Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-22М-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Окунєв М.Ю.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Дрєєв О.М.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 123 *"Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Окунєву Максиму Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів*

2. Керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 35-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Окунєв М.Ю. Дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи радіально-базисної нейронної мережі для розпізнання образів.

Метою розробки є дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Об'єктом дослідження є процес радіально-базисної нейронної мережі для розпізнання образів.

Предметом дослідження є методи радіально-базисної нейронної мережі для розпізнання образів.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.4.1.

Ключові слова: комп'ютерна інженерія, нейронна мережа, розпізнання образів

ABSTRACT

Okuniev M.Yu. Research and software implementation of the radial basis neural network system for pattern recognition. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the radial basis neural network system for pattern recognition.

The purpose of the development is the research and software implementation of the radial basis neural network system for pattern recognition.

The object of research is the process of the radial-basis neural network for pattern recognition.

The subject of the research is radial-basis neural network methods for pattern recognition.

Research methods are based on artificial intelligence methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the radial basis neural network system for pattern recognition.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10.4.1 environment.

Keywords: computer engineering, neural network, pattern recognition

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ААУ	–	автономне адаптивне управління
БД	–	блок датчиків
БЗ	–	база знань
БОС	–	блок оцінки стану
БПР	–	блок прийняття рішень
ГІП	–	графічний інтерфейс користувача (GUI)
ВО	–	виконавчий орган
ЕОМ	–	електрона обчислювальна машина
КА	–	кінцевий автомат
НРС	–	недетермінований автомат Рабина-Скотта
НМ	–	нейронна мережа
МНРС	–	модифікований недетермінований автомат Рабина-Скотта
НАНУ	–	Національна академія наук України
ОУ	–	об'єкт управління
ПЧО	–	просторово-часовий образ
ВВ	–	випадкова величина
ПЗ	–	програмне забезпечення
СПДНМ	–	система побудови й дослідження нейронних мереж
СРНМ	–	Система розробки нейронних мереж
УС	–	управляюча система
ФР	–	функція розподілу
ФРО	–	апарат формування й розпізнавання образів
Z^+	–	множина ненегативних цілих чисел
$G(V, N)$	–	граф із множиною вершин V і множиною ребер N
$i \to j$	–	ребро, спрямоване з вершини i у вершину j
$X \leftrightarrow Y$	–	взаємооднозначне відображення множини X на множину Y

$\pi(X)$	–	множина кінцевих підмножин множини X
$R[a,b]$	–	множина речовинних чисел на $[a,b]$, $R \equiv R[-\infty,+\infty]$
B^N	–	простір двійкових векторів розмірності N
Λ	–	порожнє слово із множини вхідних слів КА
0	–	*неправда* у вираженні тризначної логіки
1	–	*істина* у вираженні тризначної логіки
$\lozenge$	–	*невизначеність* у вираженні тризначної логіки
$\vec{X} \subset \vec{Y}$	–	$\vec{X}$ є підвектор (сукупність обраних компонентів) вектора $\vec{Y}$
$X \supset Y$	–	клас Y є нащадком класу X

ВСТУП

Актуальність теми. В останні роки розпізнавання образів знаходить все більше застосування в повсякденному житті. Розпізнавання мови й рукописного тексту значно спрощує взаємодію людини з комп'ютером, розпізнавання друкованого тексту використовується для перекладу документів в електронну форму. Алгоритми, що лежать в основі розпізнавання, досить очевидні, потрібно лише визначитися з термінами. Базовим є невизначене поняття множини. У комп'ютері множина представляється набором неповторюваних однотипних елементів. Слово "неповторюваних" означає, що якийсь елемент у множини або є, або його там немає. Універсальна множина включає всі можливі для розв'язуваного завдання елементи, порожнє не містить жодного. У класичній постановці завдання розпізнавання універсальна множина розбивається на частки-образи. Образ якого-небудь об'єкта задається набором його приватних проявів. У випадку з розпізнаванням тексту в універсальну множину увійдуть всі можливі знаки, в образ "И" – всі можливі накреслення цієї букви, а програма розпізнавання займається тим, що на основі невеликого набору прикладів накреслень кожної букви (навчальної вибірки) визначає, яку з них символізує уведена карлючка. Методика віднесення елемента до якого-небудь образу називається вирішальним правилом. Ще одне важливе поняття – метрика, спосіб визначення відстані між елементами універсальної множини. Чим менша ця відстань, тим більше схожими є символи, звуки – те, що ми розпізнаємо. Звичайно елементи задаються у вигляді набору чисел, а метрика – у вигляді функції. Від вибору подання образів і реалізації метрики залежить ефективність програми, один алгоритм розпізнавання з різними метриками буде помилятися з різною частотою.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем радіально-базисної нейронної мережі для розпізнання образів.

– Дослідження системи радіально-базисної нейронної мережі для розпізнання образів.

– Програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Об'єктом дослідження є процес радіально-базисної нейронної мережі для розпізнання образів.

Предметом дослідження є методи радіально-базисної нейронної мережі для розпізнання образів.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод радіально-базисної нейронної мережі для розпізнання образів.

– Розроблено вітчизняний продукт радіально-базисної нейронної мережі для розпізнання образів, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі радіально-базисної нейронної мережі для розпізнання образів.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для реалізації радіально-базисної нейронної мережі для розпізнання образів. Розпізнавання являє собою інформаційний процес, реалізований деяким перетворювачем інформації (інтелектуальним інформаційним каналом, системою розпізнавання), що має вхід і вихід. На вхід системи подається інформація про те, якими ознаками володіють пропоновані об'єкти. На виході системи відображається інформація про те, до яких класів (узагальнених образів) віднесені розпізнавані об'єкти.

По суті це завдання є завданням кодування. Складається список узагальнених класів, до яких можуть відноситися конкретні реалізації об'єктів, а також список ознак, якими ці об'єкти в принципі можуть володіти.

При створенні й експлуатації автоматизованої системи розпізнавання образів вирішується ряд завдань. Розглянемо коротко й спрощено ці завдання. Відзначимо, що в різних авторів формулювання цих завдань, та й сам набір не збігаються, так як він деякою мірою залежить від конкретної математичної моделі, на якій заснована та або інша система розпізнавання. Крім того, деякі завдання в певних моделях розпізнавання не мають рішення й, відповідно, не ставляться.

Завдання формування навчальної вибірки

Навчальна вибірка являє собою базу даних, що містить опису конкретних реалізацій об'єктів мовою ознак, доповнену інформацією про приналежність цих об'єктів до певних класів розпізнавання.

Завдання навчання системи розпізнавання

Навчальна вибірка використовується для формування узагальнених образів класів розпізнавання на основі узагальнення інформації про те, якими

ознаками володіють об'єкти навчальної вибірки, що ставляться до цього класу й інших класів.

Завдання зниження розмірності простору ознак

Після навчання системи розпізнавання (одержання статистики розподілу частот ознак по класах) стає можливим визначити для кожної ознаки її цінність для рішення завдання розпізнавання. Після цього найменш коштовні ознаки можуть бути вилучені із системи ознак. Потім система розпізнавання повинна бути навчена заново, так як в результаті видалення деяких ознак статистика розподілу ознак, що залишилися, по класах змінюється. Цей процес може повторюватися, тобто бути ітераційним.

Завдання розпізнавання

Розпізнаються об'єкти розпізнаваної вибірки, що, зокрема, може складатися й з одного об'єкта. Розпізнавана вибірка формується аналогічно навчальної, але не містить інформації про приналежність об'єктів до класів, так як саме це й визначається в процесі розпізнавання. Результатом розпізнавання кожного об'єкта є розподіл або список всіх класів розпізнавання в порядку убування ступеня подібності розпізнаваного об'єкта з ними.

Завдання контролю якості розпізнавання

Після розпізнавання може бути встановлена його адекватність. Для об'єктів навчальної вибірки це може бути зроблене відразу, так як для них просто відомо, до яких класів вони ставляться. Для інших об'єктів ця інформація може бути отримана пізніше. У кожному разі може бути визначена фактична середня ймовірність помилки по всіх класах розпізнавання, а також імовірність помилки при віднесенні розпізнаваного об'єкта до певного класу.

Результати розпізнавання повинні інтерпретуватися з урахуванням наявної інформації про якість розпізнавання.

Завдання адаптації

Якщо в результаті виконання процедури контролю якості установлено, що воно незадовільне, то опису неправильно розпізнаних об'єктів можуть бути

скопійовані з розпізнаваної вибірки в навчальну, доповнені адекватною класифікаційною інформацією й використані для переформування вирішальних правил, тобто враховані. Більше того, якщо ці об'єкти не ставляться до вже наявних класів розпізнавання, що й могло бути причиною їхнього невірного розпізнавання, те цей список може бути розширений. У результаті система розпізнавання адаптується й починає адекватно класифікувати ці об'єкти.

Зворотне завдання розпізнавання

Завдання розпізнавання полягає в тому, що для даного об'єкта по його відомих ознаках системою встановлюється його приналежність до якогось раніше невідомому класу. У зворотному завданні розпізнавання, навпаки, для даного класу розпізнавання системою встановлюється, які ознаки найбільш характерні для об'єктів даного класу, а які немає (або які об'єкти навчальної вибірки ставляться до даного класу).

Завдання кластерного й конструктивного аналізу

Кластерами називаються такі групи об'єктів, класів або ознак, що усередині кожного кластера вони максимально подібні, а між різними кластерами – максимально різні. Конструктом (у контексті, розглянутому в даному розділі) називається система протилежних кластерів. Таким чином, у певному змісті конструкти є результат кластерного аналізу кластерів. У кластерному аналізі кількісно виміряється ступінь подібності й розходження об'єктів (класів, ознак), і ця інформація використовується для класифікації. Результатом кластерного аналізу є сама класифікація об'єктів по кластерах. Ця класифікація може бути представлена у формі семантичних мереж.

Завдання когнітивного аналізу

У когнітивному аналізі інформація про подібність і розходження класів або ознак цікавить дослідника сама по собі, а не для того, щоб використовувати неї для класифікації, як у кластерному і конструктивному аналізі. Якщо для двох класів розпізнавання є характерним той самий ознака, то це вносить вклад у подібність цих двох класів. Якщо ж для одного із класів ця ознака є

нехарактерним, то це вносить вклад у розходження. Якщо дві ознаки корелюють одна з одною, то в певному змісті їх можна розглядати як одну ознаку, а якщо антикорелюють, то як різні. З урахуванням цієї обставини наявність різних ознак у різних класів також вносить певний вклад у їхню подібність і розходження. Результати когнітивного аналізу можуть бути представлені у формі когнітивних діаграм.

1.2 Область застосування

Область застосування розробляє мого у даній роботі програмного забезпечення наведемо на прикладі продуктів Google.

Людина й комп'ютер по-різному сприймають зображення й відео. Глянувши на фотографію, ви відразу впізнаєте кращого друга – навіть якщо знімок не кращої якості. Для комп'ютера та ж картинка являє собою дані, з яких можна витягти інформацію про контури предметів і їхній колір. Незважаючи на те що в нього відсутні органи почуттів, його можна навчити розпізнавати певні об'єкти, у тому числі особи. Google використовує таку технологію, щоб забезпечити конфіденційність. Наприклад, на зображеннях перегляду вулиць комп'ютер визначає особи перехожих, що потрапили в об'єктив камери, і розмиває їх. Також завдяки цій технології система знаходить на знімках і у відеороликах Google+ знайомих і пропонує їх відзначити (хоча й не може сама визначити, хто це).

При розпізнаванні образів можуть ураховуватися й різні деталі. Існують шаблони, по яких можна припустити, що людина має певні відмітні ознаки, наприклад носить бороду або окуляри. Ця ж технологія дозволяє справлятися з ефектом червоних очей. А якщо ви користуєтеся сервісом Hangouts, то комп'ютер зможе підняти вам настрій, помістивши під вашим носом шикарні вуси або надягши на вас пенсне.

Як випливає з назви, ця технологія дозволяє комп'ютеру порівнювати вже відомі йому особи з новими, і Google використовує її в ряді своїх функцій.

Система може визначати схожі зображення і ймовірні збіги. Допустимо, користувач завантажив фотографію й хоче їй поділитися. При використанні функції "Знайди мене на фото" йому буде запропонований список людей, які можуть бути присутнім на знімку і яких можна на ньому відзначити. Те ж саме стосується й відео

Включивши Голосовий пошук у додатку Google на пристрої, можливо вимовляти запити, а не друкувати їх. Цей інструмент теж використовує технологію розпізнавання образів. Після кожного запиту в Голосовому пошуку ми зберігаємо дані про мову й країну, саме висловлення й сприйняті системою слова. Збережені аудіодані не будуть зв'язуватися з ідентифікатором аккаунта Google, поки не активується ця можливість у налаштуваннях. Нам відсилаються не всі висловлення, а тільки ті, які були вимовлені після включення Голосового пошуку: наприклад, якщо ви нажали значок мікрофона на панелі швидкого пошуку або на віртуальній клавіатурі або вимовили вголос "гугл", коли пристрій показує, що функція Голосового пошуку доступна. Наші сервери обробляють аудіодані й визначають, що саме ви сказали, а також зберігають їх, щоб краще розпізнавати інші запити.

Можливо також навести наступний приклад. Дві групи дослідників, незалежно друг від друга, одночасно розробили технології розпізнавання образів, які здатні описувати зображення людською мовою. Перша група працювала в Стенфордському університеті, а друга – у компанії Google. Обидві програми виконують однакове завдання: розпізнавання об'єктів у кадрі. Після цього вони становлять словесний опис кадру. Як показав експеримент, у деяких випадках цей опис досить точний й практично не відрізняється від опису людиною.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

OCR Pro

Недорога програма із простим і некрасивим інтерфейсом. Хоч на працездатності це й не позначається, але приставка Pro у назві може ввести користувача в оману. З такою назвою хотілося б програму з більш кращим інтерфейсом.

Рисунок 2.1 – Інтерфейс користувача OCR Pro

Те ж саме стосується й функціонала. Робимо фото або вибираємо його з бібліотеки, вибираємо мова розпізнавання, натискаємо розпізнати й бачимо текст, який можна скопіювати або розмножити вісьма різними способами, включаючи друк. Втім, це можна зробити й пізніше, документи зберігаються автоматично. Додаток вимагає підключення до мережі, без її зробити нічого не вдасться. Зате швидкість і точність розпізнавання тут на висоті. Звичайно, огріхи є, як і у всіх програм подібного роду, але їх дуже мало.

Властиво, основну функцію OCR Pro виконує добре, але може віджахнути корявий інтерфейс.

Переваги: швидкість і точність розпізнавання, підтримує велика кількість мов, багаті соціальні можливості.

Недоліки: корявий інтерфейс, для розпізнавання потрібен Інтернет, невеликого функціонала.

OCR Scanner

Майже брат-близнюк розглянутої вище програми OCR Pro. Такий же слабкий інтерфейс, тільки із ще меншими можливостями. Додаток має статус "тріал". Тобто, безкоштовно він може розпізнати всього п'ять документів.

Працює всі теж досить просто. Після обраного або зробленого фото, програма надає текст, який можливо скопіювати або відправити по електронній пошті. Швидкість і точність розпізнавання точнісінько як в OCR Pro. Видимо це пов'язане з тим, що додаток теж працює через Інтернет і не виключено, що на тому самому движку.

Основний плюс OCR Scanner у тому, що якщо вам не потрібно розпізнавати велика кількість документації, те й заплатити за це можна поменше.

Переваги: безкоштовна демо-версія, швидкість і точність розпізнавання, підтримує велику кількість мов.

Недоліки: слабкий інтерфейс, для розпізнавання потрібен Інтернет, дуже слабкий функціонал.

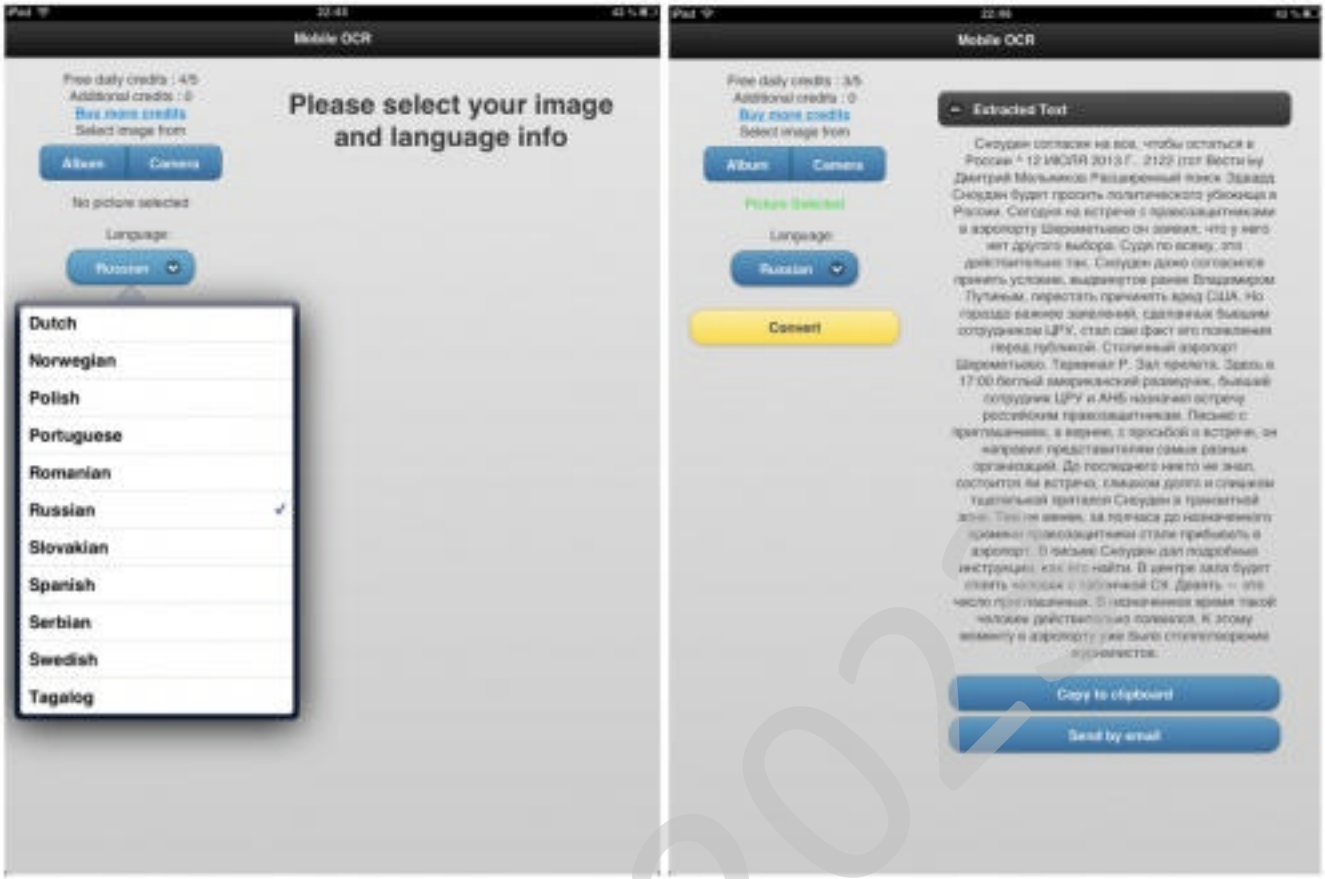

Рисунок 2.2 – Інтерфейс користувача OCR Scanner

TextGrabber + Translator

Не дивно, що такий монстр як ABBYY вирішив освоїти мобільний ринок, адже компанія розвивається аж з 1989 року. Дивно тільки, що їхні додатки дотепер не оптимізовані для iPad. Хоча й з успіхом на ньому працюють.

Ми вирішили не розглядати FineReader, а написати про більш функціональний продукт TextGrabber, хоча по суті це те саме, тільки з убудованим перекладачем.

Інтерфейс додатку приємний і дружелюбний. Все ясно з першого разу, тим більше є українська локалізація. Далі, теж одні плюси. По-перше, розпізнавання не вимагає підключення до мережі. По-друге, текст розпізнається мультимовно, чого немає в жодній із представлених програм. Інакше кажучи, якщо в документі присутні слова на двох мовах, то додаток безпомилково їх

визначить. По-третє, є обрізка фото, для виділення тільки тексту і як наслідок, більше швидкої роботи. По-четверте, у програму інтегровані популярні соціальні мережі, пошук і словник. В-п'ятих, це убудований перекладач і про нього поговоримо окремо.

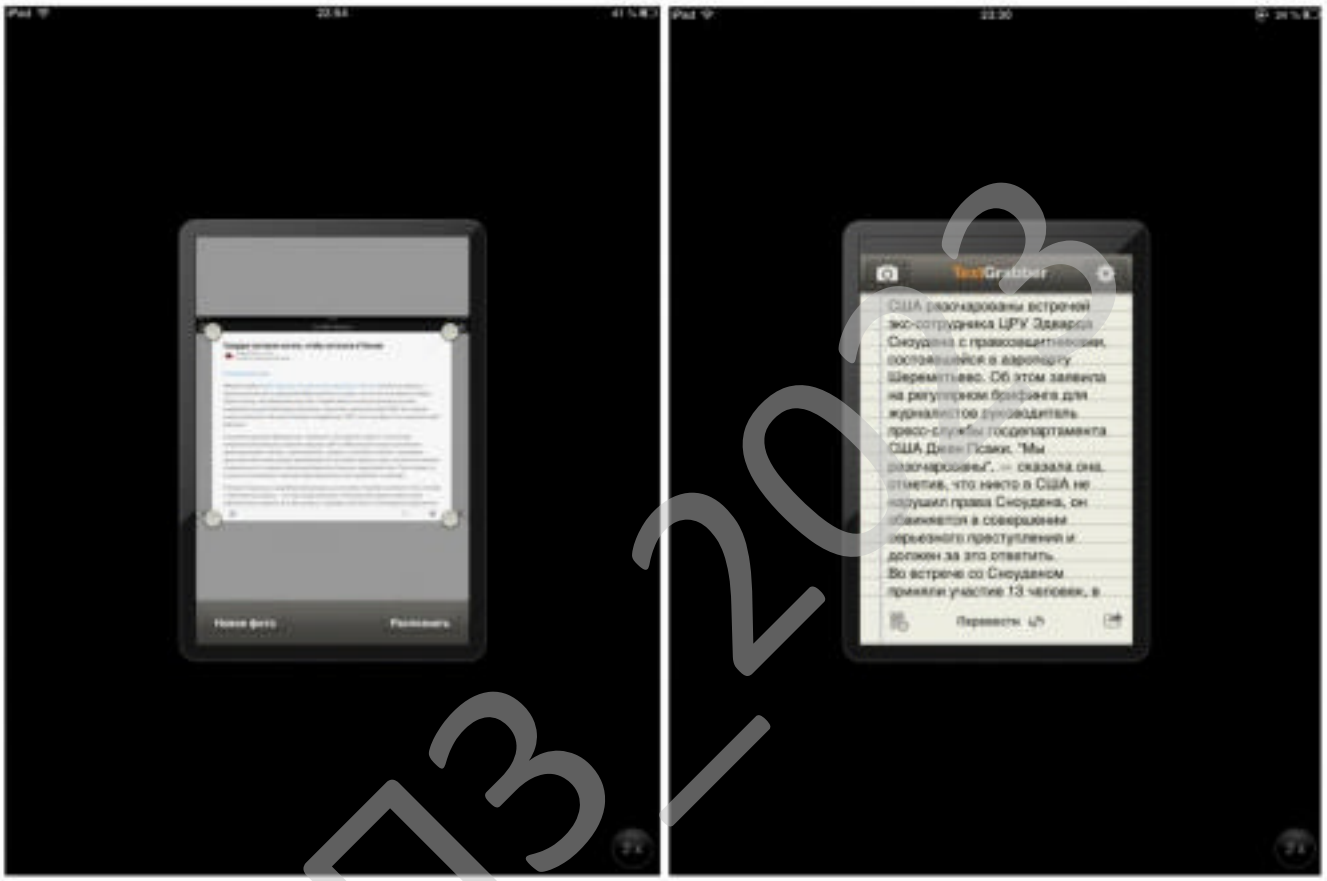

Рисунок 2.3 – Інтерфейс користувача TextGrabber + Translator

Мені наприклад, дуже зручно робити скріншоти англомовних додатків з App Store, розпізнавати й відразу переводити, адже копіювати текст звідти не можна. Правда на відміну від розпізнавання, перекладач працює через Інтернет. Це накладає деякі обмеження при роботі поза мережею, але проте фішка дуже приємна.

Відмінний продукт, що подійде більшості користувачів, незважаючи на маленьке віконце на великому екрані.

Переваги: приємний інтерфейс, швидкість і точність розпізнавання, підтримує велика кількість мов, мультимовне розпізнавання тексту, для розпізнавання не потрібен Інтернет, убудований перекладач, гарний функціонал.

Недоліки: для перекладача потрібен Інтернет, не оптимізована для iPad.

Prizmo – Scanning, OCR, and Speech

Самий функціональний і найдорожчий додаток із цього огляду. Відразу хотілося б відзначити, що існують настільна й мобільна версії програми, із синхронізацією через iCloud. Правда, для Mac OS ціна досить висока.

З убудованих можливостей тут присутні: розпізнавання тексту, розпізнавання візиток, перекладач і перетворення тексту в мову (купується окремо). Почнемо один по одному.

Інтерфейс непоганий, але не дуже дружелюбний у плані юзабіліті. Розібратися начебто не складно, а от працювати небагато не зручно.

Функція розпізнавання тексту може працювати оффлайн, що вже радує. Безліч мов розпізнавання, а от по-умовчанню коштують не все. Їх потрібно захитати окремо, через меню настроювань. Працює OCR досить точно, але декілька повільніше інших додатків. Зате, є кілька фішок для редагування документа на місці. Наприклад, можна відразу видалити непотрібні абзаци й картинки. До речі, програма вміє перетворювати в PDF.

Візитні картки – (ім'я й прізвище, назва компанії, посада, номери телефонів, електронної пошти, веб-сайт, адреса), все це можна витягти, щоб створити контакти для телефону або поділитися ними в якості vCard.

Перекладач працює тільки через Інтернет, що в загальному не дивно. Ну, і додаткова функція у вигляді перетворення тексту в мову, комусь може виявитися корисною.

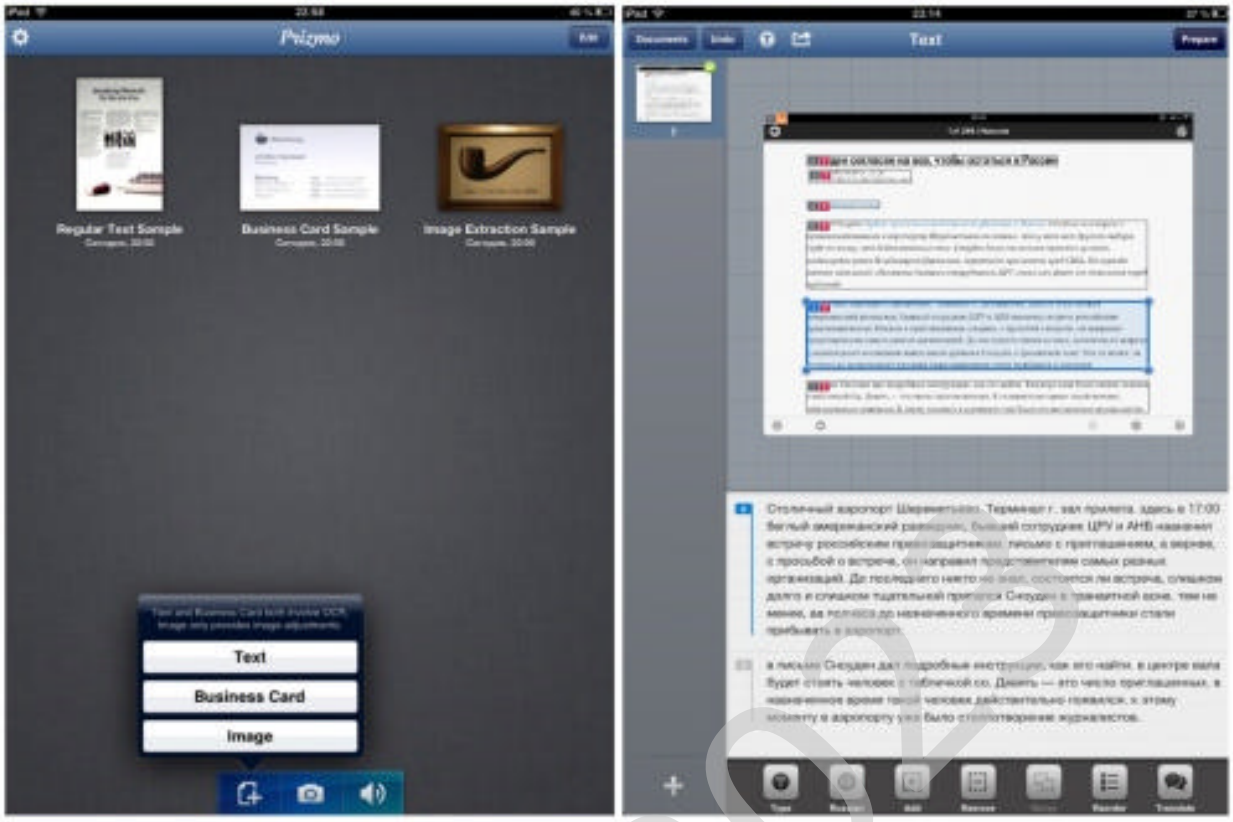

Рисунок 2.4 – Інтерфейс користувача Prizmo – Scanning, OCR, and Speech

Досить якісний, багатофункціональний додаток послужить гарну службу як у професійному плані, так і в аматорському.

Переваги: точність розпізнавання, підтримує велику кількість мов, для розпізнавання не потрібен Інтернет, убудований перекладач, відмінний функціонал.

Недоліки: низька швидкість розпізнавання, небагато заплутаний інтерфейс, для перекладача потрібен Інтернет.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою

Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з

підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи радіально-базисної нейронної мережі для розпізнання образів.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Методи розпізнавання образів і їхні характеристики

Розпізнаванням образів називаються завдання побудови й застосування формальних операцій над числовими або символьними відображеннями об'єктів реального або ідеального миру, результати рішення яких відбивають відносини еквівалентності між цими об'єктами. Відносини еквівалентності виражають приналежність оцінюваних об'єктів до яких-небудь класів, розглянутих як самостійні семантичні одиниці.

При побудові алгоритмів розпізнавання класи еквівалентності можуть задаватися дослідником, що користується власними змістовними поданнями або використовує зовнішню додаткову інформацію про подібність і розходження об'єктів у контексті розв'язуваного завдання. Тоді говорять про «розпізнавання із учителем» [1]. У противному випадку, тобто коли автоматизована система вирішує завдання класифікації без залучення зовнішньої навчальної інформації, говорять про автоматичну класифікацію або «розпізнаванні без учителя». Великість алгоритмів розпізнавання образів вимагає залучення досить значних обчислювальних потужностей, які можуть бути забезпечені тільки високопродуктивною комп'ютерною технікою.

Різні автори дають різну типологію методів розпізнавання образів. Одні автори розрізняють параметричні, непараметричні й евристичні методи, інші – виділяють групи методів, виходячи з історично сформованих шкіл і напрямків у даній області. Наприклад, у роботі [1], у якій даний академічний огляд методів розпізнавання, використовується наступна типологія методів розпізнавання образів:

− методи, засновані на принципі поділу;

− статистичні методи;

− методи, побудовані на основі «потенційних функцій»;

− методи обчислення оцінок (голосування);

− методи, засновані на вирахуванні висловлень, зокрема на апарату алгебри логіки.

В основі даної класифікації лежить розходження у формальних методах розпізнавання образів і тому опущено розгляд евристичного підходу до розпізнавання, що одержало повний і адекватний розвиток в експертних системах. Евристичний підхід заснований на важко формалізуємих знаннях і інтуїції дослідника. При цьому дослідник сам визначає, яку інформацію і який образ система повинна використовувати для досягнення необхідного ефекту розпізнавання.

Подібна типологія методів розпізнавання з тим або іншим ступенем деталізації зустрічається в багатьох роботах з розпізнавання. У той же час відомі типології не враховують одну дуже істотну характеристику, що відбиває специфіку способу подання знань про предметну область за допомогою якого-небудь формального алгоритму розпізнавання образів.

Д.А.Поспєлов виділяє два основних способи подання знань [2]:

− інтенсіональне, у вигляді схеми зв'язків між атрибутами (ознаками);

− екстенсіональне, за допомогою конкретних фактів (об'єкти, приклади).

Інтенсіональне подання фіксують закономірності й зв'язку, який пояснюється структура даних. Стосовно до діагностичних завдань така фіксація полягає у визначенні операцій над атрибутами (ознаками) об'єктів, що приводять до необхідного діагностичного результату. Інтенсіональні подання реалізуються за допомогою операцій над значеннями атрибутів і не припускають добутку операцій над конкретними інформаційними фактами (об'єктами).

У свою чергу, екстенсіональні подання знань пов'язані з описом і фіксацією конкретних об'єктів із предметної області й реалізуються в операціях, елементами яких служать об'єкти як цілісні системи.

Можна провести аналогію між інтенсіональними й екстенсіональними поданнями знань і механізмами, що лежать в основі діяльності лівої й правої півкуль головного мозку людини. Якщо для правої півкулі характерна цілісна прототипна репрезентація навколишнього світу, то ліва півкуля оперує закономірностями, що відбивають зв'язку атрибутів цього миру [2].

Описані вище два фундаментальних способи подання знань дозволяють запропонувати наступну класифікацію методів розпізнавання образів:

− інтенсіональні методи, засновані на операціях з ознаками;

− екстенсіональні методи, засновані на операціях з об'єктами.

Необхідно особливо підкреслити, що існування саме цих двох (і тільки двох) груп методів розпізнавання: оперуючих з ознаками, і оперуючих із об'єктами, глибоко закономірно. Із цього погляду жоден із цих методів, узятий окремо від іншого, не дозволяє сформувати адекватне відбиття предметної області. На думку авторів, між цими методами існує відношення додатковості в змісті Н.Бору [3], тому перспективні системи розпізнавання повинні забезпечувати реалізацію обох цих методів, а не тільки якого-небудь одного з них.

Таким чином, в основу класифікації методів розпізнавання, запропонованої Д. А. Поспєловим, покладені фундаментальні закономірності, що лежать в основі людського способу пізнання взагалі, що відносить її в зовсім особливе (привілейоване) положення в порівнянні з іншими класифікаціями, які на цьому тлі виглядають більше легковагими й штучними.

Інтенсіональні методи

Відмінною рисою інтенсіональних методів є те, що як елементи операцій при побудові й застосуванні алгоритмів розпізнавання образів вони використовують різні характеристики ознак і їхніх зв'язків. Такими елементами можуть бути окремі значення або інтервали значень ознак, середні величини й дисперсії, матриці зв'язків ознак і т.п., над якими виробляються дії, що виражаються в аналітичній або конструктивній формі. При цьому об'єкти в даних

методах не розглядаються як цілісні інформаційні одиниці, а виступають у ролі індикаторів для оцінки взаємодії й поводження своїх атрибутів.

Група інтенсіональних методів розпізнавання образів велика, і її розподіл на підкласи носить у певній мері умовний характер.

Методи, засновані на оцінках щільностей розподілу значень ознак

Ці методи розпізнавання образів запозичені із класичної теорії статистичних рішень, у якій об'єкти дослідження розглядаються як реалізації багатомірної випадкової величини, розподіленої в просторі ознак за яким-небудь законом. Вони базуються на байєсовській схемі прийняття рішень, що апелює до апріорних імовірностей приналежності об'єктів до того або іншого розпізнаваного класу й умовним щільностям розподілу значень вектору ознак. Дані методи зводяться до визначення відносини правдоподібності в різних областях багатомірного простору ознак.

Група методів, заснованих на оцінці щільностей розподілу значень ознак, має пряме відношення до методів дискримінантного аналізу. Байєсовський підхід до прийняття рішень і відноситься до найбільш розробленим у сучасній статистиці так званим параметричним методам, для яких уважається відомим аналітичне вираження закону розподілу (у цьому випадку нормальний закон) і потрібно оцінити лише невелика кількість параметрів (вектори середніх значень і коваріаційні матриці).

Основними труднощами застосування зазначених методів уважаються необхідність запам'ятовування всієї навчальної вибірки для обчислення оцінок локальних щільностей розподілу ймовірностей і висока чутливість до непоказності навчальної вибірки.

Методи, засновані на припущеннях про клас вирішальних функцій

У даній групі методів вважається відомим загальний вид вирішальної функції й заданий функціонал її якості. На підставі цього функціонала по навчальній послідовності знаходять найкраще наближення вирішальної функції [1]. Найпоширенішими є подання вирішальних функцій у вигляді лінійних і

узагальнених нелінійних поліномів. Функціонал якості вирішального правила звичайно зв'язують із помилкою класифікації.

Основним достоїнством методів, заснованих на припущеннях про клас вирішальних функцій, є ясність математичної постановки завдання розпізнавання, як завдання пошуку екстремуму. Різноманіття методів цієї групи пояснюється широким спектром використовуваних функціоналів якості вирішального правила й алгоритмів пошуку екстремуму. Узагальненням розглянутих алгоритмів, до яких ставляться, зокрема, алгоритм Ньютона, алгоритми перцептроного типу й ін., є метод стохастичної апроксимації.

Можливості градієнтних алгоритмів пошуку екстремуму, особливо в групі лінійних вирішальних правил, досить добре вивчені. Збіжність цих алгоритмів доведена тільки для випадку, коли розпізнавані класи об'єктів відображаються в просторі ознак компактними геометричними структурами.

Досить висока якість вирішального правила може бути досягнуте за допомогою алгоритмів, що не мають строгого математичного доказу збіжності рішення до глобального екстремуму. До таких алгоритмів відноситься більша група процедур евристичного програмування, що представляють напрямок еволюційного моделювання. Еволюційне моделювання є біонічним методом, запозиченим у природи. Воно засновано на використанні відомих механізмів еволюції з метою заміни процесу змістовного моделювання складного об'єкта феноменологічним моделюванням його еволюції. Відомим представником еволюційного моделювання в розпізнаванні образів є метод групового обліку аргументів (МГОА) [1]. В основу МГОА покладений принцип самоорганізації, і алгоритми МГОА відтворюють схему масової селекції.

Однак досягненню практичних цілей у цьому випадку не супроводжує добування нових знань про природу розпізнаваних об'єктів. Можливість добування цих знань, зокрема знань про механізми взаємодії атрибутів (ознак), тут принципово обмежена заданою структурою такої взаємодії, зафіксованої в обраній формі вирішальних функцій.

Логічні методи

Логічні методи розпізнавання образів базуються на апарату алгебри логіки й дозволяють оперувати інформацією, укладеної не тільки в окремих ознаках, але й у сполученнях значень ознак. У цих методах значення якої-небудь ознаки розглядаються як елементарні події [4].

У самому загальному виді логічні методи можна охарактеризувати як різновид пошуку по навчальній вибірці логічних закономірностей і формування деякої системи логічних вирішальних правил (наприклад, у вигляді кон'юнкцій елементарних подій), кожне з яких має власну вагу. Група логічних методів різноманітна й включає методи різної складності й глибини аналізу. Для дихотомічних (булевих) ознак популярними є так звані деревоподібні класифікатори, метод тупикових тестів, алгоритм «Кора» і ін.

Алгоритм «Кора», як і інші логічні методи розпізнавання образів, є досить трудомістким в обчислювальному відношенні, оскільки при відборі кон'юнкцій необхідний повний перебір. Тому при застосуванні логічних методів пред'являються високі вимоги до ефективної організації обчислювального процесу, і ці методи добре працюють при порівняно невеликих розмірностях простору ознак і тільки на потужних комп'ютерах.

Лінгвістичні (структурні) методи

Лінгвістичні методи розпізнавання образів засновані на використанні спеціальних граматик, що породжують мови, за допомогою яких може описуватися сукупність властивостей розпізнаваних об'єктів [4].

Для різних класів об'єктів виділяються непохідні (атомарні) елементи (підобрази, ознаки) і можливі відносини між ними. Граматикою називають правила побудови об'єктів із цих непохідних елементів.

Таким чином, кожний об'єкт являє собою сукупність непохідних елементів, «з'єднаних» між собою тими або іншими способами або, інакше кажучи, «пропозицією» деякого «мови». Хотілося б особливо підкреслити дуже значну світоглядну цінність цієї думки [4].

Шляхом синтаксичного аналізу (граматичного розбору) «пропозиції» визначається його синтаксична «правильність» або, що еквівалентно, чи може деяка фіксована граматика, що описує клас, породити наявний опис об'єкта.

Однак завдання відновлення (визначення) граматик по деякій множині висловлень (пропозицій – описів об'єктів), що породжують дану мову, є важко формалізуємою.

Екстенсіональні методи

У методах даної групи, на відміну від інтенсіонального напрямку, кожному досліджуваному об'єкту в великій або меншій мері надається самостійне діагностичне значення. По своїй суті ці методи близькі до клінічного підходу, що розглядає людей не як проранжировану по тому або іншому показнику ланцюжок об'єктів, а як цілісні системи, кожна з яких індивідуальна й має особливу діагностичну цінність [1]. Таке дбайливе відношення до об'єктів дослідження не дозволяє виключати або втрачати інформацію про кожний окремий об'єкт, що відбувається при застосуванні методів інтенсіонального напрямки, що використовує об'єкти тільки для виявлення й фіксації закономірностей поводження їхніх атрибутів.

Основними операціями в розпізнаванні образів за допомогою обговорюваних методів є операції визначення подібності й розходження об'єктів. Об'єкти в зазначеній групі методів відіграють роль діагностичних прецедентів. При цьому залежно від умов конкретного завдання роль окремого прецеденту може мінятися в самих широких межах: від головної й визначальної й до досить непрямої участі в процесі розпізнавання. У свою чергу умови завдання можуть вимагати для успішного рішення участі різної кількості діагностичних прецедентів: від одного в кожному розпізнаваному класі до повного обсягу вибірки, а також різних способів обчислення мер подібності й розходження об'єктів. Цими вимогами пояснюється подальший поділ екстенсіональних методів на підкласи.

Метод порівняння із прототипом

Це найбільш простий екстенсіональний метод розпізнавання. Він застосовується, наприклад, у тому випадку, коли розпізнавані класи відображаються в просторі ознак компактними геометричними угрупованнями. У такому випадку звичайно як крапка – прототипу вибирається центр геометричного угруповання класу (або найближчий до центра об'єкт).

Для класифікації невідомого об'єкта перебуває найближчий до нього прототип, і об'єкт відноситься до того ж класу, що й цей прототип. Очевидно, ніяких узагальнених образів класів у даному методі не формується.

Як міра близькості можуть застосовуватися різні типи відстаней. Часто для дихотомічних ознак використовується відстань Хеммінгу, що у цьому випадку дорівнює квадрату евклідової відстані. При цьому вирішальне правило класифікації об'єктів еквівалентно лінійної вирішальної функції.

Зазначений факт слід особливо зазначити. Він наочно демонструє зв'язок прототипної і признакової репрезентації інформації про структуру даних. Користуючись наведеним поданням, можна, наприклад, будь-яку традиційну вимірювальну шкалу, що є лінійною функцією від значень дихотомічних ознак, розглядати як гіпотетичний діагностичний прототип. У свою чергу, якщо аналіз просторової структури розпізнаваних класів дозволяє зробити вивід про їхню геометричну компактність, то кожний із цих класів досить замінити одним прототипом, що фактично еквівалентний лінійної діагностичної моделі.

На практиці, безумовно, ситуація часто буває відмінною від описаного ідеалізованого приклада. Перед дослідником, що наміряється застосувати метод розпізнавання, заснований на порівнянні із прототипами діагностичних класів, встають непрості проблеми.

По-перше, це вибір міри близькості (метрики), від якого може істотно змінитися просторова конфігурація розподілу об'єктів. По-друге, самостійною проблемою є аналіз багатомірних структур експериментальних даних. Обидві ці проблеми особливо гостро встають перед дослідником в умовах високої розмірності простору ознак, характерної для реальних завдань.

Метод k найближчих сусідів

Метод k найближчих сусідів для рішення завдань дискримінантного аналізу був уперше запропонований ще в 1952 році [2]. Він полягає в наступному.

При класифікації невідомого об'єкта перебуває задане число (k) геометрично найближчих до нього в просторі ознак інших об'єктів (найближчих сусідів) із уже відомою приналежністю до розпізнаваних класів. Рішення про віднесення невідомого об'єкта до того або іншого діагностичного класу приймається шляхом аналізу інформації про цю відому приналежність його найближчих сусідів, наприклад, за допомогою простого підрахунку голосів.

Спочатку метод k найближчих сусідів розглядався як непараметричний метод оцінювання відносини правдоподібності. Для цього методу отримані теоретичні оцінки його ефективності в порівнянні з оптимальним байєсовським класифікатором. Доведено, що асимптотичні ймовірності помилки для методу k найближчих сусідів перевищують помилки правила Байєса не більш ніж у два рази.

При використанні методу k найближчих сусідів для розпізнавання образів дослідникові доводиться вирішувати складну проблему вибору метрики для визначення близькості діагностуємих об'єктів. Ця проблема в умовах високої розмірності простору ознак надзвичайно загострюється внаслідок достатньої трудомісткості даного методу, що стає значимої навіть для високопродуктивних комп'ютерів. Тому тут так само, як і в методі порівняння із прототипом, необхідно вирішувати творче завдання аналізу багатомірної структури експериментальних даних для мінімізації числа об'єктів, що представляють діагностичні класи.

Необхідність зменшення числа об'єктів у навчальній вибірці (діагностичних прецедентів) є недоліком даного методу, так як зменшує показність навчальної вибірки.

Алгоритми обчислення оцінок («голосування»)

Принцип дії алгоритмів обчислення оцінок (АОО) складається в обчисленні пріоритетів (оцінок подібності), що характеризують «близькість» розпізнаваного й еталонного об'єктів по системі ансамблів ознак, що представляє собою систему підмножин заданої множині ознак.

На відміну від усіх раніше розглянутих методів алгоритми обчислення оцінок принципово по-новому оперують описами об'єктів. Для цих алгоритмів об'єкти існують одночасно в самих різних підпросторах простору ознак. Клас АОО доводить ідею використання ознак до логічного кінця: оскільки не завжди відомо, які сполучення ознак найбільш інформативні, то в АОО ступінь подібності об'єктів обчислюється при зіставленні всіх можливих або певних сполучень ознак, що входять в описи об'єктів [1].

Використовувані сполучення ознак (підпростору) автори називають опорними множинами або множинами часткових описів об'єктів. Уводиться поняття узагальненої близькості між розпізнаваним об'єктом і об'єктами навчальної вибірки (з відомою класифікацією), які називають еталонними об'єктами. Ця близькість представляється комбінацією близькостей розпізнаваного об'єкта з еталонними об'єктами, обчислених на множинах часткових описів. Таким чином, АОО є розширенням методу k найближчих сусідів, у якому близькість об'єктів розглядається тільки в одному заданому просторі ознак.

Ще одним розширенням АОО є те, що в даних алгоритмах завдання визначення подібності й розходження об'єктів формулюється як параметрична й виділений етап налаштування АОО по навчальній вибірці, на якому підбираються оптимальні значення уведених параметрів. Критерієм якості служить помилка розпізнавання, а параметризується буквально все:

- правила обчислення близькості об'єктів по окремих ознаках;
- правила обчислення близькості об'єктів у підпросторах ознак;

− ступінь важливості того або іншого еталоного об'єкта як діагностичного прецеденту;

− значимість внеску кожної опорної множині ознак у підсумкову оцінку подібності розпізнаваного об'єкта з яким-небудь діагностичним класом.

Параметри АОО задаються у вигляді значень порогів і (або) як ваги зазначених складових.

Теоретичні можливості АОО принаймні не нижче можливостей будь-якого іншого алгоритму розпізнавання образів, так як за допомогою АОО можуть бути реалізовані всі мислимі операції з досліджуваними об'єктами.

Але, як це звичайно буває, розширення потенційних можливостей натрапляє на великі труднощі при їхньому практичному втіленні, особливо на етапі побудови (налаштування) алгоритмів даного типу.

Окремі труднощі відзначалися раніше під час обговорення методу k найближчих сусідів, які можна було інтерпретувати як усічений варіант АОО. Його теж можна розглядати в параметричному виді й звести завдання до пошуку зваженої метрики обраного типу. У той же час уже тут для високорозмірних завдань виникають складні теоретичні питання й проблеми, пов'язані з організацією ефективного обчислювального процесу.

Для АОО, якщо спробувати використовувати можливості даних алгоритмів у повному обсязі, зазначені труднощі зростають багаторазово.

Відзначені проблеми пояснюють те, що на практиці застосування АОО для рішення високорозмірних завдань супроводжується введенням яких-небудь евристичних обмежень і допущень. Зокрема, відомий приклад використання АОО в психодіагностиці, у якому апробована різновид АОО, фактично еквівалентна методу k найближчих сусідів.

Колективи вирішальних правил

У завершення огляду методів розпізнавання образів зупинимося ще на одному підході. Це так звані колективи вирішальних правил (КВП) [2].

Так як різні алгоритми розпізнавання проявляють себе по-різному на одній і тій же вибірці об'єктів, то закономірно встає питання про синтетичне вирішальне правило, адаптивно використовуючий сильні сторони цих алгоритмів. У синтетичному вирішальному правилі застосовується дворівнева схема розпізнавання. На першому рівні працюють приватні алгоритми розпізнавання, результати яких поєднуються на другому рівні в блоці синтезу. Найпоширеніші способи такого об'єднання засновані на виділенні областей компетентності того або іншого приватного алгоритму. Найпростіший спосіб знаходження областей компетентності полягає в апріорній розбивці простору ознак виходячи із професійних міркувань конкретної науки (наприклад розшарування вибірки по деякій ознаці). Тоді для кожної з виділених областей будується власний алгоритм, що розпізнає. Інший спосіб базується на застосуванні формального аналізу для визначення локальних областей простору ознак як околиць розпізнаваних об'єктів, для яких доведена успішність роботи якого-небудь приватного алгоритму розпізнавання.

Самий загальний підхід до побудови блоку синтезу розглядає результуючі показники приватних алгоритмів як вихідні ознаки для побудови нового узагальненого вирішального правила. У цьому випадку можуть використовуватися всі перераховані вище методи інтенсіонального й екстенсіонального напрямків у розпізнаванні образів. Ефективними для рішення завдання створення колективу вирішальних правил є логічні алгоритми типу «Кора» і алгоритми обчислення оцінок (АОО), покладені в основу так званого алгебраїчного підходу, що забезпечує дослідження й конструктивний опис алгоритмів розпізнавання, у рамки якого укладаються всі існуючі типи алгоритмів [1].

Порівняльний аналіз методів розпізнавання образів

Зрівняємо описані вище методи розпізнавання образів і оцінимо ступінь їхньої адекватності. Для рішення реальних завдань із групи методів інтенсіонального напрямку практичну цінність представляють параметричні

методи й методи, засновані на пропозиціях про вид вирішальних функцій. Параметричні методи становлять основу традиційної методології конструювання показників. Застосування цих методів у реальних завданнях пов'язане з накладенням сильних обмежень на структуру даних, які приводять до лінійних діагностичних моделей з дуже приблизними оцінками їхніх параметрів. При використанні методів, заснованих на припущеннях про вид вирішальних функцій, дослідник також змушений звертатися до лінійних моделей. Це обумовлено високою розмірністю простору ознак, характерної для реальних завдань, що при підвищенні ступеня поліноміальної вирішальної функції дає величезний ріст числа її членів при проблематичному супутнім підвищенні якості розпізнавання. Таким чином, зпроєціювавши область потенційного застосування інтенсіональних методів розпізнавання на реальну проблематику, одержимо картину, що відповідає добре відпрацьованої традиційної методології лінійних діагностичних моделей.

Властивості лінійних діагностичних моделей, у яких діагностичний показник представлений зваженою сумою вихідних ознак, добре вивчені. Результати цих моделей (при відповідному нормуванні) інтерпретуються як відстані від досліджуваних об'єктів до деякої гіперплощини в просторі ознак або, що еквівалентно, як проекції об'єктів на деяку пряму лінію в даному просторі. Тому лінійні моделі адекватні тільки простим геометричним конфігураціям областей простору ознак, у які відображаються об'єкти різних діагностичних класів. При більше складних розподілах ці моделі принципово не можуть відбивати багато особливостей структури експериментальних даних. У той же час такі особливості здатні нести коштовну діагностичну інформацію.

Разом з тим поява в якому-небудь реальному завданні простих багатомірних структур (зокрема, багатомірних нормальних розподілів) варто скоріше розцінювати як виключення, чим як правило. Часто діагностичні класи формуються на основі складносоставних зовнішніх критеріїв, що автоматично спричиняє геометричну неоднорідність даних класів у просторі ознак. Це

особливо стосується «життєвих», що найбільше часто зустрічаються на практиці критеріїв. У таких умовах застосування лінійних моделей фіксує тільки самі «грубі» закономірності експериментальної інформації.

Застосування екстенсіональних методів не пов'язане з яким-небудь припущеннями про структуру експериментальної інформації, крім того, що усередині розпізнаваних класів повинні існувати одна або кілька груп чимсь схожих об'єктів, а об'єкти різних класів повинні чимсь відрізнятися друг від друга. Очевидно, що при будь-якій кінцевій розмірності навчальної вибірки (а іншої вона бути й не може) ця вимога виконується завжди просто з тієї причини, що існують випадкові розходження між об'єктами. У якості міри подібності застосовуються різні міри близькості (відстані) об'єктів у просторі ознак. Тому ефективне використання екстенсіональних методів розпізнавання образів залежить від того, наскільки вдало визначені зазначені міри близькості, а також від того, які об'єкти навчальної вибірки (об'єкти з відомою класифікацією) виконують роль діагностичних прецедентів. Успішне рішення даних завдань дає результат, що наближається до теоретично досяжних меж ефективності розпізнавання.

Перевагам екстенсіональних методів розпізнавання образів протипоставлена, у першу чергу, висока технічна складність їхнього практичного втілення. Для високорозмірних просторів ознак зовні просте завдання знаходження пара найближчих крапок перетворюється в серйозну проблему. Також багато авторів відзначають як проблема необхідність запам'ятовування досить великої кількості об'єктів, що представляють розпізнавані класи.

Саме по собі це не є проблемою, однак сприймається як проблема (наприклад, у методі k найближчих сусідів) з тієї причини, що при розпізнаванні кожного об'єкта відбувається повний перебір всіх об'єктів навчальної вибірки.

Тому доцільно застосувати модель системи розпізнавання, у якій проблема повного перебору об'єктів навчальної вибірки при розпізнаванні знімається, так як він здійснюється лише один раз при формуванні узагальнених

образів класів розпізнавання. При самому ж розпізнаванні здійснюється порівняння ідентифікуємого об'єкту лише з узагальненими образами класів розпізнавання, кількість яких фіксоване й зовсім не залежить від розмірності навчальної вибірки. Даний підхід дозволяє збільшувати розмірність навчальної вибірки доти, поки не буде досягнута необхідна висока якість узагальнених образів, зовсім при цьому не побоюючись, що це може привести до неприйнятного збільшення часу розпізнавання (так як час розпізнавання в даній моделі взагалі не залежить від розмірності навчальної вибірки).

Теоретичні проблеми застосування екстенсіональних методів розпізнавання пов'язані із проблемами пошуку інформативних груп ознак, знаходження оптимальних метрик для виміру подібності й розходження об'єктів і аналізу структури експериментальної інформації. У той же час успішне рішення перерахованих проблем дозволяє не тільки конструювати ефективні алгоритми, що розпізнають, але й здійснювати перехід від екстенсіонального знання емпіричних фактів до інтенсіональному знанню про закономірності їхньої структури.

Перехід від екстенсіонального знання до інтенсіонального відбувається на тій стадії, коли формальний алгоритм розпізнавання вже сконструйована і його ефективність продемонстрована. Тоді виробляється вивчення механізмів, за рахунок яких досягається отримана ефективність. Таке вивчення, пов'язане з аналізом геометричної структури даних, може, наприклад, привести до виводу про те, що досить замінити об'єкти, що представляють той або інший діагностичний клас, одним типовим представником (прототипом). Це еквівалентно, як відзначалося вище, завданню традиційної лінійної діагностичної шкали. Також можливо, що кожний діагностичний клас досить замінити декількома об'єктами, осмисленими як типові представники деяких підкласів, що еквівалентно побудові віяла лінійних шкал. Можливі й інші варіанти, які будуть розглянуті нижче.

Таким чином, огляд методів розпізнавання показує, що в цей час теоретично розроблений цілий ряд різних методів розпізнавання образів. У літературі приводиться розгорнута їхня класифікація. Однак для більшості цих методів їхня програмна реалізація відсутня, і це глибоко закономірно, можна навіть сказати «визначено» характеристиками самих методів розпізнавання. Про це можна судити по тому, що такі системи мало згадуються в спеціальній літературі й інших джерелах інформації.

Отже, залишається недостатньо розробленим питання про практичну застосовність тих або інших теоретичних методів розпізнавання для рішення практичних завдань при реальних (тобто досить значних) розмірностях даних і на реальних сучасних комп'ютерах.

Вищезгадана обставина може бути зрозуміло, якщо нагадати, що складність математичної моделі експоненційно збільшує трудомісткість програмної реалізації системи й у такому ж ступені зменшує шанси на те, що ця система буде практично працювати. Це означає, що реально на ринку можна реалізувати тільки такі програмні системи, в основі яких лежать досить прості й «прозорі» математичні моделі. Тому розроблювач, зацікавлений у тиражуванні свого програмного продукту, підходить до питанню про вибір математичної моделі не із чисто наукової точки зору, а як прагматик, з урахуванням можливостей програмної реалізації. Він уважає, що модель повинна бути як можна більше простій, а значить реалізуватися з меншими витратами й більш якісно, а також повинна обов'язково працювати (бути практично ефективної).

У цьому зв'язку особливо актуальної представляється завдання реалізації в системах розпізнавання механізму узагальнення описів об'єктів, що ставляться до одного класу, тобто механізму формування компактних узагальнених образів. Очевидно, що такий механізм узагальнення дозволить «стиснути» кожну по розмірності навчальну вибірку до заздалегідь відомого по розмірності базі узагальнених образів. Це дозволить також повідносити й вирішити ряд завдань, які навіть не можуть бути сформульовані в таких методах розпізнавання, як метод порівняння із прототипом, метод k найближчих сусідів і АОО.

Це завдання:

− визначення інформаційного внеску ознак в інформаційний портрет узагальненого образа;

− кластерно-конструктивний аналіз узагальнених образів;

− визначення семантичного навантаження ознаки;

− семантичний кластерно-конструктивний аналіз ознак;

− змістовне порівняння узагальнених образів класів один з одним і ознак один з одним (когнітивні діаграми, у т.ч. діаграми Мерліна [2]).

Метод, що дозволив досягти рішення цих завдань, також відрізняє засновану на ньому перспективну систему від інших систем, як компілятори відрізняються від інтерпретаторів, так як завдяки формуванню узагальнених образів у цій перспективній системі досягається незалежність часу розпізнавання від обсягів навчальної вибірки. Відомо, що саме існування цієї залежності приводить до практично неприйнятних витрат машинного часу на розпізнавання в таких методах, як метод k найближчих сусідів, АОО й КВП при таких розмірностях навчальної вибірки, коли можна говорити про достатню статистику.

3.2 Розробка структурної схеми

Для реалізації завдання розпізнання образів використовується радіально-базисна нейронна мережа.

Штучні нейронні мережі, що використовують у якості активаційних функцій радіально-базисні (такі мережі скорочено називаються RBF-мережами). Загальний вид радіально-базисної функції:

$$f(x) = \varphi\left(\frac{x^2}{\sigma^2}\right), \text{ наприклад, } f(x) = e^{-\frac{x^2}{\sigma^2}}, \qquad (3.1)$$

де x – вектор вхідних сигналів нейронна,

σ – ширина вікна функції,

$\varphi(y)$ – убутна функція (найчастіше, рівна нулю поза деяким відрізком).

Радіально-базисна мережа характеризується трьома особливостями:

1. Єдиний схований шар

2. Тільки нейрони схованого шару мають нелінійну активаційну функцію

3. Синаптичні ваги зв'язків вхідного й схованого шарів дорівнюють одиниці

Виходом мережі є лінійна комбінація радіальних базисних функцій входів і параметрів нейрона. Мережі радіальних базисних функцій мають безліч застосувань, у тому числі функції наближення, прогнозування тимчасових рядів, класифікації й системи керування.

Описані вище багатошарові мережі сигмоїдального типу з математичної точки зору виконують апроксимацію функції декількох змінних X=RN у безліч вихідних змінних Y=RM. Оскільки сигмоїдальна функція, що грає роль функції активації нейронів, має ненульове значення на всьому діапазоні вхідних даних, то в перетворенні мережею вхідних даних у вихідні беруть участь багато хто (якщо не все) її нейрони. Внаслідок цього апроксимація сигмоїдальними (і, природно, лінійними) нейронами називається глобальною апроксимацією.

Радіальні мережі будуються з використанням радіальних нейронів, функція активації яких має ненульові значення тільки на околицях свого центра . Тому апроксимація за допомогою таких мереж називається локальною апроксимацією.

Радіальна мережа має двошарову структуру, перший шар становлять радіальні нейрони, вихідний – один або декілька лінійних.

Навчання радіальної мережі

Процес навчання радіальної мережі розпадається на два етапи:

– підбор параметрів радіальної функції fi для кожного радіального нейрона (у випадку функції Гауса це центр Ci і параметр ширини si);

– підбор ваг вихідного шару нейронів.

При цьому другий етап значно простіше першого, оскільки зводиться до обчислення вираження $W=G^{+}*D$, де основні обчислювальні витрати – розрахунок псевдоінверсії матриці Гріна G.

Завдання відшукання параметрів радіальних функцій Гаусса для всіх нейронів першого шару у свою чергу розпадаються на дві підзадачі:

− визначення центрів Ci;

− розрахунок параметрів ширини si.

Зрозуміло, що основними вимогами до розташування центрів Ci в області визначення вхідних даних X є:

− повнота покриття області визначення;

− рівномірність розподілу.

Саме цим вимогам відповідає рішення по кластеризації даних, що дається нейронною мережею із самоорганізацією на основі конкуренції. Отже, алгоритми навчання, використовувані в цих мережах для відшукання усереднених векторів у кластерах даних, безпосередньо застосовні й у радіальних мережах для відшукання центрів радіальних функцій.

Після визначення місця розташування всіх центрів радіальних функцій Ci відбувається підбор параметрів si, що визначають величину області охвату, у якій значення радіальної функції перевищує граничне значення e. Такий підбор повинен забезпечити, з одного боку, покриття всього простору вхідних даних і, з іншого боку, незначність перекриття сусідніх зон.

На рисунку 3.1 представлена структурна схема системи управління радіально-базисної нейронної мережі для розпізнання образів, під якою будемо розуміти середовище, у яку вкладений ОУ, у свою чергу утримуючий у собі УС. Як видно з рисунка, можна затверджувати, що УС управляє не тільки ОУ, але всією системою. Під середовищем у системі можна розуміти різні об'єднання об'єктів. Будемо називати середовищем W сукупність об'єктів, що лежать поза УС; середовищем S – сукупність об'єктів, що лежать поза ОУ; середовищем U – всю систему.

Блок формування бази знань [4-6] (БЗ) призначений для автоматичного подання емпірично знайдених УС знань про функціональні властивості системи. Елементарною конструкцією бази знань (БЗ) у методі ААУ є статистично

достовірна відомість про те, як певна дія Y_j впливає на прообраз певного сформованого образа. Дією Y_j названа підмножина множини припустимих впливів, елементи якого абсолютно ідентичні для УС по їхньому впливі на сформовані образи. Непуста відомість може мати одне із двох значень: або дія Y_j тягне розпізнавання образа O_i , або дія Y_j тягне витиснення образа O_i. За допомогою БЗ можна бачити, як конкретна дія впливає на всю сукупність сформованих образів.

Блок оцінки стану [7] (БОС) виробляє інтегральну оцінку якості стану ОУ S_t. Оцінка S_t використовується для розрахунку оцінки (ваги) p_i кожного зі знову сформованих образів деяким статистичним способом. У свою чергу, S_t функціонально залежить від оцінок p_i розпізнаних образів. Є деяка множина споконвічна сформованих і оцінених образів. Оцінка S_t використовується також для розрахунку темпу прийняття рішень.

Блок датчиків поставляє УС вхідну інформацію у вигляді двійкового вектора. Цей блок необхідний у реальних системах для сполучення середовища й УС, тому при моделюванні УС на ЕОМ не використовувався й ми не акцентуємо увагу на ньому в даній роботі.

Роботу блоку формування й розпізнавання образів (ФРО) можна представити в такий спосіб (докладний опис див. у роботах [3, 8]). У блоці ФРО на підставі апріорної інформації про можливі функціональні властивості середовища задані деякі об'єкти, назвемо їхніми нейронами (наприклад, нейрони спеціального виду, описані в роботі [8]), на які відображаються деякі класи часових-просторово-часових явищ, які потенційно можуть існувати в системі. Відображення задається топологією мережі. У класі, відображуваному на нейрон, виділяється підклас, що може сприйматися даним нейроном. Кожний нейрон може статистично аналізувати сприйманий їм підклас. Накопичуючи статистичну інформацію про сприйманий підклас, нейрон може прийняти рішення, чи є цей підклас випадковим або невипадковим явищем у системі. Якщо який-небудь нейрон приймає рішення, що відображуваний на нього підклас є невипадковою

подією, то він переходить у деяке відмінне від вихідного "навчене" стан. Якщо нейрон навчений, то будемо говорити також, що сформовано образ, цей образ ідентифікується номером даного нейрона. Підклас явищ, сприйманий нейроном, і викликавший його навчання, тобто просторово-часові явища, що статистично вірогідно існують у системі, називається прообразом даного образа. Сформований образ може бути розпізнаний блоком ФРО, коли прообраз даного образа спостерігається БД. Блок ФРО вказує, які зі сформованих образів розпізнані в сучасний момент. Одночасно із цим розпізнані образи беруть участь у формуванні образів більше високих порядків, тобто має місце агрегування й абстрагування образів.

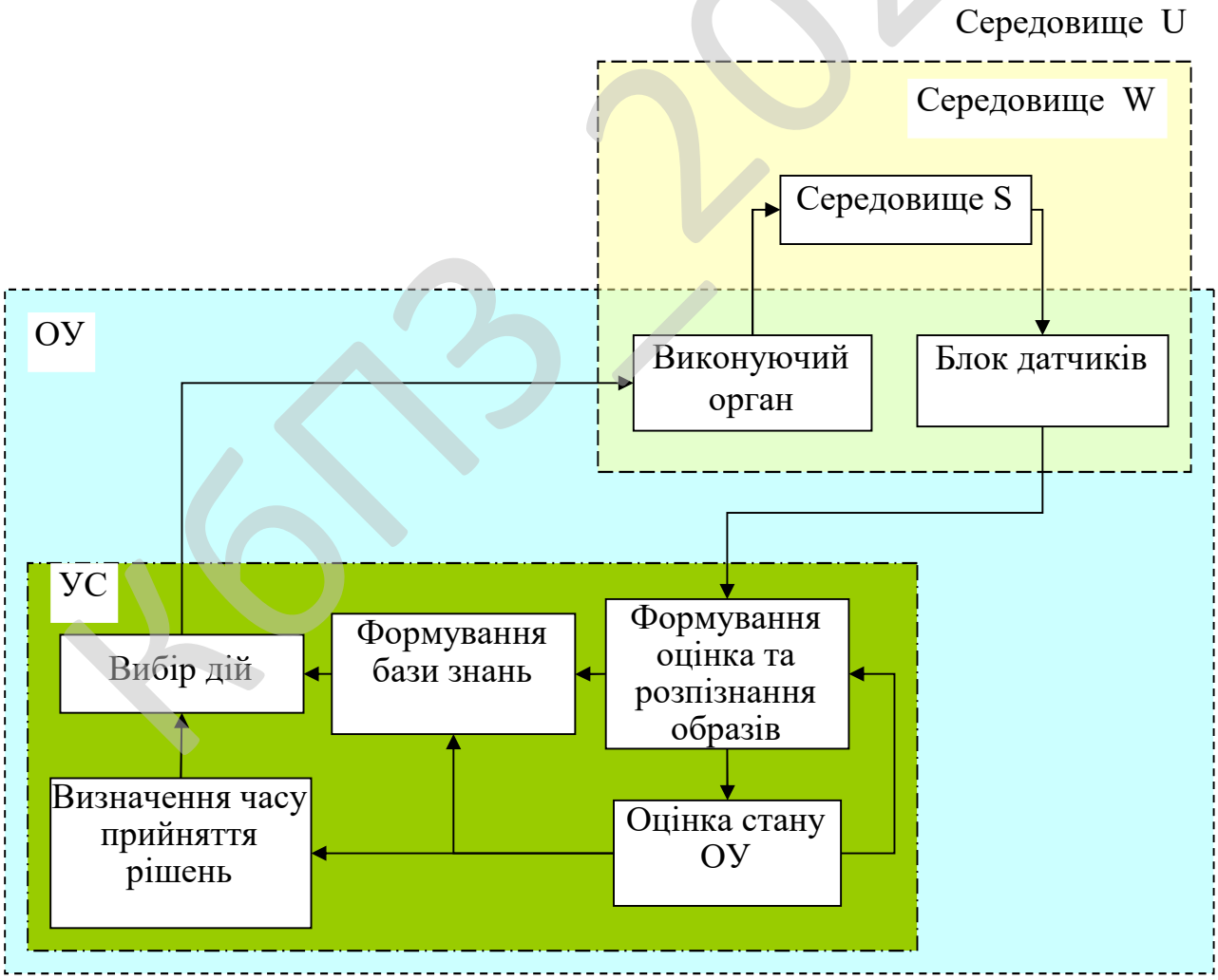

Рисунок 3.1 – Структурна схема системи

Блок вибір дії [4-6] або, надалі, блок прийняття рішень (БПР) реалізує процедуру ухвалення рішення, засновану на аналізі поточної ситуації, цільових функцій, умісту БЗ, а також оцінки поточного значення оцінки S_t. Фактична інформація про поточну ситуацію представлена множиною образів, розпізнаних у сучасний момент блоком ФРО, а інформація про якість поточного стану представлена оцінкою S_t. Множина розпізнаних образів визначає в БЗ той її розділ, що адекватний поточної ситуації (ті знання, які дійсні в поточних умовах). Відповідно до цільової функції, що припускає прагнення УС до поліпшення якості стану ОУ, УС вибирає по БЗ ту дію, що має максимальну суму оцінок викликуваних і образів, що витісняються. Із множини вихідних впливів, що відповідає обраній дії Y_j, конкретне вихідний вплив вибирається випадковим способом, що відповідає другій цільовій функції, що передбачає прагнення до одержання нових знань.

Блок визначення часу ухвалення рішення визначає глибину перегляду БЗ залежно від поточної оцінки S_t. Чим вище значення S_t, тим більше образів (у порядку убування модуля їхньої ваги) може врахувати УС при ухваленні рішення, тим менше темп прийняття рішень. При моделюванні цей блок не використовувався й у даній роботі розглядатися не буде.

У УС можуть бути засоби для апріорного аналізу наслідків альтернативних обираних дій на кілька кроків уперед.

Такий загалом алгоритм управління, реалізований УС. Основні властивості процесу управління полягають у тому, що УС автоматично накопичує емпіричні знання про властивості пред'явленого їй об'єкта управління й приймає рішення, опираючись на накопичені знання. Якість управління росте в міру збільшення об'єму накопичених знань. Помітимо також, що управління складається не в тому, що УС реагує на вхідну інформацію (у певному змісті – негативний зворотний зв'язок), а в тому, що УС постійно активно шукає можливий у поточних умовах спосіб поліпшити стан ОУ (позитивний зворотний зв'язок). Тим самим УС має внутрішню активність.

3.3 Розробка функціональної схеми

На наведеній функціональній схемі (рисунок 3.2) зазначені основні класи об'єктів ядра системи і їхня взаємодія. Стрілками показані потоки даних при роботі системи. Кожному з основних блоків УС відповідає свій блок у системі. Чотири блоки: ФРО, БЗ, БОС і БПР становлять УС. Нагадаємо, що в підрозділі 3.1 ми визначили такі поняття як блок, вихідна функція блоку, шаблон, нейронна мережа й формальна модель нейрона. З формальної моделі НМ слідує, що блок – це ієрархічна структура, у якій елементи одного рівня з'єднані в мережу й кожний з елементів рівня може бути мережею, що складається з елементів більш низького рівня.

Розглядаючи обраний елемент якого-небудь рівня, можна вважати його «чорним ящиком», тобто абстрагуватися від його вмісту й внутрішнього пристрою. Наприклад, можна на деякому проміжному етапі конструювання УС абстрагуватися від нейромережної реалізації якого-небудь блоку верхнього рівня й спробувати різні реалізації, причому необов'язково нейромережні. Система не накладає обмежень на внутрішній пристрій кожного блоку, тому вона може не мати внутрішньої ієрархії, а просто представлятися деякою функцією виходу. Далі, у процесі розвитку УС, вміст окремих блоків може помінятися, можливо стати більш складним і ієрархічним, при цьому поводження системи не зміниться, якщо новий вміст забезпечує функціональність старого в змісті еквівалентності вихідних функцій.

Таким чином, полегшується розробка системи, так як з'являється можливість конструювання «зверху долілиць», немає необхідності реалізовувати блок відразу через НМ, можна поставити часову «заглушку», а в процесі розвитку системи ускладнювати, доповнювати або заміняти на зовсім іншу внутрішню конструкцію блоків.

Крім зазначених блоків, у систему входять ще два важливих класи об'єктів: конструктори мережі й аналізатори роботи мережі. Перші, як видно з

назви, призначені для створення робочих копій НМ у пам'яті комп'ютера по різних джерелах, наприклад по специфікації мережі з файлу. Властиво, для кожного джерела й створюється свій об'єкт. (Варто відрізняти дані об'єкти від конструкторів мереж, призначених для створення за допомогою ГІП файли специфікації мереж; ці конструктори в ядро СРНМ не входять). Специфікація мережі може посилатися на шаблони блоків з бібліотеки, які, таким чином, також можуть бути джерелом для конструкції. Аналізатори потрібні при налагодженні мереж. Справа в тому, що мережі можуть містити тисячі й десятки тисяч елементів (принципових обмежень немає, мають місце обмеження тільки по пам'яті й продуктивності комп'ютера), роботу яких одночасно простежити просто неможливо, особливо якщо часовий інтервал роботи становить сотні й більше тактів. Тому необхідно якось узагальнювати інформацію про стан мережі (яке є сукупність станів кожного елемента) у кожний момент часу й видавати користувачеві сумарну інформацію, можливо, з деякою деталізацією по розсуду користувача. Для такої задачі й потрібні спеціальних об'єктів – аналізатори. Ці об'єкти можуть зберігати історію станів обраних елементів в обрані інтервали часу й згодом нею аналізувати, тобто визначати статистичного роду інформацію. Кожний об'єкт вирішує цю задачу по-своєму й може бути обраний залежно від роду необхідної інформації про роботу мережі.

Відзначимо тут на наш погляд дуже корисну класифікацію об'єктів на інструменти й матеріали [19]. Матеріалами називаються об'єкти, що є свого роду контейнерами інформації й утримуючих методів тільки для нагромадження й нескладних перетворень цієї інформації. Інструментами називаються об'єкти, призначені для обробки матеріалів, тобто для більше інтелектуальних і складних перетворень тої інформації, що зберігають об'єкти – матеріали. Таким чином, з погляду цієї класифікації, ми вважаємо нейронні мережі (блоки) матеріалами, а конструктори й аналізатори – інструментами. Варто не плутати ці інструменти-об'єкти з інструментами-додатками, що є надбудовами над ядром.

У реалізації програми ми істотно використовували ідеї об'єктних шаблонів з [15]. Далі, в описі реалізації системи ми будемо використовувати російськомовні аналоги термінів, уведених в [15], тому, щоб не виникло плутанини, відзначимо, що Фабрика відповідає Factory, об'єктні шаблони – design patterns, Синглетон – Singleton, Chain of Responsibility – Ланцюжок Оброблювачів. Назви класів об'єктів будуть виділені курсивом і починатися із заголовної букви. Відзначимо, що ідея шаблонів у програмуванні й computer science виявилася досить плідної й слово «шаблон» тут ми використовуємо в трьох різних змістах: об'єктний шаблон (design pattern) та просто шаблон. Ми опишемо тільки реалізацію ядра системи.

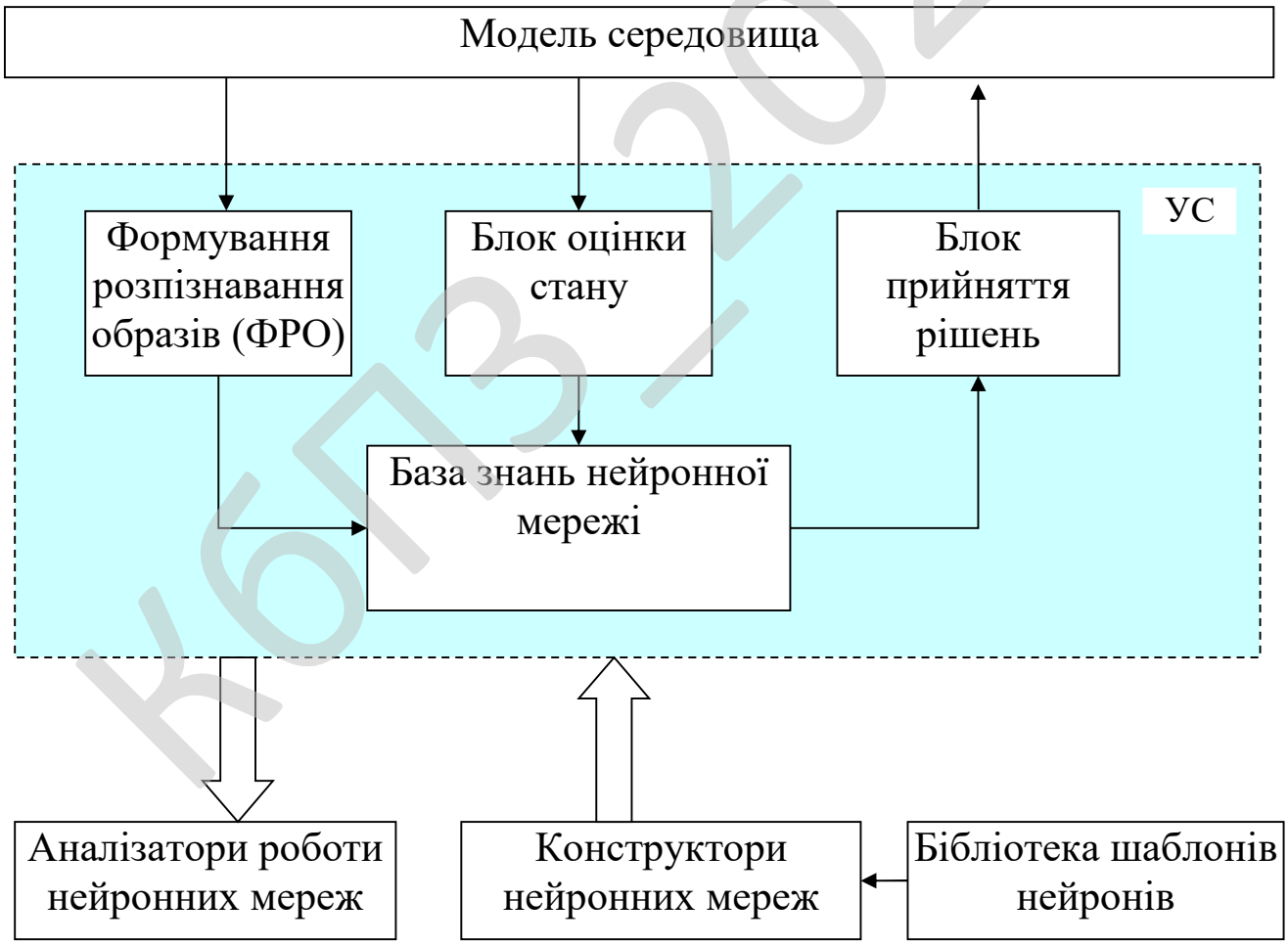

Рисунок 3.2 – Функціональна схема системи

Проходження принципам відкритості припускає закладання можливості розвитку системи через додавання надбудов над ядром. Ми, по можливості, намагалися додержуватися даного принципу. Зокрема, одним з напрямків розвитку ми бачимо створення конструкторів бібліотек шаблонів (а, отже, і мереж) за допомогою ГІП. Передбачається, що вихідним продуктом цих конструкторів будуть файли специфікації шаблонів, з якими вже вміє працювати ядро, з яких і будуть формуватися бібліотеки шаблонів. Далі, можна було б створити тривимірний візуалізатор БЗ, також ми вважаємо, знадобиться окремий інструмент для конструювання самих БЗ, а, можливо, при певному рівні складності блоків УС, і для кожного з них по окремому інструменті, які б ураховували повною мірою специфіку блоків УС.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.10. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Головне вікно ПЗ.

– Аналізатор радіально-базисної нейронних мереж.

– Визначення швидкості прийняття рішень.

– Оцінка ефективності роботи мережі.

– Виведення результатів.

– Конструктор радіально-базисної нейронної мережі.

– База знань.

– Навчання радіально-базисної нейронної мережі.

– Виведення результаті.

– Розпізнавання образів.

– Бібліотека налаштувань.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

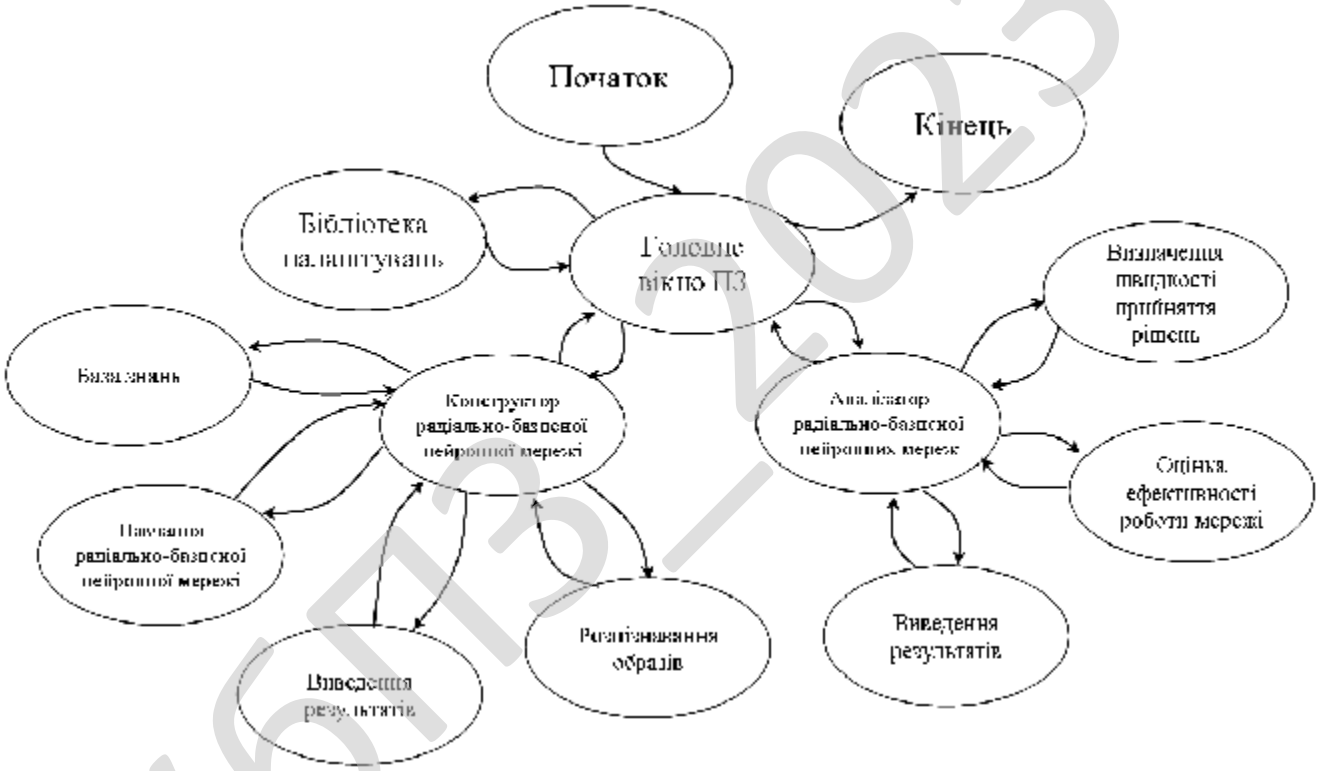

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З якої видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем я враховував, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня.

Розглянемо виконані основні напрацювання. Робота з файлами.

1. Зберегти файл у сховище.

```
CFIO_WriteFileToStorage (hStorage, hFile, lpNameInStorage);
```

HStorage – дескриптор сховища даних нейронної мережі.

HFile – дескриптор файлу.

lpNameInStorage – ім'я файлу зберігається в загальному сховищі.

Файл приписується до сховища й при закритті сховища даних нейронної мережі буде в нього занесений. Сам файл диска видаляється.

Повертає TRUE у випадку успіху або FALSE у випадку помилки.

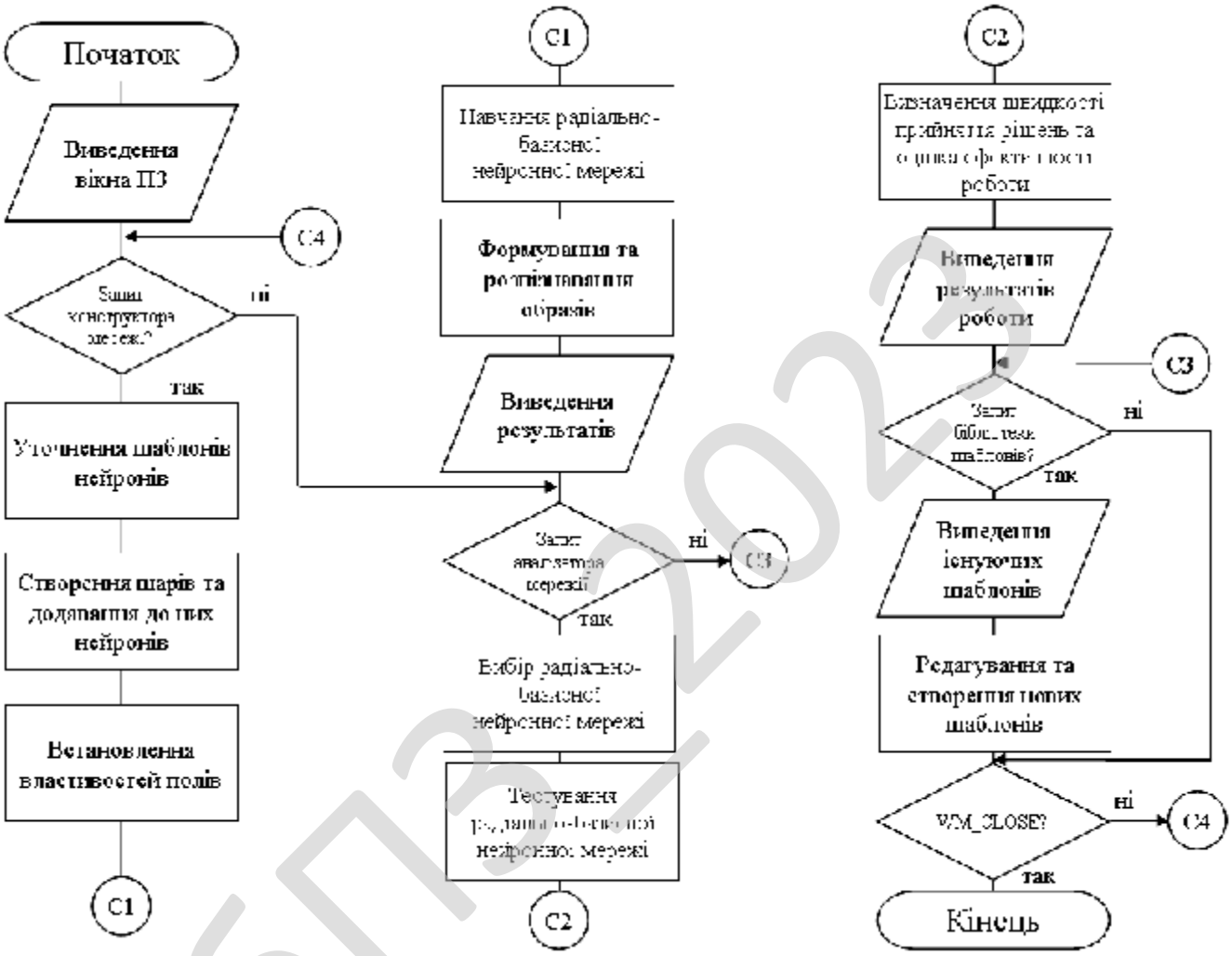

Рисунок 4.1 – Блок схема основної програми

2. Одержати дескриптор файлу зі сховища даних нейронної мережі.

```
Cfio_ReadFileFromStorage (hStorage, lpName);
```

HStorage – дескриптор сховища даних нейронної мережі.

LpName – ім'я файлу, що витягається із загального сховища даних нейронної мережі (під яким він записаний у сховище). Повертає файлу у випадку успіху або NULL у випадку помилки: файлу з таким ім'ям у сховище немає.

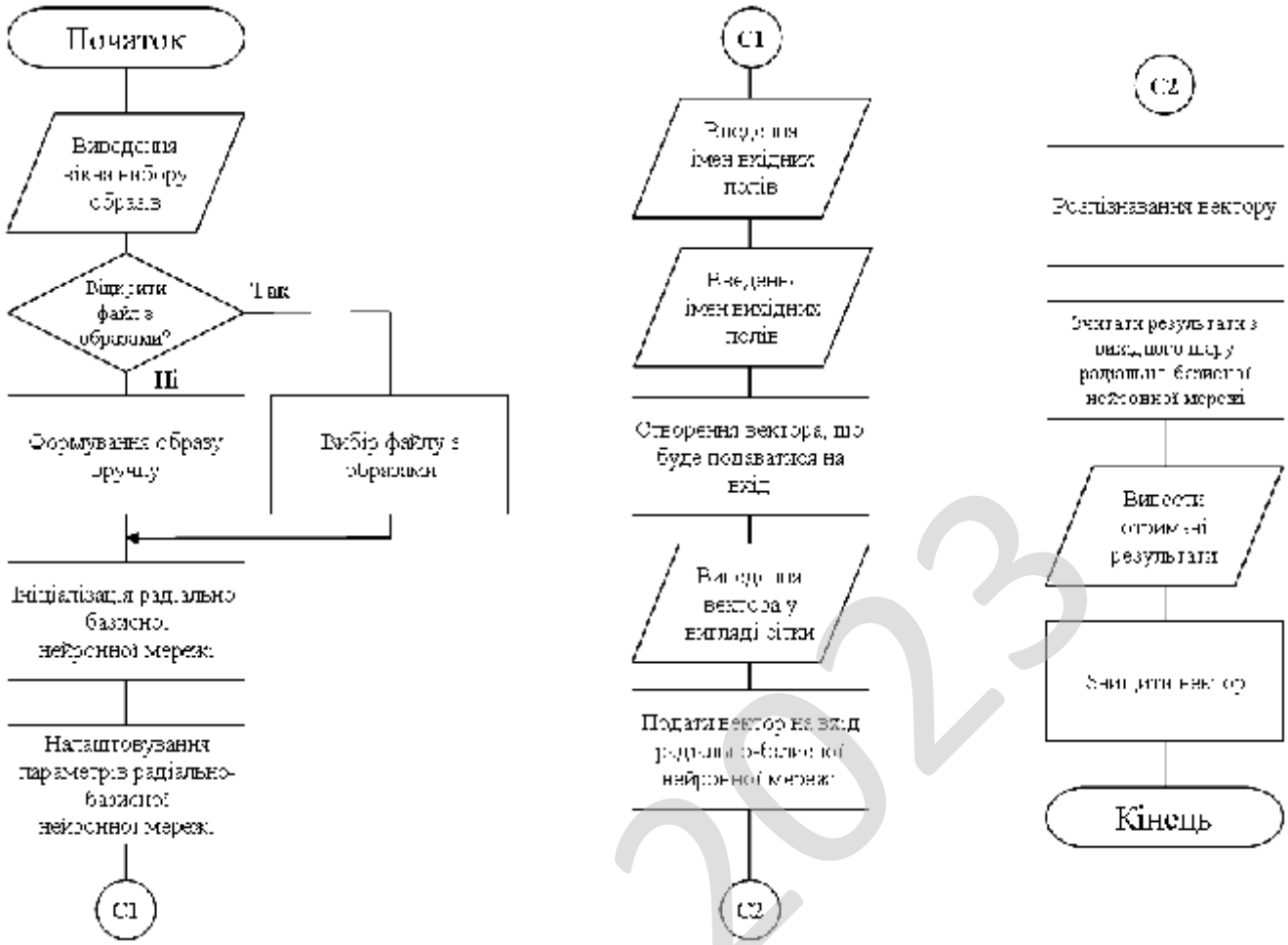

Рисунок 4.2 – Блок схема підпрограм

3. Відкрити файл.

```
CFIO_OpenFreeFile(hStorage, lpName, dwFlag);
```

HStorage – дескриптор сховища даних нейронної мережі до якого буде приписаний файл. Якщо NULL, файл до сховища не прив’язується.

lpName – ім'я файлу.

wdType – тип даних. Може приймати комбінацію з наступних значень:

– osf_create – створити новий файл;

– osf_open – відкрити існуючий файл;

– csf_read – відкрити файл на читання;

– osf_write – відкрити файл на запис;

– osf_binary – відкрити файл у двійковому виді;

– osf_in_memory – не виводити на диск.

– osf_temporary – тимчасовий;

Значення покажчику, що повертається, на дані або NULL у випадку помилки:

– Файлу з таким ім'ям не існує (при використанні прапора OSF_ OPEN).

4. Закрити файл.

```
CFIO_CloseFreeFile(hFile, dwFlag);
```

hFile – дескриптор файлу.

dwFlag – указує на дії вироблені при закритті.

Може приймати комбінацію з наступних значень:

– CSF_SAVEDISK – закрити зі збереженням на диску. Файл відкреплюється від сховища даних нейронної мережі.

– CSF_SAVESTORAGE – закрити зі збереженням у загальному сховищі.

– CSF_DELETE – закрити й видалити файл із відкріпленням від сховища даних нейронної мережі.

Значення, що повертається: TRUE у випадку успіху.

5. Записати дані на диск.

```
CFIO_WriteToFile (hFile, lpData, dwSize);
```

hFile – покажчик отриманий від Open.

lpData – покажчик на дані, що зберігаються.

dwSize – число зберігаємих байт.

Повертає число записаних байт.

6. Прочитати дані з диска.

```
CFIO_ReadFromFile (hFile, lpData, dwSize);
```

hFile – покажчик отриманий від Open;

lpData – покажчик на зчитувальні дані;

dwSize – число зчитувальних байт;

Повертає число прочитаних байт.

7. Перемістити покажчик.

```
CFIO_SeekFilePointer (hFile, dwBytes, dwFrom);
```

hFile – покажчик отриманий від MFOpen.

dwBytes – число байт на яке відбувається зсув.

dwFrom – прапор, звідки відбувається зсув. Може приймати одне з наступних значень:

– FS_END – з кінця файлу.

– FS_BEGIN – з початку файлу.

– FS_CUR – з поточного значення.

Повертає позицію, починаючи з початку файлу.

8. Прочитати поточний покажчик. Повертає позицію, починаючи з початку файлу.

```
CFIO_TellFilePointer (hFile);
```

hFile – дескриптор файлу, отриманий від Open;

dwBytes – число байт на яке відбувається зсув.

9. Скинути кешовані дані на диск.

```
CFIO_FlushFile(hFile);
```

hFile – дескриптор, отриманий від Open;

Робота з пам'яттю.

10. Одержати покажчик на блок пам'яті.

```
CFIO_AllocMemory (dwSize, dwFlag);
```

DwSize – розмір блоку.

DwFlag – атрибути пам'яті. Може приймати одне з наступних значень:

– maf_gptr – одержати покажчик на пам'ять; пам'ять обнуляється; значення, що повертається – покажчик на пам'ять;

– maf_ghnd – одержати покажчик на глобальну пам'ять; пам'ять обнуляється; значення, що повертається – покажчик на глобальний об'єкт;

– maf_gall_gmem_fixed – описаний у прапорах globalalloc;

– maf_gall_gmem_moveable – описаний у прапорах globalalloc;

– maf_gall_gptr – описаний у прапорах globalalloc;

– maf_gall_ghnd – описаний у прапорах globalalloc;

– maf_gall_gmem_ddeshare – описаний у прапорах globalalloc;

– maf_gall_gmem_discardable – описаний у прапорах globalalloc;

– maf_gall_gmem_lower – описаний у прапорах globalalloc;

– maf_gall_gmem_nocompact – описаний у прапорах globalalloc;

– maf_gall_gmem_nodiscard – описаний у прапорах globalalloc;

– maf_gall_gmem_not_banked – описаний у прапорах globalalloc;

– MAF_GALL_GMEM_NOTIFY – описаний у прапорах GlobalAlloc;

– MAF_GALL_GMEM_SHARE – описаний у прапорах GlobalAlloc;

– MAF_GALL_GMEM_ZEROINIT – описаний у прапорах GlobalAlloc.

11. Перевиділити блок пам'ять.

```
Cfio_ReAllocMemory (hMemory, dwSize, dwFlag);
```

hMemory – дескриптор блоку пам'яті.

dwFlag – параметр, може приймати наступні значення:

– MRF_NEW_MEMORY – виділяє новий блок, копіює в нього вміст старого й звільняє старий.

– MRF_GALL_GMEM_DISCARDABLEGPTR – описаний у прапорах GlobalReAlloc.

– MRF_GALL_GMEM_MOVEABLE – описаний у прапорах GlobalReAlloc.

– MRF_GALL_GMEM_NOCOMPACT – описаний у прапорах GlobalReAlloc.

– MRF_GALL_GMEM_ZEROINIT – описаний у прапорах GlobalReAlloc.

12. Звільнити блок пам'яті.

CFIO_FreeMemory(hMem);

hMem – пам’ять, що звільняється.

13. Закріпити глобальний об'єкт і одержати покажчик на пам'ять. Повертає покажчик на пам'ять або NULL.

```
CFIO_LockMemory(hMem);
```

hMem – покажчик на глобальний об'єкт.

14. Відкріпити глобальний об'єкт.

```
CFIO_UnlockMemory(hMem);
```

hMem – покажчик на глобальний об'єкт.

15. Записати дані з виділеного блоку пам'яті на диск.

```
CFIO_WriteMemoryToFile(hMem, lpName);
```

hMem – покажчик на пам'ять або глобальний об'єкт.

lpName – ім'я файлу, куди зберігати.

Повертає число записаних байт.

16. Виділити блок пам'яті потрібного розміру й зчитати дані з диска до пам’яті.

```
ReadMemFromFile (PChar lpName, lphMem);
```

lphMem – (out) посилання на покажчик на пам'ять або глобальний об'єкт.

lpName – (in) ім'я файлу, звідки читати.

Повертає число зчитаних байт.

17. Записати дані з виділеного блоку в сховище.

```
CFIO_ReadMemoryFromFile(lpName,  phMem);
```

phMem – покажчик на пам'ять або глобальний об'єкт.

lpName – ім'я, під яким зберігати в сховище.

Повертає число записаних байт.

18. Виділити блок пам'яті потрібного розміру й уважати в нього дані зі сховища даних нейронної мережі.

```
CFIO_ReadMemoryFromStorage(hStorage, lpName, phMem);
```

hStorage – дескриптор сховища даних нейронної мережі.

phMem – (out) посилання на покажчик на дескриптор пам'яті, створеної в процесі читання.

lpName – (in) ім'я, під яким вони зберігаються у сховищі.

Повертає число зчитаних байт.

Робота з диском.

19. Відкрити сховище загальних даних.

```
CFIO_OpenStorage(lpName, dwTypes);
```

LpName – ім'я загального файлу-сховища даних нейронної мережі

DwType – тип дій. Може приймати комбінацію з наступних значень:

– OS_CREATE – Створити нове сховище.

– OS_OPEN – Відкрити існуюче сховище. Файли зі сховища даних нейронної мережі відписуються в тимчасову директорію.

Повертає дескриптор відкритого сховища даних нейронної мережі або NULL у випадку помилки:

– сховище не існує (при використанні прапора OS_OPEN);

– при відкритті існуючого сховища даних нейронної мережі на диску перебувають файли, що збігаються по ім'ю з файлами сховища даних нейронної мережі.

20. Закрити сховище загальних даних.

```
CFIO_CloseStorage(hStorage,  dwFlag);
```

HStorage – дескриптор сховища даних нейронної мережі.

DwFlag – тип дій. Може приймати комбінацію з наступних значень:

– CS_DELETE – закрити й видалити сховище.

– CS_SAVE – закрити зі збереженням.

– CS_FORSE_SAVE – при закритті сховища даних нейронної мережі всі прикріплені до нього файли заносяться в сховище.

Повертає TRUE у випадку успіху або FALSE у випадку помилки.

21. Видалити сховище.

```
CFIO_DeleteStorage (lpName);
```

LpName – ім'я сховища даних, що видаляється, нейронної мережі.

Повертає TRUE у випадку успіху або FALSE у випадку помилки:

– сховище не існує.

Крім цього було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація. UML може бути

застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм.

Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем. Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту. UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об'єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи. Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

4.2 Захист розробленого програмного забезпечення

Дані у системі захищаються за допомогою алгоритму обміну ключа Діффі-Хеллмана.

Ціль алгоритму полягає в тому, щоб два учасники могли безпечно обмінятися ключем, що надалі може використовуватися в якому-небудь алгоритмі симетричного шифрування. Сам алгоритм Діффі-Хеллмана може застосовуватися тільки для обміну ключами.

Алгоритм заснований на труднощі обчислень дискретних логарифмів. Дискретний логарифм визначається в такий спосіб. Уводиться поняття примітивного кореня простого числа Q як числа, чиї ступені створюють всі цілі від 1 до Q – 1. Це означає, що якщо A є примітивним коренем простого числа Q, тоді числа:

$$A \bmod Q, A^2 \bmod Q, \ldots, A^{Q-1} \bmod Q,$$

є різними й складаються із цілих від 1 до Q – 1 з деякими перестановками. У цьому випадку для будь-якого цілого Y < Q і примітивного кореня A простого числа Q можна знайти єдину експоненту X, таку, що:

$$Y = A^X \bmod Q,$$

де $0 \leq X \leq (Q - 1)$.

Експонента X називається дискретним логарифмом, або індексом Y, по підставі A mod Q. Це позначається як $\text{ind}_{,Q}(Y)$.

Тепер опишемо алгоритм обміну ключів Діффі-Хеллмана.

Загальновідомі елементи

Q – просте число.

A – A < Q і A є примітивним коренем Q.

Створення пари ключів користувачем I

Вибір випадкового числа X_i (закритий ключ):

$$X_i < Q.$$

Обчислення числа Y_i (відкритий ключ):

$$Y_i = A^{Xi} \bmod Q.$$

Створення відкритого ключа користувачем J

Вибір випадкового числа X_j (закритий ключ):

$$X_j < Q.$$

Обчислення випадкового числа Y_j (відкритий ключ):

$$Y_j = A^{Xj} \bmod Q.$$

Створення загального секретного ключа користувачем I

$$K = (Y_j)^{Xi} \bmod Q.$$

Створення загального секретного ключа користувачем J

$$K = (Y_i)^{Xj} \bmod Q.$$

Передбачається, що існують два відомих усім числа: просте число Q і ціле A, що є примітивним коренем Q.

Тепер припустимо, що користувачі I і J хочуть обмінятися ключем для алгоритму симетричного шифрування.

Користувач I вибирає випадкове число $X_i < Q$ і обчислює:

$$Y_i = A^{Xi} \bmod Q.$$

Аналогічно користувач J незалежно вибирає випадкове ціле число $X_j < Q$ і обчислює:

$$Y_j = A^{Xj} \bmod Q.$$

Кожна сторона тримає значення X у секреті й робить значення Y доступним для іншої сторони.

Тепер користувач I обчислює ключ як $K = (Y_j)^{Xi} \bmod Q$, і користувач J обчислює ключ як $K = (Y_i)^{Xj} \bmod Q$. У результаті обоє одержать те саме значення:

$$K = (Y_j)^{Xi} \bmod Q = (A^{Xj} \bmod Q)^{Xi} \bmod Q = (A^{Xj})^{Xi} \bmod Q =$$

за правилами модульної арифметики

$$= A^{Xj\,Xi} \bmod Q = (A^{Xj})^{Xj} \bmod Q = (A^{Xi} \bmod Q)^{Xj} \bmod Q = (Y_i)^{Xj} \bmod Q$$

Таким чином, дві сторони обмінялися секретним ключем. Так як X_i і X_j є закритими, супротивник може одержати тільки наступні значення: Q, A, Y_i і Y_j. Для обчислення ключа атакуючий повинен зламати дискретний логарифм, тобто обчислити:

$$X_j = \mathrm{ind}_{a,\,q}(Y_j).$$

Безпека обміну ключа в алгоритмі Діффі-Хеллмана випливає з того факту, що, хоча відносно легко обчислити експоненти по модулю простого числа, дуже важко обчислити дискретні логарифми. Для великих простих чисел задача вважається нерозв'язною.

Варто помітити, що даний алгоритм уразливий для атак типу "in-the-middle". Якщо супротивник може здійснити активну атаку, тобто має можливість

не тільки перехоплювати повідомлення, але й заміняти їх іншими, він може перехопити відкриті ключі учасників Y_i і Y_j , створити свою пару відкритого й закритого ключа ($X_{оп}$, $Y_{оп}$) і послати кожному з учасників свій відкритий ключ. Після цього кожний учасник обчислить ключ, що буде загальним із супротивником, а не з іншим учасником. Якщо немає контролю цілісності, то учасники не зможуть виявити подібну підміну.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс головного вікна програми, у режимі конструктора радіально-базисної нейронної мережі на етапі навчання. З цього рисунка видно, що головне вікно включає в себе наступні елементи:

– Блок меню.

– Блок режимів роботи.

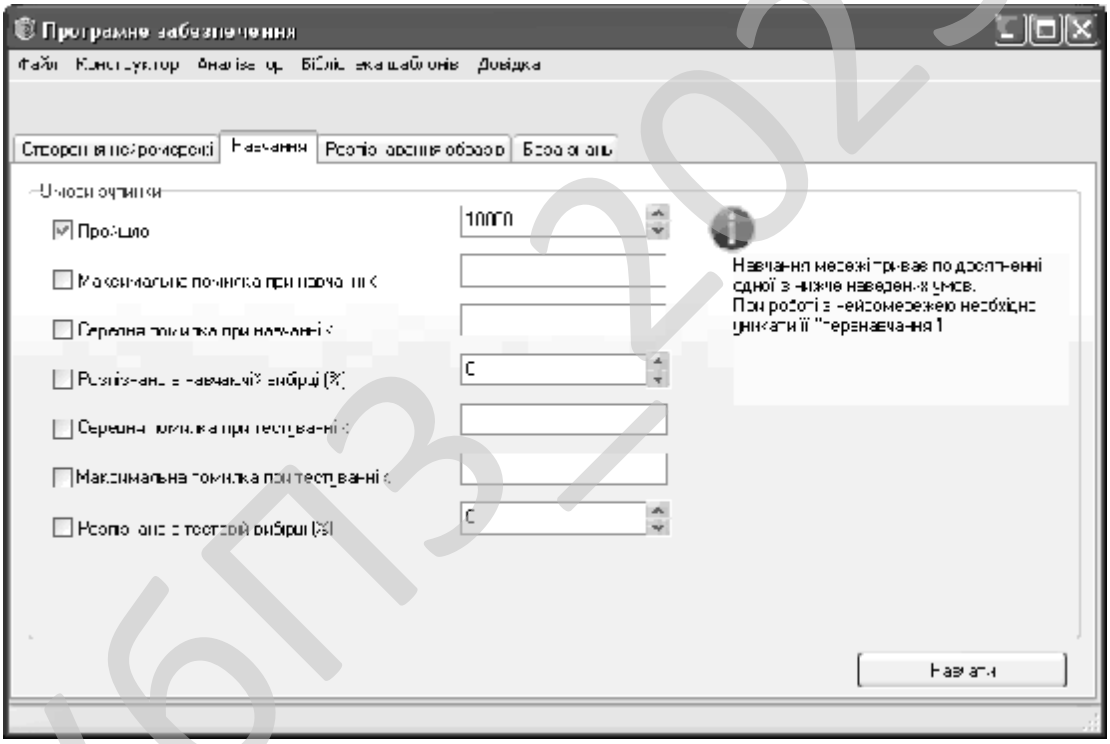

Рисунок 5.1 – Головне вікно ПЗ

Блок меню включає в себе наступні елементи:

– Файл.

– Конструктор.

– Аналізатор.

– Бібліотека шаблонів.

– Довідка.

Блок режимів роботи включає в себе наступні режими:

– Створення радіально-базисної нейронної мережі.

– Навчання радіально-базисної нейронної мережі.

– Розпізнання образів.

– База знань.

У режимі конструктора радіально-базисної нейронної мережі, на етапі навчання, ми бачимо, що користувач, може задати наступну інформацію для зупинки навчання:

– Кількість епох, які пройшли.

– Максимальна помилка при навчанні.

– Середня помилка при навчанні.

– Кількість розпізнаних образів, в відсотках.

– Величину максимальної помилки при тестуванні.

– Середня помилка при тестуванні.

– Кількість розпізнаних у тестовій вибірці, у відсотках.

На рисунку 5.3 наведено вікно авторського права. Обрано умови розповсюдження – proprietary software. Програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримує обмежені права користування ним: може бути заборонено або закрито доступ до коду (вивчення), внесення змін, тиражування, розповсюдження та перепродаж. Програмне забезпечення вважається власницьким, якщо наявне хоча б одне з перелічених обмежень. Найчастіше основним методом захисту майнових прав на власницьке ПЗ, поза ліцензійною угодою, власник обирає закриття сирцевого коду, захищаючи свій продукт від модифікації і вбудовуючи системи обмеження користування через авторизацію. Таке програмне забезпечення називається закритим. Проте, код власницького продукту може бути і відкритим, але власник може обмежити права користувача умовами користувальницької ліцензії.

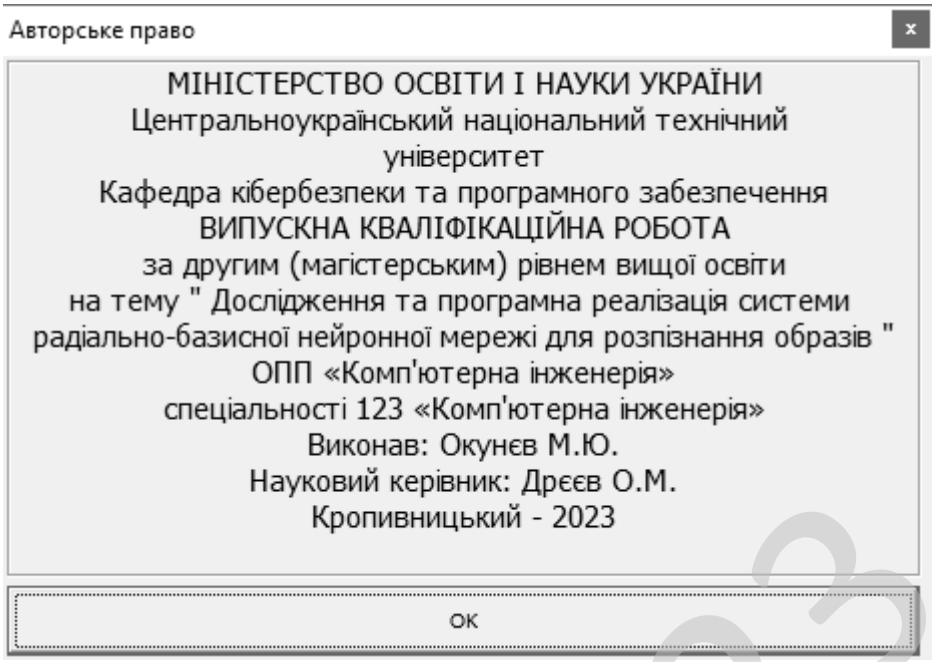

Рисунок 5.3 – Вікно авторського права

Власницьке програмне забезпечення та комерційне програмне забезпечення не є синонімами – власницьким може бути і безплатне (тобто, некомерційне) програмне забезпечення. На противагу власницькому ПЗ існує вільне програмне забезпечення, автори і власники якого дозволяють вивчати, модифікувати і поширювати свій продукт. Саме визначення власницького програмного забезпечення виникло в результаті діяльності громадського руху вільного програмного забезпечення (представленого Фондом вільного програмного забезпечення та іншими організаціями) і осмислення умов свободи користування програмами. Сама назва власницьке ПЗ підкреслює визначальне значення власника у способі використання і можливостях розвитку цього програмного забезпечення.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи радіально-базисної нейронної мережі для розпізнання образів.

Метою розробки є дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Об'єктом дослідження є процес радіально-базисної нейронної мережі для розпізнання образів.

Предметом дослідження є методи радіально-базисної нейронної мережі для розпізнання образів.

Методи дослідження базуються на методах штучного інтелекту, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод радіально-базисної нейронної мережі для розпізнання образів.

– Розроблено вітчизняний продукт радіально-базисної нейронної мережі для розпізнання образів, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці). В магістерській роботі було проведене дослідження та виконана програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	100
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	100000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$

$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); *S* – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 49 = 83 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	83	Ф 7.1-7.4
Впровадження	13	Д13
Всього	124	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{124 \cdot 1}{60 - 5} = 2,25 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	2	60	1
Кабельні господарства ЛОМ на 1 м.п.	2,5	200	500	8,33
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	39,49

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф^{с}_{др} = \frac{З_ч \cdot n_{міс}}{1,2}, \quad (7.6)$$

$$Ф^{с}_{др} = \frac{39,49 \cdot 3}{1,2} = 99 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 99/(60 \cdot 8) = 0{,}2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2016, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	0,25	11124	8343
Продакт-менеджер	0,25	9052	6789
Інженер-програміст	2,25	12200	82350
Інженер - електронщик	0,2	8000	4800
Інженер-системотехнік	0,25	8000	6000
Адміністратор мережі	0,5	8000	12000
Системний програміст	0,25	8000	6000
Дизайнер WEB	0,25	8000	6000
Інженер-верстальник	0,25	8000	6000
Бухгалтер-економіст	0,5	8000	12000
Всього за період розробки	$R_{сп}$ = 4,95	-	$Ф_{роб}$ = 150282

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{150282}{4,95 \cdot 60} = 506 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 29000 = 1858000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 185800 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Комп'ютерторг за 20.10.23 – джерело http://computorg.ua/ru/price.html

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	AMD Athlon ™ 3000G (YD3000C6M2OFB) AM4, 2 ядра, 4 потоки, 3.5 GHz	-
Системна плата	GIGABYTE A520M S2H сокет - AM4, DDR4, LAN - 1 Гбит/с, D-Sub (VGA), DVI, HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Відеокарта	GeForce GT1030 2048Mb ASUS (GT1030-2G-BRK) PCI-Express 3.0, 2 ГБ, GDDR5	-
Жорсткий диск	SSD M.2 2280 240GB Apacer (AP240GAST280-1) 240 GB, TLC, M.2, SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2400 MHz eXceleram (E408247D)	-
DVD-привод	відсутній	-
Корпус	ATX Middle Tower FOXCONN Pro, 3GTLA-489, PSU 350W(FSP Brand: ATX-450PNR, 12cm), black, (front bezel – black+light silver; body material – 0.6mm), 80mm fan (rear), 2xUSB2.0/AUDIO/MIC, Air Duct, Tool-less chassis design,Thermally Advantaged Chassis	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кардрідер внутрішній	USB 3.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black	220
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1 170/160, D-SUB, Wide)	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	15	10947	16420,5	180625,5
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	199177

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1858000	-	-
2. Передавальні пристрої	185800	-	-
Всього по групі	2043800	5	102190
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
4. Нематеріальні активи	100000	10	10000
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	143000	20	28600
7. Господарський інвентар	28000	25	7000
Всього по групі	180031	-	37857,75
Разом	$К_р$ = 2523008		$А_р$ = 249636,25

Примітка: вартість автомобіля Renault Scenic 2006 взята за даними електронного ресурсу, джерело https://auto.ria.com/uk/auto_renault_scenic_33598032.html, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 506 \cdot 164/100 = 629 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 629 \cdot 10 \cdot 0{,}01 = 63 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(629+63) = 256 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 629 \cdot 15 \cdot 0{,}01 = 94 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.; $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.; $З_{М3}$ – вартість фарби, картриджів, тонеру, грн.; $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=206 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 206 \cdot 1,5 = 309 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 50):

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 24 грн./шт., DVD-R box – 39 грн./шт.

$$З_{М2} = 49 \cdot 24 + 1 \cdot 39 = 1215 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з ., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574 + 558 + 570 = 1702 \text{ грн.}$$

$$З_М = (309 + 1215 + 1702)/100 = 32 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 629 \cdot 15 \cdot 0,01 = 94 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 100 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 249636 \cdot 3/(100 \cdot 12) = 624 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	629
2. Додаткова зарплата виконавців	$З_д$	63
3. Відрахування на соціальні потреби	$С_{оц}$	256
4. Загальногосподарські витрати	$Г_{осп}$	94
5. Витрати на матеріали	$З_М$	32
6. Освоєння нових операційних систем, мов програмування	$О_п$	94
7. Амортизація основних фондів	$А_м$	624
8. Повна собівартість програмного забезпечення	$С_п$	1792
9. Плановий прибуток	$П_р$	896
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	2688
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	537,6
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	3225,6

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 629+63+256+94+32+94+624 = 1792 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (P_n) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_n \cdot C_n, \quad (7.22)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 1792 = 896 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	3226
Всього капітальних витрат	–	3226

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування)	$З_р$	32208	12078
2. Витрати на електроенергію	$З_{ел}$	276	58
3. Витрати на амортизацію	$З_{ам}$	0	807
Всього витрат за рік	$І$	32484	12943

Витрати на профілактичні роботи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування кожного комп'ютера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 240 годин на рік до 90 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\,баз} = 240 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 32208 \text{ грн},$$

до:

$$З_{р\,нов} = 90 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 12078 \text{ грн.}$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \qquad (7.24)$$

$$З_{ел\,баз} = 0{,}5 \cdot 240 \cdot 2{,}3 = 276 \text{ грн.}$$

$$З_{ел\,нов} = 0{,}5 \cdot 50 \cdot 2{,}3 = 58 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	3226	–	806,5
Всього відрахувань	-	–	3226	–	806,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (2688\text{-}1792) \cdot 100\text{-}(0{,}05 \cdot 2043800 + 0{,}4 \cdot 199177 + 0{,}25 \cdot 37031 + $$
$$+ 0{,}1 \cdot 100000 + 0{,}2 \cdot 143000) \cdot 3/12 = 32170 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{479208}{(2688\text{-}1792)\cdot 100\cdot 12/3} = 1{,}3 \text{ років}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	100
2. Повна собівартість розробленої програми	Грн.	1792
3. Ціна розробленої програми	Грн.	2688
4. Плановий прибуток від реалізації розробленої програми	Грн.	896
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2523008
7. Загальний прибуток від реалізації програмної продукції	Грн.	89600
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	32170
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	1,3
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3226
11.Величина економічного ефекту у користувача програмної продукції	Грн.	18735
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,16

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \qquad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (32484\text{-}12943)\text{-}0{,}25 \cdot 3226 = 18735 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \tag{7.28}$$

$$T_{сп} = \frac{3226}{32484 - 12943} = 0{,}16 \text{ року}.$$

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Техніка безпеки – це система правил і заходів, які допомагають запобігти травмам, хворобам і аваріям на робочому місці або в повсякденному житті. Знання техніки безпеки дуже важливе, бо воно рятує життя і здоров'я людей. Наприклад, якщо людина працює на шахті, то вона повинна знати правила безпеки у вугільних шахтах, щоб не потрапити під обвал або не підірватися на міні. Або якщо людина хоче навчитися програмувати, то вона повинна дотримуватися гігієнічних вимог і не сидіти за комп'ютером занадто довго, щоб не зашкодити своїм очам і спині.

Охорона праці та здоров'я у сфері ІТ – це комплекс заходів, які спрямовані на забезпечення безпечних і здорових умов праці для працівників, які використовують інформаційні технології, а також на запобігання травматизму, професійним захворюванням і стресу.

Охорона праці та здоров'я у сфері ІТ включає такі напрями, як:

– Ергономіка – це наука про адаптацію робочого середовища до фізичних і психологічних особливостей людини[2]. Ергономіка вимагає врахування таких факторів, як розміри, форми, кольори, освітлення, шум, температура, вологість, вентиляція, постава, рухи, пози, втома, навантаження на очі та ін. Егономіка допомагає покращити комфорт, продуктивність і задоволення працівників.

– Комп'ютерна безпека – це захист комп'ютерних систем і даних від несанкціонованого доступу, зміни, знищення або блокування. Комп'ютерна безпека вимагає використання антивірусних програм, фаєрволів, паролей, шифрування, резервного копіювання та інших технологій. Комп'ютерна безпека допомагає запобігти крадіжці, шпигунству, шантажу, саботажу та іншим загрозам.

– Соціальна взаємодія – це процес спілкування між людьми у робочому колективі або через мережеві сервіси. Соціальна взаємодія вимагає дотримання правил етикету, поваги, толерантності, співробітництва та конструктивного діалогу. Соціальна взаємодія допомагає покращити настрій, мотивацію, комунікацію та творчий потенціал працівників.

Правила охорони праці і здоров'я для програмістів:

– Регулярно роби перерви в роботі. Вставай із-за столу і розминай м'язи.

– Налаштуй яскравість і контрастність монітору так, щоб не напружувати очі.

– Використовуй ергономічну мишку і клавіатуру, які зручно лягають у руку і не виклкають болю.

– Слідкуй за своєю поставою. Сиди прямо і не нахиляйся до екрану.

– Захищай свій комп'ютер від вірусів, шпигунських програм і хакерів. Оновлюй ативірусне програмне забезпечення і не відкривай підозрілі файли і посилання.

– Не забувай про соціальну взаємодію. Спілкуйся з колегами, друзями і родиною. – Відвідуй заходи, які тебе цікавлять. Не ізолюй себе від світу.

– Люби свою професію, але не забувай про інші сфери життя. Розвивай свої захоплення, хобі і таланти. Знаходь рівновагу між роботою і відпочинком.

Закон України “Про охорону праці” визначає основні принципи, завдання, права і обов’язки суб’єктів відносин з охорони праці, а також організаційні та правові основи державного управління і контролю за дотриманням законодавства про охорону праці.

Згідно з цим законом, ІТ компанії повинні впроваджувати такі заходи з охорони праці:

– Створювати на підприємстві службу охорони праці або призначати відповідальних осіб, які забезпечують розроблення, реалізацію та контроль за дотриманням заходів з охорони праці.

– Забезпечувати безпечні і нешкідливі умови праці для працівників, використвючи сучасні засоби техніки безпеки, санітарно-гігієнічні умови, засоби клектинго та індивідуального захисту, оптимальні режими праці та відпочинку.

– Проводити атестацію робочих місць на відповідність нормативно-правовим атам з охорони праці та аудит з охорони праці.

– Проводити навчання та інструктаж з питань охорони праці, з надання першої медичної допомоги потерпілим від нещасних випадків і правил поведінки у разі виникнення аварії[5].

– Забезпечувати лікувально-профілактичне обслуговування працюючих, санітарно-побутове обслуговування, пільги і компенсації для працівників, які працюють у важких і шкідливих умовах.

– Нести відповідальність за порушення законодавства про охорону праці та заподіяння шкоди життю і здоров'ю працівників.

8.2 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об'єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов'язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об'єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами. Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам. Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань. Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах. [4]

8.3 Характеристика умов праці програміста

В приміщенні, в якому проводиться розробка і дослідження програмного продукту, відсутні умови, які можуть створювати підвищену або особливо підвищену небезпеку, тому воно відноситься до класу звичайних приміщень згідно Правил улаштування електроустановок (ПУЕ). Джерелом живлення є трифазна мережа напруги 380/220 В з глухо заземленою нейтралі, з частотою 50 Гц згідно За пожежо-вибухонебезпекою приміщення відноситься до класу В. В таблиці 8.2 наведена загальна характеристика приміщення щодо вибухопожеженобезпеки та важкістю робіт.

Таблиця 8.1 – Загальна характеристика приміщення щодо вибухопожежонебезпеки та важкістю робіт

Характеристика приміщень за вибухопожежною категорією та класом зони	Загальна характеристика приміщення	Категорія за важкістю робіт згідно ГН 3.3.5-8.6.6.1 -2002
В – пожежонебезпечне клас П – II	Звичайне без ознак хімічного забруднення та нормальної вологості і за санітарними нормами	1а…….до 139 Вт/м^2 1б…….до 140-174 Вт/м^2 Клас умов праці – оптимальний

Температура повітря в приміщенні визначається температурою зовнішнього повітря і тепловою енергією, що виділяється всередині приміщення. Джерелами теплоти в даному приміщенні є люди, електроустаткування, а також освітлювальні прилади в темний час доби. Зовнішнім джерелом надлишкового тепла є сонячна радіація у світлий час доби. Робота, виконувана в даному

приміщенні, відноситься до категорії І-а. Людиною в цьому випадку виділяється до 120 ккал теплової енергії в годину. Вологість повітря в приміщенні визначається вологістю атмосферного і видихуваного людьми повітря, а також випарами з поверхні шкіри.

У приміщенні немає виділення шкідливих газів. Тому що в ньому не проводиться монтажних робіт, пайки чи інших робіт, при яких виділяються шкідливі гази.

Для нормалізації параметрів повітряного середовища також періодично здійснюється провітрювання приміщення і вологе прибирання. У всьому будинку діє встановлена загально обмінна витяжна вентиляція.

Раціональне освітлення приміщення сприяє кращому виконанню виробничого завдання і забезпеченню комфорту при роботі. Для забезпечення нормального освітлення застосовуються природне, однобічне, бічне і штучне освітлення, а також сполучене, нормуються зідно ДБН В.2.5-28-2006 Природне і штучне освітлення [1].

За результатами виміру освітленості величина освітленості від системи загального штучного освітлення дорівнює 310 лк, що відповідає вимогам, які пред'являються до даного приміщення.

Основними джерелами шуму на робочих місцях , обладнаних відео дисплейними терміналами, є принтер, сканер факс і обладнання для кондиціювання повітря, в самих відео дисплейних терміналів – вентилятори систем охолодження і трансформатори.

Згідно ДСанПіН 3.3.2.007-98 [2] допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА (акустичних децибела).

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга) [4, 10, 11].

8.5 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток *63·63·6 мм*., (згідно з ДСТУ 2251-93 «Кутики сталеві гарячекатані рівнополичні. Сортамент») довжиною *L=2 м.,* та горизонтальний електрод – металева полоса з перетином *60·5 мм*. Напруга – *220/380 В*. Розрахункова схема розташування заземлюючих електродів – по контуру (прямокутником).

Розрахунок проведем за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина (питомий опір $\rho_2 = 40$ *Ом·м)*. Умовна товщина верхнього шару грунта: *H=0,5 м*. Відстань між вертикальними

заземлювачами (електродами) *А=2 м*. Глибина закладення горизонтального контура заземлення *t=0,6 м*. Опір заземлювача, який нормується: $R_{ЗН}$ = *4 Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,6+2/2=1,6\ м.$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi\rho = 1,36 \cdot 40 = 54,5\ Ом \cdot м.$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідной климатичной зони у багатошаровому грунті [12];

$\rho_2 = 40\ Ом \cdot м.$ – табличне значення питомого опору нижнього шару грунта (глина) [12].

Еквівалентний діаметр ветикальньного електрода (кутка) [12]:

$$Dв=0,95 \cdot K = 0,95 \cdot 63 = 59,85\ мм. = 0,0598\ м.$$

де *К= 63 мм.* – розмір метелевого кутка (задан).

Відношення *А/L=2/2=1*

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [12]:

$$R_O = 0,366(\rho/L)[\lg(2L/Dв)+(1/2)\lg((4T+L)/(4T-L))] =$$

$$= 0,366(54,5/2)[(\lg(2 \cdot 2/0,0598)+(1/2)\lg((4 \cdot 1,6+2)/(4 \cdot 1,6-2))] =$$

$$= 19,6\ Ом.$$

Визначаємо коефіцієнт екранування вертикальних електродів *Кев=0,62* при орієтовній кількості вертикальних електродів, яке дорівнює 5 [12].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без врахування горизонтального заземлювача), при $R_{ЗН}$ = 4 *Ом*:

$$N = R_O / (Кев\ R_{ЗН}) = 19,6 / (0,62 \cdot 4) = 7,8 \approx 8\ шт.$$

Визначаємо длину з'єднуючої полоси:

$$L_П = 1,05 \cdot A \cdot N = 1,05 \cdot 2 \cdot 8 = 16,6 \approx 16 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [12]:

$$R_П = 0,366(\rho \cdot K_П / L_П) lg(2 \cdot L_П^2 / (B \cdot t)) =$$

$$= 0,366(40 \cdot 5/16) \cdot lg((2 \cdot 16^2)/(0,06 \cdot 0,6)) = 20,2 \text{ Ом.}$$

де $K_П = 5$ – табличне значення кліматичного коефіцієнта питомого опору грунта для відповідной климатичной зони для з'єднуючої полоси [12]:

$B = 60$ мм. $= 0,06$ м.- ширина з'єднуючої полоси (задана).

Загальньний опір розтіканню електричного струму заземлювача [12]:

$$R = (R_O \cdot R_П)/(R_O \cdot \eta_П + N \cdot R_П \cdot K_{ев}) =$$

$$= (19,6 \cdot 20,2)/(19,6 \cdot 0,6 + 8 \cdot 20,2 \cdot 0,62) = 3,53 \text{ Ом.}$$

де $\eta_П = 0,6$ – табличне значення коефіцієнта екранування з'єднуючої полоси [12].

Умова $R \le R_{ЗН}$ виконується ($3,53 \le 4$).

Остаточно отримали: кількість вертикальних електродів дорівнює 8 при $R = 3,53$ Ом.

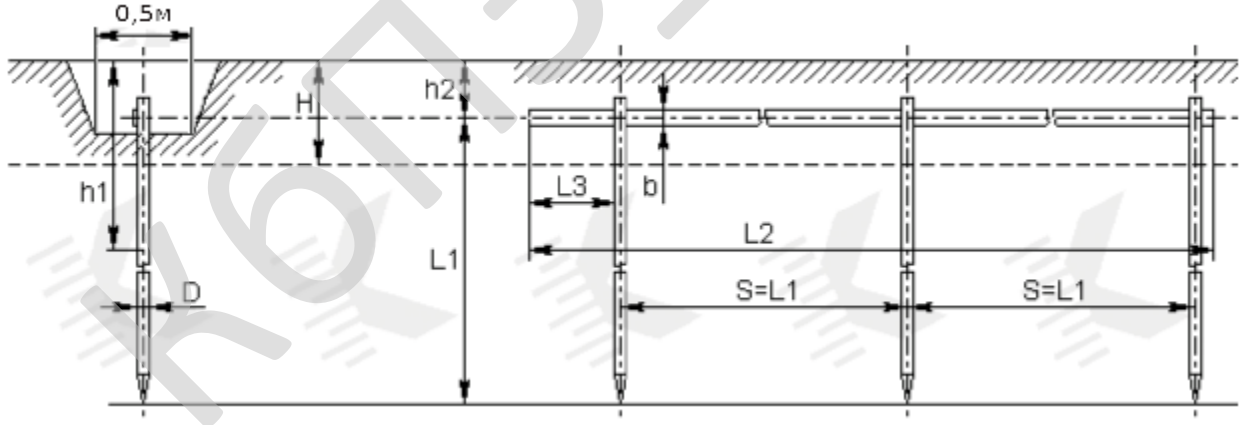

Рисунок 8.1 – Схема штучного заземлення .

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів з питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп'ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи радіально-базисної нейронної мережі для розпізнання образів.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів радіально-базисної нейронної мережі для розпізнання образів.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем радіально-базисної нейронної мережі для розпізнання образів.

– Досліджена система радіально-базисної нейронної мережі для розпізнання образів.

– На основі отриманих результатів досліджень створена програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання радіально-базисної нейронної мережі для розпізнання образів.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10.4.1. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Діффі-Хеллмана.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 18735 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,16 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Окунєв М.Ю. Дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023. Moody and C. J. Darken, "Fast learning in networks of locally tuned processing units, " Neural Computation, 1, 281—294 (1989). Also see Radial basis function networks according to Moody and Darken

2. T. Poggio and F. Girosi, "Networks for approximation and learning, " Proc. IEEE 78(9), 1484—1487 (1990).

3. Roger D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, and S. Qian, ?Function approximation and time series prediction with neural networks,? Proceedings of the International Joint Conference on Neural Networks, June 17-21, p. I-649 (1990).

4. Martin D. Buhmann Radial Basis Functions: Theory and Implementations. – Cambridge University, 2003. – ISBN ISBN 0-521-63338-9.

5. Yee, Paul V. and Haykin, Simon Regularized Radial Basis Function Networks: Theory and Applications. – John Wiley, 2001. – ISBN ISBN 0-471-35349-3.

6. John R. Davies, Stephen V. Coggeshall, Roger D. Jones, and Daniel Schutzer, "Intelligent Security Systems, " in Freedman, Roy S., Flein, Robert A., and Lederman, Jess, Editors Artificial Intelligence in the Capital Markets. – Chicago: Irwin, 1995. – ISBN ISBN 1-55738-811-3.

7. Simon Haykin Neural Networks: A Comprehensive Foundation. – 2nd edition. – Upper Saddle River, NJ: Prentice Hall, 1999. – ISBN ISBN 0-13-908385-5.

8. S. Chen, C. F. N. Cowan, and P. M. Grant, «Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks», IEEE Transactions on Neural Networks, Vol 2, No 2 (Mar) 1991.

9. Aggarwal C.C. Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition. – Springer, 2018. – 520 p

10. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 c.

11. Brink H., Richards J., Fetherolf M. Real-World Machine Learning. – Manning, 2016. – 474 p.

12. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

13. Fenner M. Machine Learning with Python for Everyone (Addison-Wesley Data & Analytics Series) 1st Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 586 p.

14. Foreman J.W. Data Smart: Using Data Science to Transform Information into Insight 1st Edition. – Wiley, 2013. – 432 p.

15. Hurbans R. Grokking Artificial Intelligence Algorithms. – Manning, 2020. – 631 p.

16. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.

17. Kotu V., Deshpande B. Data Science: Concepts and Practice. – Elsevier Science, 2018. – 953 p.

18. Knowledge Base A Complete Guide – 2021 Edition // The Art of Service – Knowledge Base Publishing, 2020. – 306 p.

19. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

20. Mattmann C. Machine Learning with TensorFlow, Second Edition. – Manning, 2020. – 1124 p.

21. Mueller J.P., Massaron L. Machine Learning For Dummies. – Wiley, 2016. – 714 p.

22. Teofili T. Deep Learning for Search. – Manning, 2019. – 695 p.

23. Rungta K. TensorFlow in 1 Day: Make your own Neural Network. – Publishdrive, 2019. – 587 p.

24. Weidman S. Deep Learning from Scratch: Building with Python from First Principles. – O'Reilly. 2019. – 252 p.

25. Rajasekaran S., Vijayalakshmi Pai G.A. Neural networks, fuzzy logic, and genetic algorithms: synthesis and applications (with cd-rom) Kindle Edition. – PHI, 2013. – 628 p.

26. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

27. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022, pp. 1-12.

28. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

29. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.

30. Kuznetsov, A., Oleshko, I., Chernov, K., Bagmut, M., Smirnova, T. «Biometric authentication using convolutional neural networks». Lecture Notes in Networks and Systems. Volume 152, 2021, Pages 85-98.

31. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

32. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». CEUR Workshop Proceedings Volume 3101, 2021, Pages 192-207.

33. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

34. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

35. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

36. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

37. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

38. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

39. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties».

International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

40. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

41. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

42. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». CEUR Workshop Proceedings, Vol 2588, P. 215-227, 2019.

43. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

44. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

45. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

46. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising

Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

47. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

48. Smirnov, S., Bulekbaeva, G., Kikvidze, O.G., Lakhno, V., Brzhanov, R., Tabylov, A. «Computer simulation in the MathCAD package of plastic deformation of the deposited layer on the flat surface of the part». Journal of Theoretical and Applied Information Technology Volume 97, Issue 20, 2019, Pages 2467-2484. (Scopus).

49. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

50. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи радіально-базисної нейронної мережі для розпізнання образів.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 35-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи радіально-базисної нейронної мережі для розпізнання образів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Delphi 10.4.1.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинна бути розглянута пожежна безпека.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 107 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 13.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Дрєєв О.М.

Дослідження та програмна реалізація системи радіально-базисної нейронної мережі для розпізнання образів

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 63

Літера: РП

Кропивницький – 2023 року

RadialBasisNeuralNetExtend.pas – навчання мережі

```
unit RadialBasisNeuralNetExtend;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, Neural_network_Comp, ExtCtrls, StdCtrls, Spin, Grids,
Neural_network_Types,
  IniFiles;

const
  FormCaption = 'Навчання мережі';

type
  TfrmRadialBasisNeuralNetExtend = class(TForm)
    PageControl: TPageControl;
    pnlNavigation: TPanel;
    Tab1: TTabSheet;
    btnBack: TButton;
    rgrFileType: TRadioGroup;
    btnNext: TButton;
    btnCancel: TButton;
    Tab2: TTabSheet;
    lblFileName: TLabel;
    btnOpenFile: TButton;
    edtFileName: TEdit;
    OpenDialog: TOpenDialog;
    Tab3: TTabSheet;
    ltbFieldName: TListBox;
    Label2: TLabel;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    edtMin: TEdit;
    Label4: TLabel;
    edtMax: TEdit;
    Label5: TLabel;
    edt: TEdit;
    Tab4: TTabSheet;
    speLayers: TSpinEdit;
    Label6: TLabel;
    stgNeuronsInLayer: TStringGrid;
    Label7: TLabel;
    Tab5: TTabSheet;
    Label8: TLabel;
    tbrAlpha: TTrackBar;
    sttAlpha: TStaticText;
    Label9: TLabel;
    edtMomentum: TEdit;
    Label10: TLabel;
    edtTeachRate: TEdit;
    Tab6: TTabSheet;
    Label11: TLabel;
    Label12: TLabel;
    btnContinueTeach: TButton;
    sttMaxTeachError: TStaticText;
    sttEpochCount: TStaticText;
    GroupBox2: TGroupBox;
    Label13: TLabel;
    edtIdentError: TEdit;
    speEpochCount: TSpinEdit;
    cbxEpoch: TCheckBox;
    cbxMaxTeachError: TCheckBox;
```

```
edtMaxTeachErrorValue: TEdit;
cbxMidTeachError: TCheckBox;
edtMidTeachErrorValue: TEdit;
cbxTeachIdent: TCheckBox;
speTeachIdentValue: TSpinEdit;
btnBeginTeach: TButton;
cbxMaxTestError: TCheckBox;
cbxMidTestError: TCheckBox;
cbxTestIdent: TCheckBox;
edtMaxTestErrorValue: TEdit;
edtMidTestErrorValue: TEdit;
speTestIdentValue: TSpinEdit;
Tab7: TTabSheet;
Label14: TLabel;
stgInput: TStringGrid;
Label15: TLabel;
stgOutput: TStringGrid;
btnCompute: TButton;
Memo1: TMemo;
Label16: TLabel;
Label17: TLabel;
Memo2: TMemo;
Bevel1: TBevel;
Memo3: TMemo;
Label1: TLabel;
sttMaxTestError: TStaticText;
Label18: TLabel;
sttMidTeachError: TStaticText;
Label19: TLabel;
sttMidTestError: TStaticText;
SaveDialog: TSaveDialog;
edtUseForTeach: TEdit;
Label20: TLabel;
btnSave: TButton;
procedure btnCancelClick(Sender: TObject);
procedure btnNextClick(Sender: TObject);
procedure btnBackClick(Sender: TObject);
procedure btnOpenFileClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure ltbFieldNameClick(Sender: TObject);
procedure rdgFieldTypeClick(Sender: TObject);
procedure rdgNormTypeClick(Sender: TObject);
procedure edtAChange(Sender: TObject);
procedure tbrAlphaChange(Sender: TObject);
procedure edtMomentumChange(Sender: TObject);
procedure edtTeachRateChange(Sender: TObject);
procedure RadialBasisNeuralNetExtendedEpochPassed(Sender: TObject);
procedure btnContinueTeachClick(Sender: TObject);
procedure edtIdentErrorChange(Sender: TObject);
procedure cbxEpochClick(Sender: TObject);
procedure speEpochCountChange(Sender: TObject);
procedure cbxMaxTeachErrorClick(Sender: TObject);
procedure edtMaxTeachErrorValueChange(Sender: TObject);
procedure cbxMidTeachErrorClick(Sender: TObject);
procedure edtMidTeachErrorValueChange(Sender: TObject);
procedure cbxTeachIdentClick(Sender: TObject);
procedure speTeachIdentValueChange(Sender: TObject);
procedure cbxMaxTestErrorClick(Sender: TObject);
procedure edtMaxTestErrorValueChange(Sender: TObject);
procedure cbxMidTestErrorClick(Sender: TObject);
procedure edtMidTestErrorValueChange(Sender: TObject);
procedure cbxTestIdentClick(Sender: TObject);
procedure speTestIdentValueChange(Sender: TObject);
procedure RadialBasisNeuralNetExtendedAfterTeach(Sender: TObject);
procedure btnBeginTeachClick(Sender: TObject);
procedure btnComputeClick(Sender: TObject);
procedure speLayersChange(Sender: TObject);
procedure btnSaveClick(Sender: TObject);
procedure edtUseForTeachChange(Sender: TObject);
```

```
  private
    { Private declarations }
    Teach: boolean;
    NotSaved: boolean;
    procedure ChangePage;
    procedure OpenFile;
    procedure LoadFile;
    procedure Tune;
    procedure CreateNet;
    procedure RunTeach;
    procedure TestNet;
  public
    { Public declarations }
  end;

var
  frmRadialBasisNeuralNetExtend: TfrmRadialBasisNeuralNetExtend;
implementation

{$R *.DFM}

// Запуск програми
procedure TfrmRadialBasisNeuralNetExtend.FormActivate(Sender: TObject);
begin
  Caption := FormCaption;
  PageControl.ActivePage := PageControl.Pages[0];
  Teach := false;
  NotSaved := false;
end;

// Вихід із програми
procedure TfrmRadialBasisNeuralNetExtend.btnCancelClick(Sender: TObject);
begin
  if NotSaved then
    if MessageDlg('Є незбережені дані. Зберегти?',mtConfirmation, [mbYes, mbNo],
0) = mrYes then
      btnSave.Click;
  Close;
end;

// Натискання кнопки "Вперед"
procedure TfrmRadialBasisNeuralNetExtend.btnNextClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, true, false);
    ChangePage;
    if ActivePage.PageIndex = PageCount - 1 then
      btnNext.Enabled := false
    else
      btnNext.Enabled := true;
    btnBack.Enabled := true;
  end;
end;

// Натискання кнопки "Назад"
procedure TfrmRadialBasisNeuralNetExtend.btnBackClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, false, false);
    if ActivePage.PageIndex = 0 then
      btnBack.Enabled := false
    else
      btnBack.Enabled := true;
    btnNext.Enabled := true;
  end;
end;
```

```
// Дія на зміну сторінки
procedure TfrmRadialBasisNeuralNetExtend.ChangePage;
begin
  case PageControl.ActivePage.PageIndex of
    1: OpenFile;
    2: LoadFile;
    3: Tune;
    4: CreateNet;
    6: TestNet;
  end;
end;

// Вибрати файл - джерело даних
procedure TfrmRadialBasisNeuralNetExtend.OpenFile;
begin
  if rgrFileType.ItemIndex = 0 then
  begin
    OpenDialog.Filter := 'NNW files (*.nnw)|*.nnw';
    lblFileName.Caption := 'Виберіть nnw-файл';
  end
  else
  begin
    OpenDialog.Filter := 'Text files (*.txt)|*.txt';
    lblFileName.Caption := 'Виберіть txt-файл';
  end;
end;

// Вибір файлу в діалоговому вікні
procedure TfrmRadialBasisNeuralNetExtend.btnOpenFileClick(Sender: TObject);
begin
  OpenDialog.Execute;
  Caption := FormCaption + ' - ' + ExtractFileName(OpenDialog.FileName);
  edtFileName.Text := OpenDialog.FileName;
end;

// Завантажити в компонент обраний файл
procedure TfrmRadialBasisNeuralNetExtend.LoadFile;
var
  i: integer;
begin
  try
    if rgrFileType.ItemIndex = 0 then
      RadialBasisNeuralNetExtended.FileName := edtFileName.Text   // nnw-файл
    else
    begin
      RadialBasisNeuralNetExtended.SourceFileName := edtFileName.Text; //
текстовий файл
      RadialBasisNeuralNetExtended.LoadDataFrom; // завантажує дані з текстового
файлу
      // конфігурація нейронної мережі за замовчуванням
      RadialBasisNeuralNetExtended.AddLayer(2);
      RadialBasisNeuralNetExtended.AddLayer(3);
      RadialBasisNeuralNetExtended.AddLayer(1);
    end;
  except
    raise Exception.Create('Помилка при відкритті файлу');
  end;
  RadialBasisNeuralNetExtended.Init;  // Ініціалізація мережі
  // Формування списку полів для StringList-А
  ltbFieldName.Clear;
  for i := 0 to RadialBasisNeuralNetExtended.AvailableFieldsCount - 1 do
    ltbFieldName.Items.Add(RadialBasisNeuralNetExtended.Fields[i].Name);
  ltbFieldName.ItemIndex := 0;
  ltbFieldNameClick(Self);
end;

// Настроювання параметрів мережі
procedure TfrmRadialBasisNeuralNetExtend.Tune;
var
```

```
  i: integer;
begin
  speLayers.Value := RadialBasisNeuralNetExtended.LayerCount - 2;
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  stgNeuronsInLayer.Cells[0, 0] := '№ шаруючи';
  stgNeuronsInLayer.Cells[1, 0] := 'Нейронів';
  for i := 0 to speLayers.Value - 1 do
  begin
    stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
    stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(RadialBasisNeuralNetExtended.Layers[i + 1].NeuronCount);
  end;

  tbrAlpha.Position := trunc(RadialBasisNeuralNetExtended.Alpha * 100);
  edtMomentum.Text := FloatToStr(RadialBasisNeuralNetExtended.Momentum);
  edtTeachRate.Text := FloatToStr(RadialBasisNeuralNetExtended.TeachRate);
  edtIdentError.Text := FloatToStr(RadialBasisNeuralNetExtended.IdentError);
  edtUseForTeach.Text := FloatToStr(RadialBasisNeuralNetExtended.UseForTeach);

  cbxEpoch.Checked := RadialBasisNeuralNetExtended.Epoch;
  speEpochCount.Text := IntToStr(RadialBasisNeuralNetExtended.EpochCount);

  cbxMaxTeachError.Checked := RadialBasisNeuralNetExtended.MaxTeachError;
  edtMaxTeachErrorValue.Text :=
FloatToStr(RadialBasisNeuralNetExtended.MaxTeachErrorValue);

  cbxMidTeachError.Checked := RadialBasisNeuralNetExtended.MidTeachError;
  edtMidTeachErrorValue.Text :=
FloatToStr(RadialBasisNeuralNetExtended.MidTeachErrorValue);

  cbxTeachIdent.Checked := RadialBasisNeuralNetExtended.TeachIdent;
  speTeachIdentValue.Value := RadialBasisNeuralNetExtended.TeachIdentCount;

  cbxMaxTestError.Checked := RadialBasisNeuralNetExtended.MaxTestError;
  edtMaxTestErrorValue.Text :=
FloatToStr(RadialBasisNeuralNetExtended.MaxTestErrorValue);

  cbxMidTestError.Checked := RadialBasisNeuralNetExtended.MidTestError;
  edtMidTestErrorValue.Text :=
FloatToStr(RadialBasisNeuralNetExtended.MidTestErrorValue);

  cbxTestIdent.Checked := RadialBasisNeuralNetExtended.TestIdent;
  speTestIdentValue.Value := RadialBasisNeuralNetExtended.TeachIdentCount;
end;

// Відображення інформації про обране поле (тип поля, нормалізація та інше)
procedure TfrmRadialBasisNeuralNetExtend.ltbFieldNameClick(Sender: TObject);
begin
  // Тип поля - вхідне, вихідне, не використовувати
  rdgFieldType.ItemIndex :=
RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  // Тип нормалізації
  rdgNormType.ItemIndex :=
RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  // Параметр нормалізації
  edt.Text :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  // Мінінум та максимум (для лінійної нормалізації)
  edtMin.Text :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin)
;
  edtMax.Text :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax)
;
end;

// Змінити тип поля
procedure TfrmRadialBasisNeuralNetExtend.rdgFieldTypeClick(Sender: TObject);
begin
```

```
  RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
end;

// Змінити тип нормалізації
procedure TfrmRadialBasisNeuralNetExtend.rdgNormTypeClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
end;

// Змінити параметр нормалізації
procedure TfrmRadialBasisNeuralNetExtend.edtAChange(Sender: TObject);
begin
 RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha :=
StrToFloat(edt.Text);
end;

// Змінити параметр Alpha мережі
procedure TfrmRadialBasisNeuralNetExtend.tbrAlphaChange(Sender: TObject);
begin
  sttAlpha.Caption := FloatToStr(tbrAlpha.Position / 100);
end;

// Зміна кількості схованих шарів
procedure TfrmRadialBasisNeuralNetExtend.speLayersChange(Sender: TObject);
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
end;

// Створення мережі обраної топології
procedure TfrmRadialBasisNeuralNetExtend.CreateNet;
var
  i: integer;
  xInput, xOutput: integer;
begin
  // Змінюється тільки кількість нейронів у схованих шарах,
  // Кількість нейронів у вхідному й вихідному шарі залежить від
  // типів полів
  with RadialBasisNeuralNetExtended do
  begin
    xInput := InputFieldCount;
    xOutput := OutputFieldCount;
    ResetLayers;
    AddLayer(xInput);
    for i := 0 to speLayers.Value - 1 do
      AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i + 1]));
    AddLayer(xOutput);
  end;
end;

// Змінити момент мережі
procedure TfrmRadialBasisNeuralNetExtend.edtMomentumChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.Momentum := StrToFloat(edtMomentum.Text);
end;

// Змінити швидкість навчання мережі
procedure TfrmRadialBasisNeuralNetExtend.edtTeachRateChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.TeachRate := StrToFloat(edtTeachRate.Text);
end;

// Дія по проходженню однієї епохи навчання
procedure
TfrmRadialBasisNeuralNetExtend.RadialBasisNeuralNetExtendedEpochPassed(Sender:
TObject);
begin
  sttMaxTestError.Caption := '';
```

```
  sttMidTestError.Caption := '';
  with RadialBasisNeuralNetExtended do
  begin
    sttMaxTeachError.Caption := FloatToStr(MaxTeachResidual); // Показати макс.
помилку на навчальній множині
    sttMidTeachError.Caption := FloatToStr(MidTeachResidual); // Показати серед.
помилку на навчальній множині
    if RadialBasisNeuralNetExtended.UseForTeach <> 100 then
    begin
      sttMaxTestError.Caption := FloatToStr(MaxTestResidual); // Показати макс.
помилку на тестовій множині
      sttMidTestError.Caption := FloatToStr(MidTestResidual); // Показати сред.
помилку на тестовій множині
    end;
    sttEpochCount.Caption := FloatToStr(EpochCurrent); // Показати номер
поточної епохи
  end;
  Application.ProcessMessages; // Дати можливість Windows перемалювати форму
end;

// Натискання кнопки "Продовжити навчання"
procedure TfrmRadialBasisNeuralNetExtend.btnContinueTeachClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.ContinueTeach := true; // Скинути прапор -
"Продовжити навчання"
  RunTeach;  // Запуск
end;

// Натискання кнопки "Навчити"
procedure TfrmRadialBasisNeuralNetExtend.btnBeginTeachClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.ContinueTeach := false; // Скинути прапор -
"Почати навчання знову"
  RunTeach;
end;

procedure TfrmRadialBasisNeuralNetExtend.edtIdentErrorChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.IdentError := StrToFloat(edtIdentError.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.edtUseForTeachChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.UseForTeach := StrToInt(edtUseForTeach.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxEpochClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.Epoch := cbxEpoch.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.speEpochCountChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.EpochCount := StrToInt(speEpochCount.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxMaxTeachErrorClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.MaxTeachError := cbxMaxTeachError.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.edtMaxTeachErrorValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.MaxTeachErrorValue :=
StrToFloat(edtMaxTeachErrorValue.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxMidTeachErrorClick(Sender: TObject);
```

```
begin
  RadialBasisNeuralNetExtended.MidTeachError := cbxMidTeachError.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.edtMidTeachErrorValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.MidTeachErrorValue :=
StrToFloat(edtMidTeachErrorValue.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxTeachIdentClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.TeachIdent := cbxTeachIdent.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.speTeachIdentValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.TeachIdentCount := speTeachIdentValue.Value;
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxMaxTestErrorClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.MaxTestError := cbxMaxTestError.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.edtMaxTestErrorValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.MaxTestErrorValue :=
StrToFloat(edtMaxTestErrorValue.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxMidTestErrorClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.MidTestError := cbxMidTestError.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.edtMidTestErrorValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.MidTestErrorValue :=
StrToFloat(edtMidTestErrorValue.Text);
end;

procedure TfrmRadialBasisNeuralNetExtend.cbxTestIdentClick(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.TestIdent := cbxTestIdent.Checked;
end;

procedure TfrmRadialBasisNeuralNetExtend.speTestIdentValueChange(Sender:
TObject);
begin
  RadialBasisNeuralNetExtended.TeachIdentCount := speTestIdentValue.Value;
end;

// Зупинка навчання
procedure
TfrmRadialBasisNeuralNetExtend.RadialBasisNeuralNetExtendedAfterTeach(Sender:
TObject);
begin
  btnBack.Enabled := true;
  btnNext.Enabled := true;
  btnCancel.Enabled := true;
  btnContinueTeach.Caption := 'Навчити';
  RadialBasisNeuralNetExtended.StopTeach := true;
end;
```

```
procedure TfrmRadialBasisNeuralNetExtend.RunTeach;
begin
  Teach := not Teach;  // Перемикач стану "вчимося/не вчимося"
  if Teach then
  begin
    btnBeginTeach.Enabled := false;
    btnBack.Enabled := false;
    btnNext.Enabled := false;
    btnCancel.Enabled := false;
    RadialBasisNeuralNetExtended.StopTeach := false;
    btnContinueTeach.Caption := 'Зупинити навчання';
    NotSaved := true;
    RadialBasisNeuralNetExtended.Train;  // Запуск нейромережі на навчання
    btnCompute.Enabled := true;
    btnSave.Enabled := true;
  end
  else
  begin
    btnBeginTeach.Enabled := true;
    btnBack.Enabled := true;
    btnNext.Enabled := true;
    btnCancel.Enabled := true;
    btnContinueTeach.Caption := 'Продовжити навчання';
    RadialBasisNeuralNetExtended.StopTeach := true; // Зупинити навчання
  end;
end;

// Відкриття сторінки - тестування навченої нейромережі
procedure TfrmRadialBasisNeuralNetExtend.TestNet;
var
  i, j: integer;
begin
  stgInput.RowCount := RadialBasisNeuralNetExtended.InputFieldCount + 1;
  stgInput.Cells[0, 0] := 'Поле';
  stgInput.Cells[1, 0] := 'Значення';

  // Проставити імена вхідних полів
  j := 0;
  for i := 0 to RadialBasisNeuralNetExtended.AvailableFieldsCount - 1 do
    if (RadialBasisNeuralNetExtended.Fields[i].KindName = fdInput) then //
Ознака того, що поле вхідне
    begin
      Inc(j);
      stgInput.Cells[0, j] := RadialBasisNeuralNetExtended.Fields[i].Name;
    end;
  stgOutput.RowCount := RadialBasisNeuralNetExtended.OutputFieldCount + 1;
  stgOutput.Cells[0, 0] := 'Поле';
  stgOutput.Cells[1, 0] := 'Значення';

  // Проставити імена вихідних полів
  j := 0;
  for i := 0 to RadialBasisNeuralNetExtended.AvailableFieldsCount - 1 do
    if (RadialBasisNeuralNetExtended.Fields[i].KindName = fdOutput) then  //
Ознака того, що поле вихідне
    begin
      Inc(j);
      stgOutput.Cells[0, j] := RadialBasisNeuralNetExtended.Fields[i].Name;
    end;
end;

// Натискання кнопки "Обчислити"
procedure TfrmRadialBasisNeuralNetExtend.btnComputeClick(Sender: TObject);
var
  xVectorFloat: TVectorFloat;
  i: integer;
begin
  // Створити вектор, що будемо подавати на вхід
  // довжиною, рівною кількості нейронів на вхідному шарі
  SetLength(xVectorFloat, RadialBasisNeuralNetExtended.InputFieldCount);
```

```
  // Заповнити значення елементів вектора
  for i := 0 to RadialBasisNeuralNetExtended.InputFieldCount - 1 do
    xVectorFloat[i] := StrToFloat(stgInput.Cells[1, i + 1]);
  // Подати на вхід нейромережі. Результати будуть у вихідному шарі нейромережі
  RadialBasisNeuralNetExtended.ComputeUnPrepData(xVectorFloat);
  // Відобразити отримані результати
  for i := 0 to RadialBasisNeuralNetExtended.OutputFieldCount - 1 do
    stgOutput.Cells[1, i + 1] :=
FloatToStr(RadialBasisNeuralNetExtended.Output[i]);
  // Знищити вектор
  SetLength(xVectorFloat, 0);
  xVectorFloat := nil;
end;

// Зберегти навчену нейромережу
procedure TfrmRadialBasisNeuralNetExtend.btnSaveClick(Sender: TObject);
begin
  SaveDialog.InitialDir :=
ExtractFilePath(RadialBasisNeuralNetExtended.FileName);
  SaveDialog.FileName := ExtractFileName(RadialBasisNeuralNetExtended.FileName);
  if SaveDialog.Execute then
  begin
    RadialBasisNeuralNetExtended.NnwFile :=
TIniFile.Create(SaveDialog.FileName);
    RadialBasisNeuralNetExtended.SavePhase1;
    RadialBasisNeuralNetExtended.SavePhase2;
    RadialBasisNeuralNetExtended.SavePhase4;
    RadialBasisNeuralNetExtended.SaveNetwork;
    NotSaved := false;
  end;
end;

end.
```

Hopf.pas - Блок формування та розпізнавання образів

```
unit Hopf;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, Neural_network_Comp, Neural_network_Types, Db, DBTables, ExtCtrls,
DBCtrls, StdCtrls,
  ToolWin, ComCtrls;

type
  TForm1 = class(TForm)
    Table: TTable;
    btnExecute: TButton;
    DBNavigator: TDBNavigator;
    DataSource: TDataSource;
    btnEdit: TButton;
    stgDatabase: TStringGrid;
    stgInput: TStringGrid;
    stgOutput: TStringGrid;
    RadialBasisNeuralNetHopf: TRadialBasisNeuralNetHopf;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    Bevel: TBevel;
    TableLETTERS: TStringField;
    dbMain: TDatabase;
    procedure DataSourceDataChange(Sender: TObject; Field: TField);
    procedure FormActivate(Sender: TObject);
    procedure GridClick(Sender: TObject);
    procedure btnEditClick(Sender: TObject);
    procedure btnExecuteClick(Sender: TObject);
    procedure GridDrawCell(Sender: TObject; ACol, ARow: Integer;
      Rect: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
  public
    { Public declarations }
    procedure AddPattern(Value: string);
    procedure Clear(Grid: TStringGrid);
    procedure Init;
    procedure ShowMatrix(Grid: TStringGrid; Value: string);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

// Показати вектор у вигляді сітки
procedure TForm1.ShowMatrix(Grid: TStringGrid; Value: string);
var
  i, j: integer;
begin
  Clear(Grid);
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
    begin
      try
        if Value[i * Grid.RowCount + j + 1] = '1' then
          Grid.Cells[i, j] := '1'
        else
          Grid.Cells[i, j] := ' '
      except
        Grid.Cells[i, j] := ' '
```

```
      end;
    end;
end;

// Очистити сітку
procedure TForm1.Clear(Grid: TStringGrid);
var
  i, j: integer;
begin
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
      Grid.Cells[i, j] := ' ';
end;

// Показати символ з таблиці
procedure TForm1.DataSourceDataChange(Sender: TObject; Field: TField);
begin
  ShowMatrix(stgDatabase, TableLETTERS.Value);
end;

// Ініціалізація мережі значеннями з таблиці
procedure TForm1.Init;
begin
  // Очистити мережу від зразків
  RadialBasisNeuralNetHopf.ResetPatterns;

  // Додати зразки з таблиці до мережі
  Table.First;
  while not Table.Eof do
  begin
    AddPattern(TableLETTERS.AsString);
    Table.Next;
  end;
  Table.First;

  // Ініціалізувати ваги
  RadialBasisNeuralNetHopf.InitWeights;
  // Мережа підготовлена до розпізнавання
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  Clear(stgDatabase);
  Clear(stgInput);
  Clear(stgOutput);
  Init;
end;

procedure TForm1.GridClick(Sender: TObject);
begin
  with Sender as TStringGrid do
    if Cells[Col, Row] = '1' then
      Cells[Col, Row] := ' '
    else
      Cells[Col, Row] := '1'
end;

// Додавання нового образа до мережі
procedure TForm1.AddPattern(Value: string);
var
  i: integer;
  xVector: TVectorInt;
begin
  SetLength(xVector, stgDatabase.RowCount * stgDatabase.ColCount);

  // Перетворення символьного рядка у вектор
  for i := 1 to stgDatabase.RowCount * stgDatabase.ColCount do
    try
      if TableLETTERS.AsString[i] = '1' then
```

```
        xVector[i - 1] := 1
      else
        xVector[i - 1] := -1;
    except
      xVector[i - 1] := -1;
    end;


  RadialBasisNeuralNetHopf.AddPattern(xVector);
end;

procedure TForm1.btnEditClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  xString := '';
  for i := 0 to stgDatabase.ColCount - 1 do
    for j := 0 to stgDatabase.RowCount - 1 do
      if stgDatabase.Cells[i, j] = '1' then
        xString := xString + '1'
      else
        xString := xString + ' ';
  Table.Edit;
  TableLETTERS.AsString := xString;
  Table.Post;
end;

// Розпізнавання символу
procedure TForm1.btnExecuteClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  // Подаємо сигнали на вихід мережі
  for i := 0 to stgInput.ColCount - 1 do
    for j := 0 to stgInput.RowCount - 1 do
      if stgInput.Cells[i, j] = '1' then
        RadialBasisNeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount +
j].Output := 1
      else
        RadialBasisNeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount +
j].Output := -1;

  // Запуск процесу розпізнавання
  RadialBasisNeuralNetHopf.Calc;

  // Перетворення виходів мережі до рядка
  xString := '';
  for i := 1 to stgOutput.RowCount * stgOutput.ColCount do
    if RadialBasisNeuralNetHopf.Layers[1].Neurons[i - 1].Output = 1 then
      xString := xString + '1'
    else
      xString := xString + ' ';

  // Відобразити результат
  ShowMatrix(stgOutput, xString);
end;

procedure TForm1.GridDrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  with Sender as TStringGrid do
  begin
    Canvas.Brush.Color := clBlack;
    if Cells[ACol,ARow] <> ' ' then
      Canvas.FillRect(Rect)
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  dbMain.Params.Values['Path'] := ExtractFilePath(Application.ExeName);
  dbMain.Open;
  Table.Open;

end;

end.
```

Neural_network_Editor.pas – розробка нейронних мереж

```
unit Neural_network_Editor;

interface

uses
  Classes,
  DesignIntf, DesignEditors,
  Neural_network_Comp, Neural_network_EditorForm,
  SysUtils, Dialogs, Controls, Neural_network_EditorFieldsForm;

type
  TNeuronsInLayerFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

  TFileNameFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure Edit; override;
  end;

  TOptionsFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

procedure Register;

implementation

function TOptionsFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TOptionsFieldsProperty.Edit;
var
  i: integer;
  xRadialBasisNeuralNetExtended: TRadialBasisNeuralNetExtended;
  frmNeuroFields: TfrmNeuroFields;
  xCurrent: integer;
begin
  frmNeuroFields := TfrmNeuroFields.Create(nil);
  try
    with frmNeuroFields do
    begin
      xRadialBasisNeuralNetExtended := (GetComponent(0) as
TRadialBasisNeuralNetExtended);
      for i := 0 to xRadialBasisNeuralNetExtended.AvailableFieldsCount - 1 do
        ltbFieldName.Items.Add(xRadialBasisNeuralNetExtended.Fields[i].Name);
      xCurrent := 0;
      rdgFieldType.ItemIndex :=
xRadialBasisNeuralNetExtended.Fields[xCurrent].Kind;
      rdgNormType.ItemIndex :=
xRadialBasisNeuralNetExtended.Fields[xCurrent].NormType;
      edtAlpha.Text :=
FloatToStr(xRadialBasisNeuralNetExtended.Fields[xCurrent].Alpha);
      sttMin.Caption :=
FloatToStr(xRadialBasisNeuralNetExtended.Fields[xCurrent].ValueMin);
```

```
      sttMax.Caption :=
FloatToStr(xRadialBasisNeuralNetExtended.Fields[xCurrent].ValueMax);
      RadialBasisNeuralNetExtended := xRadialBasisNeuralNetExtended;
      ShowModal;
    end;
  except
    frmNeuroFields.Free;
  end;
end;

function TOptionsFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TRadialBasisNeuralNetExtended) do
    if AvailableFieldsCount > 1 then
    begin
      for i := 0 to AvailableFieldsCount - 1 do
        Result := Result + Fields[i].Name + ',';
      Result[Length(Result)] := ']'
    end
    else
      Result[Length(Result) + 1] := ']';
end;

function TNeuronsInLayerFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

function TNeuronsInLayerFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TRadialBasisNeuralNetBP) do
    if LayerCount > 0 then
    begin
      for i := 0 to LayerCount - 1 do
        Result := Result + IntToStr(LayersBP[i].NeuronCount) + ',';
      Result[Length(Result)] := ']';
    end
    else
      Result[Length(Result) + 1] := ']';
end;

procedure TNeuronsInLayerFieldsProperty.Edit;
var
  i: integer;
  xPreviousCount: integer;
  xRadialBasisNeuralNetBP: TRadialBasisNeuralNetBP;
  frmNeuronsInLayer: TfrmNeuronsInLayer;
  xChangesMade: boolean;
begin
  xChangesMade := False;
  frmNeuronsInLayer := TfrmNeuronsInLayer.Create(nil);
  try
    with frmNeuronsInLayer do
    begin
      xRadialBasisNeuralNetBP := (GetComponent(0) as TRadialBasisNeuralNetBP);
      speLayers.Value := xRadialBasisNeuralNetBP.LayerCount;
      stgNeuronsInLayer.RowCount := xRadialBasisNeuralNetBP.LayerCount + 1;
      for i := 0 to xRadialBasisNeuralNetBP.LayerCount - 1 do
      begin
        stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
```

```
        stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(xRadialBasisNeuralNetBP.LayersBP[i].NeuronCount);
      end;
      if ShowModal = mrOk then
      begin
        xRadialBasisNeuralNetBP.ResetLayers;
        if speLayers.Value = 0 then
        begin
          xRadialBasisNeuralNetBP.ResetLayers;
          Exit;
        end;
        if xRadialBasisNeuralNetBP.LayerCount <> speLayers.Value then
          xChangesMade := True
        else
          for i := 0 to xRadialBasisNeuralNetBP.LayerCount - 1 do
            if xRadialBasisNeuralNetBP.LayersBP[i].NeuronCount <>
StrToInt(stgNeuronsInLayer.Cells[1, i + 1]) then
            begin
              xChangesMade := True;
              Break;
            end;
        if xChangesMade then
        begin
          if xRadialBasisNeuralNetBP.LayerCount = speLayers.Value then
            for i := 0 to speLayers.Value - 1 do
              xRadialBasisNeuralNetBP.NeuronsInLayer[i] :=
stgNeuronsInLayer.Cells[1, i + 1]
          else
            if xRadialBasisNeuralNetBP.LayerCount < speLayers.Value then
            begin
              for i := 0 to xRadialBasisNeuralNetBP.LayerCount - 1 do
                xRadialBasisNeuralNetBP.NeuronsInLayer[i] :=
stgNeuronsInLayer.Cells[1, i + 1];
              for i := xRadialBasisNeuralNetBP.LayerCount to speLayers.Value - 1
do

xRadialBasisNeuralNetBP.AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i + 1]));
            end
            else
            begin
              if speLayers.Value > 0 then
                for i := 0 to speLayers.Value - 1 do
                  xRadialBasisNeuralNetBP.NeuronsInLayer[i] :=
stgNeuronsInLayer.Cells[1, i + 1];
              xPreviousCount := xRadialBasisNeuralNetBP.LayerCount -
speLayers.Value;
              for i := 1 to xPreviousCount do

xRadialBasisNeuralNetBP.DeleteLayer(xRadialBasisNeuralNetBP.LayerCount - 1);
            end;
          if not xRadialBasisNeuralNetBP.AutoInit then
            xRadialBasisNeuralNetBP.AutoInit := True;
        end;
      end;
    end;
  except
    frmNeuronsInLayer.Free;
  end;
end;

function TFileNameFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TFileNameFieldsProperty.Edit;
var
  xOpenDialog: TOpenDialog;
begin
```

```
  xOpenDialog := TOpenDialog.Create(nil);
  if UpperCase(GetName) = 'FILENAME' then
    xOpenDialog.Filter := 'Нейронна мережа(*.nnw)|*.nnw|всі файли (*.*)|*.*';
  if UpperCase(GetName) = 'SOURCEFILENAME' then
    xOpenDialog.Filter := 'Текстові файли (*.txt)|*.txt|всі файли (*.*)|*.*';

  if xOpenDialog.Execute then
  begin
    if UpperCase(GetName) = 'FILENAME' then
      (GetComponent(0) as TRadialBasisNeuralNetExtended).FileName :=
xOpenDialog.FileName;
    if UpperCase(GetName) = 'SOURCEFILENAME' then
      (GetComponent(0) as TRadialBasisNeuralNetExtended).SourceFileName :=
xOpenDialog.FileName;
  end;
  xOpenDialog.Free;
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(TStrings), TRadialBasisNeuralNetBP,
'NeuronsInLayer', TNeuronsInLayerFieldsProperty);
  RegisterPropertyEditor(TypeInfo(TFileName), TRadialBasisNeuralNetExtended, '',
TFileNameFieldsProperty);
  RegisterPropertyEditor(TypeInfo(string), TRadialBasisNeuralNetExtended,
'Options', TOptionsFieldsProperty);
end;

end.
```

Neural_network_EditorForm.pas – редактор нейронних мереж

```
unit Neural_network_EditorForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, StdCtrls, ExtCtrls, Neural_network_Comp, Spin, Neural_network_Types;

type
  TfrmNeuronsInLayer = class(TForm)
    Bevel1: TBevel;
    btnOk: TButton;
    btnCancel: TButton;
    stgNeuronsInLayer: TStringGrid;
    Label1: TLabel;
    speLayers: TSpinEdit;
    procedure stgNeuronsInLayerGetEditMask(Sender: TObject; ACol,
      ARow: Integer; var Value: String);
    procedure stgNeuronsInLayerSetEditText(Sender: TObject; ACol,
      ARow: Integer; const Value: String);
    procedure speLayersChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmNeuronsInLayer: TfrmNeuronsInLayer;

implementation

{$R *.DFM}

procedure TfrmNeuronsInLayer.stgNeuronsInLayerGetEditMask(Sender: TObject;
  ACol, ARow: Integer; var Value: String);
begin
  Value := '0000';
end;

procedure TfrmNeuronsInLayer.stgNeuronsInLayerSetEditText(Sender: TObject;
  ACol, ARow: Integer; const Value: String);
begin
{  with stgNeuronsInLayer do
    try
      Cells[ACol, ARow] := IntToStr(StrToInt(trim(Value)));
    except
      Cells[ACol, ARow] := IntToStr(DefaultNeuronCount);
    end;}
end;

procedure TfrmNeuronsInLayer.speLayersChange(Sender: TObject);
var
  i: integer;
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  for i := 1 to stgNeuronsInLayer.RowCount do
    if trim(stgNeuronsInLayer.Cells[1, i]) = '' then
    begin
      stgNeuronsInLayer.Cells[0, i] := IntToStr(i);
      stgNeuronsInLayer.Cells[1, i] := IntToStr(DefaultNeuronCount);
    end;
end;

procedure TfrmNeuronsInLayer.FormCreate(Sender: TObject);
```

```
begin
  stgNeuronsInLayer.Cells[0,0] := '# шару';
  stgNeuronsInLayer.Cells[1,0] := 'нейронів';
end;

end.
```

Neural_network_EditorFieldsForm.pas – редактор властивостей полів нейроної мережі

```
unit Neural_network_EditorFieldsForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Neural_network_Comp;

type
  TfrmNeuroFields = class(TForm)
    ltbFieldName: TListBox;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    sttMin: TStaticText;
    sttMax: TStaticText;
    edtAlpha: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    Bevel2: TBevel;
    Label4: TLabel;
    procedure ltbFieldNameClick(Sender: TObject);
    procedure rdgFieldTypeClick(Sender: TObject);
    procedure rdgNormTypeClick(Sender: TObject);
    procedure edtAlphaChange(Sender: TObject);
  private
    { Private declarations }
  public
    RadialBasisNeuralNetExtended: TRadialBasisNeuralNetExtended;
    { Public declarations }
  end;

var
  frmNeuroFields: TfrmNeuroFields;

implementation

{$R *.DFM}

procedure TfrmNeuroFields.ltbFieldNameClick(Sender: TObject);
begin
  rdgFieldType.ItemIndex :=
RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  rdgNormType.ItemIndex :=
RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  edtAlpha.Text :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  sttMin.Caption :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin)
;
  sttMax.Caption :=
FloatToStr(RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax)
;
end;

procedure TfrmNeuroFields.rdgFieldTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
```

```
    RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        RadialBasisNeuralNetExtended.Fields[i].Kind := rdgFieldType.ItemIndex;
end;

procedure TfrmNeuroFields.rdgNormTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        RadialBasisNeuralNetExtended.Fields[i].NormType :=
rdgNormType.ItemIndex;
end;

procedure TfrmNeuroFields.edtAlphaChange(Sender: TObject);
begin
  RadialBasisNeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha :=
StrToFloat(edtAlpha.Text);
end;

end.
```

Neural_network_Types.pas – ініціалізація базових констант

```
unit Neural_network_Types;

interface

const

  DefaultAlpha = 1;               // Параметр активаційної функції
  DefaultEpochCount = 10000;      // Кількість етапів для навчання
  DefaultErrorValue = 0.05;       // Помилка за замовчуванням для всіх видів
  DefaultHopfLayerCount = 2;      // Кількість шарів у мережі Хопфилда
  DefaultLayerCount = 0;          // Мінімальна кількість шарів у мережі back-
propagation
  DefaultMaxIterCount = 10;       // Максимальна кількість ітерацій в алгоритмі
Хопфилда
  DefaultMomentum = 0.9;          // Імпульс - момент
  DefaultNeuronCount = 0;         // Кількість нейронів у прихованому шарі за
замовчуванням
  DefaultPatternCount = 0;        // Кількість прикладів
  DefaultTeachRate = 0.1;         // Швидкість навчання
  DefaultTeachIdentCount = 100;   // Необхідний відсоток розпізнаних прикладів з
навчальної множини
  DefaultTestIdentCount = 100;    // Необхідний відсоток розпізнаних прикладів з
тестової множини
  DefaultUseForTeach = 100;       // Відсоток прикладів використовуваних як
навчальна множина
  DefaultDeltaBarAcceleratingConst = 0.095;
  DefaultDeltaBarDecFactor = 0.85;
  DefaultRPropInitValue = 0.1;
  DefaultRPropMaxStepSize = 50;
  DefaultRPropMinStepSize = 1 E-6;
  DefaultRPropDecFactor = 0.5;
  DefaultRPropIncFactor = 1.2;
  DefaultSuperSABDecFactor = 0.5;
  DefaultSuperSABIncFactor = 1.05;

  SensorLayer = 0;                // Сенсорний шар
  BiasNeuron = 1;                 // Зсув у мережі back-propagation

  Separators = ['.', ','];
  Letters    = ['a'..'z'];
  Capitals   = ['A'..'Z'];
  DigitChars = ['0'..'9'];
  SpaceChar = #9;

resourcestring

  SFieldNorm = 'Не існує типу нормалізації %d';
  SFieldKind = 'Не існує типу поля %d';
  SFieldIndexRange = 'Неправильно зазначений номер поля %d';
  SInFieldCount = 'Неправильно встановлена кількість вхідних полів';
  SInNeuronCount = 'Неправильно встановлена кількість вхідних нейронів';
  SInVectorCount = 'Неправильно встановлена розмірність вхідного вектора';
  SLayerRangeIndex = 'Неправильно зазначений номер шару %d';
  SNeuronRangeIndex = 'Неправильно зазначений номер нейрона %d';
  SNeuronCount = 'Неправильно зазначена кількість нейронів';
  SOutFieldCount = 'Неправильно встановлена кількість вихідних полів';
  SOutNeuronCount = 'Неправильно встановлена кількість вихідних нейронів';
  SOutVectorCount = 'Неправильно встановлена розмірність вихідного вектора';
  SPatternRangeIndex = 'Вихід за межі масиву прикладів %d';
  SStreamCannotRead = 'Помилка читання з потоку';
  SWeightRangeIndex = 'Неправильно зазначений номер ваги %d';
  SWrongFileName = 'Неправильно зазначене ім'я файлу %s';
  SCannotBeNumber = 'Помилка, вираз %s неможливо привести до числового типу';
  SBPStopCondition = 'Не задана умова зупинки процесу навчання';
```

```
type

  TVectorInt = array of integer;
  TVectorFloat = array of double;
  TVectorString = array of string;
  TMatrixInt = array of array of integer;
  TMatrixFloat = array of array of double;
  TNormalize = (nrmLinear, nrmSigmoid, nrmAuto,  nrmNone,
                nrmLinearOut, nrmAutoOut);
  TNeuroFieldType = (fdInput, fdOutput, fdNone);

implementation


end.
```

Neural_network_Comp.pas – формування бібліотеки шаблонів та дослідження нейронних мереж

```
unit Neural_network_Comp;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, PumpData, Neural_network_Types, IniFiles, Math;

type

  // Класи виключень
  EInOutDimensionError = class(Exception);
  ENeuronCountError  = class(Exception);
  ENeuronNotEqualFieldError  = class(Exception);
  EBPStopCondition  = class(Exception);

  // Процедурні типи
  TActivation = function (Value: double): double of object;

  // Упереджуюче оголошення класів
  TNeuron = class;
  TLayer = class;

  // Базовий клас нейрона

  TNeuron = class(TObject)
  private
    FOutput: double;
    // Вектор ваг
    FWeights: TVectorFloat;
    // Покажчик на шар, у якому перебуває нейрон
    Layer: TLayer;
    function GetWeights(Index: integer): double;
    procedure SetWeights(Index: integer; Value: double);
    procedure SetWeightCount(const Value: integer);
  public
    constructor Create(ALayer: TLayer); virtual;
    destructor Destroy; override;
    // Ініціалізація ваг
    procedure InitWeights;  virtual;
    // Зважена сума
    procedure ComputeOut(const AInputs: TVectorFloat); virtual;
    property Output: double read FOutput write FOutput;
    property WeightCount: integer  write SetWeightCount;
    property Weights[Index: integer]: double read GetWeights write SetWeights;
  end;

  // Клас нейрона для мережі Хопфілда

  TNeuronHopf = class(TNeuron)
  public
    procedure ComputeOut(const AInputs: TVectorFloat); override;
  end;

  // Клас нейрона для мережі

  TNeuronBP = class(TNeuron)
  private
    // Локальна помилка
    FDelta: double;
    // Значення швидкості навчання на попередньому етапі
    FLearningRate: TVectorFloat;
    // Значення частинної похідної на попередньому етапі
    FPrevDerivative: TVectorFloat;
```

```
    // Значення корекції ваги на попередньому етапі
    FPrevUpdate: TVectorFloat;
    // Функція активації
    FOnActivation: TActivation;
    // Похідна функції активації
    FOnActivation: TActivation;
    function GetPrevUpdate(Index: integer): double;
    function GetPrevDerivative(Index: integer): double;
    function GetLearningRate(Index: integer): double;
    function GetPrevUpdateCount: integer;
    procedure SetPrevDerivative(Index: integer; const Value: double);
    procedure SetPrevDerivativeCount(const Value: integer);
    procedure SetDelta(Value: double);
    procedure SetPrevUpdate(Index: integer; Value: double);
    procedure SetPrevUpdateCount(const Value: integer);
    procedure SetLearningRate(Index: integer; const Value: double);
    procedure SetLearningRateCount(const Value: integer);
  public
    destructor Destroy; override;
    procedure ComputeOut(const AInputs: TVectorFloat); override;
    property Delta: double read FDelta write SetDelta;
    property LearningRate[Index: integer]: double read GetLearningRate write
SetLearningRate; //
    property LearningRateCount: integer write SetLearningRateCount;
    property PrevDerivativeCount: integer write SetPrevDerivativeCount;
    property PrevDerivative[Index: integer]: double read GetPrevDerivative write
SetPrevDerivative; //
    property PrevUpdateCount: integer read GetPrevUpdateCount write
SetPrevUpdateCount;
    property PrevUpdate[Index: integer]: double read GetPrevUpdate write
SetPrevUpdate;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
  end;

  // Базовий клас шару
  TLayer = class(TPersistent)
  private
    FNumber: integer;
    // Розмірність NeuronCount
    FNeurons: array of TNeuron;
    function GetNeurons(Index: integer): TNeuron;
    function GetNeuronCount: integer;
    procedure SetNeurons(Index: integer; Value: TNeuron);
    procedure SetNeuronCount(Value: integer);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); virtual;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property Neurons[Index: integer]: TNeuron read GetNeurons write SetNeurons;
    property NeuronCount: integer read GetNeuronCount write SetNeuronCount;
  end;

  // Клас шару для мережі Хопфілда
  TLayerHopf = class(TLayer)
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
  end;

  // Клас шару для мережі
  TLayerBP = class(TLayer)
  private
    function GetNeuronsBP(Index: integer): TNeuronBP;
    procedure SetNeuronsBP(Index: integer; Value: TNeuronBP);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
```

```
    property NeuronsBP[Index: integer]: TNeuronBP read GetNeuronsBP write
SetNeuronsBP;
  end;

  // Базовий клас мережі
  TRadialBasisNeuralNet = class(TComponent)
  private
    // Масив шарів
    FLayers: array of TLayer;
    // Число вибірок
    FPatternCount: integer;
    // Розмірність FPatternCount, InputNeuronCount
    FPatternsInput: TMatrixFloat;
    // Розмірність FPatternCount, OutputNeuronCount
    FPatternsOutput: TMatrixFloat;
    function GetLayers(Index: integer): TLayer;
    function GetOutputNeuronCount: integer;
    function GetPatternsOutput(PatternIndex: integer; OutputIndex: integer):
double;
    function GetPatternsInput(PatternIndex: integer; InputIndex: integer):
double;
    procedure SetLayers(Index: integer; Value: TLayer);
    procedure SetPatternsInput(PatternIndex: integer; InputIndex: integer;
Value: double);
    procedure SetPatternsOutput(PatternIndex: integer; InputIndex: integer;
Value: double);
  protected
    function GetLayerCount: integer; virtual;
    function GetInputNeuronCount: integer; virtual;
    procedure Clear; virtual;
    procedure ResizeInputDim; virtual;
    procedure ResizeOutputDim; virtual;
    procedure SetPatternCount(const Value: integer); virtual;
    procedure SetLayerCount(Value: integer); virtual;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  public
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); virtual; abstract;
    procedure AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat); overload; virtual;
    procedure DeleteLayer(Index: integer); virtual; abstract;
    procedure DeletePattern(Index: integer); virtual;
    procedure Init(const ANeuronsInLayer: TVectorInt); overload; virtual;
    property InputNeuronCount: integer read GetInputNeuronCount;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Layers[Index: integer]: TLayer read GetLayers write SetLayers;
    property OutputNeuronCount: integer read GetOutputNeuronCount;
    property PatternsInput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsInput write SetPatternsInput;
    property PatternsOutput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsOutput write SetPatternsOutput;
    procedure ResetPatterns; virtual;
  end;

  // Клас мережі Хопфілда
  TRadialBasisNeuralNetHopf = class(TRadialBasisNeuralNet)
  private
    FAutoInit: boolean;
    FInputNeuronCount: integer;
    FMaxIterCount: integer;
    FPatternCount: integer;
    FPatterns: TMatrixInt;
    FOnAfterInit: TNotifyEvent;
    FOnBeforeInit: TNotifyEvent;
    FOnPatternRecognized: TNotifyEvent;
    function GetInput(Index: integer): double;
    function GetPatterns(InputIndex: integer; PatternIndex: integer): integer;
    function Stabled: boolean;
    procedure SetInput(Index: integer; Value: double);
```

```
    procedure SetPatterns(InputIndex: integer; PatternIndex: integer; Value:
integer);
  protected
    function GetInputNeuronCount: integer; override;
    function GetLayerCount: integer; override;
    procedure SetInputNeuronCount(Value: integer);
    procedure SetPatternCount(const Value: integer); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddPattern(const ANewPattern: TVectorInt); reintroduce; overload;
    procedure Calc; virtual;
    procedure DeletePattern(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure InitWeights; virtual;
    procedure ResetPatterns; override;
    procedure ResizePatternsDim; virtual;
    property Input[Index: integer]: double read GetInput write SetInput;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Patterns[InputIndex: integer; PatternIndex: integer]: integer read
GetPatterns write SetPatterns;
  published
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property InputNeuronCount: integer read GetInputNeuronCount write
SetInputNeuronCount;
    property MaxIterCount: integer read FMaxIterCount write FMaxIterCount;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnPatternRecognized: TNotifyEvent read FOnPatternRecognized write
FOnPatternRecognized;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  end;

  // Клас мережі
  TRadialBasisNeuralNetBP = class(TRadialBasisNeuralNet)
  private
    // Коефіцієнт крутості граничної сигмоїдальної функції
    FAlpha: double;
    // Прапор автоініціалізації топології мережі
    FAutoInit: boolean;
    // Прапор продовження навчання
    FContinueTeach: boolean;
    // Бажаний вихід нейромережі розмірність OutputNeuronCount
    FDesiredOut: TVectorFloat;
    // Прапор зупинки при досягненні FEpochCount
    FEpoch: boolean;
    // Лічильник етапів (пред'явлення мережі всіх прикладів з навчальної
вибірки)
    FEpochCount: integer;
    // Номер поточної епохи
    FEpochCurrent: integer;
    // Значення помилки, при якій приклад вважається розпізнаним
    FIdentError: double;
    // Значення максимальної помилки на навчальній множині
    FMaxTeachResidual: double;
    // Значення максимальної помилки на тестовій множині
    FMaxTestResidual: double;
    // Значення середньої помилки на навчальній множині
    FMidTeachResidual: double;
    // Значення середньої помилки на тестовій множині
    FMidTestResidual: double;
    // Помилка на навчальній множині
    FTeachError: double;
    // Коефіцієнт інерційності
    FMomentum: double;
    // Кількість нейронів у шарах
    FNeuronsInLayer: TStrings;
    // Подія після ініціалізації
    FOnAfterInit: TNotifyEvent;
```

```
    FOnAfterNeuronCreated: TNotifyEvent;
    // Подія після навчання
    FOnAfterTeach: TNotifyEvent;
    // Подія до ініціалізації
    FOnBeforeInit: TNotifyEvent;
    // Подія до початку навчання
    FOnBeforeTeach: TNotifyEvent;
    // Подія після проходження одного етапу
    FOnEpochPassed: TNotifyEvent;
    // Число прикладів у навчальній безлічі
    FPatternCount: integer;
    // Масив утримуючий псевдовипадкову послідовність
    FRandomOrder: TVectorInt;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTeachCount: integer;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTestCount: integer;
    // Прапор зупинки навчання
    FStopTeach: boolean;
    FTeachStopped: boolean;
    // Коефіцієнт швидкості навчання - величина градієнтного кроку
    FTeachRate: double;
    // Число прикладів у тестовій множині
    FTestSetPatternCount: integer;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatterns: TMatrixFloat;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatternsOut: TMatrixFloat;
    function GetDesiredOut(Index: integer): double;
    function GetLayersBP(Index: integer): TLayerBP;
    function GetTestSetPatterns(InputIndex, PatternIndex: integer): double;
    function GetTestSetPatternsOut(InputIndex, PatternIndex: integer): double;
    procedure NeuronCountError;
    procedure NeuronsInLayerChange(Sender: TObject);
    procedure SetAlpha(Value: double);
    procedure SetDesiredOut(Index: integer; Value: double);
    procedure SetEpochCount(Value: integer);
    procedure SetLayersBP(Index: integer; Value: TLayerBP);
    procedure SetMomentum(Value: double);
    procedure SetTeachRate(Value: double);
    procedure SetTestSetPatternCount(const Value: integer);
    procedure SetTestSetPatterns(InputIndex, PatternIndex: integer; const Value:
double);
    procedure SetTestSetPatternsOut(InputIndex, PatternIndex: integer; const
Value: double);
    // Перетасування набору даних
    procedure Shuffle;
  protected
    function GetLayerCount: integer; override;
    function GetOutput(Index: integer): double; virtual;
    // Активаційна функція
    function Activation(Value: double): double; virtual;
    // Похідна активаційної функції
    function Activation(Value: double): double; virtual;
    // Середня квадратична помилка
    function QuadError: double; virtual;
    // Підстроювання ваг
    procedure AdjustWeights; virtual;
    // Розраховує  локальну помилку - дельту
    procedure CalcLocalError; virtual;
    // Перевірка мережі на тестовій множині
    procedure CheckTestSet; virtual;
    procedure DoOnAfterInit; virtual;
    procedure DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex: integer);
virtual;
    procedure DoOnAfterTeach; virtual;
    procedure DoOnBeforeInit; virtual;
    procedure DoOnBeforeTeach; virtual;
    procedure DoOnEpochPassed; virtual;
```

```
    // Ініціалізація ваг мережі псевдовипадковими значеннями
    procedure InitWeights; virtual;
    // Пред'явлення мережі вхідних значень приклада
    procedure LoadPatternsInput(APatternIndex :integer); virtual;
    // Пред'явлення мережі вхідних значень приклада
    procedure LoadPatternsOutput(APatternIndex :integer); virtual;
    // Поширює сигнал у прямому напрямку
    procedure Propagate; virtual;
    // Установка значень за замовчуванням
    procedure SetDefaultProperties; virtual;
    procedure SetPatternCount(const Value: integer); override;
    // Струс мережі
    procedure ShakeUp; virtual;
    property TeachStopped: boolean read FTeachStopped write FTeachStopped;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); override;
    procedure Compute(AVector: TVectorFloat); virtual;
    procedure DeleteLayer(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure ResetLayers; virtual;
    procedure TeachOffLine; virtual;
    property DesiredOut[Index: integer]: double read GetDesiredOut write
SetDesiredOut;
    property EpochCurrent: integer read FEpochCurrent;
    property IdentError: double read FIdentError write FIdentError;
    property LayersBP[Index: integer]: TLayerBP read GetLayersBP write
SetLayersBP;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Output[Index: integer]: double read GetOutput;
    property StopTeach: boolean read FStopTeach write FStopTeach;
    property TeachError: double read FTeachError;
    property MaxTeachResidual: double read FMaxTeachResidual;
    property MaxTestResidual: double read FMaxTestResidual;
    property MidTeachResidual: double read FMidTeachResidual;
    property MidTestResidual: double read FMidTestResidual;
    property RecognizedTeachCount: integer read FRecognizedTeachCount;
    property RecognizedTestCount: integer read FRecognizedTestCount;
    property TestSetPatternCount: integer read FTestSetPatternCount write
SetTestSetPatternCount;
    property TestSetPatterns[InputIndex: integer; PatternIndex: integer]: double
read GetTestSetPatterns write SetTestSetPatterns;
    property TestSetPatternsOut[InputIndex: integer; PatternIndex: integer]:
double read GetTestSetPatternsOut write SetTestSetPatternsOut;
  published
    property Alpha: double read FAlpha write SetAlpha;
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property ContinueTeach: boolean read FContinueTeach write FContinueTeach;
    property Epoch: boolean read FEpoch write FEpoch;
    property EpochCount: integer read FEpochCount write SetEpochCount;
    property Momentum: double read FMomentum write SetMomentum;
    property NeuronsInLayer: TStrings read FNeuronsInLayer write
FNeuronsInLayer;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnAfterNeuronCreated: TNotifyEvent read FOnAfterNeuronCreated write
FOnAfterNeuronCreated;
    property OnAfterTeach: TNotifyEvent read FOnAfterTeach write FOnAfterTeach;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnBeforeTeach: TNotifyEvent read FOnBeforeTeach write
FOnBeforeTeach;
    property OnEpochPassed: TNotifyEvent read FOnEpochPassed write
FOnEpochPassed;
    property PatternCount: integer read FPatternCount write SetPatternCount;
    property TeachRate: double read FTeachRate write SetTeachRate;
  end;

  // Клас мережі back-propagation TRadialBasisNeuralNetExtended }
  TRadialBasisNeuralNetExtended = class(TRadialBasisNeuralNetBP)
```

```
private
  // Файл даних
  FNeuroDataSource: TNeuroDataSource;
  // Ім'я файлу даних *.txt
  FSourceFileName: TFileName;
  // Ім'я конфігураційного файлу *.nnw
  FFileName: TFileName;
  // Конфігураційний файл
  FNnwFile: TIniFile;
  // Поля
  FFields: TNeuroFields;
  // Кількість доступних полів
  FAvailableFieldsCount: integer;
  FMaxTeachError: boolean;
  FMaxTeachErrorValue: double;
  FMaxTestError: boolean;
  FMaxTestErrorValue: double;
  FMidTeachError: boolean;
  FMidTeachErrorValue: double;
  FMidTestError: boolean;
  FMidTestErrorValue: double;
  FOptions: string;
  FSettingsLoaded: boolean;
  FTestAsValid: boolean;
  FTeachIdent: boolean;
  FTeachIdentCount: integer;
  FTestIdent: boolean;
  FTestIdentCount: integer;
  FUseForTeach: integer;
  FIdentError: double;
  FRealOutputIndex: TVectorInt;
  FRealInputIndex: TVectorInt;
  function GetFields(Index: integer): TNeuroField;
  function GetInputFieldCount: integer;
  function GetOutputFieldCount: integer;
  function GetRealInputIndex(Index: integer): integer;
  function GetRealOutputIndex(Index: integer): integer;
  procedure SetFields(Index: integer; Value: TNeuroField);
  procedure SetFileName(Value: TFilename);
  procedure SetAvailableFieldsCount(Value: integer);
  procedure SetUseForTeach(const Value: integer);
  procedure SetTeachIdentCount(const Value: integer);
  procedure SetRealOutputIndex(Index: integer; const Value: integer);
  procedure SetRealOutputIndexCount(const Value: integer);
  procedure SetRealInputIndex(Index: integer; const Value: integer);
  procedure SetRealInputIndexCount(const Value: integer);
protected
  function GetOutput(Index: integer): double; override;
  procedure DoOnBeforeTeach; override;
  procedure DoOnEpochPassed; override;
  procedure SetDefaultProperties; override;
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  procedure ComputeUnPrepData(AVector: TVectorFloat);
  // Завантажує дані з текстового файлу
  procedure LoadDataFrom;
  // Завантажує настроювання мережі
  procedure LoadNetwork;
  // Завантажує настроювання мережі
  procedure LoadPhase1;
  // Завантажує настроювання мережі
  procedure LoadPhase2;
  // Завантажує настроювання мережі
  procedure LoadPhase4;
  // Нормалізує набір даних
  procedure NormalizeData;
  // Зберігає настроювання мережі
  procedure SaveNetwork;
```

```
    // Зберігає настроювання мережі
    procedure SavePhase1;
    // Зберігає настроювання мережі
    procedure SavePhase2;
    // Зберігає настроювання мережі
    procedure SavePhase4;
    // Навчання нейронної мережі
    procedure Train;
    property AvailableFieldsCount: integer read FAvailableFieldsCount write
SetAvailableFieldsCount;
    property Fields[Index: integer]: TNeuroField read GetFields write SetFields;
    property InputFieldCount: integer read GetInputFieldCount;
    property OutputFieldCount: integer read GetOutputFieldCount;
    property SettingsLoaded: boolean read FSettingsLoaded write FSettingsLoaded;
    property RealOutputIndex[Index: integer]: integer read GetRealOutputIndex
write SetRealOutputIndex;
    property RealOutputIndexCount: integer write SetRealOutputIndexCount;
    property RealInputIndex[Index: integer]: integer read GetRealInputIndex
write SetRealInputIndex;
    property RealInputIndexCount: integer write SetRealInputIndexCount;
    property NnwFile: TIniFile read FNnwFile write FNnwFile;
  published
    property FileName: TFileName read FFileName write SetFileName;
    property IdentError: double read FIdentError write FIdentError;
    property MaxTeachError: boolean read FMaxTeachError write FMaxTeachError;
    property MaxTeachErrorValue: double read FMaxTeachErrorValue write
FMaxTeachErrorValue;
    property MaxTestError: boolean read FMaxTestError write FMaxTestError;
    property MaxTestErrorValue: double read FMaxTestErrorValue write
FMaxTestErrorValue;
    property MidTeachError: boolean read FMidTeachError write FMidTeachError;
    property MidTeachErrorValue: double read FMidTeachErrorValue write
FMidTeachErrorValue;
    property MidTestError: boolean read FMidTestError write FMidTestError;
    property MidTestErrorValue: double read FMidTestErrorValue write
FMidTestErrorValue;
    property Options: string read FOptions write FOptions;
    property SourceFileName: TFileName read FSourceFileName write
FSourceFileName;
    property TestAsValid: boolean read FTestAsValid write FTestAsValid;
    property TeachIdent: boolean read FTeachIdent write FTeachIdent;
    property TeachIdentCount: integer read FTeachIdentCount write
SetTeachIdentCount;
    property TestIdent: boolean read FTestIdent write FTestIdent;
    property TestIdentCount: integer read FTestIdentCount write FTestIdentCount;
    property UseForTeach: integer read FUseForTeach write SetUseForTeach;
  end;


procedure Register;

implementation

{$R *.RES}

{ TNeuron }

constructor TNeuron.Create(ALayer: TLayer);
begin
  inherited Create;
  // покажчик на шар у якому перебуває нейрон
  Layer := ALayer;
end;

destructor TNeuron.Destroy;
begin
  WeightCount := 0;
  FWeights := nil;
  Layer := nil;
```

```
  inherited;
end;

procedure TNeuron.ComputeOut(const AInputs: TVectorFloat);
var
  i: integer;
begin
  FOutput := 0;
  // Підраховується зважена сума нейрона
  for i := Low(AInputs) to High(AInputs) do
    FOutput := FOutput + FWeights[i] * AInputs[i];
end;

function TNeuron.GetWeights(Index: integer): double;
begin
  try
    Result := FWeights[Index];
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

procedure TNeuron.InitWeights;
var
  i: integer;
begin
  // Ініціалізація ваг нейрона
  for i := Low(FWeights) to High(FWeights) do
    FWeights[i] := Random
end;

procedure TNeuron.SetWeightCount(const Value: integer);
begin
  SetLength(FWeights, Value);
end;

procedure TNeuron.SetWeights(Index: integer; Value: double);
begin
  try
    FWeights[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

{ Кінець опису TNeuron }

{ TNeuronHopf }

procedure TNeuronHopf.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // гранична функція
  if FOutput >= 0 then
    FOutput := 1
  else
    FOutput := -1
end;

{ Кінець опису TNeuronHopf}

{ TNeuronBP }

destructor TNeuronBP.Destroy;
begin
  FOnActivation := nil;
  FOnActivation := nil;
```

```
  PrevUpdateCount := 0;
  FPrevUpdate := nil;
  inherited;
end;

function TNeuronBP.GetLearningRate(Index: integer): double;
begin
  Result := FLearningRate[Index];
end;

function TNeuronBP.GetPrevDerivative(Index: integer): double;
begin
  Result := FPrevDerivative[Index];
end;

function TNeuronBP.GetPrevUpdateCount: integer;
begin
  Result := High(FPrevUpdate) + 1;
end;

function TNeuronBP.GetPrevUpdate(Index: integer): double;
begin
  Result := FPrevUpdate[Index];
end;

procedure TNeuronBP.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // Задає зсув нейрона
  FOutput := FOutput + Weights[High(AInputs) + 1];
  FOutput := OnActivation(FOutput);
end;

procedure TNeuronBP.SetDelta(Value: double);
begin
  FDelta := Value;
end;

procedure TNeuronBP.SetLearningRate(Index: integer; const Value: double);
begin
  FLearningRate[Index] := Value;
end;

procedure TNeuronBP.SetLearningRateCount(const Value: integer);
begin
  SetLength(FLearningRate, Value)
end;

procedure TNeuronBP.SetPrevUpdate(Index: integer; Value: double);
begin
  FPrevUpdate[Index] := Value;
end;

procedure TNeuronBP.SetPrevUpdateCount(const Value: integer);
begin
  SetLength(FPrevUpdate, Value)
end;

procedure TNeuronBP.SetPrevDerivative(Index: integer; const Value: double);
begin
  FPrevDerivative[Index] := Value;
end;

procedure TNeuronBP.SetPrevDerivativeCount(const Value: integer);
begin
  SetLength(FPrevDerivative, Value)
end;

{ Кінець опису TNeuronBP }
```

```
{ TLayer }

procedure TLayer.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayer).FNumber;
  NeuronCount := (Source as TLayer).NeuronCount;
  // Створюються нейрони
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

constructor TLayer.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  inherited Create;
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

destructor TLayer.Destroy;
var
  i: integer;
begin
  for i := 0 to NeuronCount - 1 do
    FNeurons[i].Free;
  NeuronCount := 0;
  FNeurons := nil;
  inherited;
end;

function TLayer.GetNeuronCount: integer;
begin
  Result := High(FNeurons) + 1;
end;

function TLayer.GetNeurons(Index: integer): TNeuron;
begin
  Result := FNeurons[Index];
end;

procedure TLayer.SetNeuronCount(Value: integer);
begin
  if Value <> High(FNeurons) + 1 then
    SetLength(FNeurons, Value);
end;

procedure TLayer.SetNeurons(Index: integer; Value: TNeuron);
begin
  try
    FNeurons[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

{ TLayerHopf }

constructor TLayerHopf.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
```

```
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronHopf.Create(Self);
end;

{ TLayerBP }

procedure TLayerBP.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayerBP).FNumber;
  NeuronCount := (Source as TLayerBP).NeuronCount;
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

constructor TLayerBP.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

destructor TLayerBP.Destroy;
begin
  inherited;
end;

function TLayerBP.GetNeuronsBP(Index: integer): TNeuronBP;
begin
  Result := FNeurons[Index] as TNeuronBP;
end;

procedure TLayerBP.SetNeuronsBP(Index: integer; Value: TNeuronBP);
begin
  FNeurons[Index] := Value as TNeuronBP;
end;

{ TRadialBasisNeuralNet }

destructor TRadialBasisNeuralNet.Destroy;
begin
  Clear;
  SetLength(FPatternsInput, 0, 0);
  FPatternsInput := nil;
  SetLength(FPatternsOutput, 0 ,0);
  FPatternsOutput := nil;
  FLayers := nil;
  inherited;
end;

procedure TRadialBasisNeuralNet.Clear;
var
  i, xCount: integer;
begin
  xCount := LayerCount;
  if  xCount > 0 then
  begin
    for i := 0 to xCount - 1 do
      FLayers[i].Free;
    LayerCount := 0;
  end;
end;

function TRadialBasisNeuralNet.GetInputNeuronCount: integer;
```

```
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TRadialBasisNeuralNet.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TRadialBasisNeuralNet.GetLayers(Index: integer): TLayer;
begin
  Result := FLayers[Index];
end;

function TRadialBasisNeuralNet.GetOutputNeuronCount: integer;
begin
  Result := Layers[LayerCount - 1].NeuronCount;
end;

function TRadialBasisNeuralNet.GetPatternsInput(PatternIndex: integer;
InputIndex: integer): double;
begin
  Result := FPatternsInput[PatternIndex, InputIndex];
end;

procedure TRadialBasisNeuralNet.AddPattern(const AInputs: TVectorFloat; const
AOutputs: TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AInputs) + 1 then
    raise EInOutDimensionError.Create(SInVectorCount);
  if OutputNeuronCount <> High(AOutputs) + 1 then
    raise EInOutDimensionError.Create(SOutVectorCount);
  PatternCount := PatternCount + 1;
  ResizeInputDim;
  ResizeOutputDim;
  for i := Low(AInputs) to High(AInputs) do
    PatternsInput[PatternCount - 1, i] := AInputs[i];
  for i := Low(AOutputs) to High(AOutputs) do
    PatternsOutput[PatternCount - 1, i] := AOutputs[i];
end;

procedure TRadialBasisNeuralNet.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    // видаляє вхідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to InputNeuronCount - 1 do
        FPatternsInput[i, j] := FPatternsInput[i + 1, j];
    // видаляє вихідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to OutputNeuronCount - 1 do
        FPatternsOutput[i, j] := FPatternsOutput[i + 1, j];
    Dec(FPatternCount);
    ResizeInputDim;
    ResizeOutputDim;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TRadialBasisNeuralNet.ResetPatterns;
begin
  FPatternCount := DefaultPatternCount;
  ResizeInputDim;
```

```
  ResizeOutputDim;
end;

procedure TRadialBasisNeuralNet.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TRadialBasisNeuralNet.SetPatternsOutput(PatternIndex: integer;
InputIndex: integer; Value: double);
begin
  FPatternsOutput[PatternIndex, InputIndex] := Value;
end;

procedure TRadialBasisNeuralNet.SetPatternsInput(PatternIndex: integer;
InputIndex: integer; Value: double);
begin
  FPatternsInput[PatternIndex, InputIndex] := Value;
end;

procedure TRadialBasisNeuralNet.Init(const ANeuronsInLayer: TVectorInt);
var
  i,j: integer;
begin
  LayerCount := High(ANeuronsInLayer) + 1;
  // FLayers[0] нульовий шар і виконує роль розподільного,
  // використовується тільки поле Output
  FLayers[0] := TLayer.Create(0, ANeuronsInLayer[0]);
  // для нульового шару не потрібні вагові коефіцієнти
  for i := 1 to LayerCount - 1 do
  begin
    FLayers[i] := TLayer.Create(i, ANeuronsInLayer[i]);
    for j := 0 to ANeuronsInLayer[i] - 1 do
    with FLayers[i].FNeurons[j] do
      // задає кількість елементів у векторі ваг нейрона j в
      //  шарі i рівним кількості виходів попереднього шару
      WeightCount := FLayers[ i-1].NeuronCount;
  end;
end;

procedure TRadialBasisNeuralNet.ResizeInputDim;
begin
  SetLength(FPatternsInput, FPatternCount, InputNeuronCount)
end;

procedure TRadialBasisNeuralNet.ResizeOutputDim;
begin
  SetLength(FPatternsOutput, FPatternCount, OutputNeuronCount)
end;

procedure TRadialBasisNeuralNet.SetLayerCount(Value: integer);
begin
  SetLength(FLayers, Value);
end;

procedure TRadialBasisNeuralNet.SetLayers(Index: integer; Value: TLayer);
begin
  try
    FLayers[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;
```

```
{  TRadialBasisNeuralNetHopf }

constructor TRadialBasisNeuralNetHopf.Create(AOwner: TComponent);
begin
  inherited;
  PatternCount := DefaultPatternCount;
  InputNeuronCount := DefaultNeuronCount;
  MaxIterCount := DefaultMaxIterCount;
  AutoInit := False;
end;

destructor TRadialBasisNeuralNetHopf.Destroy;
begin
  FOnAfterInit := nil;
  FOnBeforeInit := nil;
  FOnPatternRecognized := nil;
  SetLength(FPatterns, 0, 0);
  FPatterns := nil;
  inherited;
end;

function TRadialBasisNeuralNetHopf.GetInput(Index: integer): double;
begin
  Result := Layers[1].Neurons[Index].Output;
end;

function TRadialBasisNeuralNetHopf.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TRadialBasisNeuralNetHopf.GetLayerCount: integer;
begin
  Result := DefaultHopfLayerCount;
end;

function TRadialBasisNeuralNetHopf.GetPatterns(InputIndex: integer;
PatternIndex: integer): integer;
begin
  Result := FPatterns[InputIndex, PatternIndex];
end;

function TRadialBasisNeuralNetHopf.Stabled: boolean;
var
  i: integer;
begin
  // Порівнює вихідні значення попередньої
  //  ітерації зі значеннями поточної
  Result := True;
  for i := 0 to InputNeuronCount - 1  do
    if FLayers[1].FNeurons[i].FOutput <> FLayers[0].FNeurons[i].FOutput then
    begin
      Result := False;
      Exit
    end;
end;

procedure TRadialBasisNeuralNetHopf.AddPattern(const ANewPattern: TVectorInt);
var
  i: integer;
begin
  if InputNeuronCount <> High(ANewPattern)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  PatternCount := PatternCount + 1;
  ResizePatternsDim;
  for i := 0 to FInputNeuronCount - 1 do
    FPatterns[FPatternCount - 1, i] := ANewPattern[i];
  if AutoInit then
```

```
    InitWeights;
end;

procedure TRadialBasisNeuralNetHopf.Calc;
var
  i: integer;
  xCurrentIter: integer;
  xArray: TVectorFloat;
begin
  SetLength(xArray, InputNeuronCount);
  // Цикл працює поки не стабілізуються виходи
  xCurrentIter := 0;
  repeat
    for i := 0 to InputNeuronCount - 1 do
    begin
      // Запам'ятовує попередній крок ітерації, для
      //  цього використовується нульовий шар
      Layers[SensorLayer].Neurons[i].Output := Layers[1].Neurons[i].Output;
      xArray[i] := Layers[1].Neurons[i].Output;
    end;
    for i := 0 to InputNeuronCount - 1 do
      with Layers[1].Neurons[i] do
        // Розраховується новий стан нейронів і аксонів
        ComputeOut(xArray);
    Inc(xCurrentIter);
  until Stabled or (MaxIterCount = xCurrentIter);
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TRadialBasisNeuralNetHopf.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    for i := Index to FPatternCount - 2 do
      for j := 0 to FInputNeuronCount - 1 do
        FPatterns[i, j] := FPatterns[i + 1, j];
    Dec(FPatternCount);
    ResizePatternsDim;
    if AutoInit then
      InitWeights;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TRadialBasisNeuralNetHopf.Init;
var
  i, j: integer;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
  LayerCount := DefaultHopfLayerCount;
  for i := 0 to LayerCount - 1 do
    FLayers[i] := TLayerHopf.Create(i, FInputNeuronCount);
  // Для нульового шару не потрібні вагові коефіцієнти
  for j := 0 to FInputNeuronCount - 1 do
    with FLayers[1].FNeurons[j] do
      // задає кількість елементів у векторі
      WeightCount := FInputNeuronCount;
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TRadialBasisNeuralNetHopf.InitWeights;
```

```
var
  i, j, k : integer;
begin
  // Ініціалізує вагову матрицю
  for i := 0 to InputNeuronCount - 1 do
    for j := 0 to InputNeuronCount - 1 do
    with Layers[1].Neurons[i] do
    begin
      Weights[j] := 0;
      if i <> j then
        for k := 0 to PatternCount - 1 do
          Weights[j] := Weights[j]+ Patterns[k, i] * Patterns[k, j]
    end;
end;

procedure TRadialBasisNeuralNetHopf.ResetPatterns;
begin
  PatternCount := DefaultPatternCount;
  if AutoInit then
    InitWeights;
end;

procedure TRadialBasisNeuralNetHopf.ResizePatternsDim;
begin
  SetLength(FPatterns, FPatternCount, FInputNeuronCount);
end;

procedure TRadialBasisNeuralNetHopf.SetInput(Index: integer; Value: double);
begin
  try
    Layers[1].Neurons[Index].Output := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TRadialBasisNeuralNetHopf.SetInputNeuronCount(Value: integer);
begin
  if Value > DefaultNeuronCount then
    FInputNeuronCount := Value
  else
    FInputNeuronCount := DefaultNeuronCount;
  ResizePatternsDim;
  Init;
end;

procedure TRadialBasisNeuralNetHopf.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
end;

procedure TRadialBasisNeuralNetHopf.SetPatterns(InputIndex: integer;
PatternIndex: integer; Value: integer);
begin
  FPatterns[InputIndex, PatternIndex] := Value;
end;

{ TRadialBasisNeuralNetBP }

constructor TRadialBasisNeuralNetBP.Create(AOwner: TComponent);
var
  i: integer;
begin
  inherited;
  FNeuronsInLayer := TStringList.Create;
```

```
  for i := 0 to DefaultLayerCount do
    AddLayer(DefaultNeuronCount);
  TStringList(FNeuronsInLayer).OnChange := NeuronsInLayerChange;
  AutoInit := True;
  StopTeach := False;
  TeachStopped := False;
  NeuronsInLayerChange(Self);
  SetDefaultProperties;
end;

destructor TRadialBasisNeuralNetBP.Destroy;
begin
  FNeuronsInLayer.Free;
  SetLength(FRandomOrder, 0);
  FRandomOrder := nil;
  SetLength(FDesiredOut, 0);
  FDesiredOut := nil;
  SetLength(FTestSetPatterns, 0, 0);
  FTestSetPatterns := nil;
  SetLength(FTestSetPatternsOut, 0, 0);
  FTestSetPatternsOut := nil;
  FOnAfterInit := nil;
  FOnAfterTeach := nil;
  FOnBeforeInit := nil;
  FOnBeforeTeach := nil;
  FOnEpochPassed := nil;
  inherited;
end;

function TRadialBasisNeuralNetBP.GetLayersBP(Index: integer): TLayerBP;
begin
  Result := FLayers[Index] as TLayerBP;
end;

function TRadialBasisNeuralNetBP.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TRadialBasisNeuralNetBP.GetDesiredOut(Index: integer): double;
begin
  Result := FDesiredOut[Index];
end;

function TRadialBasisNeuralNetBP.GetOutput(Index: integer): double;
begin
  try
    Result := LayersBP[LayerCount - 1].NeuronsBP[Index].Output;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

function TRadialBasisNeuralNet.GetPatternsOutput(PatternIndex: integer;
OutputIndex: integer): double;
begin
  Result := FPatternsOutput[PatternIndex, OutputIndex];
end;

function TRadialBasisNeuralNetBP.QuadError: double;
var
  i: integer;
begin
  // розраховує середньоквадратичну помилку
  Result := 0;
  for i := 0 to OutputNeuronCount - 1 do
    Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
DesiredOut[i]);
```

```
  Result := Result/2;
end;

function TRadialBasisNeuralNetBP.Activation(Value: double): double;
begin
  // Активаційна функція - сигмоїд
  Result := 1/( 1 + exp(-FAlpha * Value) )
end;

function TRadialBasisNeuralNetBP.Activation(Value: double): double;
begin
  // Похідна сигмоїди
  Result := FAlpha * Value * (1 - Value)
end;

function TRadialBasisNeuralNetBP.GetTestSetPatterns(InputIndex, PatternIndex:
integer): double;
begin
  Result := FTestSetPatterns[InputIndex, PatternIndex];
end;

function TRadialBasisNeuralNetBP.GetTestSetPatternsOut(InputIndex, PatternIndex:
integer): double;
begin
  Result := FTestSetPatternsOut[InputIndex, PatternIndex];
end;

procedure TRadialBasisNeuralNetBP.AddLayer(ANeurons: integer);
begin
  if ANeurons < DefaultNeuronCount then
    NeuronCountError
  else
    NeuronsInLayer.Add(IntToStr(ANeurons));
end;

procedure TRadialBasisNeuralNetBP.AdjustWeights;
var
  i, j, k: integer;
  xCurrentUpdate: double;
begin
  // Підстроювання ваг починаючи з першого шару
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount  do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // коректує вагу з'єднуючого j-нейрона шару i
        //  з k-нейроном шару i-1:  добутком дельта j-нейрона
        //  на вихід k-нейрона шару i-1
        if k = LayersBP[ i-1].NeuronCount then
           // якщо це нейрон, що задає зсув
           xCurrentUpdate := -TeachRate * Delta + Momentum * PrevUpdate[k]
        else
           xCurrentUpdate := -TeachRate * Delta *
           LayersBP[ i-1].NeuronsBP[k].Output + Momentum * PrevUpdate[k];
        Weights[k]:= Weights[k] + xCurrentUpdate;
        PrevUpdate[k] := xCurrentUpdate;
      end;
    end
end;

procedure TRadialBasisNeuralNetBP.CalcLocalError;
var
  i, j, k: integer;
begin
  // Дельта-правило з останнього шару до першого
  for i := LayerCount - 1 downto 1 do
    // для останнього шару
```

```
    if i = LayerCount - 1 then
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Delta := (LayersBP[i].NeuronsBP[j].Output-
DesiredOut[j])
        * Activation(LayersBP[i].NeuronsBP[j].Output)
    else
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        with LayersBP[i].NeuronsBP[j] do
        begin
          Delta := 0;
          // Підсумує добуток локальної помилки k-нейрона шару i+1
          //  на вагу з'єднуючий k-нейрон шару i+1 з j-нейроном шару i
          for k := 0 to LayersBP[i+1].NeuronCount - 1 do
            Delta := Delta + LayersBP[i+1].NeuronsBP[k].Delta *
                             LayersBP[i+1].NeuronsBP[k].Weights[j];
          Delta := Delta * Activation(Output)
        end;
end;

procedure TRadialBasisNeuralNetBP.CheckTestSet;
var
  i, j: integer;
  xArray: TVectorFloat;
  xFirstTestSample: boolean;
  xQuadError: double;
  // функція розраховує середньоквадратичну помилку
  function QuadError(APatternCount: integer): double;
  var
    i: integer;
  begin
    Result := 0;
    for i := 0 to OutputNeuronCount - 1 do
      Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
TestSetPatternsOut[APatternCount, i]);
    Result := Result/2;
  end;
begin
  SetLength(xArray, InputNeuronCount);
  xFirstTestSample := True;
  FRecognizedTestCount := 0;
  FMidTestResidual := 0;
  FMaxTestResidual := 0;
  for i := 0 to TestSetPatternCount - 1 do
  begin
    for j := 0 to InputNeuronCount - 1 do
      xArray[j] := TestSetPatterns[i, j];
    Compute(xArray);
    xQuadError := QuadError(i);
    // перевірка - чи розпізнаний приклад з тестової множини
    if xQuadError < IdentError then
      Inc(FRecognizedTestCount);
    FMidTestResidual := FMidTestResidual + xQuadError;
    // максимальна помилка на тестовій множині
    if xFirstTestSample then
    begin
      FMaxTestResidual := xQuadError;
      xFirstTestSample := False;
    end
    else
      if FMaxTestResidual < xQuadError then
        FMaxTestResidual := xQuadError;
  end;
  // середня помилка на тестовій множині
  FMidTestResidual := FMidTestResidual/TestSetPatternCount;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TRadialBasisNeuralNetBP.Compute(AVector: TVectorFloat);
```

```
var
  i: integer;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  AVector[i];
  Propagate;
end;

procedure TRadialBasisNeuralNetBP.DoOnAfterInit;
begin
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TRadialBasisNeuralNetBP.DoOnAfterNeuronCreated(ALayerIndex,
ANeuronIndex: integer);
var
  i: integer;
begin
  with LayersBP[ALayerIndex].NeuronsBP[ANeuronIndex] do
    for i := 0 to PrevUpdateCount - 1 do
      PrevUpdate[i] := 0;
  if Assigned(FOnAfterNeuronCreated) then
    FOnAfterNeuronCreated(Self);
end;

procedure TRadialBasisNeuralNetBP.DoOnAfterTeach;
begin
  if Assigned(FOnAfterTeach) then
    FOnAfterTeach(Self);
end;

procedure TRadialBasisNeuralNetBP.DoOnBeforeInit;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
end;

procedure TRadialBasisNeuralNetBP.DoOnBeforeTeach;
begin
  if Assigned(FOnBeforeTeach) then
    FOnBeforeTeach(Self);
end;

procedure TRadialBasisNeuralNetBP.DoOnEpochPassed;
begin
  if Assigned(FOnEpochPassed) then
    FOnEpochPassed(Self);
end;

procedure TRadialBasisNeuralNetBP.DeleteLayer(Index: integer);
var
  i: integer;
begin
  try
    NeuronsInLayer.Delete(Index);
    for i := Index to LayerCount - 2 do
      LayersBP[i].Assign(LayersBP[i + 1]);
    FLayers[LayerCount - 1].Free;
    LayerCount := LayerCount - 1;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

procedure TRadialBasisNeuralNetBP.Init;
```

```
var
  i, j: integer;
begin
  DoOnBeforeInit;
  if NeuronsInLayer.Count > 0 then
  begin
  LayerCount := NeuronsInLayer.Count;
    // FLayers[0] нульовий шар, використовується тільки поле Output
    FLayers[0] := TLayerBP.Create(0, StrToInt(NeuronsInLayer.Strings[0]));
    // для нульового шару не потрібні вагові коефіцієнти
    for i := 1 to LayerCount - 1 do
    begin
      FLayers[i] := TLayerBP.Create(i, StrToInt(NeuronsInLayer.Strings[i]));
      for j := 0 to StrToInt(NeuronsInLayer.Strings[i]) - 1 do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // задає кількість елементів у векторі ваг + зсув
        WeightCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        // задає кількість  у векторі утримуючих попередню
        //  корекцію елементів  + зсув
        PrevUpdateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        PrevDerivativeCount := LayersBP[ i-1].NeuronCount + BiasNeuron; // для
швидких алгоритмів
        LearningRateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;   // для
швидких алгоритмів
        OnActivation := Activation;
        OnActivation := Activation;
        Randomize;
        DoOnAfterNeuronCreated(i, j);
      end
    end;
    // установлює розмірність масиву виходів
    //  число нейронів в останньому шарі = числу виходів
    SetLength(FDesiredOut, OutputNeuronCount);
  end;
  DoOnAfterInit;
end;

procedure TRadialBasisNeuralNetBP.InitWeights;
var
  i,j: integer;
begin
  Randomize;
  // Ініціалізація ваг
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
       LayersBP[i].NeuronsBP[j].InitWeights;
end;

procedure TRadialBasisNeuralNetBP.LoadPatternsInput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to InputNeuronCount - 1 do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  PatternsInput[APatternIndex,
i];
end;

procedure TRadialBasisNeuralNetBP.LoadPatternsOutput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to OutputNeuronCount - 1 do
    DesiredOut[i] := PatternsOutput[APatternIndex, i];
end;

procedure TRadialBasisNeuralNetBP.NeuronsInLayerChange(Sender: TObject);
begin
  if AutoInit then
```

```
    Init;
end;

procedure TRadialBasisNeuralNetBP.NeuronCountError;
begin
  raise ENeuronCountError.Create(SNeuronCount)
end;

procedure TRadialBasisNeuralNetBP.Propagate;
var
  i, j, xIndex: integer;
  xArray: TVectorFloat;
begin
   // Поширення сигналу в прямому напрямку з першого шару
  for i := 1 to LayerCount - 1 do
  begin
    // формування масиву входів з виходів попереднього шару
    SetLength(xArray, LayersBP[ i-1].NeuronCount);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := LayersBP[ i-1].NeuronsBP[xIndex].Output;
    // обчислення виходу нейрона
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      with LayersBP[i].NeuronsBP[j] do
        ComputeOut(xArray);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := 0;
  end;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TRadialBasisNeuralNetBP.ResetLayers;
begin
  Clear;
  FNeuronsInLayer.Clear;
end;

procedure TRadialBasisNeuralNetBP.SetDesiredOut(Index: integer; Value: double);
begin
  FDesiredOut[Index] := Value;
end;

procedure TRadialBasisNeuralNetBP.SetLayersBP(Index: integer; Value: TLayerBP);
begin
  FLayers[Index] := Value as TLayerBP;
end;

procedure TRadialBasisNeuralNetBP.SetAlpha(Value: double);
begin
  if (Value > 10) or (Value < 0.01) then
    FAlpha := DefaultAlpha
  else
    FAlpha := Value;
end;

procedure TRadialBasisNeuralNetBP.SetTeachRate(Value: double);
begin
  if (Value > 1) or (Value <= 0) then
    FTeachRate := DefaultTeachRate
  else
    FTeachRate := Value;
end;

procedure TRadialBasisNeuralNetBP.SetTestSetPatterns(InputIndex, PatternIndex:
integer; const Value: double);
begin
  FTestSetPatterns[InputIndex, PatternIndex] := Value;
end;
```

```
procedure TRadialBasisNeuralNetBP.SetTestSetPatternsOut(InputIndex,
PatternIndex: integer; const Value: double);
begin
  FTestSetPatternsOut[InputIndex, PatternIndex] := Value;
end;

procedure TRadialBasisNeuralNetBP.SetTestSetPatternCount(const Value: integer);
begin
  FTestSetPatternCount := Value;
  SetLength(FTestSetPatterns, FTestSetPatternCount, InputNeuronCount);
  SetLength(FTestSetPatternsOut, FTestSetPatternCount, OutputNeuronCount);
end;

procedure TRadialBasisNeuralNetBP.SetMomentum(Value: double);
begin
  if (Value > 1) or (Value < 0) then
    FMomentum := DefaultMomentum
  else
    FMomentum := Value;
end;

procedure TRadialBasisNeuralNetBP.SetEpochCount(Value: integer);
begin
  if Value < 1 then
    FEpochCount := 1
  else
    FEpochCount := Value;
end;

procedure TRadialBasisNeuralNetBP.ShakeUp;
var
  i, j, k: integer;
begin
  Randomize;
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      for k := 0 to LayersBP[ i-1].NeuronCount  do
        with LayersBP[i].NeuronsBP[j] do
          Weights[k]:= Weights[k] + Random*0.1-0.05;
end;

procedure TRadialBasisNeuralNetBP.Shuffle;
var
  i, j, xNewInd, xLast: integer;
  xIsUnique : boolean;
begin
  xNewInd := 0;
  FRandomOrder[0] := Round(Random(FPatternCount));
  xLast := 0;
  for i := 1 to PatternCount - 1 do
  begin
    xIsUnique := False;
    while not xIsUnique do
    begin
      xNewInd := Round((Random(FPatternCount)));
      xIsUnique := True;
      for j := 0 to xLast do
        if xNewInd = FRandomOrder[j] then
          xIsUnique := False;
    end;
    FRandomOrder[i] := xNewInd;
    xLast := xLast +1;
  end;
end;

procedure TRadialBasisNeuralNetBP.TeachOffLine;
var
  j: integer;
  xQuadError: double;
```

```
  xNewEpoch: boolean;
begin
  DoOnBeforeTeach;
  if not ContinueTeach then
  begin
    // ваги ініціалізуються, якщо мережа навчається з "нуля"
    InitWeights;
    FEpochCurrent := 1;
  end;
  Randomize;
  SetLength(FRandomOrder, FPatternCount);
  TeachStopped := False;
  while (FEpochCurrent <= EpochCount) do
  begin
    FTeachError := 0;
    FMaxTeachResidual := 0;
    FRecognizedTeachCount := 0;
    xNewEpoch := True;
    Shuffle;
    for j := 0 to PatternCount - 1 do
    begin
      LoadPatternsInput(FRandomOrder[j]);
      LoadPatternsOutput(FRandomOrder[j]);
      Propagate;
      xQuadError := QuadError;
      // перевірка - чи розпізнаний приклад з навчальної множини
      if xQuadError < IdentError then
        Inc(FRecognizedTeachCount);
      FTeachError := FTeachError + xQuadError;
      // максимальна помилка на навчальній множині
      if xNewEpoch then
      begin
        FMaxTeachResidual := xQuadError;
        xNewEpoch := False;
      end
      else
        if MaxTeachResidual < xQuadError then
          FMaxTeachResidual := xQuadError;
      CalcLocalError;
      AdjustWeights;
    end;
    // середня помилка на навчальній множині
    FMidTeachResidual := TeachError/PatternCount;
    // перевірка мережі на узагальнення
    if TestSetPatternCount > 0 then
      CheckTestSet;
    DoOnEpochPassed;
    if StopTeach then
    begin
      TeachStopped := True;
      Exit;
    end;
    Inc(FEpochCurrent);
  end;
  DoOnAfterTeach;
end;

procedure TRadialBasisNeuralNetBP.SetPatternCount(const Value: integer);
begin
  FPatternCount := Value;
  inherited;
end;

procedure TRadialBasisNeuralNetBP.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Alpha := DefaultAlpha;
  ContinueTeach := False;
  Epoch := True;
```

```
  EpochCount := DefaultEpochCount;
  Momentum := DefaultMomentum;
  TeachRate := DefaultTeachRate;
  ResizeInputDim;
  ResizeOutputDim;
end;

{ TRadialBasisNeuralNetExtended }

constructor TRadialBasisNeuralNetExtended.Create(AOwner: TComponent);
begin
  inherited;
  SetDefaultProperties;
end;

destructor TRadialBasisNeuralNetExtended.Destroy;
var
  i: integer;
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  FNeuroDataSource.Free;
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i].Free;
  inherited;
end;

function TRadialBasisNeuralNetExtended.GetFields(Index: integer): TNeuroField;
begin
  Result := FFields[Index];
end;

function TRadialBasisNeuralNetExtended.GetInputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdInput then
      Inc(Result);
end;

function TRadialBasisNeuralNetExtended.GetOutputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdOutput then
      Inc(Result);
end;

function TRadialBasisNeuralNetExtended.GetOutput(Index: integer): double;
var
  xTmp: double;
begin
  with Fields[RealOutputIndex[Index]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := -ln(1/LayersBP[LayerCount - 1].NeuronsBP[Index].Output -
1);
                 LayersBP[LayerCount - 1].NeuronsBP[Index].Output := xTmp *
Dispersion + ValueMid;
               end;
      nrmLinear:  Result := (LayersBP[LayerCount - 1].NeuronsBP[Index].Output +
1)*(ValueMax - ValueMin)/2 + ValueMin;
      nrmLinearOut: Result := LayersBP[LayerCount -
1].NeuronsBP[Index].Output*(ValueMax - ValueMin) + ValueMin;
```

```
      nrmSigmoid:  Result := - Ln(1/LayersBP[LayerCount -
1].NeuronsBP[Index].Output - 1)/Alpha;
    end;
end;

function TRadialBasisNeuralNetExtended.GetRealInputIndex(Index: integer):
integer;
begin
  Result := FRealInputIndex[Index];
end;

function TRadialBasisNeuralNetExtended.GetRealOutputIndex(Index: integer):
integer;
begin
  Result := FRealOutputIndex[Index];
end;

procedure TRadialBasisNeuralNetExtended.ComputeUnPrepData(AVector:
TVectorFloat);
var
  i: integer;
  xTmp: double;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
  with FFields[RealInputIndex[i]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := (LayersBP[SensorLayer].NeuronsBP[i].Output -
ValueMid)/Dispersion;
                 LayersBP[SensorLayer].NeuronsBP[i].Output := 1/(1 + exp(-
xTmp));
               end;
      nrmLinear: LayersBP[SensorLayer].NeuronsBP[i].Output := 2*(AVector[i] -
ValueMin)/(ValueMax - ValueMin) - 1;
      nrmLinearOut: LayersBP[SensorLayer].NeuronsBP[i].Output := (AVector[i] -
ValueMin)/(ValueMax - ValueMin);
      nrmSigmoid: LayersBP[SensorLayer].NeuronsBP[i].Output := 1/( 1 + exp(-
Alpha * AVector[i]));
    end;
  Propagate;
end;

procedure TRadialBasisNeuralNetExtended.DoOnBeforeTeach;
begin
  if InputNeuronCount <> InputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SInFieldCount);
  if OutputNeuronCount <> OutputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SOutFieldCount);
  if InputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  if OutputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SOutNeuronCount);
  if (not FMaxTeachError) and (not FMaxTestError) and
     (not FMidTeachError) and (not FMidTestError) and (not FEpoch) then
    raise EBPStopCondition.Create(SBPStopCondition);
  inherited DoOnBeforeTeach;
end;

procedure TRadialBasisNeuralNetExtended.DoOnEpochPassed;
begin
  if MaxTeachError and (MaxTeachResidual < MaxTeachErrorValue) then
    StopTeach := True;
  if MidTeachError and (MidTeachResidual < MidTeachErrorValue) then
    StopTeach := True;
  if MaxTestError and (MaxTestResidual < MaxTestErrorValue) then
    StopTeach := True;
  if MidTestError and (MidTestResidual < MidTestErrorValue) then
```

```
    StopTeach := True;
  if TeachIdent and (Round((FRecognizedTeachCount * 100)/PatternCount) <
TeachIdentCount) then
    StopTeach := True;
  if TestIdent and (Round((FRecognizedTestCount * 100)/TestSetPatternCount) <
TestIdentCount) then
    StopTeach := True;
  inherited DoOnEpochPassed;
end;

procedure TRadialBasisNeuralNetExtended.LoadDataFrom;
var
  xTempStream: TFileStream;
  i, j: integer;
  xFieldCount: integer;
  xArray: TVectorFloat;
  xPatternsList: TStringList;
begin
  // створюється потік
  xTempStream := TFileStream.Create(FSourceFileName, fmOpenRead);
  // створюється список
  xPatternsList := TStringList.Create;
  xPatternsList.LoadFromStream(xTempStream);
  try
    if SettingsLoaded then
    begin
      xFieldCount := FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
      if AvailableFieldsCount <> xFieldCount then
        if MessageDlg('Кількість полів у файлі даних не відповідає значенню
AvailableFieldsCount'+ #13 + 'Установити нове значення AvailableFieldsCount = '+
IntToStr(xFieldCount),
                      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
          AvailableFieldsCount := xFieldCount;
    end
    else
      AvailableFieldsCount :=
FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
    FNeuroDataSource.ExtractHeaders(FFields, xPatternsList.Strings[0]);
    // установлюється розмірність часового масиву
    SetLength(xArray, FAvailableFieldsCount);
    // установлюється розмірність масиву даних
    if FUseForTeach = 100 then
      PatternCount := xPatternsList.Count - 1
    else
    begin
      PatternCount := Round((xPatternsList.Count - 1) * FUseForTeach / 100);
      TestSetPatternCount := xPatternsList.Count - PatternCount - 1;
    end;
    for i := 0 to FAvailableFieldsCount - 1 do
      FFields[i].DataInCount := xPatternsList.Count - 1;
    for j := 0 to xPatternsList.Count - 2 do
    begin
      FNeuroDataSource.ExtractValues(xArray, xPatternsList.Strings[j + 1]);
      for i := 0 to FAvailableFieldsCount - 1 do
        FFields[i].DataIn[j] := xArray[i]
    end;
  finally
    xTempStream.Free;
    xPatternsList.Free;
    SetLength(xArray, 0);
    xArray := nil;
  end;
end;

procedure TRadialBasisNeuralNetExtended.LoadPhase1;
begin
  FSourceFileName := FNnwFile.ReadString('Phase1', 'LearnSampleFileName', '');
  FNeuroDataSource.Name := FSourceFileName;
end;
```

```
procedure TRadialBasisNeuralNetExtended.LoadPhase2;
var
  i: integer;
begin
  AvailableFieldsCount := FNnwFile.ReadInteger('Phase2', 'AvailableFieldsCount',
1);
  for i := 0 to AvailableFieldsCount - 1 do
  with FFields[i] do
  begin
    Name := FNnwFile.ReadString('Phase2', 'FieldName_'+IntToStr(i), '');
    Kind := FNnwFile.ReadInteger('Phase2', 'FieldType_'+IntToStr(i), 0);
    NormType := FNnwFile.ReadInteger('Phase2', 'NormType_'+IntToStr(i), 0);
    ValueMax := FNnwFile.ReadFloat('Phase2', 'Max_'+IntToStr(i), 0);
    ValueMin := FNnwFile.ReadFloat('Phase2', 'Min_'+IntToStr(i), 0);
    ValueMid := FNnwFile.ReadFloat('Phase2', 'Mid_'+IntToStr(i), 0);
    Dispersion := FNnwFile.ReadFloat('Phase2', 'Disp_'+IntToStr(i), 0);
    Alpha := FNnwFile.ReadFloat('Phase2', 'Alpha_'+IntToStr(i), 0);
    Ind := FNnwFile.ReadBool('Phase2', 'Ind_'+IntToStr(i), False);
  end;
  SettingsLoaded := True;
end;

procedure TRadialBasisNeuralNetExtended.LoadPhase4;
begin
  UseForTeach := FNnwFile.ReadInteger('Phase4', 'UseForTeach',
DefaultUseForTeach);
  IdentError:= FNnwFile.ReadFloat('Phase4', 'IdentErr', DefaultErrorValue);
  TestAsValid := FNnwFile.ReadBool('Phase4', 'TestAsValid', False);
  Epoch:= FNnwFile.ReadBool('Phase4', 'Epoch', False);
  EpochCount:= FNnwFile.ReadInteger('Phase4', 'Epoch', DefaultEpochCount);
  MaxTeachError:= FNnwFile.ReadBool('Phase4', 'MaxTeachErr', False);
  MaxTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTeachErr',
DefaultErrorValue);
  MidTeachError:= FNnwFile.ReadBool('Phase4', 'MidTeachErr', False);
  MidTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTeachErr',
DefaultErrorValue);
  TeachIdent:= FNnwFile.ReadBool('Phase4', 'TeachIdent', False);
  TeachIdentCount:= FNnwFile.ReadInteger('Phase4', 'TeachIdent',
DefaultTeachIdentCount);
  MaxTestError:= FNnwFile.ReadBool('Phase4', 'MaxTestErr', False);
  MaxTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTestErr',
DefaultErrorValue);
  MidTestError:= FNnwFile.ReadBool('Phase4', 'MidTestErr', False);
  MidTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTestErr',
DefaultErrorValue);
  TestIdent:= FNnwFile.ReadBool('Phase4', 'TestIdent', False);
  TestIdentCount:= FNnwFile.ReadInteger('Phase4', 'TestIdent',
DefaultTestIdentCount);
end;

procedure TRadialBasisNeuralNetExtended.LoadNetwork;
var
  i,j,k: integer;
  xLayerCount: integer;
begin
  // знищується поточна конфігурація нейромережі
  ResetLayers;
  Alpha := FNnwFile.ReadFloat('Network', 'Alpha', DefaultAlpha);
  Momentum := FNnwFile.ReadFloat('Network', 'Miu', DefaultMomentum);
  TeachRate := FNnwFile.ReadFloat('Network', 'TeachSpeed', DefaultTeachRate);
  EpochCount := FNnwFile.ReadInteger('Network', 'Epoch', DefaultEpochCount);
  xLayerCount := FNnwFile.ReadInteger('Network','CountLayers',
DefaultLayerCount);
  // задається кількість нейронів у шарах
  AutoInit := False;
  for i := 0 to xLayerCount - 1 do
    AddLayer(FNnwFile.ReadInteger('Network','Layer_'+IntToStr(i),
DefaultNeuronCount));
```

```
  AutoInit := True;
  // ініціалізація нової конфігурації нейромережі
  Init;
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Weights[k] := FNnwFile.ReadFloat('Network',
                              'W_'+IntToStr( i-
1)+'_'+IntToStr(k)+'_'+IntToStr(j), 0);
      LayersBP[i].NeuronsBP[j].Weights[LayersBP[ i-1].NeuronCount] :=
FNnwFile.ReadFloat('Network',
       'WT_'+IntToStr( i-1)+'_'+IntToStr(j), 0);
    end;
end;

procedure TRadialBasisNeuralNetExtended.SavePhase1;
begin
  FNnwFile.WriteString('Phase1', 'LearnSampleFileName', FNeuroDataSource.Name);
end;

procedure TRadialBasisNeuralNetExtended.SavePhase2;
var
  i: integer;
begin
  FNnwFile.WriteInteger('Phase2', 'AvailableFieldsCount',
FAvailableFieldsCount);
  for i := 0 to AvailableFieldsCount - 1 do
  with Fields[i] do
  begin
    FNnwFile.WriteString('Phase2', 'FieldName_'+IntToStr(i), Name);
    FNnwFile.WriteInteger('Phase2', 'FieldType_'+IntToStr(i), Kind);
    FNnwFile.WriteInteger('Phase2', 'NormType_'+IntToStr(i), NormType);
    FNnwFile.WriteFloat('Phase2', 'Max_'+IntToStr(i), ValueMax);
    FNnwFile.WriteFloat('Phase2', 'Min_'+IntToStr(i), ValueMin);
    FNnwFile.WriteFloat('Phase2', 'Mid_'+IntToStr(i), ValueMid);
    FNnwFile.WriteFloat('Phase2', 'Disp_'+IntToStr(i), Dispersion);
    FNnwFile.WriteFloat('Phase2', 'Alpha_'+IntToStr(i), Alpha);
    FNnwFile.WriteBool('Phase2', 'Ind_'+IntToStr(i), Ind);
  end;
end;

procedure TRadialBasisNeuralNetExtended.SavePhase4;
begin
  FNnwFile.WriteBool('Phase4', 'Epoch', Epoch);
  FNnwFile.WriteInteger('Phase4', 'Epoch', EpochCount);
  FNnwFile.WriteFloat('Phase4', 'IdentErr', IdentError);
  FNnwFile.WriteBool('Phase4', 'MaxTeachErr', MaxTeachError);
  FNnwFile.WriteFloat('Phase4', 'MaxTeachErr', MaxTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MaxTestErr', MaxTestError);
  FNnwFile.WriteFloat('Phase4', 'MaxTestErr', MaxTestErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTeachErr', MidTeachError);
  FNnwFile.WriteFloat('Phase4', 'MidTeachErr', MidTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTestErr', MidTestError);
  FNnwFile.WriteFloat('Phase4', 'MidTestErr', MidTestErrorValue);
  FNnwFile.WriteFloat('Phase4', 'Miu', Momentum);
  FNnwFile.WriteBool('Phase4', 'TeachIdent', TeachIdent);
  FNnwFile.WriteFloat('Phase4', 'TeachSpeed', TeachRate);
  FNnwFile.WriteInteger('Phase4', 'TeachIdent', TeachIdentCount);
  FNnwFile.WriteBool('Phase4', 'TestAsValid', TestAsValid);
  FNnwFile.WriteBool('Phase4', 'TestIdent', TestIdent);
  FNnwFile.WriteInteger('Phase4', 'TestIdent', TestIdentCount);
  FNnwFile.WriteInteger('Phase4', 'UseForTeach', UseForTeach);
end;

procedure TRadialBasisNeuralNetExtended.SaveNetwork;
var
  i, j, k: integer;
```

```
begin
  FNnwFile.WriteFloat('Network', 'TeachSpeed', TeachRate);
  FNnwFile.WriteFloat('Network', 'Miu', Momentum);
  FNnwFile.WriteFloat('Network', 'Alpha', Alpha);
  FNnwFile.WriteInteger('Network', 'Epoch', EpochCount);
  FNnwFile.WriteInteger('Network','CountLayers', LayerCount);
  // задається кількість нейронів у шарах
  for i := 0 to LayerCount - 1 do
    FNnwFile.WriteInteger('Network','Layer_'+IntToStr(i),
StrToInt(NeuronsInLayer[i]));
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to StrToInt(NeuronsInLayer[i]) - 1 do
    begin
      for k := 0 to StrToInt(NeuronsInLayer[ i-1]) do
        FNnwFile.WriteFloat('Network','W_'+IntToStr( i-1)+'_'+IntToStr(k)+
                            '_'+IntToStr(j),
LayersBP[i].NeuronsBP[j].Weights[k]);
      FNnwFile.WriteFloat('Network','WT_'+IntToStr( i-1)+'_'+IntToStr(j),
         LayersBP[i].NeuronsBP[j].Weights[StrToInt(NeuronsInLayer[j])]);
   end;
end;

procedure TRadialBasisNeuralNetExtended.NormalizeData;
var
  i: integer;
begin
  // нормалізація вхідних і вихідних значень
  for i := 0 to FAvailableFieldsCount - 1 do
  begin
    FFields[i].FindMinMax;
    FFields[i].Normalize;
  end;
end;

procedure TRadialBasisNeuralNetExtended.Train;
var
  i, j, k: integer;
begin
  if FUseForTeach = 100 then
  begin
    PatternCount := FFields[0].DataInCount;
    TestSetPatternCount := 0;
  end
  else
  begin
    PatternCount := Round((FFields[0].DataInCount - 1) * FUseForTeach / 100);
    TestSetPatternCount := FFields[0].DataInCount - PatternCount;
  end;
  if not TeachStopped then
    NormalizeData;
  // формування вхідних значень навчальної множини
  RealOutputIndexCount := OutputFieldCount;
  RealInputIndexCount := InputFieldCount;
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsInput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealInputIndex[k] := i;
      Inc(k);
    end;
  // формування вихідних значень навчальної множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
```

```
      for j := 0 to PatternCount - 1 do
        FPatternsOutput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealOutputIndex[k] := i;
      Inc(k);
    end;
  // формування вхідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatterns[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // формування вихідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatternsOut[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // навчання або донавчання мережі
  TeachOffLine;
end;

procedure TRadialBasisNeuralNetExtended.SetAvailableFieldsCount(Value :
integer);
var
  i: integer;
begin
  FAvailableFieldsCount := Value;
  // встановлюється кількість полів
  SetLength(FFields, Value);
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i] := TNeuroField.Create;
end;

procedure TRadialBasisNeuralNetExtended.SetFields(Index: integer; Value:
TNeuroField);
begin
  try
    FFields[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SFieldIndexRange, [Index])
  end;
end;

procedure TRadialBasisNeuralNetExtended.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Epoch := False;
  IdentError:= DefaultErrorValue;
  MaxTeachError := False;
  MaxTeachErrorValue := DefaultErrorValue;
  MaxTestError:= False;
  MaxTestErrorValue:= DefaultErrorValue;
  MidTestError:= False;
  MidTestErrorValue:= DefaultErrorValue;
  MidTeachError := False;
  MidTeachErrorValue :=  DefaultErrorValue;
  SettingsLoaded := False;
  TeachIdent := False;
  TeachIdentCount:= DefaultTeachIdentCount;
  TestAsValid := False;
  TestIdent:= False;
```

```
  TestIdentCount:= DefaultTestIdentCount;
  UseForTeach := DefaultUseForTeach;
end;

procedure TRadialBasisNeuralNetExtended.SetFileName(Value: TFilename);
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  try
    FNnwFile := TIniFile.Create(Value);
    FFileName := Value;
  except
    on E: EInOutError do
      raise E.CreateFmt(SWrongFileName, [Value]);
  end;
  FNeuroDataSource := TNeuroDataSource.Create;
  LoadPhase1;
  LoadPhase2;
  LoadPhase4;
  LoadNetwork;
  LoadDataFrom;
end;

procedure TRadialBasisNeuralNetExtended.SetTeachIdentCount(const Value:
integer);
begin
  if (Value <= 0) or (Value > 100) then
    FTeachIdentCount := DefaultTeachIdentCount
  else
    FTeachIdentCount := Value;
end;

procedure TRadialBasisNeuralNetExtended.SetUseForTeach(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FUseForTeach := DefaultUseForTeach
  else
    FUseForTeach := Value;
end;

procedure TRadialBasisNeuralNetExtended.SetRealOutputIndex(Index: integer; const
Value: integer);
begin
  FRealOutputIndex[Index] := Value;
end;

procedure TRadialBasisNeuralNetExtended.SetRealOutputIndexCount(const Value:
integer);
begin
  SetLength(FRealOutputIndex, Value)
end;
procedure TRadialBasisNeuralNetExtended.SetRealInputIndex(Index: integer; const
Value: integer);
begin
  FRealInputIndex[Index] := Value;
end;
procedure TRadialBasisNeuralNetExtended.SetRealInputIndexCount(const Value:
integer);
begin
  SetLength(FRealInputIndex, Value)
end;
procedure Register;
begin
  RegisterComponents('Neural_network_', [TRadialBasisNeuralNetHopf,
TRadialBasisNeuralNetBP, TRadialBasisNeuralNetExtended]);
end;
end.
```

Pumpdata.pas – формування бази знань

```
unit PumpData;

interface

uses
  SysUtils, IniFiles, Classes, Neural_network_Types;

type

  EFieldNormError = class(Exception);
  EFieldKindError = class(Exception);

  TNeuroField = class;
  TNeuroFields = array of TNeuroField;

  TNeuroField = class(TObject)
  private
    FAlpha: double;
    FDataIn: TVectorFloat;
    FDispersion: double;
    FInd: boolean;
    FKind: byte;
    FName: string;
    FNormType: byte;
    FValueMax: double;
    FValueMid: double;
    FValueMin: double;
    function GetDataIn(Index: integer): double;
    function GetKindName: TNeuroFieldType;
    function GetNormTypeName: TNormalize;
    function GetDataInCount: integer;
    procedure SetDataIn(Index: integer; Value: double);
    procedure SetKind(Value: byte);
    procedure SetNormType(Value: byte);
    procedure SetDataInCount(Value: integer);
  public
    procedure FindMinMax;
    procedure CalcMid;
    procedure CalcDispersion;
    procedure Normalize;
    procedure DeNormalize;
    property Alpha: double read FAlpha write FAlpha;
    property DataIn[Index: integer]: double read GetDataIn write SetDataIn;
    property DataInCount: integer read GetDataInCount write SetDataInCount;
    property Dispersion: double read FDispersion write FDispersion;
    property Ind: boolean read FInd write FInd;
    property Kind: byte read FKind write SetKind;
    property KindName: TNeuroFieldType read GetKindName;
    property Name: string read FName write FName;
    property NormType: byte read FNormType write SetNormType;
    property NormTypeName: TNormalize read GetNormTypeName;
    property ValueMax: double read FValueMax write FValueMax;
    property ValueMin: double read FValueMin write FValueMin;
    property ValueMid: double read FValueMid write FValueMid;
  end;

  TNeuroDataSource = class(TObject)
  private
    FName: TFileName;
    function IsHeaderChar(AValue: char): boolean;
  public
    function FieldCount(AHeader: string): integer;
    procedure ExtractHeaders(const AFields: TNeuroFields; AHeader: string);
    procedure ExtractValues(const AVector:TVectorFloat; AHeader: string);
    property Name: TFileName read FName write FName;
  end;
```

```
implementation

{ Клас TNeuroField }

function TNeuroField.GetDataIn(Index: integer): double;
begin
  Result := FDataIn[Index];
end;

function TNeuroField.GetDataInCount: integer;
begin
  Result := High(FDataIn) + 1;
end;

function TNeuroField.GetKindName: TNeuroFieldType;
begin
  case FKind of
    0 : Result := fdInput;
    1 : Result := fdOutput;
    2 : Result := fdNone;
  end;
end;

function TNeuroField.GetNormTypeName: TNormalize;
begin
  case FNormType of
    0 : if KindName = fdInput then
          Result := nrmLinear
        else if KindName = fdOutput then
          Result := nrmLinearOut;
    1 : Result := nrmSigmoid;
    2 : Result := nrmAuto;
    3 : Result := nrmNone;
  end;
end;

procedure TNeuroField.CalcMid;
var
  i: integer;
begin
  FValueMid := 0;
  for i := Low(FDataIn) to High(FDataIn) do
    FValueMid := FValueMid + FDataIn[i];
  FValueMid := FValueMid/(High(FDataIn) + 1);
end;

procedure TNeuroField.CalcDispersion;
var
  i: integer;
begin
  if High(FDataIn) > 1 then
  begin
    FDispersion := 0;
    for i := Low(FDataIn) to High(FDataIn) do
      FDispersion := FDispersion + sqr(FDataIn[i] - ValueMid);
    FDispersion := sqrt(FDispersion/High(FDataIn));
  end
  else
    FDispersion := 0;
end;

(*procedure TNeuroField.DeNormalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
```

```
                FDataIn[i] := (FDataIn[i] + 1)*(FValueMax - FValueMin)/2 +
FValueMin;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := FDataIn[i]*(FValueMax - FValueMin) + FValueMin;
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := - Ln(1/FDataIn[i] - 1)/Alpha;
  end;
end;*)

procedure TNeuroField.FindMinMax;
var
  i: integer;
begin
  FValueMax:= FDataIn[0];
  FValueMin:= FDataIn[0];
  for i := 1 to High(FDataIn) do
  begin
    if FValueMin > FDataIn[i] then
      FValueMin := FDataIn[i];
    if FValueMax < FDataIn[i] then
      FValueMax := FDataIn[i]
  end;
end;

procedure TNeuroField.Normalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmAuto:  begin
                CalcMid;
                CalcDispersion;
                for i := Low(FDataIn) to High(FDataIn) do
                begin
                  xTmp := (FDataIn[i] - FValueMid)/FDispersion;
                  FDataIn[i] := 1/(1 + exp(-xTmp));
                end;
              end;
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 2*(FDataIn[i] - FValueMin)/(FValueMax - FValueMin)-
1;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
              FDataIn[i] := (FDataIn[i] - FValueMin)/(FValueMax - FValueMin);
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
              FDataIn[i] := 1/( 1 + exp(-Alpha * FDataIn[i]));
  end;
end;

procedure TNeuroField.SetNormType(Value: byte);
begin
  if (Value < 0) or (Value > 3) then
    raise EFieldNormError.CreateFmt(SFieldNorm, [Value])
  else
    FNormType := Value;
end;

procedure TNeuroField.SetKind(Value: byte);
begin
  if (Value < 0) or (Value > 2) then
    raise Exception.CreateFmt(SFieldKind, [Value])
  else
    FKind := Value;
end;

procedure TNeuroField.SetDataIn(Index: integer; Value: double);
begin
  FDataIn[Index] := Value;
end;
```

```
procedure TNeuroField.SetDataInCount(Value: integer);
begin
  SetLength(FDataIn, Value)
end;

{ Клас TNeuroDataSource }

function TNeuroDataSource.IsHeaderChar(AValue: char): boolean;
begin
  if (AValue in Letters) or (AValue in Capitals) or (AValue in DigitChars) then
    Result := True
  else
    Result := False;
end;

procedure TNeuroDataSource.ExtractValues(const AVector:TVectorFloat; AHeader:
string);
var
  s: string;
  i, xCurPos: integer;
begin
  i := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  try
    while xCurPos > 0 do
    begin
      s := Copy(AHeader, 1, xCurPos - 1);
      AVector[i] := StrToFloat(s);
      Inc(i);
      Delete(AHeader, 1, xCurPos - 1);
      AHeader := Trim(AHeader);
      xCurPos := Pos(SpaceChar, AHeader);
    end;
    s := AHeader;
    AVector[i] := StrToFloat(s);
  except
    on EConvertError do
      EConvertError.CreateFmt(SCannotBeNumber, [s])
  end;
end;

procedure TNeuroDataSource.ExtractHeaders(const AFields: TNeuroFields; AHeader:
string);
var
  s: string;
  xFieldCount, j, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    s := Copy(AHeader, 1, xCurPos - 1);
    AFields[xFieldCount].FName := '';
    for j := 1 to Length(s) do
      if isHeaderChar(s[j]) then
        AFields[xFieldCount].FName := AFields[xFieldCount].FName + s[j];
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  AFields[xFieldCount].FName := '';
  for j := 1 to Length(AHeader) do
    if isHeaderChar(AHeader[j]) then
      AFields[xFieldCount].FName := AFields[xFieldCount].FName + AHeader[j];
```

```
  { повертає кількість полів }
end;

function TNeuroDataSource.FieldCount(AHeader: string): integer;
var
  xFieldCount, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  { повертає кількість полів }
  Result := xFieldCount + 1;
end;


end.
```