісти графіки хотіли б використовувати кращі формати, такі як формати даних з плаваючою комою(силка), щоб отримати ефекти, такі як високий динамічний діапазон зображень. Багато GPGPU додатків вимагають точності формату з плаваючою комою, яка прийшла з відеокартами, відповідно до специфікації DirectX 9. DirectX 9 Shader Model 2.x запропонувала підтримку двох типів точності: повної і часткової. Повна точність може бути FP32 або FP24 (floating point 32- або 24-біт на компонент) або більше, в той час як часткова точність була FP16. Серія графічних процесорів ATI R300 підтримувала точність FP24 тільки в трубопроводі програмованого фрагмента (хоча FP32 підтримувалась в вертексних процесорах), а серія Nvidia NV30 підтримує як FP16, так і FP32. Інші виробники, такі як S3 Graphics і XGI підтримують суміш форматів до FP24. Shader Model 3.0 змінила специфікацію, збільшуючи вимоги до повної точності як мінімум до підтримки FP32 в трубопроводі фрагмента. Shader Model 3.0 в сумістності з поколінням R5xx (Radeon X1000 серії) підтримує тільки FP32 через трубопровід, а серії Nvidia NV4x і G7X раніше підтримували і повну точність FP32, і часткову FP16. Хоча це і не передбачено Shader Model 3.0, як графічні процесори ATI, так і Nvidia Shader Model 3.0 представили підтримку для змішування цілей візуалізації FP16, сприяючи підтримці високодинамічного відображення діапазону.

*GPU проти CPU*

Спочатку, дані просто передавалися в одному напрямку з центрального процесора до GPU, а потім до пристрою відображення. Однак, з плином часу, для графічних процесорів стало цінним зберігати спершу прості, потім складні структури даних, які передаються назад в процесор, що аналізує зображення, або набір науково представлених даних у форматі 2D або 3D, що відеокарта може зрозуміти. Так як GPU має доступ до кожної операції малювання, він може аналізувати дані в цих формах дуже швидко; в той час як процесор повинен опитати кожен піксель або елемент даних набагато повільніше, так як швидкість доступу між центральним процесором і його більшим простором оперативної пам'яті (або ще у гіршому випадку, жорсткого диску) є повільніша, ніж на графічних процесорах і відеокартах, які, як правило, містять менші кількості більш дорогої пам'яті, до якої доступ можна отримати набагато швидше.

Передача частини набору даних, які будуть проаналізовані на цю GPU пам'ять у вигляді текстур або інших легко зрозумілих GPU форм призводить до збільшення швидкості. Відмінною особливістю конструкції GPGPU є здатність передавати інформацію в обох напрямках назад від GPU до CPU; як правило, швидкість передачі даних в обох напрямках є ідеально висока, в результаті чого ефект множення впливає на швидкість конкретного алгоритму високого використання. GPGPU конвеєри можуть підвищити ефективність на особливо великих наборах даних та / або даних, що містять 2D або 3D зображення. Він використовується в складних графічних конвеєрах, а також наукових обчисленнях; зокрема, в областях з великими наборами даних, таких як відображення геному, або там, де дво- або тривимірній аналіз корисний — зокрема, аналіз біомолекул, дослідження білків, і в інших складних галузях органічної хімії. Такі конвеєри можуть також значно поліпшити ефективність обробки зображень та комп'ютерного зору, поміж інших областей.

GPGPU це концепт програмного забезпечення, а не апаратного. Тим не менш, спеціалізовані конструкції устаткування можуть ще більше підвищити ефективність GPGPU трубопроводів, які традиційно виконують відносно мало алгоритмів на дуже великих обсягах даних. Масово розпаралелені завдання гігантських рівнів даних, таким чином, можуть бути розпаралелені ще більше за допомогою спеціалізованих установок, таких як обчислювальні стійки (багато подібних, вузькоспеціалізованих машин, побудованих в «стійці»), який додає третій шар — багато обчислювальних блоків кожен з яких використовує багато центральних процесорів щоб відповідати багатьом графічним.



*Рис. 3.2. Порівняння розробки програм CPU та GPU*

### **3.2.3 OpenCV**

Бібліотека (в програмуванні) - це файл або набір файлів, що містять підпрограми, функції, які використовуються для розробки програмного забезпечення. Кожна мова програмування має свій набір бібліотек, ці бібліотеки називаються стандартними, крім стандартних бібліотек є ще призначені для користувача бібліотеки, вони призначені для використання спеціалізованих завдань, розробляються програмістами. Сьогодні ми познайомимося з графічною бібліотекою розробленої компанією Intel на мові програмування С / С ++, яка широко застосовується для створення систем відеоспостереження [15].

Отже, OpenCV (open source computer vision library) - це бібліотека комп'ютерного зору з відкритим вихідним кодом. Також, вона існує для деяких інших мов, наприклад, для Java. Включає в себе різні алгоритми комп'ютерного зору, розпізнавання зображень і багато іншого, що працюють в реальному режимі часу. Всі бажаючі можуть використовувати бібліотеку OpenCV безкоштовно, як в освітніх цілях, так і в комерційних проектах.

Вона включає в себе наступні алгоритми:

* Розпізнавання об'єктів в потоці.
* Розпізнавання друкованого і рукописного тексту.
* Усунення спотворень картинки.
* Виявлення подібності та форми об'єктів.
* Стеження за переміщенням об'єкта.
* Розпізнавання рухів, жестів і багато іншого.

Підтримувані платформи та інструменти для OpenCV.

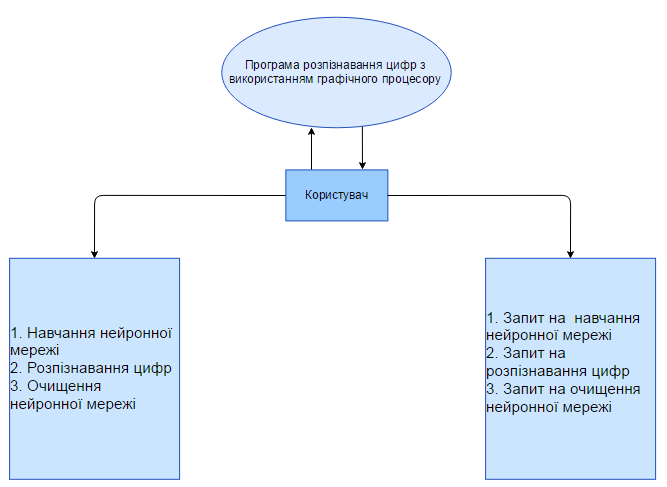
* Самі бібліотеки:
* Windows: компілятори Microsoft Visual C ++ (6.0, .NET 2003), Intel Compiler, Borland C ++, Mingw (GCC 3.x).
* Linux: GCC (2.9x, 3.x), Intel Compiler: "./configure-make-make install", RPM (spec файл включений в поставку)
* Використовуються C і "полегшений" C ++. Прагма і умовна компіляція використовуються дуже обмежено.

Засоби GUI, захоплення відео:

* Windows: DirectShow, VFW, MIL, CMU1394
* Linux: V4L2, DC1394, FFMPEG
* Документація: статичний HTML

# РОЗДІЛ 4. РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ

На даному рисунку показано діаграму, котра демонструє варіанти використання системи. На ній користувачі поставлені у відповідність узагальнені функції, які доступні їм у процесі роботи із системою.



*Рис. 4.1. Концептуальна модель*

## **4.1 Відношення з іншими програмами**

Програма успішно працює на різних операційних системах, з використанням різних графічних процесорів.

## **4.2 Модель взаємодії системи з сутностями**

Таблиця 4.1 показує взаємодію сутності «Користувач» з програмою відповідно до функцій програми.

Таблиця 4.1

|  |  |  |
| --- | --- | --- |
| **Користувач** | **Потік повідомлень** | **Програма** |
| 1. Запит на навчання нейронної мережі 2. Запит на розпізнавання цифр   3. Запит на очищення нейронної мережі | 1. Текстові стрічки  2. Текстові стрічки  3. Текстові стрічки | 1. Навчання нейронної мережі  2. Розпізнавання цифр  3. Очищення нейронної мережі |

Таблиця 4.2 показує взаємодію системи з сутністю «Користувач» відповідно до функцій програми.

Таблиця 4.2

|  |  |  |
| --- | --- | --- |
| **Програма** | **Потік повідомлень** | **Користувач** |
| 1. Навчання нейронної мережі 2. Розпізнавання цифр   3. Очищення нейронної мережі | 1. Інформація   2.Інформація  3. Інформація. | 1. Отримання результатів виконання  3.Отримання результатів виконання  3. Отримання результатів виконання |

# РОЗДІЛ 5. ВИБІР, РОЗРОБКА МЕТОДІВ І АЛГОРИТМІВ ФУНКЦІОНУВАННЯ АПАРАТНО-ПРОГРАМНИХ ЗАСОБІВ

**5.1. Опис вибраної методології проектування та життєвого циклу системи**

Для проектування системи використовуватиметься структурний підхід (Structured systems analysis and design method - SSADM). Сутність структурного підходу до розробки ІС полягає в її декомпозиції (розбитті) на під функції. Система розбивається на функціональні підсистеми, які в свою чергу діляться на під функції, що підрозділяються на завдання і так далі [16].

У структурному підході є дві групи засобів, що ілюструють функції та відношення між даними у системі. Кожній групі засобів відповідають певні види діаграм. Найбільш поширені з них є:

• SADT (Structured Analysis and Design Technique) моделі і відповідні функціональні діаграми;

• DFD (Data Flow Diagrams) діаграми потоків даних ;

• ERD (Entity-Relationship Diagrams) діаграми "сутність-зв'язок".

Для функціонального моделювання системи використаємо методологію IDEF0, яка заснована на методології SADT [17].

Для моделювання послідовностей та взаємозязків різних бізнес-процесів у системі у системі використаємо діаграму IDEF3 [18].

Для моделювання потоків даних системи використаємо діаграму потоків DFD [19].

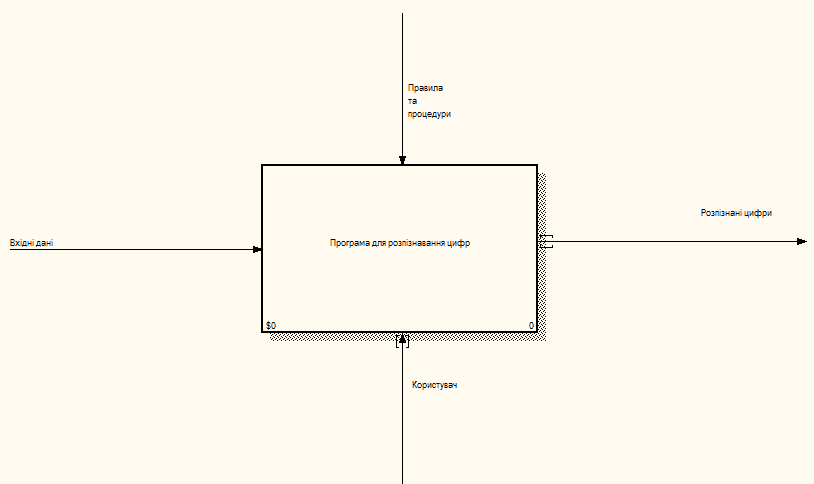
Для моделювання схеми даних використаємо діаграму «сутність-звязок» (ERD).

Для проектування моделей та діаграм обрані наступні case-засоби:

1. All Fusion Process Modeler - инструмент для моделюванния, анализу, документації та оптимізації бізнес-процесів. Засіб буде використовуватися для побудови моделей: IDEF0, IDEF3 та DFD;
2. Rational Rose - це програма для UML-моделювання та проектування. Засіб буде використовуватися для побудови UML діаграм: класів, послідовностей та компонентів;
3. Enterprice Architect - це програма для UML-моделювання та проектування. Засіб буде використовуватися для побудови UML діаграм: прецедентів та діяльності.

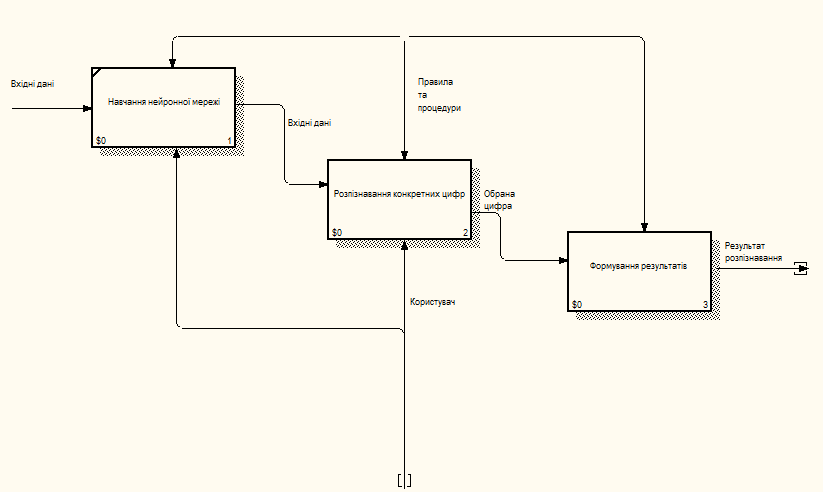
## **5.2. Побудова моделей бізнес-процесів за допомогою діаграм IDEF0**

Для функціонування моделювання і створення моделей основних бізнес-процесів використано діаграму IDEF0. На рис. 5.1 зображена контекстна діаграма основного бізнес-процесу роботи даної програми.

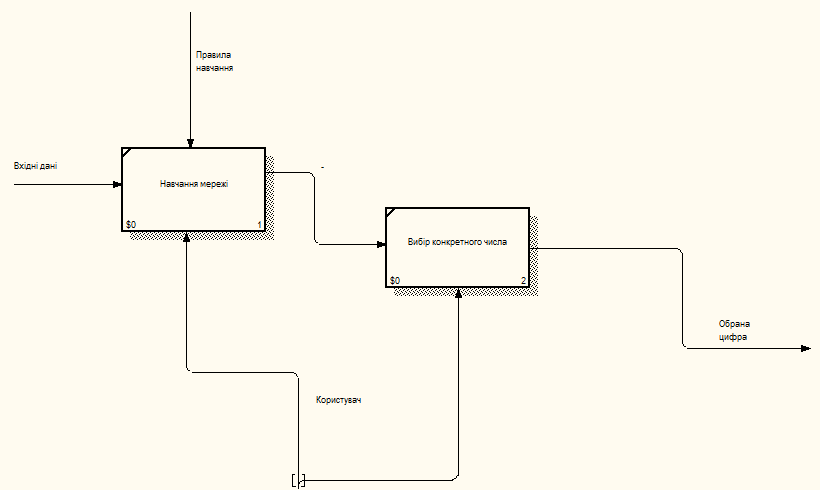


*Рис. 5.1. Контекстна діаграма*

Діаграми, які зображені на рис. 5.2 і 5.3 відображаються декомпозицію першого рівня та декомпозицію другого рівня «Навчання нейронної мережі».



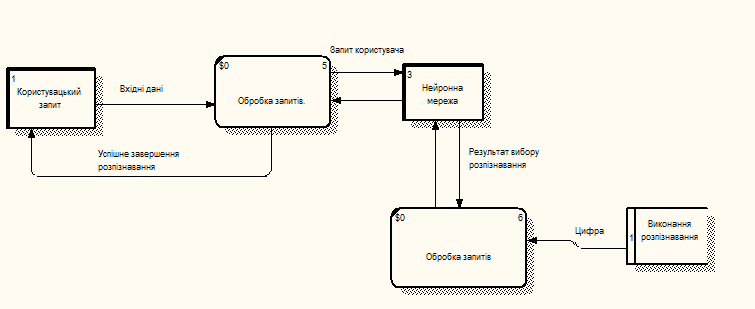
*Рис. 5.2. Діаграма декомпозиції першого рівня*



*Рис. 5.3. Діаграма декомпозиції другого рівня «Навчання нейронної мережі»*

## **5.3. Відображення потоків даних за допомогою DFD діаграми**

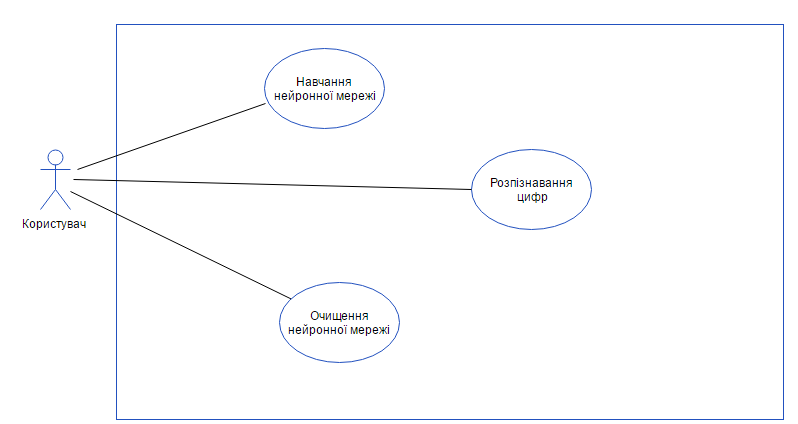
Для відображення потоків даних у системі використаємо діаграму DFD (Data Flow Diagram). На даній діаграмі відображаються потоки даних під час роботи системи. Діаграма зображена на рис. 5.4.



*Рис. 5.4. DFD діаграма*

5.4. Діаграма прецендентів

На даній діаграмі зображено основні функції програми.



*Таблиця 5.1. Специфікація прецеденту «Навчання нейронної мережі»*

|  |  |
| --- | --- |
| **Прецедент** | *Навчання нейронної мережі* |
| Короткий опис | Користувач може навчити нейронну мережу розпізнавати ті цифри, які він захоче |
| Суб’єкти | Користувач |
| Передумови | Користувач починає роботу з програмою |
| Основний потік | Успішне навчання нейронної мережі |
| Альтернативний потік | Навчання нейронної мережі не вибрано |

*Таблиця 5.2. Специфікація прецеденту «Очищення нейронної мережі»*

|  |  |
| --- | --- |
| **Прецедент** | *Очищення нейронної мережі* |
| Короткий опис | Користувач очищає нейронну мережу |
| Суб’єкти | Користувач |
| Передумови | Користувач має можливість очистити мережу, і навчити її спочатку |
| Основний потік | Успішне очищення нейронної мережі |
| Альтернативний потік | Очищення нейронної мережі не вибрано і Користувач буде працювати зі старою мережею, яку він навчив (або вона була навчена) |

**РОЗДІЛ 6. РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ**

**6.1. Опис використаних сторонніх бібліотек та модулів**

Оскільки завдання полягало розробити програму для розпізнавання цифр з використанням графічного процесору**,** тому для реалізації обчислень на GPU було вибрано бібліотеку OpenCV. Дана бібліотека містить понад 2500 оптимізованих алгоритмів, серед яких повний набір як класичних так і практичних алгоритмів машинного навчання і комп’ютерного зору. Алгоритми OpenCV застосовуються у різних сферах. Ця бібліотека популярна за рахунок своєї відкритості і можливості безплатно її використовувати.

**6.2. Принцип роботи програми**

Програма вчиться розпізнавати цифри на зображеннях. Побудована на нейронних мережах за принципом навчання з вчителем. Чим більше різних цифр буде вивчено програмою, тим краще буде працювати алгоритм розпізнавання.

Що ж відбувається при вивченні зображень:

1. Задається яка цифра зображена на ньому.

2. На зображення накладаються різні фільтри, щоб виокремити контур головного об’єкту на зображенні.

3. Зображення уніфікується по шаблону (зводиться до одного розміру).

4. Після уніфікування зображення додається до типового образу правдивої цифри і до не типових усіх інших.

Що ж відбувається при розпізнаванні зображення:

1. На зображення накладаються різні фільтри, щоб виокремити контур головного об’єкту на зображенні.

2. Зображення уніфікується по шаблону (зводиться до одного розміру).

3. Образ, який утворився накладається на типові образи кожної цифри і кількість співпадінь – буде ймовірністю, що це число правдиве.

4. Після порівння усіх образів вибирається число з найбільшою ймовірністю і повертається.

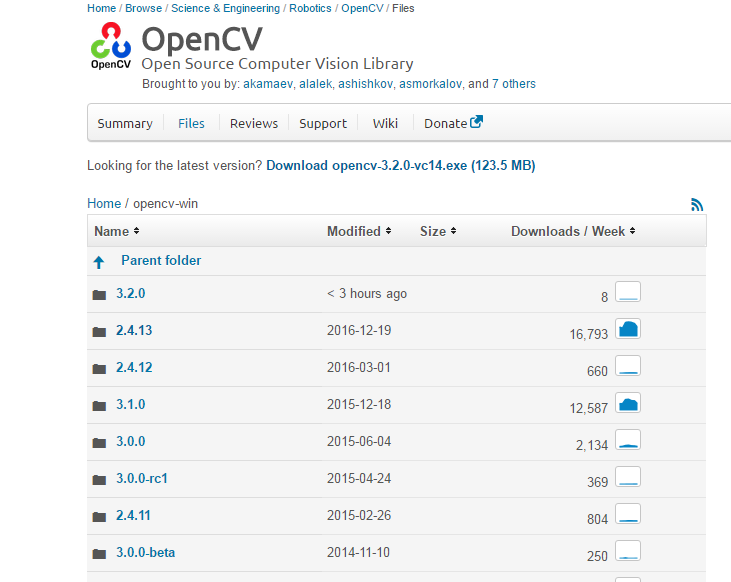
Для прискорення процесу навчання можливо завантажити у папку усі зображення, які бажаєте, щоб програма вивчила і передати адресу папки у програму. Програма тоді запустить процес навчання.

Також можна розпізнати образ на зображенні просто задавши адресу зображення.

**6.3. Установка**

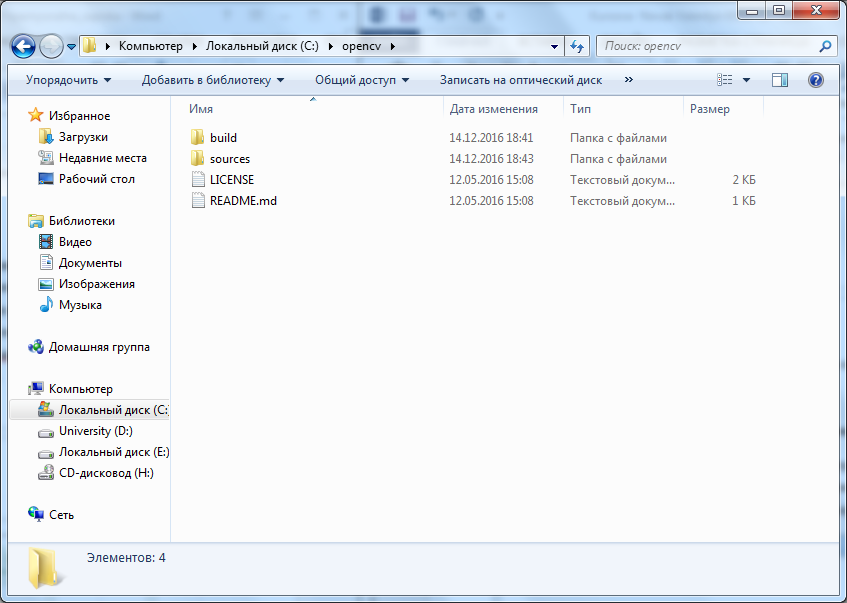
Програма має таку залежність, як бібліотеку OpenСV, тому без установки даної бібліотеки, програма не буде працювати.

1. Якщо ж OpenCV не встановлена на комп’ютер, то потрібно скачати та розпакувати її. Мінімальною версією повинна бути 2.2. Існує багато способів установки, но основним способом являється установка із вихідних кодів. Потрібно зайти на сайт *sourceforge.net/projects/opencvlibrary/files/opencv-win/,* вибрати версію, а також той інсталятор, який відповідає вашій операційній системі.



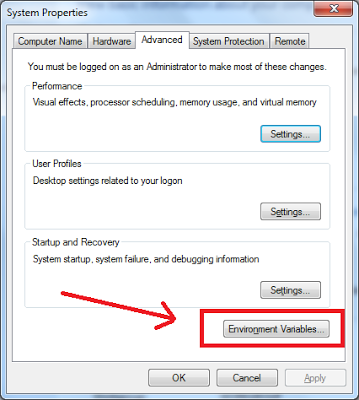
*Рис. 6.1. Вибір версії*

2. Отже, завантаживши файл, необхідно розпакувати на операційний диск.

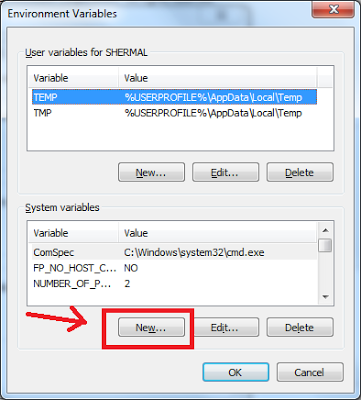


*Рис. 6.2. Розпаковка*

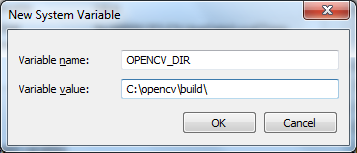
3. Наступним кроком буде настроювання OpenCV на комп’ютері. Заходимо «в розширені настройки комп’ютер» і вибираєм «Змінні середовища».



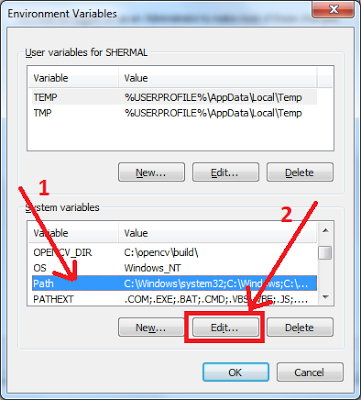
*Рис. 6.3. Вхід в змінні середовища*



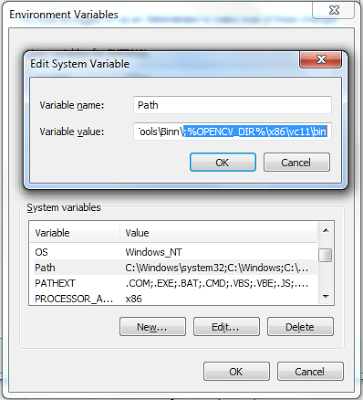
*Рис. 6.4. Вибираєм створення нового*



*Рис. 6.5. Створюємо нову змінну і заносим шлях до нашої бібліотеки*



*Рис. 6.6. Редагуємо системну змінну*

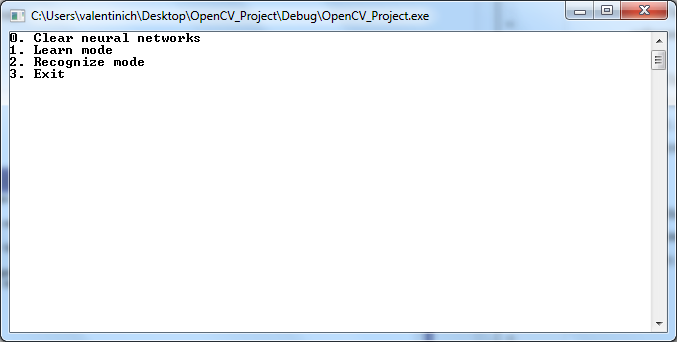


*Рис. 6.7. Додаємо нашу бібліотеку та вибираєм тип компілятора*

Тепер наша бібліотека налаштована на комп’ютері, тому спокійно можна запускати програму, оскільки у Visual уже настроєно всі шляхи до наших бібліотек.

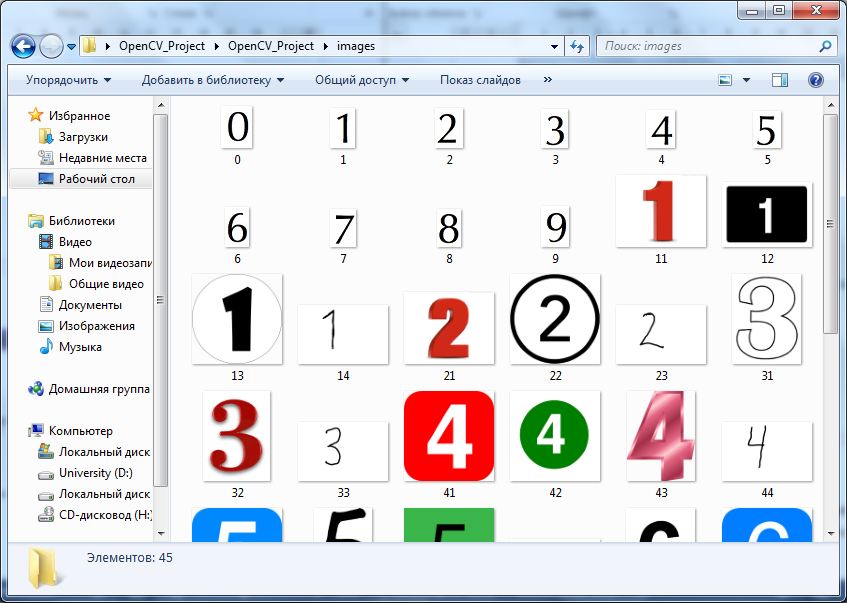
**6.4. Розробка та опис інтерфейсу користувача**

На рис.6.8 зображено головний інтерфейс програми. Інтерфейс побудований досить зрозумілим. З даного інтерфейсу користувач може очистити мережу, навчити, розпізнати цифри, а також вийти з даної програми.



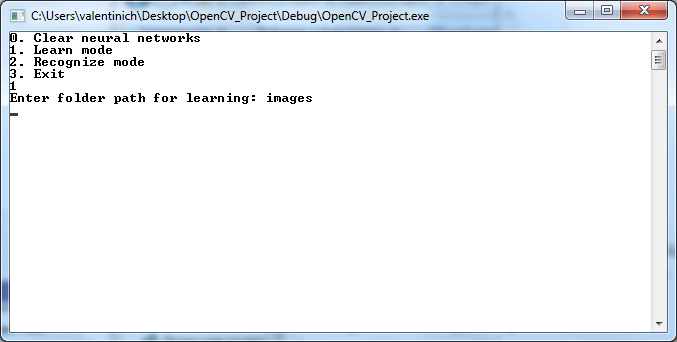
*Рис. 6.8. Головний інтерфейс*

Програма просить вибрати один з чотирьох пунктів. Якщо мережа не навчена, користувач має її навчити. Він повинен занести в коріневу папку свої зображення цифр по яких він хоче навчити дану мережу.



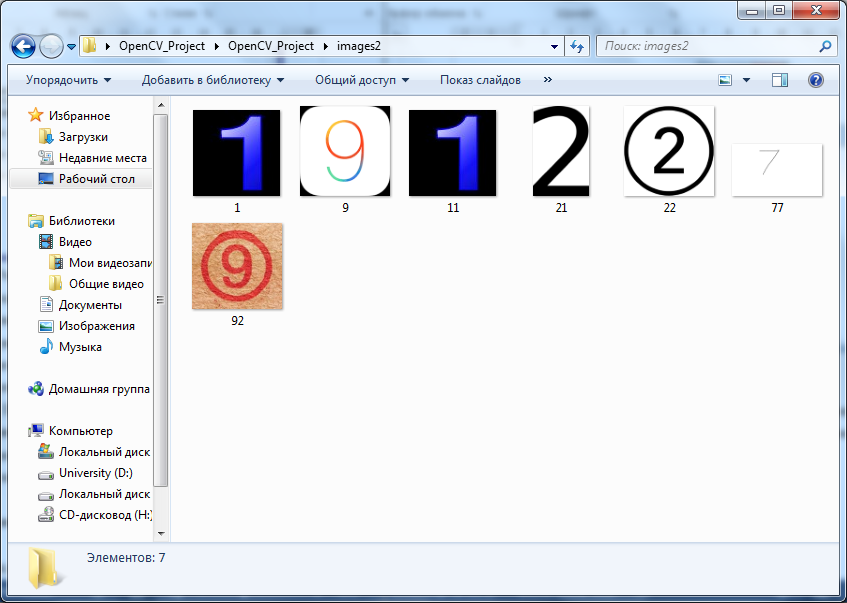
*Рис. 6.9. Папка із зображеннями по яких буде навчена мережа*

Користувача приступає до навчання мережі. Вибираючи пункт «1. Learn mode».



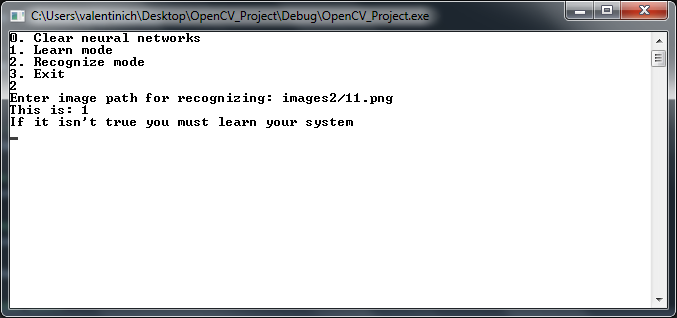
*Рис. 6.10. Навчання мережі*

Навчивши свою мережу користувач може далі розпізнавати зображення, йому необхідно занести в папку «images2» всі зображення, котрі він хоче розпізнати.



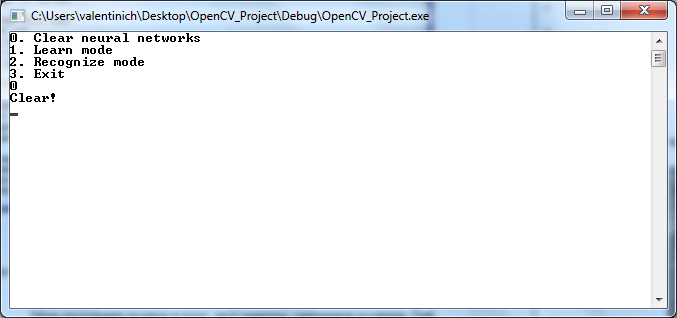
*Рис. 6.11. Папка із зображеннями, котрі необхідно розпізнати*

Після того як він занесе зображення, котрі потрібно розпізнати, він приступає до кроку розпізнавання.



*Рис. 6.12. Результат виконання програми*

На рис. 6.13. зображено очищення мережі.



*Рис. 6.13. Очищення мережі*

**ВИСНОВКИ**

В результаті виконання цієї курсової роботи було розроблено програму розпізнавання цифр на графічному процесорі з використанням такої технології, як OpenCV. Дана програма досить зручна у використанні, містить в собі велику кількість функцій, котрі оптимізують обробку зображень, а також операцій з матрицями.

Протягом виконання роботи було проведено детальний аналіз предметної області. Також було проаналізовано основні аналоги програми, які зараз присутні на ринку.

Після цього всього було створено детальний проект даної програми, який би задовольняв поставленим раніше вимогам. Користувацький інтерфейс проектувався таким чином, щоб він міг бути зрозумілим та інтуїтивним для користувачів.

Якщо ж говорити про те як система може розвиватись в майбутньому, то відразу можна сказати, що є велика кількість можливих шляхів розвитку. Планується створення можливості розпізнавати не тільки цифри, а також тексти, як друковані, так і рукописні.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. Розпізнавання образів [Електронний ресурс] – Режим доступу до ресурсу:

<https://uk.wikipedia.org/wiki/розпізнавання_образів>.

2. Біологічний нейрон [Електронний ресурс] – Режим доступу до ресурсу:

[https://](https://en.wikipedia.org/wiki/SADT) techtrend.com.ua/index..

3. Штучні нейронні мережі/ Е.В. Бодянський, О.Г. Руденко 2004 – 369с.

4. Одношарові нейронні мережі [Електронний ресурс] – Режим доступу до ресурсу: https:// www.studfiles.ru/preview/5461803.

5. Нейрон персептрона [Електронний ресурс] – Режим доступу до ресурсу:

https:// www.machinelearning.ru/wiki.

6. Нейронні мережі/ Хайкин Саймон 2006 – 1104с.

7. CuneiForm [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CuneiForm

8. FineaReader [Електронний ресурс] – Режим доступу до ресурсу: <https://www.abbyy.com/ru-ru/finereader/professional/>

9. ReadirisPro [Електронний ресурс] – Режим доступу до ресурсу:

https://readiris-pro.en.softonic.com/

10. OmniPage [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/OmniPage>

11. Дерево цілей [Електронний ресурс] – Режим доступу до ресурсу: <https://buklib.net/books>

12. MODI [Електронний ресурс] – Режим доступу до ресурсу: <https://support.microsoft.com/uk-ua>

13. Дерево цілей [Електронний ресурс] – Режим доступу до ресурсу: <https://buklib.net/books>

14. GPGPU[Електронний ресурс] – Режим доступу до ресурсу:

gpgpu.org/developer

15. OpenCV [Електронний ресурс] – Режим доступу до ресурсу:

opencv.org/

16. SSADM [Електронний ресурс] – Режим доступу до ресурсу: <http://en.wikipedia.org/wiki/Structured_systems_analysis_and_design_method>.

17. SADT [Електронний ресурс] – Режим доступу до ресурсу:

<https://en.wikipedia.org/wiki/SADT>.

18. IDEF3 [Електронний ресурс] – Режим доступу до ресурсу:

https://uk.wikipedia.org/wiki/IDEF3.

19. DFD [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/DFD>.

**ДОДАТОК А**

int main()

{

Mat image;

wstring wpath;

string path;

uint mode, digit;

while (true) // Вічний цикл (переривання всередині)

{

cout << "0. Clear neural networks\n"

"1. Learn mode\n"

"2. Recognize mode\n"

"3. Exit" << endl;

cin >> mode;

switch (mode)

{

case 0:

initInfoFiles();

cout << "Clear!" << endl;

getch();

break;

case 1:

cout << "Enter folder path for learning: ";

wcin >> wpath;

learnImageArray(wpath + L'/');

break;

case 2:

cout << "Enter image path for recognizing: ";

cin >> path;

image = imread(path, CV\_LOAD\_IMAGE\_GRAYSCALE);

if (!image.data)

{

cout << "Couldn't load image" << endl;

getch();

break;

}

digit = recognizeDigitOnImage(image);

cout << "This is: " << digit << endl;

cout << "If it isn't true you must learn your system" << endl;

getch();

break;

default:

exit(0);

break;

}

system("cls");

}

}

void initInfoFiles()

{

string fileName = "im\_3\_5.bmp";

for (uint i = 0; i < nDigits; ++i)

{

// Задаємо цифру

fileName[3] = (char)i + '0';

// Задаєм атрибут файлу - типовий

fileName[5] = fileAttrTypical;

// Створюємо 0 матрицю відповідних розмірів і записуємо у файл

imwrite(fileName, Mat::zeros(ppImageRows, ppImageCols, CV\_8U));

// Задаєм атрибут файлу - не типовий

fileName[5] = fileAttrATypical;

// Створюємо 0 матрицю відповідних розмірів і записуємо у файл

imwrite(fileName, Mat::zeros(ppImageRows, ppImageCols, CV\_8U));

}

}

void printMat(const Mat &mat)

{

for (int i = 0; i < mat.rows; ++i)

{

for (int j = 0; j < mat.cols; ++j)

{

cout << +mat.at <uchar>(i, j);

}

cout << endl;

}

}

void printInfoFiles()

{

string fileName = "im\_3\_5.bmp";

for (uint i = 0; i < nDigits; ++i)

{

// Задаєм цифру в імя файлу

fileName[3] = (char)i + '0';

// Задаємо атрибут - типовий

fileName[5] = fileAttrTypical;

// Зчитуємо матрицю з довільною глибиною

Mat ones = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

cout << "Image " << i << endl;

printMat(ones);

cout << endl;

fileName[5] = fileAttrATypical;

// Зчитуємо матрицю з довільною глибиною

Mat zeros = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

// Виводимо матрицю

printMat(zeros);

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

system("pause >> void");

system("cls");

}

}

void PreProcessImage(const Mat &inImage, Mat &outImage)

{

// Змінні для різних фільтрів

Mat grayImage, blurredImage, thresholdImage, contourImage, regionOfInterest;

// дво вимірний Вектор контурів

vector<vector<Point> > contours;

//cvtColor(inImage, grayImage, COLOR\_BGR2GRAY);

// Накладаємо фільтри

GaussianBlur(inImage, blurredImage, Size(5, 5), 2, 2);

adaptiveThreshold(blurredImage, thresholdImage, 255, 1, 1, 11, 2);

thresholdImage.copyTo(contourImage);

// Пошук контурів

findContours(contourImage, contours, RETR\_LIST, CHAIN\_APPROX\_SIMPLE);

int idx = 0;

size\_t area = 0;

for (size\_t i = 0; i < contours.size(); i++)

{

if (area < contours[i].size())

{

idx = i;

area = contours[i].size();

}

}

// Формуємо прямокутник по краях контуру

Rect rec = boundingRect(contours[idx]);

// Обрізаємо до мінімального розміру

regionOfInterest = thresholdImage(rec);

// Підлаштовуємо під один шаблон

resize(regionOfInterest, outImage, Size(ppImageCols, ppImageRows));

}

int getDigitProbabilityOn(const Mat &image, const uint digit)

{

string fileName = "im\_3\_5.bmp";

// Задаєм цифру

fileName[3] = (char)digit + '0';

// Задаєм атрибут

fileName[5] = fileAttrTypical;

// Зчитуєм матрицю з файлу

const Mat imTypical = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

// Міняєм атрибут

fileName[5] = fileAttrATypical;

// Зчитуєм матрицю з файлу

const Mat imATypical = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

// Формуємо матрицю 1 і 0 ( є там щось чи нема)

const MatExpr bitImage = (image != 0) - 254;

// Сумуємо елементи матриці, повертаємо результат

return (int)cv::sum(imTypical.mul(bitImage))[0];// -(int)cv::sum(imATypical.mul(bitImage))[0];

}

uint recognizeDigitOnImage(const Mat &image)

{

int maxProbability = INT\_MIN;

uint digit = 0;

Mat outImage;

// Обробляємо зоображення

PreProcessImage(image, outImage);

for (uint i = 0; i < nDigits; ++i)

{

// Дістаємо ймовірність і"ї цифри на зоображенні

const int &probability = getDigitProbabilityOn(outImage, i);

// Обираємо цифру з максимальною ймовірністю

if (maxProbability < probability)

{

digit = i;

maxProbability = probability;

}

}

return digit;

}

void type(const uint digit, const Mat &image)

{

// Задаємо маску файлу

string fileName = "im\_3\_5.bmp";

Mat imTypical, ppImage;

fileName[3] = (char)digit + '0';

fileName[5] = fileAttrTypical;

// Обробляємо зоображення

PreProcessImage(image, ppImage);

// Зчитуєм матрицю позитивних образів

imTypical = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

// Додаємо образ до типового зоображення для цифри

imTypical += (ppImage != 0) - 254;

// Записуємо у файл

imwrite(fileName, imTypical);

}

void untype(const uint digit, const Mat &image)

{

// Маска назви файлу

string fileName = "im\_3\_5.bmp";

Mat imATypical, ppImage;

// Задаєм цифру

fileName[3] = (char)digit + '0';

// Атрибут

fileName[5] = fileAttrATypical;

// Обробляємо зоображення

PreProcessImage(image, ppImage);

// Зчитуємо матрицю негативних образів

imATypical = imread(fileName, CV\_LOAD\_IMAGE\_ANYDEPTH);

// Зводимо до матриці 0 і 1

imATypical += (ppImage != 0) - 254;

imwrite(fileName, imATypical);

}

void learnImage(Mat image, uint trueDigit)

{

//probablyDigit = recognizeDigitOnImage(image);

for (uint i = 0; i < nDigits; ++i)

{

if (i == trueDigit)

{

// Додаєм образ до типових

type(trueDigit, image); // assist true digit

}

else

{

// Додаєм образ до не типових

untype(i, image);

}

}

}

void learnImageArray(const wstring &path)

{

// Створюємо змінні

Mat image;

wstring fileName, fileType;

uint digit;

WIN32\_FIND\_DATA f;

// Створюємо файл для зчитування

HANDLE h = FindFirstFile((path + L'\*').c\_str(), &f);

if (h != INVALID\_HANDLE\_VALUE)

{

// Ящко папка існує

do

{

// Зчитуємо імя наступного файлу

fileName = path + wstring(f.cFileName);

// Вибираємо тип файлу

fileType = fileName.substr(fileName.find\_last\_of('.'));

if (fileType == L".jpg" || fileType == L".png" || fileType == L".bmp")

{

// Якщо тип дозволений зчитуєм матрицю у сірих тонах

image = imread(string(fileName.begin(), fileName.end()), CV\_LOAD\_IMAGE\_GRAYSCALE);

// Виводимо на екран чекаючи нажаття клавіші користувачем

imshow("Image", image); waitKey(0);

// Після нажаття клавіші знищуємо вікно

destroyWindow("Image");

// Запитуємо що це за цифра

cout << "What digit it is? " << endl;

cin >> digit;

// Вивчаємо дане зоображення

learnImage(image, digit);

}

} while (FindNextFile(h, &f));//

}

else

{

fprintf(stderr, "Error opening directory\n");

}

}