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 створення та навчання нейронної мережі для розпізнавання образів”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-2
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Михайлов Д.В.
«____» _____ 2023 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 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 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

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

Михайлову Дмитру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів*

2. Керівник роботи *Петренюк Володимир Ілліч, канд. фіз.-мат. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 33-13 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 створення та навчання нейронної мережі для розпізнавання 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

Михайлов Д.В. Дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи створення та навчання нейронної мережі для розпізнавання образів.

Метою розробки є дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів.

Об'єктом дослідження є процес створення та навчання нейронної мережі для розпізнавання образів.

Предметом дослідження є методи створення та навчання нейронної мережі для розпізнавання образів.

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

Результат роботи – програмна реалізація системи створення та навчання нейронної мережі для розпізнавання 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'ютерні науки, розпізнавання образів

ABSTRACT

Mykhailov D.V. Research and software implementation of a system for creating and training a neural network for pattern recognition. 122 Computer Science. 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 system of creating and training a neural network for pattern recognition.

The purpose of the development is research and software implementation of a system for creating and training a neural network for pattern recognition.

The object of research is the process of creating and training a neural network for pattern recognition.

The subject of research is the methods of creating and training a neural network for pattern recognition.

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

The result of the work is a software implementation of a system for creating and training a neural network 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 science, 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. Розпізнавання зображень – це здатність комп'ютерів ідентифікувати та класифікувати конкретні об'єкти, місця, людей, текст і дії в цифрових зображеннях і відео. Як додаток комп'ютерного зору, програмне забезпечення для розпізнавання зображень працює шляхом аналізу та обробки візуального вмісту зображення чи відео та порівняння його з отриманими даними, що дозволяє програмному забезпеченню автоматично «бачити» та інтерпретувати те, що є, так, як це може робити людина. здатний.

Розпізнавання зображень – це застосування комп'ютерного зору, у якому машини ідентифікують і класифікують конкретні об'єкти, людей, текст і дії в цифрових зображеннях і відео. По суті, це здатність комп'ютерного програмного забезпечення «бачити» та інтерпретувати речі у візуальних медіа так, як це може зробити людина.

Розпізнавання зображень є невід'ємною частиною технології, яку ми використовуємо щодня – від функції розпізнавання обличчя, яка розблоковує смартфони, до мобільних чекових депозитів у банківських програмах. Він також широко використовується в таких сферах, як медична візуалізація для виявлення пухлин, зламаних кісток та інших аберацій, а також на фабриках для виявлення дефектної продукції на конвеєрі.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів.

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

– існуючих систем створення та навчання нейронної мережі для розпізнавання образів.

– Дослідження системи створення та навчання нейронної мережі для розпізнавання образів.

– Програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів.

Об'єктом дослідження є процес створення та навчання нейронної мережі для розпізнавання образів.

Предметом дослідження є методи створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання образів.

– Розроблено вітчизняний продукт створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання 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

Програмне забезпечення, яке розробляється у даному магістерському проекті, призначено для реалізації системи створення та навчання нейронної мережі для розпізнавання образів**.**

І символи й образи – це «слова», які обробляють комп'ютери, а основне розходження між ними полягає лише в розмірності. При цьому розмір образу може бути на багато порядків більше розміру символу. Здавалося б, різниця не дуже значна й приводить лише до трохи більшого часу обробки довгих слів, але насправді розходження в розмірах даних мають принципове значення, тому що складність роботи з образами зростає нелінійно при збільшенні їхньої розрядності.

Якщо для відносно коротких символів можна описати всі можливі над ними операції й створити процесор, що передбачуваним образом обробляє всі вхідні символи, що виконують роль команд або даних, то реалізувати те ж саме для образів неможливо, оскільки подібний опис буде рости експоненційно. А виходить, будь-який процесор, призначений для обробки образів, містить лише частину можливих вхідних зразків і відповідних їм дій і повинен «додумувати» своє поводження й узагальнювати відомі йому приклади, щоб його реакція була аналогічної й прийнятної з пу рішення задачі, для якої він призначений. Таким чином, розходження між послідовними й паралельними обчисленнями полягає в принципово різних методах постановки й рішення задач, пов'язаних з обробкою інформації.

На принципі послідовних обчислень на обмежені по довжині символах засновані комп'ютери, реалізовані по традиційній архітектурі фон Неймана з алгоритмічними програмами, а паралельні обчислення й розпізнавання образів лежать в основі нейрокомп’ютерів, організованих по принципах, схожим із

пристроєм і роботою мозку. Сучасні електронно-обчислювальні машини значно перевершують людей по здатності робити чисельні розрахунки, однак людина може з легкістю й буквально за секунду довідатися людини, особа якого промайнуло в юрбі й з яким він не бачився багато років.

Як уже було сказано, основна задача нейрокомп'ютерів – обробка образів. При цьому в них, як і в мозку, відсутні загальні шини, немає поділу на активний процесор і пасивну пам'ять, а обчислення й навчання розподілені по всіх елементарних процесорах – нейронам, які функціонують паралельно. За рахунок цього нейрокомп'ютери дозволяють домогтися фантастичної продуктивності, що може в мільйони разів перевищувати продуктивність традиційних комп'ютерів з послідовною архітектурою.

Переваги нейромережного підходу полягають у наступному:

– паралелізм обробки інформації;

– єдиний і ефективний принцип навчання;

– надійність функціонування;

– здатність вирішувати неформалізовані задачі.

Біологічна еволюція, що привела до настільки ефективних рішень, ішла по шляху від образів до логіки. Так і людина після народження спочатку вчиться розпізнавати образи, а тільки потім здобуває вміння міркувати логічно й будувати алгоритми. Комп'ютери ж, навпроти, почавши з логіки, лише через кілька десятиліть освоюють розпізнавання образів за рахунок створення спеціальних програм для комп'ютерів традиційної архітектури або завдяки створенню спеціалізованих апаратних нейропроцесорів.

1.2 Область застосування

Областю застосування є системи розпізнання образів. Розпізнавання зображень – це підмножина комп'ютерного зору, яка є ширшою сферою штучного інтелекту, яка навчає комп'ютери бачити, інтерпретувати та розуміти візуальну інформацію із зображень або відео.

Комп’ютерний зір включає кілька «підпроблем» або «завдань», як висловився Кханна, включаючи класифікацію зображень або призначення однієї мітки зображенню, наприклад «автомобіль» або «яблуко». Це призводить до більш складних завдань, таких як виявлення об’єктів, що включає не лише ідентифікацію багатьох речей на зображенні (автомобілів, тварин, дерев тощо), але й локалізацію їх у межах сцени. А ще є сегментація сцени, коли машина класифікує кожен піксель зображення чи відео та визначає, який там об’єкт, що дозволяє легше ідентифікувати аморфні об’єкти, як-от кущі, небо чи стіни.

Розпізнавання зображень є ще одним завданням комп’ютерного зору. Його алгоритми розроблені для аналізу вмісту зображення та класифікації його за певними категоріями або мітками, які потім можна використовувати.

У багатьох випадках багато технологій, які використовуються сьогодні, були б неможливими без розпізнавання зображень і, як наслідок, комп’ютерного зору.

Наприклад, щоб застосувати доповнену реальність, або AR, машина повинна спочатку зрозуміти всі об’єкти в сцені, як з точки зору того, що вони собою являють, так і щодо того, де вони знаходяться по відношенню один до одного. Якщо машина не може адекватно сприймати середовище, в якому вона знаходиться, вона не зможе застосувати AR поверх нього. Те саме стосується безпілотних автомобілів і автономних мобільних роботів.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання 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 існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Актуальна діяльність компаній, орієнтованих на web-розробки й технології. Такими є Google, Twitter, FaceBook і багато хто інші. Google Inc. застосовує технології розпізнавання осіб і образів для більше розумного пошуку, Twitter оцінює настрій пишучих в онлайн-блогах людей, FaceBook недавно представив суспільству нове удосконалення за назвою Tagger, що автоматично розпізнає, відзначає й підписує особи друзів користувача соціальної мережі.

Розроблювачі компанії Face.com відомі своїми розробками, а також своєї власної SDK на JavaScript, що використовує PhotoTagger.

Недавно даною компанією була представлена технологія, що дозволяє ідентифікувати людей по фотографіях, опублікованим на Інтернет-сайтах. Програма PhotoFinder аналізує цифрові зображення, знайдені на сторінках глобальної мережі й порівнює їх з еталонним зображенням шуканої особи, дійсність якого не викликає сумніву. Для обробки еталона використовується алгоритм, що ґрунтується на унікальному розташуванні різних частин особи – очей, носа й рота. Начебто нічого нового, але таке програмне забезпечення дозволяє сконструювати величенну базу користувачів мережі Інтернет. Дана технологія реалізована у вигляді віджетів і може вбудовуватися творцями web-порталів на їхні сайти. Пошук ведеться як по фотографіях, так і по відео. Даний сервіс уже оцінили такі компанії як Flickr і YouTube. Прихильники технології вважають, що розробка спростить пошук і встановлення особистості. Але є політика конфіденційності, що порушувати не можна.

Не відступають і компанії, що представляють дані інновації у своїх продуктах, таких як цифрові фото– і відеокамери. Таких компаній дуже багато, серед них Ricoh, Fujifilm, Canon, Nikon® і інші.

Google Cloud Vision API

Виявляйте та класифікуйте кілька об’єктів, зображень тощо за допомогою попередньо навченого Vision API Google Cloud або спеціально навченого Vision AutoML. Google Cloud Vision AI допомагає розробникам легко використовувати потужність машинного навчання, щоб розуміти зображення з найвищою в галузі точністю передбачення.

Користувачі:

- Інженер-програміст.

Галузі промисловості:

- Програмне забезпечення.
- Інформаційні технології та послуги.

Сегмент ринку:

- 44% малий бізнес.
- 23% середнього ринку.

Vue.ai

Vue.ai – це наскрізна платформа автоматизації роздрібної торгівлі, якій довіряють понад 100 роздрібних торговців по всьому світу, включаючи Diesel, Nordstrom, Tata Cliq, Mercado Libre, ThredUp та багато інших. Vue.ai переробляє майбутнє роздрібної торгівлі за допомогою штучного інтелекту. Використовуючи алгоритми Visual AI і ML, набір продуктів Vue.ai вирішує найбільші проблеми роздрібної торгівлі – від підвищення продуктивності до підвищення доходів. Наша платформа штучного інтелекту використовується для: Автоматизованого керування каталогом Автоматизованої модерації зображень (для торговельних майданчиків) Автоматизованих зображень на моделях Створення стилю та обладнання за допомогою ШІ Динамічної персоналізації 1:1 Персоналізованих подорожей покупців

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Одяг і мода.
- Роздрібна торгівля.

Сегмент ринку:

- 41% середнього ринку.
- 29% малий бізнес.

Clarifai

Clarifai надає повну стекову платформу штучного інтелекту для розробників і команд, щоб швидко та спільно впроваджувати штучний інтелект із зображенням, мовою та аудіо. Ми пропонуємо комп'ютерне бачення, магістерські програми та аудіомоделі на одній платформі, яка підтримує повний життєвий цикл штучного інтелекту, включаючи підготовку та керування даними, навчання й оцінку моделей, а також операції моделі. Розробники можуть використовувати готові моделі, створювати власні моделі або використовувати одну з багатьох програм LLM з відкритим кодом або сторонніх розробників. Моделі можна комбінувати для вирішення складніших проблем і мультимодальних варіантів використання за допомогою нашого API або простого у використанні інтерфейсу користувача. Створіть один раз і розгорніть масштабований виробничий штучний інтелект корпоративного рівня в хмарі, локально, на голому металі чи гібриді. Clarifai завершує важку інженерну роботу зі створення масштабованої, надійної та безпечної платформи ШІ, щоб ви могли зосередитися на створенні ШІ у своїх програмах. Увійдіть і запустіть свою першу програму зі штучним інтелектом менш ніж за п'ять хвилин.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 47% середнього ринку.
- 35% малий бізнес.

V7

V7 – це потужна платформа для навчання штучному інтелекту, яка дає змогу коментувати зображення, відео, документи та файли медичних зображень. Це найшвидший спосіб отримати високоякісні анотовані дані для навчання ваших моделей комп'ютерного зору. Ви можете використовувати функції автоматичного маркування, щоб прискорити свої проекти анотацій, керувати своїми робочими процесами MLOps або найняти професійних анотаторів для допомоги. – Ідеальне автоматичне маркування даних, включаючи інструменти сегментації штучного інтелекту та семантичні маски. – Інтуїтивно зрозуміле керування набором даних, робочим процесом і моделлю для кращого контролю над процесом розробки ML. – Анотаційні послуги через офіційних партнерів, які мають досвід роботи в нішевих галузях. Завдяки розширеним функціям автоматичного маркування, контролю якості (QA) і інструментам командної співпраці V7 допоможе вам скоротити час анотації даних у 10 разів і допоможе розробити точніші рішення штучного інтелекту. З легкістю додавайте анотації до будь-якого набору даних і плавно інтегруйте V7 у систему керування навчальними даними та розробки моделей. – Створення автоматичних міток на рівні пікселів для сегментації екземплярів. – Додавайте анотації до довгих відео з понад 100 тисячами кадрів і тисячами анотацій. – Використовуйте багатопланові види та вокселі для медичних зображень і об'ємних даних. – Додайте анотації до таких форматів медичних файлів, як DICOM і NIfTI, або зображення цілого слайда. – Запустіть свої дані через загальнодоступні інструменти ML, підключіть зовнішні моделі або навчіть власний штучний інтелект у хмарі. – Скануйте текст будь-якою мовою, класифікуйте файли, виявляйте об'єкти та витягуйте відповідну інформацію за допомогою ШІ. – Додайте ключові точки та скелети для розпізнавання активності або оцінки пози. – Призначайте ролі користувачів,

керуйте правами доступу та розподіляйте завдання анотацій. Ви можете бездоганно інтегрувати V7 із наявним стеком технологій і з легкістю імпортувати/експортувати анотації. Крім того, V7 дає вам повний контроль над своїми моделями, завданнями та наборами даних через відкритий API, Darwin-py SDK і CLI. Спробуйте один із найкращих інструментів MLO та маркування даних для комп'ютерного зору безкоштовно: https://www.v7labs.com/sign-up

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Інформаційні технології та послуги.
- Програмне забезпечення.

Сегмент ринку:

- 55% малий бізнес.
- 35% середнього ринку.

Encord

Encord – це наскрізна платформа для розблокування ШІ з ваших даних. Безпечно розробляйте, тестуйте та розгортайте передбачувані та генеративні системи штучного інтелекту в масштабі, щоб розкрити цінність машинного навчання. Створюйте високоякісні навчальні дані, використовуйте канали активного навчання, оцінюйте якість моделі, точно налаштовуйте моделі та багато іншого в одній простій у використанні платформі. Анотування. Ефективно позначайте будь-яку візуальну модальність і керуйте великомасштабними групами анотацій за допомогою настроюваних робочих процесів і інструментів контролю якості. Активний – тестуйте, перевіряйте та оцінюйте свої моделі та поверхню, керуйте та визначайте пріоритети для найцінніших даних для маркування, щоб підвищити продуктивність моделі. Apollo – навчайте, налаштовуйте та керуйте пропрієтарними та базовими моделями в масштабі для виробничих додатків ШІ. Accelerate – спеціалізовані послуги маркування за запитом, які допоможуть вам масштабуватись. Encord користується довірою

команд-новаторів штучного інтелекту в RapidAI, Tractable, Stanford Medicine, Memorial, King's College London, NHS, UHN, Royal Navy, Veo та багатьох інших глобальних компаній.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Програмне забезпечення.
- Лікарня та охорона здоров'я.

Сегмент ринку:

- 52% малий бізнес.
- 40% середнього ринку.

Kili

Технологія Kili – це комплексний інструмент для маркування, за допомогою якого ви можете швидко позначати свої навчальні дані, знаходити та виправляти проблеми у своєму наборі даних, а також спрощувати операції з маркування. Технологія Kili значно прискорює створення надійного ШІ. Щоб швидко створювати навчальні набори даних за допомогою технології Kili, ви можете використовувати: – Настроювані інтерфейси для всіх ваших даних: технологія Kili підтримує зображення, відео, текст, документи PDF, супутникові зображення та розмови. Використовуйте розширений UX, який пришвидшує маркування та запобігає помилкам тегування. Налаштуйте інтерфейси та правила перевірки на основі вашого випадку використання та процесу маркування. – Потужні робочі процеси для швидкого й точного додавання анотацій: керуйте своєю чергою маркування, щоб визначити пріоритетність активів, призначити активи певним етикеткам або додати правила перевірки. Налаштуйте конвеєр перегляду, щоб виявити невідповідності та надіслати ресурси назад до розміщувачів. – Інструменти автоматизації для прискорення маркування: застосовуйте інтерактивну сегментацію та відстеження для прискорення маркування без шкоди для якості. Використовуйте власні передбачення моделі

для попереднього позначення. Використовуйте активне навчання, щоб зосередитися на маркуваннях людей і перевірити, де це матиме найбільший вплив. Щоб знайти та виправити проблеми у вашому наборі даних ML, ви можете: – Зосередити на даних, які мають значення: створіть потік зв'язку між анотаторами та рецензентами. Швидко ітеруйте з анотаторами на мітках, щоб змінити. Забезпечуйте безперервний зворотний зв'язок із вашою групою маркування, щоб уникнути відхилень у якості. – Кількісно оцініть якість за допомогою розуміння передових показників якості: подивіться на консенсус за класами, щоб знати, коли ваша онтологія потребує перестановки. Подивіться на розбіжності між авторами етикеток, щоб виявити непорозуміння серед вашої групи анотаторів. Фільтрувати фрагменти даних із показниками низької якості. Порівняйте якість між етикетувальниками або з галузевим стандартом. – Підвищуйте якість даних за допомогою програмного виявлення помилок: програмно виявляйте помилки, створюючи автоматизовані сценарії контролю якості в інтерфейсі маркування. Використовуйте моделі виявлення помилок, щоб автоматично знаходити та виправляти проблеми у ваших наборах даних ML. – Організуйте всі свої стратегії якості за допомогою автоматизованих робочих процесів: повністю автоматизуйте та створюйте власні робочі процеси для масштабування операцій маркування. Щоб спростити роботу з маркуванням, технологія Kili дозволяє: – Імпортувати та експортувати дані без зусиль: безпосередньо інтегруйте хмарне сховище Amazon, Google і Microsoft, щоб автоматично почати маркування без необхідності переміщення даних. Відстежуйте всі проміжні зміни у своїх даних і експортуйте дані з версіями безпосередньо у формат вашої моделі (YOLO, PASCAL VOC тощо). – Керуйте своєю командою в масштабі: легко призначайте попередньо визначені ролі (адміністратор, менеджер, рецензент, етикетник), щоб контролювати обов'язки користувачів у проектах. Для максимальної зручності та безпеки виконуйте автентифікацію за допомогою певних постачальників ідентифікаційних даних. Забезпечте найвищий рівень керування та безпеки даних. – Автоматизуйте свою

інфраструктуру MLOps: використовуйте веб-хуки, щоб виконувати певні дії, як-от розпочати навчання моделі, виконання активного навчання для створення попередніх міток або керування версіями. Створіть цикл зворотнього зв'язку між експериментами з моделлю та новими етикетками. Використовуйте Kili API та Python SDK, щоб підключити будь-який стек ML. У Kili Technology ми віримо, що зосередження на високоякісних навчальних даних, які постійно маркуються, є способом розкрити цінність ШІ. Сьогодні ми продовжуємо наш шлях, щоб надати всім компаніям можливість перетворювати неструктуровані дані у високоякісні, щоб значно прискорити створення надійного ШІ. Щоб безкоштовно спробувати технологію Kili, приєднайтеся до нашої безкоштовної пробної версії за адресою https://cloud.kili-technology.com/label. Щоб дізнатися про розширені плани, перегляньте наші тарифні плани та поговоріть з нашою командою за адресою https://kili-technology.com/pricing

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Інформаційні технології та послуги.
- Програмне забезпечення.

Сегмент ринку:

- 39% середнього ринку.
- 37% малий бізнес.

Segments.ai

Мультисенсорна платформа маркування для робототехніки та автономного водіння. Segments.ai – це швидка й точна платформа для маркування даних для анотації даних за допомогою кількох датчиків. Ви можете отримати мітки сегментації, векторні мітки тощо за допомогою інтуїтивно зрозумілих інтерфейсів міток для зображень, відео та 3D-хмар точок (лідар і RGBD). Сегментація зображення – Семантична сегментація – Сегментація екземплярів – Паноптична сегментація – Інструменти маркування на базі ML:

DeepPixels і Autosegment Vector Labeling Image – Обмежувальні рамки – Багатокутники – Полілінії – Сегментація хмари точок ключових точок – Семантична сегментація – Сегментація екземпляра – Паноптична сегментація Позначення вектора хмари точок – Кубоїди / тривимірні обмежувальні прямокутники – Ключові точки – Багатокутники та полілінії Позначення відео – Швидке позначення послідовностей даних за допомогою інтерполяції та допомоги в ML. – Позначте об'єднані тривимірні хмари точок необмеженого розміру. – Швидше позначайте тривимірні послідовності за допомогою пакетного режиму та перегляду об'єднаної хмари точок. Об'єднання датчиків: візуалізуйте та позначайте кілька модальностей в одному інтерфейсі. Створіть свій інтелектуальний робочий процес анотацій саме так, як вам потрібно, з гнучкістю, необхідною для швидкого та ефективного виконання роботи. Segments.ai – це платформа самообслуговування з цільовою підтримкою нашої основної команди інженерів, коли вам це потрібно. – Python SDK, який нарешті має сенс – Документація, яка полегшить налаштування – Самообслуговування з підтримкою лише тоді, коли ви застрягли, щоб ми не сповільнювали вас – Автоматичний ініціатор дій за допомогою веб-хуків – Підключіть свого хмарного постачальника (AWS, Google Cloud, Azure) – експортуйте до популярних фреймворків ML (PyTorch, TensorFlow, Hugging Face) На борту вашої робочої сили або скористайтеся одним із наших партнерів. Наші інструменти керування спрощують позначення та спільний перегляд великих наборів даних. Розпочніть безкоштовну пробну версію сьогодні на https://segments.ai/join

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Програмне забезпечення.

Сегмент ринку:

- 100% малий бізнес.

Anyline

Anyline робить збір даних простим, надаючи вам можливість читати, інтерпретувати та обробляти візуальну інформацію на мобільних пристроях, веб-сайтах і вбудованих камерах. Наша найновіша технологія збору даних швидко стає необхідною для операторів у всьому світі, зосереджуючись на автомобільній промисловості та галузях обслуговування автомобілів. Незалежно від того, скануєте критичні показники DOT або розміру шин із шин клієнта чи навіть вимірюєте глибину їхнього протектора за допомогою камери мобільного телефону, наша технологія сканування не схожа ні на що раніше. З нашої домашньої бази у Відні, Австрія, і штаб-квартири США в Бостоні наша зростаюча та динамічна команда змінює спосіб, у який компанії збирають дані та керують ними.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 50% малий бізнес.
- 25% під

Pixyle

Програма розпізнавання зображень Pixyle виявляє на зображеннях модні речі. Ми робимо візуальний контент доступним для покупки. Він точний, потужний і ідеально інтегрується з будь-яким набором технологій. Ми допомагаємо роздрібним торговцям одягу розробляти візуальні рішення штучного інтелекту, які дозволяють покупцям візуально шукати продукти та отримувати релевантні рекомендації щодо подібних продуктів на їхніх платформах електронної комерції, покращуючи як залучення клієнтів, так і коефіцієнти конверсії. Наші рішення для автоматичного тегування та бізнес-аналітики дозволяють роздрібним торговцям заощаджувати час, зменшувати

витрати та приймати розумніші рішення. Дізнайтеся більше про нашу компанію на www.pixyle.ai Якщо ви хочете зв'язатися з нами, ви можете зв'язатися з нами за адресою: hello@pixyle.com Зацікавлені нашими рішеннями? Розпочніть безкоштовну пробну версію на pixyle.ai/request-demo

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Одяг і мода.

Сегмент ринку:

- 73% малий бізнес.
- 18% підприємства.

INTSIG

Як провідна компанія зі штучного інтелекту та великих даних, INTSIG розробила багато програм і сформулювала рішення як для окремих користувачів, так і для корпоративних клієнтів з усього світу. INTSIG, відомий своїми двома мобільними додатками CamScanner і CamCard, завоював серця 2,3 мільярда людей у всьому світі. Завдяки застосуванню передових технологій захоплення та вилучення даних до більшої кількості типів документів INTSIG тепер надає 5 основних продуктів для корпоративних користувачів: (1) CamScanner API/SDK, який пропонує надійні можливості сканування документів; (2) CamCard API/SDK, який забезпечує групове розпізнавання візитних карток і повну інтеграцію з корпоративними CRM; (3) CamCard Business, продукт SaaS, який оптимізує керування візитними картками та підвищує ефективність роботи в мережі; (4) CamCheckout API/SDK для розпізнавання банківських карток і (5) CamID API/SDK для розпізнавання державних документів, що посвідчують особу. Компанії можуть легко інтегрувати ці продукти у власні програми/веб/системи. INTSIG також надає 5 основних рішень для компаній усіх типів: (1) рішення eKYC, яке допомагає компаніям перевіряти та автентифікувати особистість своїх клієнтів; (2) Рішення для автоматизації

облікової заборгованості, яке забезпечує виняткову точність у розпізнаванні банківських виписок і рахунків-фактур із таблицями, зменшує введення даних вручну та кількість помилок і підвищує ефективність обробки рахунків-фактур; (3) рішення Travel & Expense для оптимізації внутрішнього управління витратами та підвищення ефективності збору рахунків і відшкодування; (4) TextIn Studio для навчання моделі для покращення ефекту розпізнавання структурованого тексту в складних сценаріях, широко використовується в банках, страхових компаніях, компаніях з цінними паперами, традиційному виробництві та інших галузях, і (5) Механізм керування рахунками-фактурами для розпізнавання рахунків-фактур, замовлень на купівлю, квитанція, контрактна записка, заява постачальника, дебетова/кредитова нота з високою точністю

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Вища освіта.
- Інформаційні технології та послуги.

Сегмент ринку:

- 48% малий бізнес.
- 33% середнього ринку.

SentiSight.ai

SentiSight.ai – це веб-платформа, яку можна використовувати для маркування зображень і розробки додатків розпізнавання зображень на основі ШІ. Він має дві головні цілі: перша – зробити задачу анотації зображень максимально зручною та ефективною, навіть для великих проектів, де багато людей працюють над маркуванням зображень, а друга – забезпечити гладкий і зручний інтерфейс для навчання та розгортання моделей глибокої нейронної мережі. Можливість виконувати обидва ці завдання на одній платформі забезпечує перевагу можливості позначати зображення, а потім навчати та вдосконалювати моделі ітеративним способом. SentiSight.ai пропонує такі

потужні функції, як: Позначення зображень. Наш інструмент міток дозволяє додавати мітки класифікації, обмежувальні рамки, багатокутники, точки, полілінії та растрові зображення. Растрові зображення можна легко перетворити на багатокутники і навпаки. Крім того, кожен позначений об'єкт може мати кілька дочірніх об'єктів, таких як ключові точки або атрибути. Зображення з мітками можна безпосередньо використовувати для навчання моделей на платформі SentiSight.ai або їх можна завантажити та використовувати для навчання моделей у компанії. Розумний інструмент маркування. Цей інструмент можна використовувати для значного збільшення швидкості маркування растрового зображення. Інтелектуальний інструмент маркування дозволяє користувачам вибрати кілька точок на передньому плані та на задньому плані та дозволити штучному інтелекту витягти позначений об’єкт. Спільні проекти маркування та відстеження часу. Щоб полегшити обробку великих проектів анотацій, SentiSight.ai дозволяє спільно використовувати проект між кількома користувачами, щоб кілька людей могли позначати зображення в одному проекті. Керівник проекту може швидко фільтрувати та переглядати зображення, позначені певним учасником проекту, відстежувати прогрес кожної особи та час, витрачений на маркування, а також керувати ролями та дозволами користувачів. Класифікаційна модель навчання. Цей тип моделі можна використовувати для ідентифікації певних об’єктів на зображенні, наприклад кішки чи собаки, але без уточнення їх розташування. Їх також можна навчити визначати більш абстрактні поняття, такі як «літо» чи «зима». Навчання моделі виявлення об'єктів. Цей тип моделі можна використовувати не тільки для ідентифікації певного об’єкта, але й для прогнозування його точного розташування на зображенні. Для кожного об’єкта, який передбачається знаходитися всередині зображення, модель також передбачає прямокутну обмежувальну рамку, яка позначає розташування об’єкта. Це дуже корисно, коли вам потрібно знати не лише те, що знаходиться всередині зображення, але й відносне розташування та кількість об’єктів. Онлайн і офлайн моделі (доступна безкоштовна 30-денна пробна версія). SentiSight.ai

пропонує можливість використовувати ваші моделі глибокого навчання як онлайн, так і офлайн. Онлайн-моделі можна використовувати через REST API або веб-інтерфейс. Обидва ці варіанти потребують підключення до Інтернету. Інший варіант – завантажити та використовувати модель розпізнавання зображень в автономному режимі. Офлайн-модель можна завантажити як безкоштовну 30-денну пробну версію, після чого користувач має можливість придбати ліцензію. Ціна ліцензії залежить від швидкості моделі, і це одноразовий платіж. Попередньо підготовлені моделі. Окрім можливості самостійного навчання моделей розпізнавання зображень, SentiSight.ai також надає кілька попередньо навчених моделей, які можна використовувати без додаткового навчання. Ці попередньо навчені моделі можна використовувати для кількох завдань, таких як модерація вмісту, класифікація товарів, автоматичні хештеги, підрахунок людей тощо. Пошук схожості зображення. Це ще одна готова до використання функція, яка дозволяє користувачам завантажувати зображення та знаходити всі схожі зображення за цим запитом у своєму наборі даних. Це також дозволяє користувачам виконувати пошук подібності NvN у своєму наборі даних, де витягуються всі подібні пари зображень.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– 67% малий бізнес.

Chooch

Chooch робить камери інтелектуальними. Рішення Chooch поєднують у собі потужність технології Generative AI і Computer Vision, щоб допомогти компаніям отримати практичну інформацію в реальному часі зі своїх відео та візуальних даних.. Технологія штучного інтелекту Chooch миттєво виявляє конкретні візуальні елементи, об’єкти та дії у відео та зображеннях, включаючи

критичні аномалії, і миттєво розуміє їх значення, надсилаючи сповіщення в реальному часі бізнес-системам для ініціювання подальших дій. Він робить це за частку часу, який може зробити людина. Підприємства використовують Chooch для автоматизації повторюваних завдань перегляду відео вручну, роблячи пошук відеоданих більш ефективним і дозволяючи компаніям перерозподіляти людські ресурси для більш цінних видів діяльності. Chooch використовується в багатьох різних програмах, включаючи виявлення роздрібних крадіжок, моніторинг безпеки на робочому місці, виявлення зброї, моніторинг самоконтролю, керування цифровими активами тощо.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 100% малий бізнес.

muse.ai

muse.ai – це платформа пошуку відео, яка дозволяє будь-кому швидко знаходити певні моменти у великій кількості відео. Це також повна платформа для зберігання та потокового передавання відео, яка дозволяє користувачам вставляти найрозширеніший пошук відео на будь-якому веб-сайті.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 87% малий бізнес.

Roboflow

Roboflow – це технологічна компанія, яка дозволяє розробникам використовувати комп’ютерний зір без досвіду машинного навчання. Понад 100

000 користувачів із компаній будь-якого розміру – від стартапів до публічних компаній – використовують наскрізну платформу компанії для збору зображень і відео, організації, анотацій, попередньої обробки, навчання моделі та розгортання. Roboflow надає компаніям інструменти для покращення своїх наборів даних і швидшої побудови точніших моделей комп'ютерного зору, щоб їхні команди могли зосередитися на проблемах своєї області, не винаходячи колесо в інфраструктурі зору. У довгостроковій перспективі Roboflow створює операційну систему для візуальних обчислень.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 100% малий бізнес.

brighter AI

brighter AIнадає рішення для анонімізації на основі найсучаснішої технології глибокого навчання для захисту кожної особистості в публічних місцях. Ми розробляємо кардинальне програмне забезпечення для анонімізації зображень і відео для захисту особистих даних. Анонімність має вирішальне значення для дотримання правил конфіденційності, таких як GDPR, CCPA, APPI та CSL. У той же час якість даних є основою інновацій ШІ та машинного навчання. З нашим програмним забезпеченням для захисту даних немає компромісу між конфіденційністю та відеоаналітикою.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

- Інформаційні технології та послуги.

Сегмент ринку:

- 50% малий бізнес.

– 27% середнього ринку.

GoSpotCheck by FORM

GoSpotCheck by FORM – це провідна в галузі програма для роботи в полі, яка направляє, відстежує та покращує продуктивність у режимі реального часу. Наше просте у користуванні мобільне рішення дає змогу виїзним командам стимулювати роботу на ринку завдяки динамічному управлінню завданнями, найсучаснішому розпізнаванню зображень, фоторепортажам, комунікації виїзної команди та розширеним звітам – і все це на одній простій у використанні платформі. Направляйте команди, покращуйте виконання та стимулюйте продажі, водночас створюючи спільний п на поле, що допомагає лідерам швидше приймати кращі рішення.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Споживчі товари.
– Роздрібна торгівля.

Сегмент ринку:

– 46% середнього ринку.
– 35% підприємства.

Plainsight

Plainsight є лідером у перевіреному баченні ШІ. Забезпечуючи унікальне поєднання стратегії штучного інтелекту, бачення платформи штучного інтелекту та досвіду глибокого навчання, Plainsight розробляє, впроваджує та контролює трансформаційні рішення комп'ютерного зору для підприємств. Завдяки найширшому спектру керованих послуг і платформі бачення AI для централізованих процесів і стандартизованих конвеєрів Plainsight робить комп'ютерне бачення повторюваним і підзвітним у всіх ініціативах корпоративного бачення AI. Plainsight вирішує проблеми, в яких інші зазнали невдачі, і дає можливість компаніям у різних галузях реалізувати весь потенціал

своїх візуальних даних із найменшими перешкодами для виробництва, найшвидшим створенням вартості та моніторингом для довгострокового успіху. Для отримання додаткової інформації відвідайте https://plainsight.ai.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 67% малий бізнес.
- 33% середнього ринку.

APISCRAPY

APISCRAPY – це інструмент веб-скопіювання та автоматизації на основі штучного інтелекту, який перетворює будь-які веб-дані в готовий до використання API даних. Інструмент здатний отримувати дані з веб-сайтів, обробляти дані, автоматизувати робочі процеси, класифікувати дані та інтегрувати готові до використання дані в базу даних або надавати дані в будь-якому бажаному форматі. Наші клієнти використовують інструмент APISCRAPY для створення продуктів і послуг штучного інтелекту, маркування даних, анотацій даних, бізнес-аналітики, дослідження ринку, моніторингу цін, агрегації даних, створення потенційних клієнтів, захисту бренду, роботизованої автоматизації процесів тощо. Основні переваги: – Перетворює будь-які веб-дані та дані додатків у готовий до використання API даних – Доповнені штучним інтелектом і готові можливості автоматизації – Попередньо створені можливості класифікації даних – Дані в реальному часі або за розкладом за допомогою інтуїтивно зрозумілих інформаційних панелей – Готова база даних можливості інтеграції -Ні кодування, ні інвестицій в інфраструктуру – Ціноутворення на основі результатів Інші інструменти від AIMLEAP: AI-Labeler: доповнений ШІ інструмент для анотацій і маркування. AI-Labeler – це доповнена штучним інтелектом платформа анотації даних, яка поєднує в собі потужність штучного

інтелекту та особисту участь для позначення, анотації та класифікації даних, що дозволяє швидше розробляти надійні та точні моделі. AI-Data-Hub: дані на вимогу для створення продуктів і послуг AI. Центр даних штучного інтелекту на вимогу для кураторських даних, попередньо анотованих даних, попередньо класифікованих даних, що дозволяє підприємствам легко й ефективно отримувати та використовувати високоякісні дані для навчання та розробки моделей ШІ. PRICE-SCRAPY: інструмент ціноутворення в режимі реального часу з підтримкою штучного інтелекту Ціноутворююче рішення на основі штучного інтелекту та автоматизації забезпечує моніторинг цін у реальному часі, аналітику цін і динамічне ціноутворення для компаній у всьому світі. API-KART: центр рішень API даних на основі штучного інтелекту API-KART – це центр даних, який дозволяє компаніям і розробникам отримувати доступ до великих обсягів даних із різних джерел і інтегрувати їх. Це центр вирішення даних для доступу до даних через API, що дозволяє компаніям використовувати дані та інтегрувати API у свої системи та програми. Про AIMLEAP AIMLEAP є сертифікованим ISO 9001:2015 та ISO/IEC 27001:2013 міжнародним постачальником технологічних консультацій і послуг, який пропонує рішення для обробки даних, розробку даних, автоматизацію, IT-послуги та послуги цифрового маркетингу. AIMLEAP було визнано «The Great Place to Work®». Зосереджуючись на штучному інтелекті та автоматизації, ми створили чимало рішень штучного інтелекту та машинного навчання, рішень веб-скрейпінгу на основі штучного інтелекту, маркування даних штучного інтелекту, центру даних штучного інтелекту та самообслуговуваних рішень BI. Ми розпочали роботу в 2012 році та успішно реалізували проекти в галузі IT та цифрової трансформації, автоматизованих рішень для обробки даних і цифрового маркетингу для понад 750 швидкозростаючих компаній у США, Європі, Новій Зеландії, Австралії, Канаді; і більше. -Сертифікація ISO 9001:2015 та ISO/IEC 27001:2013 - Обслуговування понад 750 клієнтів -11+ років досвіду роботи в галузі -98%

утримання клієнтів -Сертифікація Great Place to Work® -Глобальні центри доставки в США, Канаді,

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 67% підприємства.
- 17% малий бізнес.

Blitline

Blitline – це найдоступніше рішення SaaS для програмного забезпечення та медіа-компаній, які мають систему керування контентом (CMS) або систему керування цифровими активами (DAM) і потребують безпечної багатоформатної обробки файлів у великих масштабах для своїх програм і веб-сайтів. Blitline SaaS є кращою альтернативою незахищеним внутрішнім програмам обробки файлів і дорогим аутсорсинговим службам, які стягують плату за гігабайт і в основному орієнтовані лише на формати зображень і відео, доступні на www.blitline.com, через Amazon EventBridge і GitHub. Blitline SaaS дозволяє постачальникам цифрових програм і платформ обробляти практично будь-які типи файлів, від зображень, таких як.jpg,.png і.tif, до файлів Adobe PDF, файлів Office 365, відеофайлів, аудіофайлів тощо. За допомогою Blitline клієнти можуть виконувати численні функції обробки файлів, включаючи автоматичне конвертування та перекодування, зміну розміру, кадрування, компонування, накладання тексту, вилучення кольорів, водяні знаки, дедуплікацію, видалення фону за допомогою штучного інтелекту (AI), рендеринг HTML для SEO вихідні дані, вилучення метаданих і розпізнавання елементів за допомогою AWS Rekognition.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

CVEDIA

CVEDIA розробляє надійні та стійкі алгоритми комп'ютерного зору, використовуючи синтетичні дані. Наші синтетичні алгоритми розробляються за 2-4 тижні для сценаріїв, коли збір даних неможливий. CVEDIA є єдиною компанією, що займається виробництвом синтетичних даних, яка вирішує проблему адаптації домену, тобто безпечні алгоритми можливі без даних.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

DIY Research & Feedback Collection

Платформа DIY Research & Feedback Collection від Emozo використовує поведінкові та емоційні дані, щоб допомогти клієнтам приймати правильні рішення щодо всього цифрового контенту. У поєднанні з нашими консультаційними послугами та панелями ми допомагаємо клієнтам вийти за рамки традиційної аналітики даних клієнтів і заглибитися в серця й уми клієнтів, щоб зрозуміти ефективність і вплив усього цифрового контенту. Ми допомагаємо клієнтам створювати та розгортати цілеспрямованіший цифровий вміст – рекламу, програми, потоковий медіаконтент тощо на будь-якому каналі – в Інтернеті, на мобільних пристроях, у соціальних мережах, на телебаченні тощо. Ми використовуємо інформацію, отриману від клієнтів, щоб розв'язувати проблеми бренду, обміну повідомленнями, і відчувати виклики. Наш новий метод поєднання неусвідомлених (уваги та емоцій) і висловлених (анкета) відповідей допомагає клієнтам дуже швидко зрозуміти ефективність усього цифрового контенту. Ми використовуємо штучний інтелект, щоб забезпечити масштабне та швидке якісне дослідження на пристроях клієнтів. Клієнтам і їхнім клієнтам не потрібно нічого завантажувати, встановлювати чи підтримувати.

Платформа SaaS Emozo підтримує ітераційні процеси розробки дизайну та пропонує повністю безпечний захист даних для клієнтів та їхніх клієнтів.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

FaceMRI

FaceMRI – дослідницька група програмного забезпечення для розпізнавання облич, яка базується в США. FaceMRI – це найдосконаліша пошукова система розпізнавання облич для Mac і ПК. FaceMRI має набір програмного забезпечення для розпізнавання облич, яке може класифікувати обличчя за статтю (чоловічі, жіночі, небінарні), віковою категорією, роками та расою. Створення графіків відвідуваності та аналітики. Обличчя можна отримати за допомогою + імпорту зображень + імпорту відео + веб-пошуку (FB, LinkedIn, Instagram) + імпорту папок + веб-камери та IP-камер + IOT і камер безпеки. + USB-ключі та зовнішні пристрої

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

LandingLens

Ціна початкового рівня: від 39 доларів США

LandingLens – це хмарна платформа комп'ютерного бачення, яка дозволяє користувачам швидко й легко створювати, ітерувати та розгортати рішення на основі штучного інтелекту. Ця наскрізна платформа стандартизує створення моделей штучного інтелекту, скорочуючи час розробки та легко масштабуючи проекти для кількох випадків використання. Почніть безкоштовно: https://www.landing.ai

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

LayerNext

LayerNext – це платформа інфраструктури даних для з'єднання найкращих у своєму класі інструментів комп'ютерного зору з потужним озером даних, яке об'єднує необроблені дані, анотації, метадані, аналітику та прогнози моделей. За допомогою LayerNext ви можете: Досліджувати та впорядковувати дані – Інтегрувати LayerNext із вашими сегментами хмарного сховища S3, GCP або Azure – Шукати та досліджувати необроблені зображення та відео, підібрані дані, метадані, основну правду та моделювати результати в єдиному легкому – навігація озером даних за допомогою вбудованого засобу перегляду. – Створюйте та стандартизуйте метадані та теги. Анотуйте дані – Створюйте ідеальні піксельні багатокутники або прості анотації прямокутників кількома клацаннями миші – Додайте мітки на рівні атрибутів для подальшої класифікації об'єктів – Використовуйте автоматичне маркування, щоб пришвидшити завдання анотацій і зменшити вартість. – Переглядайте стан, швидкість і якість ваших операцій маркування. Керуйте наборами даних – керуйте своїми позначеними навчальними наборами даних за допомогою контролю версій, щоб уникнути дублювання та проблем із цілісністю – фіксуйте походження та походження даних, щоб гарантувати якість, цілісність і надійність ваших даних. Створюйте моделі – підключайте свої набори даних до процесу створення моделі – автоматично генеруйте анотацію формати залежно від вибору моделі. Аналіз продуктивності – знайдіть і заповніть прогалини в наборі даних, порівнюючи ефективність навчальних експериментів. – Візуалізуйте композиції міток і метаданих ваших наборів даних. Інтегруйте інструменти – підключіть власні інструменти анотацій, навчальні кінцеві точки або програми сторонніх розробників за допомогою нашого SDK для python

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 67% малий бізнес.
- 33% підприємства.

Mobius Labs

За останні кілька років комп'ютерний зір зробив серйозний стрибок. Ми в Mobius Labs йдемо далі: це вже не просто комп'ютерний зір, це надлюдський зір. Надлюдське бачення означає миттєве перетворення тисяч зображень і відео в розуміння, досвід і переваги. Наша технологія покращує візуальний пошук, розуміючи все про зображення чи відео та автоматично створюючи для них метадані. Superhuman Vision також означає надання периферійним пристроям (наприклад, ноутбукам і смартфонам) здатності миттєво розуміти й аналізувати зображення, відеозаписи та сценарії реального життя. Ми прагнемо створити передову технологію, яка дає змогу будь-кому взаємодіяти з медіа новаторськими способами. Це комп'ютерний зір, який нетехніки можуть навчити, використовуючи дуже невелику кількість даних, розгорнути на периферійних пристроях і дозволити своєму бізнесу захопити ринки, де є зображення чи відео для аналізу.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

Qince

Qince CRM спрямована на допомогу підприємствам у галузі споживчих товарів у здійсненні цифровізації. Він має портфель продуктів, включаючи SFA (Sales Force Automation), mERP (Purchase-sales-inventory), DMS (Distributor

Management System), TPM (Trade Promotion Management). Це допомагає компаніям точно й ефективно керувати командами продажів, включаючи SR, DSR, супервайзерів і промоутерів; керувати клієнтами, включаючи дистриб'юторів, оптовиків і магазини; і керувати товарами, включаючи замовлення та запаси.

Користувачі:

– Інформація відсутня.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

- 50% підприємства.
- 50% Середнього ринку.

Кластер SentiVeillance

SentiVeillance Cluster – це готове до використання програмне забезпечення для легкої інтеграції біометричної ідентифікації обличчя, класифікації та відстеження транспортних засобів і пішоходів, а також автоматичного розпізнавання номерних знаків у операційні системи керування відео (VMS). Програмне забезпечення аналізує прямі відеопотоки, які обслуговуються VMS з камер спостереження. Список можливих застосувань включає правоохоронні органи, моніторинг руху, безпеку, контроль відвідуваності, підрахунок відвідувачів та інші програми.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

SentiVeillance SDK

SentiVeillance SDK призначений для розробки програмного забезпечення, яке виконує біометричну ідентифікацію обличчя, виявляє рухомих пішоходів, транспортні засоби чи інші об’єкти та виконує автоматичне розпізнавання

номерних знаків за допомогою живих відеопотоків із цифрових камер спостереження. SDK використовується для пасивної ідентифікації – коли перехожі не докладають зусиль, щоб їх впізнали. Список можливих застосувань включає правоохоронні органи, безпеку, контроль відвідуваності, підрахунок відвідувачів, моніторинг руху та інші комерційні програми.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

Sky Engine AI

Поглиблене навчання віртуальної реальності платформа для розробників розвиток штучного інтелекту SKY ENGINE Платформа штучного інтелекту дозволяє створювати оптимальні індивідуальні моделі ШІ з нуля та навчати їх віртуальній реальності. Програмне забезпечення SKY ENGINE AI створює віртуальний і генеративний світ, де об’єкти та події можна моделювати в різноманітних конфігураціях та умовах навколишнього середовища (іноді дуже рідко, і їх важко помітити в реальному світі). Генерація даних зі штучним інтелектом SKY ENGINE полегшує життя спеціалістів із обробки даних, надаючи ідеально збалансовані набори даних для будь-яких програм комп’ютерного бачення, таких як виявлення та розпізнавання об’єктів, 3D-позиціонування, оцінка пози та інші складні випадки, включаючи аналіз мультисенсорних даних, наприклад радарів, лідарів, супутників, рентген та багато іншого. Послуги штучного інтелекту SKY ENGINE – разом із нашими основними технологіями, команда Sky Engine AI створює індивідуальні клієнтські рішення на вимогу, які охоплюють увесь конвеєр обробки даних, тобто дослідження, генерацію даних, навчання та перевірку моделей ШІ.

Галузі промисловості:

– Інформація відсутня.

Сегмент ринку:

– Інформація відсутня.

AI Image Recognition

AI Image Recognition Правильне рішення для вашого успіху в роздрібній торгівлі! Наше ІЧ-рішення допоможе вам краще та швидше оцінити виконання в магазині, щоб отримати чітке уявлення про ситуацію на POS. Перетворіть свої фотографії на полицях на ідеї, а ідеї – на дії за допомогою програмного забезпечення розпізнавання зображень ШІ Sterison. Висока ефективність Час перевірки на магазин скорочено з 8 хвилин до 5 секунд (ефективність збільшена на 99%) Реальні дані Система автоматично розраховує кількість товарів і частку відображення без помилок підрахунку вручну. Відсутність внутрішнього персоналу Після завантаження фотографії система автоматично перевіряє, чи кваліфікований дисплей. Перевірка вручну не потрібна.

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 створення та навчання нейронної мережі для розпізнавання 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

Як працює розпізнавання зображень?

Щоб зрозуміти, як працює розпізнавання зображень, важливо спочатку визначити цифрові зображення.

Цифрове зображення складається з елементів зображення або пікселів, які просторово організовані у двовимірну сітку або масив. Кожен піксель має числове значення, яке відповідає його інтенсивності світла або рівню сірого, пояснив Джейсон Корсо, професор робототехніки в Університеті Мічигану та співзасновник стартапу комп'ютерного зору Voxel51.

Використовуючи розпізнавання зображень, система комп'ютерного зору може розпізнавати моделі та закономірності в усіх цих числових даних, які відповідають таким речам, як люди, транспортні засоби чи пухлини. Це, по суті, автоматизує вроджену здатність людини дивитися на зображення, ідентифікувати об'єкти в ньому та відповідним чином реагувати. (Сітківка ока людини може обробляти близько 10 зображень із роздільною здатністю один мільйон точок за секунду, які аналізуються мозком, щоб ми могли швидко асимілювати, контекстуалізувати та реагувати на те, що бачимо.)

«Комп'ютерний зір в основному виконує частину роботи мозку», – сказав Корсо для Built In. «Це справді складно».

Сьогодні розпізнавання зображень базується на глибокому навчанні – підкатегорії машинного навчання, яка використовує багаторівневі структури алгоритмів, які називаються нейронними мережами, щоб постійно аналізувати дані та робити висновки про них, подібно до того, як працює людський мозок. У разі розпізнавання зображень нейронні мережі подають якомога більше

попередньо позначених зображень, щоб «навчити» їх розпізнавати подібні зображення.

Процес зазвичай розбивається на три окремі етапи:

1. Збирається набір даних

Після створення величезного набору даних із зображень і відео його необхідно проаналізувати та додати до нього будь-які значущі функції чи характеристики. Наприклад, зображення собаки має бути ідентифіковано як «собака». І якщо на одному зображенні кілька собак, їх потрібно позначити мітками або обмежувальними рамками, залежно від поставленого завдання.

2. Нейронну мережу годують і навчають

Далі на цих зображеннях живиться та навчається нейронна мережа. Як і людський мозок, машину потрібно навчити розпізнавати концепцію, показуючи їй багато різних прикладів. Якщо всі дані позначені, для розрізнення різних категорій об'єктів (наприклад, кота чи собаки) використовуються алгоритми навчання під наглядом. Якщо дані не позначені, система використовує алгоритми неконтрольованого навчання для аналізу різних атрибутів зображень і визначення важливих подібностей або відмінностей між зображеннями.

Для завдань, пов'язаних із розпізнаванням зображень, найкраще підходять згорткові нейронні мережі, або CNN, оскільки вони можуть автоматично виявляти важливі особливості зображень без нагляду людини.

Для цього CNN мають різні рівні. Перший, відомий як згортковий шар, застосовує фільтри (також відомі як ядра) до пакету вхідних зображень, щоб сканувати їх пікселі та математично порівнювати кольори та форми пікселів, вилучаючи важливі характеристики або візерунки із зображень, як-от краї і кути.

Потім CNN використовує те, що дізнався з першого шару, щоб розглянути дещо більші частини зображення, відзначаючи більш складні характеристики. Він продовжує робити це з кожним шаром, дивлячись на більші та важливіші частини зображення, доки не вирішить, що зображено на основі всіх знайдених функцій.

«У глибокому навчанні вам не потрібні ручні інженерні функції. Вам просто потрібні великі обсяги даних. І модель достатньо глибока, щоб початкові рівні спочатку витягували корисні функції, а потім класифікували, що таке об'єкт», – сказав Вікеш Кханна, головний технічний директор і співзасновник Ambient.ai, для Built In.

3. Висновки перетворюються на дії

Після того, як система розпізнавання зображень навчена, їй можна надсилати нові зображення та відео, які потім порівнюються з вихідним набором навчальних даних, щоб зробити прогнози. Це те, що дозволяє йому призначити певну класифікацію зображенню або вказати, чи присутній певний елемент.

Потім система перетворює їх на висновки, які можна застосувати: безпілотний автомобіль виявляє червоне світло та зупиняється; камера безпеки ідентифікує витягнуту зброю та надсилає сповіщення.

Штучна нейронна мережа є деякою моделлю природної нейронної мережі. Кожний елемент штучної мережі (нейрон) є прототипом, що імітує властивості й роботу біологічного нейрона [1].

Розглянемо роботу штучної одношарової нейронної мережі (рисунок 3.1). На сінапси (односпрямовані входи) штучного нейрона надходить деяка множина сигналів. Кожний сигнал множиться на відповідну вагу, що характеризує синаптичну силу. Вхідні сигнали після сінапсів надходять в осередки нейронів (блоки підсумовування), виходом яких є аксони, з яких сигнали (порушення або гальмування), що визначають рівень активації нейрона, надходять на сінапси наступних нейронів (якщо мережа багатошарова) або на основі яких приймається рішення (наприклад, граничним мажоритарним блоком). Хоча мережні парадигми досить різноманітні, у їхній основі майже завжди лежить саме ця конфігурація.

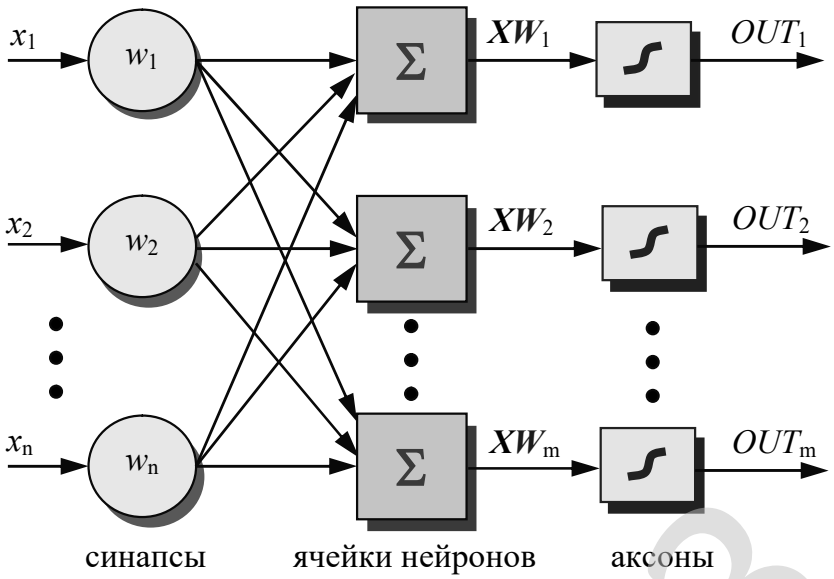

Рисунок 3.1 – Штучна одношарова нейронна мережа

Відзначимо, що сінапси служать лише для розподілу вхідних сигналів, а аксони лише для видачі сигналів, тому вони не вважаються шаром (шаром є осередки нейронів, що виконують обчислення з комбінаціями вхідних сигналів). У штучних і біологічних мережах багато з'єднань можуть бути відсутніми, всі з'єднання показані з метою спільності. Можуть мати місце також з'єднання між виходами й входами елементів в одному шарі.

Умовно цю частину нейронної мережі може представити штучним нейроном (рисунок 3.2). Ця проста модель штучного нейрона ігнорує багато властивостей свого біологічного двійника. Наприклад, вона не бере до уваги затримки в часі, які впливають на динаміку системи. Вхідні сигнали відразу ж породжують вихідний сигнал. І, що більше важливо, вона не враховує впливів функції частотної модуляції або синхронізуючої функції біологічного нейрона, які ряд дослідників уважають вирішальними.

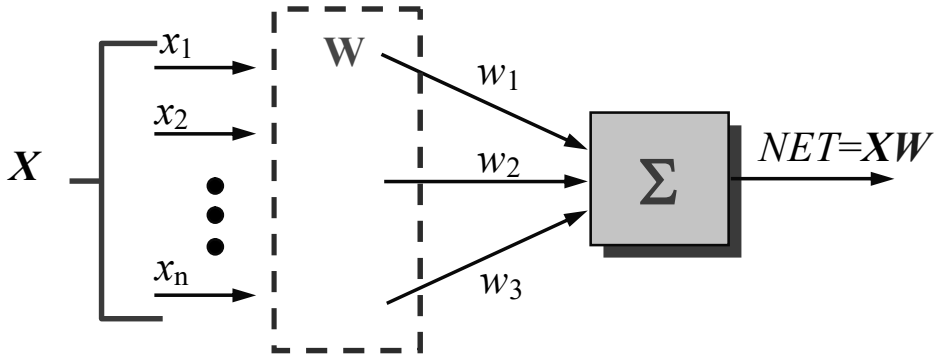

Рисунок 3.2 – Вхідна частина штучного нейрона

Сигнал *NET* далі, як правило, перетвориться деякою активаційною функцією *F* і дає вихідний нейронний сигнал *OUT*.

Активаційна функція – це нелінійна підсилювальна характеристика штучного нейрона, що визначає його рівень активації.

Коефіцієнт підсилення активаційної функції обчислюється як відношення приросту величини *OUT*, що викликав до його невеликого приросту величини *NET* – $\frac{\Delta OUT}{\Delta NET}$. Він виражається нахилом кривої при певному рівні порушення й змінюється від малих значень при більших негативних порушеннях (крива майже горизонтальна) до максимального значення при нульовому порушенні й знову зменшується, коли порушення стає більшим позитивним.

Якщо активаційна функція F звужує діапазон зміни величини *NET* так, що при будь-яких значеннях *NET* значення *OUT* належать деякому кінцевому інтервалу, то *F* називається «стискаючою» функцією.

Якість роботи штучної нейронної мережі багато в чому залежить від ваг синаптичних зв'язків і від типу застосовуваних активаційних функцій. Розглянемо типи активаційних функцій.

Типи активаційних функцій

Розрізняють наступні види активаційних функцій:

– одинична функція;

– лінійний поріг (функція гістерезису);

– сигмоідальна функція (гіперболічний тангенс).

Одинична функція

Дана функція є найпростіший, її графік представлений на рисунку 3.3.

Для цієї функції *OUT* = 1, якщо *NET* = *XW* більше деякої величини T і *OUT* = 0 в інших випадках, де T – деяка постійна гранична величина.

Якщо в одношаровій нейронній мережі застосовується одинична активаційна функція, то така мережа називається персептроном [2, 3, 4]. Перше систематичне вивчення персептронів було почато Маккалокком і Питтсом в 1943 р. [2]. В 60-і роки персептрони викликали великий інтерес і оптимізм. Розенблатт довів чудову теорему про навчання персептронів [4]. Далі первісна ейфорія змінилася розчаруванням, коли виявилося, що персептрони не здатні навчитися рішенню ряду простих задач. Мінський [3] строго проаналізував цю проблему й показав, що є тверді обмеження на те, що можуть виконувати одношарові персептрони, і, отже, на те, чому вони можуть навчатися.

Функція лінійного порога. Відмінністю від попередньої функції є не стрибкоподібна зміна виходу *OUT*, а у вигляді лінійного тренда. Графік функції представлений на рисунку 3.4.

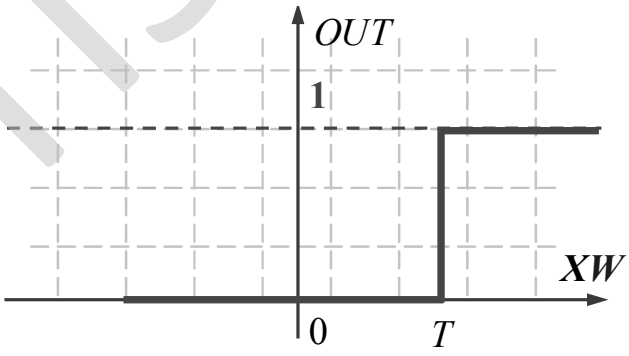

Рисунок 3.3 – Графік одиничної активаційної функції

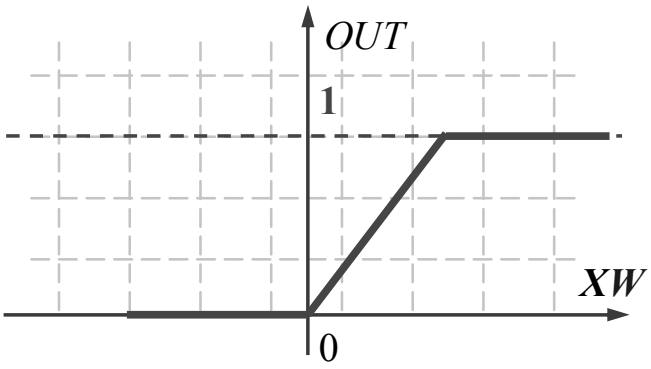

Рисунок 3.4 – Графік лінійної активаційної функції

Сигмоідальна функція. У якості «стискаючої» функції часто використовується логістична або «сигмоідальна» (S-образна) функція. Ця функція математично виражається як $F(x) = 1/(1+e^{-x})$. Таким чином,

$$OUT = 1/1 + e^{-NET}. \quad (3.1)$$

Дана функція дозволяє обробляти нейронної мережі як слабкі, так і сильні сигнали. Слабкі сигнали мають потребу у великому мережному посиленні, щоб дати придатний до використання вихідний сигнал. Однак підсилювальні каскади з більшими коефіцієнтами підсилення можуть привести до насичення виходу шумами підсилювачів (випадковими флуктуаціями), які присутні в будь-якій фізично реалізованій мережі. Сильні вхідні сигнали у свою чергу також будуть приводити до насичення підсилювальних каскадів, крім можливості корисного використання виходу. Сигмоідальна функція вирішує цю проблему. Уперше це виявив Гроссберг в 1973 році.

У якості сигмоідальних функцій може виступати як логістична функція (3.1), так і функція гіперболічного тангенса *OUT* = th(*NET*) (рисунок 3.5). Відмінністю логістичної функції від функції гіперболічного тангенса є діапазон вихідних значень. У першій він перебуває в діапазоні (0; 1), а в другої в діапазоні (-1; 1).

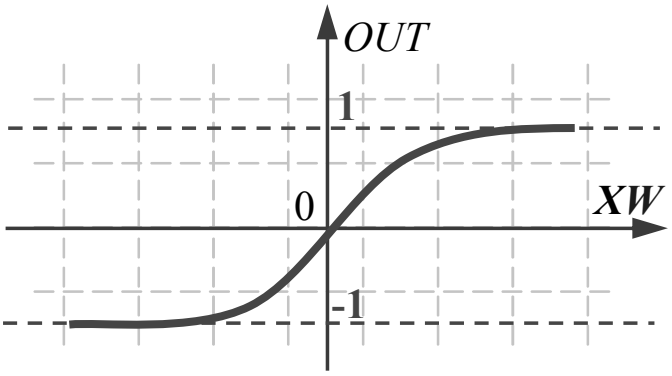

Рисунок 3.5 – Графік сигмоідальної активаційної функції

Проблема функції « що виключає або»

Один із самих песимістичних результатів Мінського [3] показує, що одношаровий персептрон не може відтворити таку просту булевську функцію, як «що виключає або». Це функція від двох аргументів, кожний з яких може бути нулем або одиницею. Вона приймає значення одиниці, коли один з аргументів дорівнює одиниці (але не обоє). Проблему можна проілюструвати за допомогою одношарової однонейронної системи із двома входами, показаної на рисунку 3.6. Позначимо один вхід через x, а інший через y, тоді всі їхні можливі комбінації будуть складатися із чотирьох точок на площині XOY, як показано на рисунку 3.7. Наприклад, точка x = 0 і y = 0 позначена на рисунку як точка A0. Таблиця 1 показує необхідний зв'язок між входами й виходом, де вхідні комбінації, які повинні давати нульовий вихід, позначені A0 і A1, а одиничний вихід – B0 і B1.

У мережі на рисунку 3.6 активаційна функція F є звичайним порогом, так що *OUT* приймає значення нуль, коли *NET* менше 0.5, і одиниця у випадку, коли *NET* більше або дорівнює 0.5. Персептрон виконує наступне обчислення:

$$NET = xw1 + yw2. \quad (3.2)$$

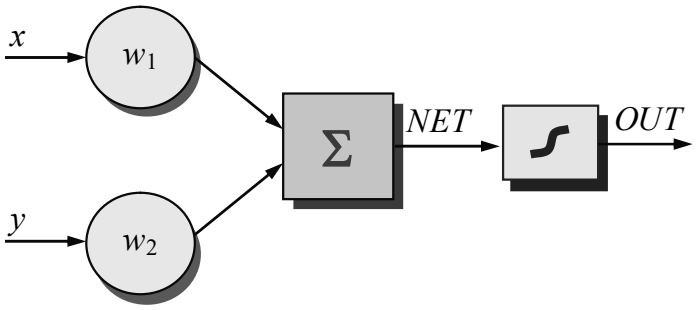

Рисунок 3.6 – Однонейрона система

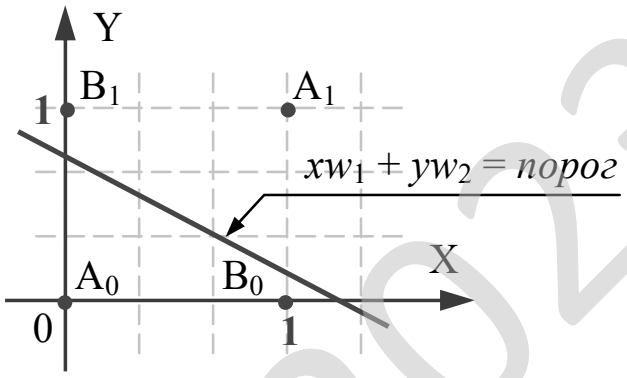

Рисунок 3.7 – Проблема «що виключає або»

Ніяка комбінація значень двох ваг не може дати співвідношення між входом і виходом, що задається таблиця 3.1 (таблиця істинності для булевої функції). Щоб зрозуміти це обмеження, зафіксуємо *NET* для (3.2) на величині порога 0.5. Персептрон у цьому випадку описується рівнянням (3.3). Це рівняння лінійно по x и y, тобто всі значення по x й y, що задовольняють цьому рівнянню, будуть лежати на деякій прямій у площині *X*OY:

$$xw1 + yw2 = 0.5. \qquad (3.3)$$

Будь-які вхідні значення для x й y на цій лінії будуть давати граничне значення 0.5 для *NET*. Вхідні значення з однієї сторони прямої забезпечать значення *NET* більше порога, отже, *OUT* = 1. Вхідні значення по іншу сторону прямої забезпечать значення *NET* менше граничного значення, роблячи *OUT* рівним 0. Зміни значень *w*1, *w*2 і порогу будуть міняти нахил і положення прямої. Для того щоб мережа реалізувала функцію « що виключає або», задану таблиця

3.1, потрібно розташувати пряму так, щоб точки А були з однієї сторони прямій, а точки В – з іншої. Спробувавши намалювати таку пряму на рисунку 3.7, переконуємося, що це неможливо. Це означає, що які би значення не приписувалися вагам і порогу, мережа нездатна відтворити співвідношення між входом і виходом, необхідне для подання функції « що виключає або».

Таблиця 3.1 – Таблиця істинності для функції «що виключає або»

Точки	Значення x	Значення y	Необхідний вихід
A0	0	0	0
B0	1	0	1
B1	0	1	1
A1	1	1	0

Неможливість намалювати пряму лінію, що розділяє площину XOY так, щоб реалізовувалася функція «що виключає або», є не єдиним випадком. Є великий клас функцій, не реалізованих одношаровою нейронною мережею. Про ці функції говорять, що вони є лінійно нероздільними, і вони накладають певні обмеження на можливості одношарових мереж.

Лінійна роздільність обмежує одношарові нейронні мережі задачами класифікації, у яких множини точок (відповідних вхідним значенням) можуть бути розділені геометрично. І оскільки лінійна роздільність обмежує можливості персептронного подання, то важливо знати, чи є дана функція роздільною. На жаль, не існує простого способу визначити це, якщо число змінних велике.

Нейрон з n двійковими входами може мати $2n$ різних вхідних образів, що складаються з нулів і одиниць. Так як кожний вхідний образ може відповідати двом різним бінарним виходам (одиниця й нуль), те всього є $22n$ функцій від n змінних (див. залежність у таблиці 3.2) [5].

Як видно з таблиці 3.2, імовірність того, що випадково обрана функція виявиться лінійно роздільної, досить мала навіть для помірного числа змінних. Із цієї причини одношарові персептрони на практиці обмежені простими задачами.

Таблиця 3.2 – Лінійно роздільні функції

n	22*n*	Число лінійно роздільних функцій
1	4	4
2	16	14
3	256	104
4	65536	1882
5	4294967296	94572

Багатошарові нейронні мережі

До кінця 60-х років проблема лінійної роздільності була добре зрозуміла. До того ж було відомо, що це серйозне обмеження представляємості одношаровими мережами можна перебороти, додавши додаткові шари.

Розглянемо просту двошарову мережу із двома входами, підведеними до двох нейронів першого шару, з'єднаними з єдиним нейроном у шарі 2 (рисунок 3.8). Нехай поріг вихідного нейрона дорівнює 0,75, а обоє його ваги рівні 0,5. У цьому випадку для того, щоб поріг був перевищений і на виході з'явилася одиниця, потрібно, щоб обидва нейрони першого рівня на виході мали одиницю. Таким чином, вихідний нейрон реалізує логічну функцію «і».

Кожний нейрон шару 1 розбиває площина *X*OY на дві напівплощини, одна забезпечує одиничний вихід для входів нижче верхньої лінії, інша – для входів вище нижньої лінії. На рисунку 3.9 показаний результат такої подвійної розбивки, де вихідний сигнал нейрона другого шару дорівнює одиниці тільки усередині V-образної області. Аналогічно в другому шарі може бути використано три нейрони з подальшою розбивкою площини й створенням області трикутної форми. Включенням достатнього числа нейронів у вхідний шар може бути утворений опуклий багатокутник будь-якої бажаної форми. Так як вони утворені за допомогою операції «і» над областями, що задаються лініями, те всі такі багатогранники опуклі, отже, тільки опуклі області й виникають. Точки, не

складові опуклої області, не можуть бути відділені від інших точок площини двошаровою мережею.

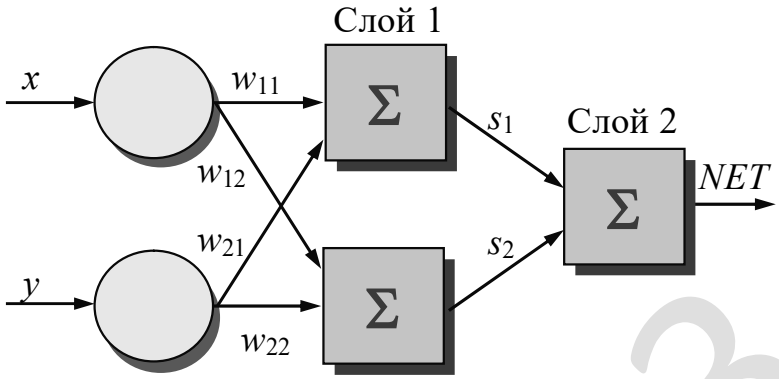

Рисунок 3.8 – Найпростіша двошарова нейронна мережа

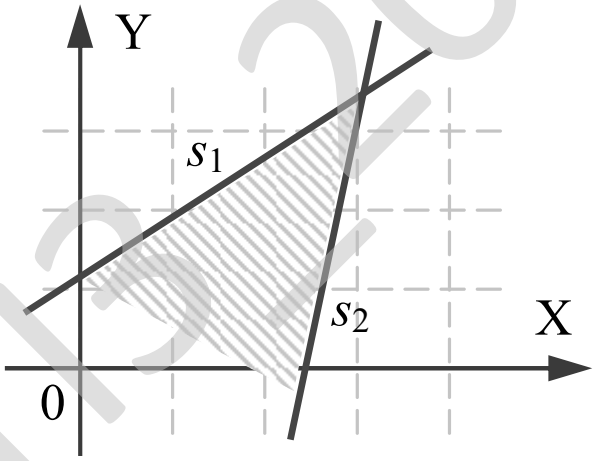

Рисунок 3.9 – Опукла область рішень, що задається двошаровою мережею

Нейрон другого шару не обов'язково обмежений функцією «і», він може реалізовувати багато інших функцій при підходящому виборі ваг і порога. Наприклад, можна зробити так, щоб одиничний вихід кожного з нейронів першого шару приводив до появи одиниці на виході нейрона другого шару, реалізувавши тим самим логічне «або». Є 16 двійкових функцій від двох змінних. Якщо вибирати підходящим образом ваги й поріг, то можна відтворити 14 з них (всі, крім «що виключає або» і «що виключає ні»).

Входи не обов'язково повинні бути двійковими. Вектор безперервних входів може являти собою довільну точку на площині XOY. У цьому випадку ми маємо справу зі здатністю мережі розбивати площина на безперервні області, а не з поділом дискретних множин точок. Для всіх цих функцій, однак, лінійна роздільність показує, що вихід нейрона другого шару дорівнює одиниці тільки в частині площини *X*OY, обмеженою багатокутною областю. Тому для поділу площин *P* і *Q* необхідно, щоб всі *P* лежали усередині опуклої багатокутної області, що не містить точок *Q* (або навпаки).

Тришарова мережа, однак, є більше загальною (рисунок 3.10). Її можливості, що класифікують, обмежені лише числом штучних нейронів і ваг. Обмеження на опуклість відсутні. Тепер нейрон третього шару приймає як вхід набір опуклих багатокутників, і їхня логічна комбінація може бути неопуклою.

На рисунку 3.11 ілюструється випадок, коли два трикутники А і В, скомбіновані за допомогою функцій «А і не В», задають неопуклу область. При додаванні нейронів і ваг число сторін багатокутників може необмежено зростати. Це дозволяє апроксимувати область будь-якої форми з будь-якою точністю. Крім того не всі вихідні області другого шару повинні перетинатися. Можливо, отже, поєднувати різні області, опуклі і неопуклі, видаючи на виході одиницю щораз, коли вхідний вектор належить однієї з них.

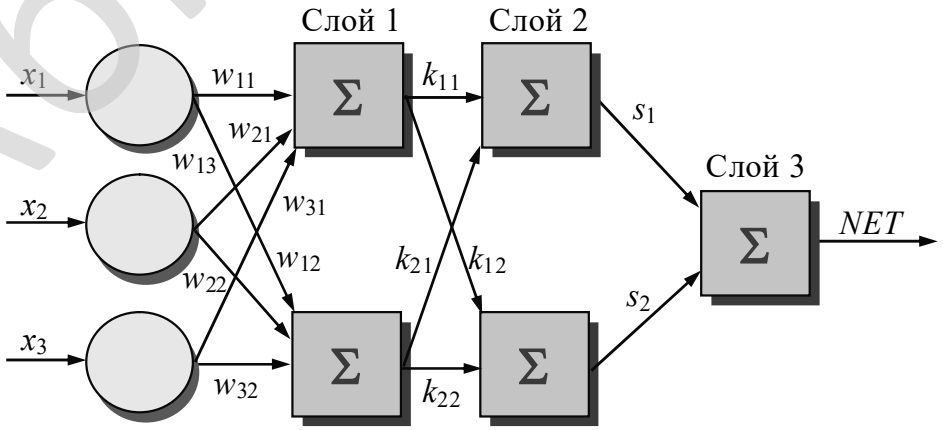

Рисунок 3.10 – Найпростіша двошарова нейронна мережа

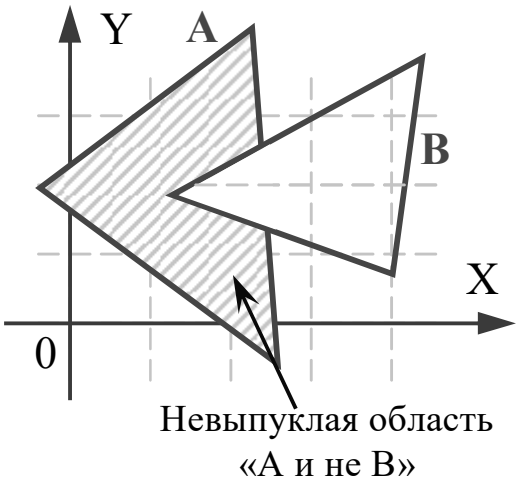

Рисунок 3.11 – Увігнута область рішень, що задається тришаровою мережею

Більші й складні нейронні мережі володіють, як правило, і більшими обчислювальними можливостями. Хоча створені мережі всіх конфігурацій, які тільки можна собі представити, пошарова організація нейронів копіює шаруваті структури певних відділів мозку.

Існують навіть тривимірні й багатомірні шаруваті нейронні структури. Однак, багатошарові мережі можуть привести до збільшення обчислювальної потужності в порівнянні з одношаровою мережею лише в тому випадку, якщо активаційна функція між шарами буде нелінійною. Обчислення виходу шару полягає в множенні вхідного вектора на першу вагову матрицю з наступним множенням (якщо відсутня нелінійна активаційна функція) результуючого вектора на другу вагову матрицю:

$$NET = (XW1)W2.$$

Так як множення матриць асоціативно, то:

$$NET = X(W1\,W2). \qquad (3.4)$$

Це показує, що двошарова лінійна мережа еквівалентна одному шару з ваговою матрицею, рівної добутку двох вагових матриць. Отже, будь-яка багатошарова лінійна мережа може бути замінена еквівалентною одношаровою мережею.

3.2 Розробка структурної схеми

Навчання нейронних мереж для розпізнання образів

Здатність штучних нейронних мереж навчатися є їх головною властивістю. Подібно біологічним системам, які вони моделюють, ці нейронні мережі самі моделюють себе в результаті спроб досягти кращої моделі поводження.

Використовуючи критерій лінійної роздільності, можна вирішити, чи здатна одношарова нейронна мережа реалізовувати необхідну функцію. Навіть у тому випадку, коли відповідь позитивний, це принесе мало користі, якщо в нас немає способу знайти потрібні значення вагових коефіцієнтів. Щоб мережа являла практичну цінність, потрібний систематичний метод (алгоритм) для обчислення цих значень. Розенблатт [4] зробив це у своєму алгоритмі навчання персептрону разом з доказом того, що персептрон може бути навчений усьому, що він може реалізовувати.

Метою навчання нейронної мережі є таке підстроювання її ваг, щоб додаток деякої множини входів приводило до необхідної множини виходів.

Для стислості ці множини входів і виходів будуть називатися векторами. При навчанні передбачається, що для кожного вхідного вектора існує парний йому цільовий вектор, що задає необхідний вихід. Разом вони називаються навчальною парою. Як правило, мережа навчається на багатьох парах. Наприклад, вхідна частина навчальної пари може складатися з набору нулів і одиниць, що представляє двійковий образ деякої букви алфавіту. Вихід може бути числом, що представляє букву «А», або іншим набором з нулів і одиниць, що може бути використаний для одержання вихідного образу. При необхідності розпізнавати за допомогою мережі всі букви алфавіту, треба було б 26 навчальних пар. Така група навчальних пар називається навчальною множиною.

Навчання може бути із вчителем або без нього. Для навчання із вчителем потрібний «зовнішній» вчитель, що оцінював би поводження системи й управляв

її наступними модифікаціями. При навчанні без вчителя мережа шляхом самоорганізації робить необхідні зміни.

Розглянемо навчання персептрону на задачі розпізнавання образів (рисунок 3.12). Навчання персептрону є навчанням із вчителем.

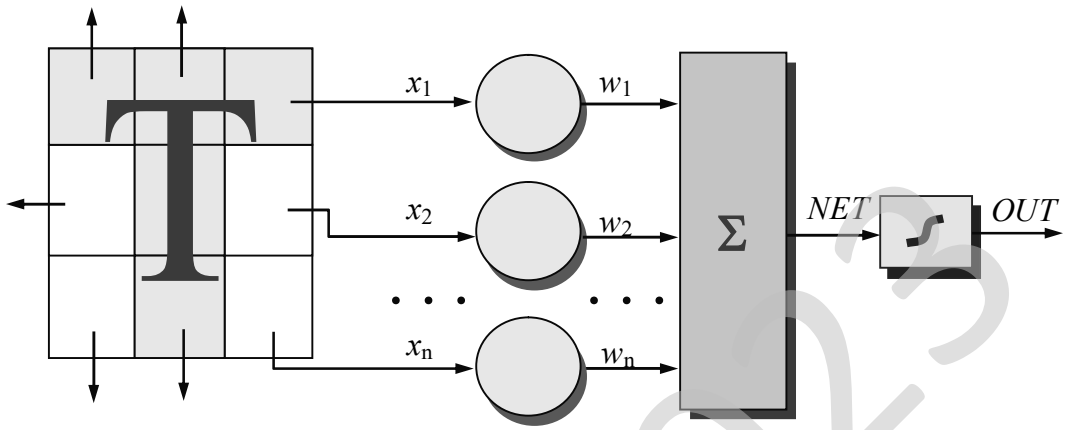

Рисунок 3.12 – Персептронна система розпізнавання образів

Персептрон навчають, подаючи множину образів по одному на його вхід і підбудовуючи ваги доти, поки для всіх образів не буде досягнутий необхідний вихід.

Нехай, для простоти, зображення буде монохромним. Тоді якщо піксель містить точку зображення, то від нього подається одиниця, у противному випадку – нуль. Множина пікселів створює вектор входу персептрону.

Допустимо, що вектор X містить значення ознак розпізнаваного образу (пікселів) – (*x*1, *x*2, …, *xn*), які множаться на відповідні компоненти вектора ваг *W* – (*w*1, *w*2,..., *wn*). Ці добутки підсумуються. Якщо сума перевищує поріг Θ, те вихід нейрона *OUT* дорівнює одиниці, у противному випадку – нулю.

Для навчання мережі образ X подається на вхід і обчислюється вихід *OUT*. Якщо *OUT* правильний, то нічого не міняється. Однак якщо вихід неправильний, то ваги, приєднані до входів, що підсилюють помилковий результат, модифікуються, щоб зменшити помилку.

Метод навчання складається з наступних кроків:

1. Вибирається черговий вхідний образ.
2. Значення ознак образи подаються на персептрон і обчислюється *OUT*.
3. Якщо вихід правильний, то переходять на перший крок.
4. Якщо вихід неправильний і дорівнює нулю, то додати значення всіх входів до відповідних їм ваг; або якщо вихід неправильний і дорівнює одиниці, то відняти значення кожного входу з відповідної йому ваги.
5. Перейти на перший крок.

За кінцеве число кроків мережа навчиться розділяти образи. Відзначимо, що це навчання глобально, тобто мережа навчається на всій множині образів. Виникає питання про те, як ця множина повинне пред'являтися, щоб мінімізувати час навчання. Чи належні елементи множини пред'являтися послідовно один за одним або їх варто вибирати випадково?

Існує кілька правил зміни кроку й подачі образів.

Дельта-правило. Важливе узагальнення алгоритму навчання персептрону, називане дельта-правилом, переносить цей метод на безперервні входи й виходи. Уводиться величина δ, що дорівнює різниці між необхідним або цільовим виходом T і реальним виходом *OUT*

$$\delta = (\mathrm{T} - OUT). \qquad (3.5)$$

Випадок, коли $\delta = 0$, відповідає кроку 3, тобто коли вихід правильний і в мережі нічого не змінюється. Крок 4 відповідає випадку $\delta <> 0$.

У кожному із цих випадків персептронний алгоритм навчання зберігається: δ множиться на величину кожного входу x*i* і цей добуток додається до відповідної ваги. З метою узагальнення вводиться коефіцієнт «швидкості навчання» η, що множиться на δx*i*, що дозволяє управляти середньою величиною зміни ваг.

В алгебраїчній формі запису:

$$\Delta i = \eta \delta x i,$$

$$w(n+1) = w(n) + \Delta i,$$

де Δi – корекція, пов'язана з *i-м* входом *xi*; *wi*(n+1) – значення *i-ї* ваги після корекції; *wi*{n) – значення *i-ї* ваги до корекції.

Дельта-правило модифікує ваги відповідно до необхідного й дійсного значень виходу кожної полярності як для безперервних, так і для бінарних входів і виходів.

Процедура зворотного поширення

На рисунку 3.13 зображена двошарова мережа, що може навчатися за допомогою процедури зворотного поширення (рисунок спрощений).

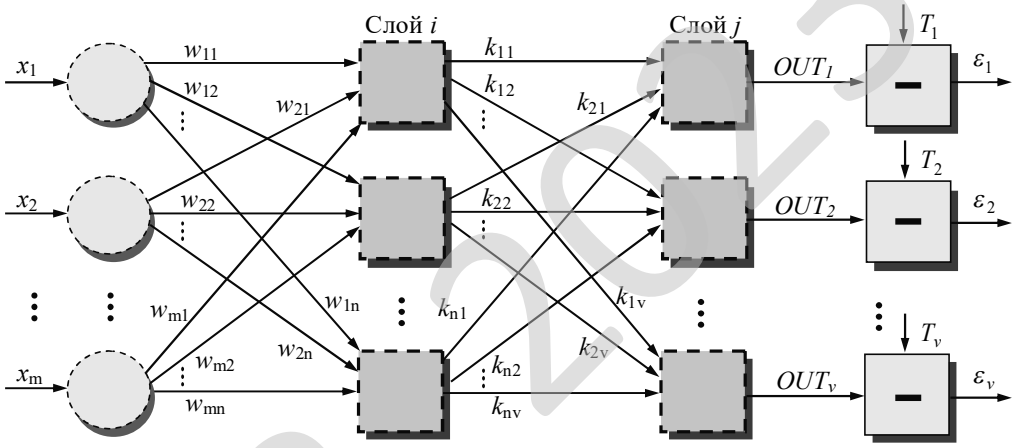

Рисунок 3.13 – Двошарова мережа зворотного поширення

На вхід мережі надходить вхідний вектор сигналів X (x1, x2, …, xm), який множиться на матрицю ваг W (на рисунку 3.13 показані всі можливі з'єднання, хоча в реальності ряд з них може бути відсутніми). На рисунку 3.13 пунктирними кружками позначені блоки множення скаляра на вектор. Далі сигнали надходять на перший шар нейронної мережі (шар *i*), де вони підсумуються й модифікуються активаційними функціями кожного нейрона, а потім отримані виходи множаться на вектор ваг для передачі сигналів на другий шар нейронної мережі (шар *j*). У шарі *j* немає блоку множення виходу нейронів на вектор. Кожний з нейронів у цих шарах на рисунку представлений пунктирним квадратом.

Процедура зворотного поширення застосовна до мереж з будь-яким числом шарів. Однак для того, щоб продемонструвати алгоритм, досить двох шарів. Зараз будуть розглядатися лише мережі прямої дії, хоча зворотне поширення застосовне й до мереж зі зворотними зв'язками.

Фактично процедура зворотного поширення є алгоритмом навчання мережі без вчителя.

Перед початком навчання всім вагам повинні бути привласнені невеликі початкові значення, обрані випадковим образом. Це гарантує, що в мережі не відбудеться насичення більшими значеннями ваг, і запобігає ряд інших патологічних випадків. Наприклад, якщо всім вагам додати однакові початкові значення, а для необхідного функціонування потрібні нерівні значення, то мережа не зможе навчитися.

Навчання мережі процедурою зворотного поширення вимагає виконання наступних операцій:

1. Вибрати чергову навчальну пару з навчальної множини й подати вхідний вектор на вхід мережі.

2. Обчислити вихід мережі.

3. Обчислити різницю між виходом мережі й необхідним виходом (3.цільовим вектором навчальної пари).

4. Підкоректувати ваги мережі так, щоб мінімізувати помилку.

Повторювати кроки з 1 по 4 для кожного вектора навчальної множини доти, поки помилка на всій множині не досягне прийнятного рівня.

Операції, виконувані кроками 1 і 2, подібні з тими, які виконуються при функціонуванні вже навченої мережі, тобто подається вхідний вектор і обчислюється вихід, що виходить. Обчислення виконуються пошарово. На рисунку 3.13 спочатку обчислюються виходи нейронів шару *i*, потім вони використовуються з деякими коефіцієнтами як входи шару *j*. Після обчислюються виходи нейронів шару *j*, які й утворять вихідний вектор мережі.

На кроці 3 кожний з виходів мережі, які на рисунку 3.13 позначені *UOTq*, віднімається з відповідного компонента цільового вектора *Tq*, щоб одержати помилку ε_q. Ця помилка використовується на кроці 4 для корекції ваг мережі, причому знак і величина змін ваг визначаються алгоритмом навчання.

Після достатнього числа повторень цих чотирьох кроків різниця між дійсними виходами й цільовими виходами повинна зменшитися до прийнятної величини, при цьому говорять, що мережа навчилася. Тоді мережа може використовуватися для розпізнавання й ваги більше не змінюються.

Кроки 1 і 2 можна розглядати як «прохід уперед», так як сигнал поширюється по мережі від входу до виходу. Кроки 3, 4 становлять «зворотний прохід», сигнал, що обчислюється тут, помилки поширюється обернено по мережі й використовується для підстроювання ваг.

Прохід уперед

Кроки 1 і 2 можуть бути виражені у векторній формі в такий спосіб: подається вхідний вектор X и на виході виходить вектор *OUT*. Векторна пара вхід-ціль X і T береться з навчальної множини. Обчислення проводяться над вектором *X*, щоб одержати вихідний вектор *OUT*.

Величина *NET* кожного нейрона першого шару обчислюється як зважена сума входів нейрона. Потім активаційна функція F «стискає» *NET* і дає величину *OUT* для кожного нейрона в цьому шарі. Коли множина виходів шару отримано, вона є вхідною множиною для наступного шару. Процес повторюється шар за шаром, поки не буде отримана заключна множина виходів мережі.

$$OUT = \mathrm{F2}(\mathrm{F1}(XW1)W2), \qquad (3.6)$$

де $W1$ – матриця вагових коефіцієнтів вхідного шару, $W2$ – матриця вагових коефіцієнтів передачі виходів першого шару нейронної мережі на входи другого шару, F1 – вектор активаційних функцій нейронів першого шару, F2 – вектор активаційних функцій нейронів другого шару.

Зворотний прохід

На даному етапі здійснюється підстроювання ваг вихідного шару. Так як для кожного нейрона вихідного шару задане цільове значення, то підстроювання ваг цього шару легко здійснюється з використанням модифікованого дельта-правила (3.5). Внутрішні шари називають «схованими шарами», для їхніх виходів не є цільових значень для порівняння. Тому навчання ускладнюється.

Вихід нейрона шару *j*, віднімаючись із цільового значення *Tq*, дає сигнал помилки ε_q. Він множиться на похідну активаційної функції (для логістичної функції (3.1) вона дорівнює *OUTq* (1 – *OUTq*)) обчислену для цього нейрона *q* шару *j*, даючи, таким чином, величину *δq*

$$\delta q = OUTq\ (3.1 - OUTq)(Tq - OUTq). \qquad (3.7)$$

Потім *δq* множиться на величину *OUTp* нейрона шару *i*, з якого виходить розглянуту вагу. Це добуток у свою чергу множиться на коефіцієнт швидкості навчання *η* (звичайно від 0.01 до 1.0), і результат додається до ваги. Така ж процедура виконується для кожної ваги від нейрона схованого шару до нейрона у вихідному шарі.

Наступні рівняння ілюструють це обчислення

$$\Delta Kpq = \eta\ \delta q\ OUTp, \qquad (3.8)$$

$$Kpq(n+1) = Kpq(n) + \Delta Kpq, \qquad (3.9)$$

де *Kpq*(*n*) – величина ваги від нейрона *p* у схованому шарі до нейрона *q* у вихідному шарі на кроці *n* (до корекції); *Kpq*(*n*+1) – величина ваги на кроці *n* + 1 (після корекції); *δq* – величина *δ* для нейрона *q*, у вихідному шарі *j*; *OUTp* – величина *OUT* для нейрона p у схованому шарі *i*.

Підстроювання ваг схованого шару

Розглянемо один нейрон у схованому шарі *i*, що передує вихідному шару *j*. При проході вперед цей нейрон передає свій вихідний сигнал нейронам у вихідному шарі через з'єднуючі їхні ваги. Під час навчання ці ваги функціонують у зворотному порядку, пропускаючи величину *δq* від вихідного шару назад до схованого шару. Кожний із цих ваг множиться на величину *δq* нейрона, до якого

він приєднаний у вихідному шарі. Величина δp, необхідна для нейрона схованого шару, виходить підсумовуванням всіх таких добутків і множенням на похідну стискаючої функції:

$$\delta_p = OUT_p\left(1 - OUT_p\right)\left[\sum_q \delta_q k_{pq}\right]. \tag{3.10}$$

Коли значення δp отримано, ваги, першого схованого шару, можуть бути підкоректовані за допомогою рівнянь

$$\Delta wxp = \eta\ \delta p\ xp, \tag{3.11}$$

$$wxp(n+1) = wxp(n) + \Delta wxp. \tag{3.12}$$

де $wxp(n)$ – величина ваги від входу x до нейрона p у схованому шарі на кроці n (до корекції); $wxp(n+1)$ – величина ваги на кроці $n + 1$ (після корекції); δp – величина δ для нейрона p, у схованому шарі i; xp – величина входу для нейрона p у схованому шарі i.

Для кожного нейрона в даному схованому шарі повинне бути обчислене δp і підбудовані всі ваги, асоційовані із цим шаром. Цей процес повторюється шар за шаром у напрямку до входу, поки всі ваги не будуть підкоректовані.

За допомогою векторних позначень операція зворотного поширення помилки може бути записана значно компактніше. Позначимо множину величин δq вихідного шару через Dq і множина ваг вихідного шару як масив Kq. Щоб одержати Dj, δ-вектор вихідного шару, що досить випливають двох операцій:

Помножити вектор вихідного шару Dq на транспоновану матрицю ваг K_q^T, що з'єднує схований рівень із вихідним рівнем.

Помножити кожний компонент отриманого добутку на похідну стискаючої функції відповідного нейрона в схованому шарі.

У символьному записі:

$$D_j = D_q K_q^T * \left[OUT_p * (I - OUT_p)\right], \tag{3.13}$$

де * – оператор заелементного добутку векторів, $OUTp$ -вихідний вектор шару i, I – вектор, всі елементи якого дорівнюють одиниці.

Імпульс

У роботі [6] описаний метод прискорення навчання для алгоритму зворотного поширення, що збільшує також стійкість процесу. Цей метод, названий імпульсом, полягає в додаванні до корекції ваги члена, пропорційного величині попередньої зміни ваги. Як тільки відбувається корекція, вона «запам'ятовується» і служить для модифікації всіх наступних корекцій. Рівняння корекції (3.8), (3.9) і за аналогією (3.11), (3.12) модифікуються в такий спосіб:

$$\Delta Kpq(n+1) = \eta\ \delta q\ OUTp + \alpha\ \Delta Kpq(n), \qquad (3.14)$$

$$Kpq(n+1) = Kpq(n) + \Delta Kpq(n+1), \qquad (3.15)$$

де α – коефіцієнт імпульсу, звичайно встановлюється близько 0.9.

Використовуючи метод імпульсу, мережа прагне йти по дну вузьких ярів поверхні помилки (якщо такі є), а не рухатися від схилу до схилу. Цей метод, очевидно, добре працює на деяких задачах, але дає слабкий або навіть негативний ефект на інші.

У роботі [7] описаний подібний метод, заснований на експонентному згладжуванні, що може мати перевага в ряді додатків. У цьому випадку формули (3.14), (3.15) будуть мати вигляд

$$\Delta Kpq(n+1) = (1\text{-}1\text{-}\alpha)\ \delta q\ OUTp + \alpha\ \Delta Kpq(n), \qquad (3.16)$$

$$Kpq(n+1) = Kpq(n) + \eta\Delta Kpq(n+1), \qquad (3.17)$$

де α – коефіцієнт згладжування, варіюємий в діапазоні від 0.0 до 1.0. Якщо α дорівнює 1.0, те нова корекція ігнорується й повторюється попередня. В області між 0 і 1 корекція ваги згладжується величиною, пропорційної α.

Метод зворотного поширення має ряд недоліків:

– параліч мережі – коли в процесі навчання значення ваг стають більшими величинами, тобто нейрони будуть мати більші значення виходів, а оскільки похідна активаційної функції при більших значеннях мала, то похідна в (3.7), (3.10), (3.13) буде також мала. У результаті процес навчання «завмре». Для запобігання паралічу звичайно зменшують розмір кроку η, але це збільшує час

навчання. Можна також спробувати ввести евристичні правила виявлення паралічу;

– локальні мінімуми – у процесі навчання будуть знайдені не найкращі значення ваг, оскільки зворотне поширення являє собою різновид градієнтного спуска, тобто здійснює спуск долілиць по поверхні помилки, безупинно підбудовуючи ваги в напрямку до мінімуму. Якщо поверхня помилки має складну форму, то навчання застряє в локальному екстремумі. Для запобігання подібної ситуації використовують статистичні методи навчання, але вони повільні;

– розмір кроку – в [6] говориться про те, що корекція ваг повинна здійснюватися нескінченно малим кроком, але на практиці крок повинен бути кінцевим і досить більшим. У теж час розмір кроку не повинен приводити до паралічу мережі й постійної нестійкості;

– тимчасова нестійкість – полягає в тому, що нейронна мережа забуває, чому її вже навчили, при подальшому навчанні. Процес навчання повинен бути таким, щоб мережа навчалася на всій навчальній множині без пропусків того, що вже виучено. Рекомендує перед корекцією ваг на вхід мережі подати вся навчальна множина. На жаль, якщо вхідні образи не можуть бути повторені даний підхід не застосуємо, оскільки будуть постійні осциляції.

Мережі зустрічного поширення

Можливості мережі зустрічного поширення перевершують можливості одношарових мереж, а час навчання, у порівнянні зі зворотним поширенням, може зменшуватися на кілька порядків. Зустрічне поширення не настільки загальне, як зворотне поширення, але воно може давати рішення в тих додатках, де довга навчальна процедура неможлива.

У зустрічному поширенні об'єднані два добре відомих алгоритми: карта, що самоорганізується, Кохонена [9] і зірка Гроссберга [8]. Їхнє об'єднання веде до властивостей, яких немає в жодного з них окремо.

Мережа зустрічного поширення володіє однією з важливих властивостей для розпізнавання образів – узагальнююча здатність мережі дозволяє одержувати правильний вихід навіть при додатку вхідного вектора, що є неповним або злегка невірним.

Структура мережі

На рисунку 3.14 показана спрощена версія мережі зустрічного поширення. На ньому ілюструються функціональні властивості цієї парадигми.

Як видно з рисунка 3.14 мережа зустрічного поширення дуже схожа на двошарову мережу зворотного поширення. Розходження полягає в операціях, виконуваних нейронами Кохонена й Гроссберга.

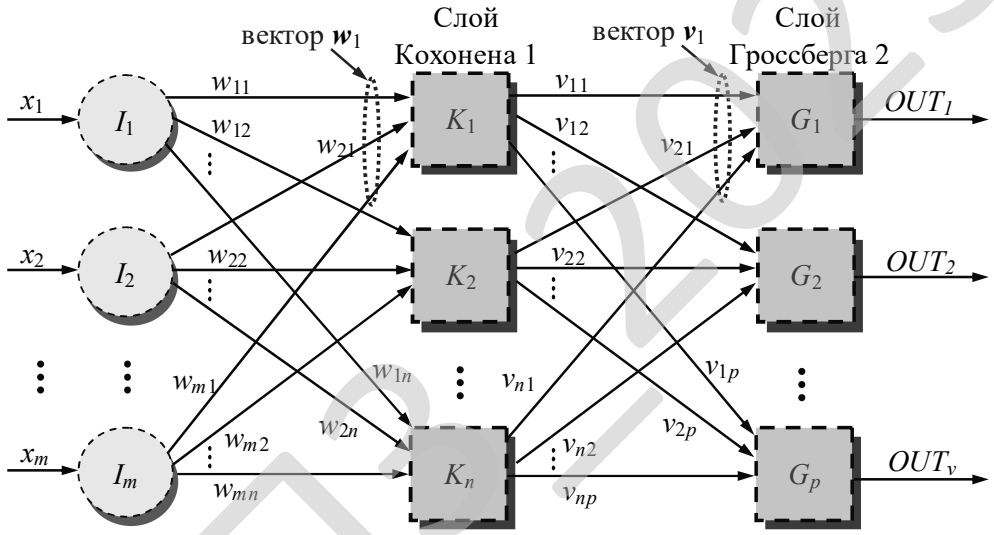

Рисунок 3.14 – Мережа зустрічного поширення без зворотних зв'язків

Шар Кохонена

У своїй найпростішій формі шар Кохонена функціонує в такий спосіб: тільки один нейрон із шару видає на виході логічну одиницю, всі інші видають нуль для заданого вхідного вектора.

Асоційоване з кожним нейроном Кохонена множина ваг з'єднує його з кожним входом. Наприклад, на рисунку 3.14 нейрон Кохонена $K1$ має ваги $w11$, $w21$, …, wm1, складовій вагарні вектор $w1$. Вони з'єднуються через вхідний шар із вхідними сигналами $x1$, $x2$, …, xm, складовими вхідний вектор X. Подібно

нейронам більшості мереж вихід *NET* кожного нейрона Кохонена є просто сумою зважених входів

$$NET_j = \sum_i x_i w_{ij}, \tag{3.18}$$

де *NETj* – це вихід *NET* нейрона Кохонена *j*.

Вихід нейрона Кохонена з максимальним значенням *NET* дорівнює одиниці, в інших він дорівнює нулю.

Шар Гроссберга

Шар Гроссберга функціонує в подібній манері. Його вихід *OUT* є зваженою сумою виходів нейронів *K*1, *K*2,..., *Kn* шару Кохонена

$$OUT_j = \sum_i NET_i v_{ij}, \tag{3.19}$$

де *NETi* – вихід *i-го* нейрона Кохонена, *OUTj* – вихід *j-го* нейрона Гроссберга.

Якщо шар Кохонена функціонує таким чином, що лише в одного нейрона величина *NET* дорівнює одиниці, а в інших дорівнює нулю, то лише один елемент векторів *v*1, *v*2, …, *vp* відмінний від нуля, і обчислення дуже прості. Фактично кожний нейрон шару Гроссберга лише видає величину ваги, що зв'язує цей нейрон з єдиним ненульовим нейроном Кохонена.

Навчання шару Кохонена

Шар Кохонена класифікує вхідні вектори в групи схожих. Це досягається за допомогою такого підстроювання ваг шару Кохонена, що близькі вхідні вектори активують той самий нейрон даного шару. Потім задачею шару Гроссберга є одержання необхідних виходів.

Навчання Кохонена є самонавчанням, що протікає без вчителя. Тому важко (і не потрібно) пророкувати, який саме нейрон Кохонена буде активуватися для заданого вхідного вектора. Необхідно лише гарантувати, щоб у результаті навчання розділялися несхожі вхідні вектори.

Попередня обробка вхідних векторів. Досить бажано (хоча й не обов'язково) нормалізувати вхідні вектори перед тим, як пред'являти їхньої мережі. Це виконується за допомогою ділення кожного компонента вхідного

вектора на довжину вектора. Ця довжина перебуває добуванням квадратного кореня із суми квадратів компонент вектора. В алгебраїчному записі:

$$x_i' = \frac{x_i}{\sqrt{x_1^2 + x_2^2 + ... + x_m^2}}. \qquad (3.20)$$

Це перетворює вхідний вектор в одиничний вектор з тим же самим напрямком, тобто у вектор одиничної довжини в m-мірному просторі.

При навчанні шару Кохонена на вхід подається вхідний вектор і обчислюються його скалярні добутки з векторами ваг, зв'язаними з усіма нейронами Кохонена. Далі ваги нейрона з максимальним значенням скалярного добутку підбудовуються. Так як скалярний добуток, використовуваний для обчислення величин *NET*, є мірою подібності між вхідним вектором і вектором ваг, то процес навчання складається у виборі нейрона Кохонена з ваговим вектором, найбільш близьким до вхідного вектора, і подальшому наближенні вагового вектора до вхідного. Мережа самоорганізується таким чином, що даний нейрон Кохонена має максимальний вихід для даного вхідного вектора. Рівняння, що описує процес навчання має такий вигляд:

$$w_н = w_с + \alpha(x - w_с), \qquad (3.21)$$

де $w_н$ – нове значення ваги, що з'єднує вхідний компонент x з нейроном, що має одиницю на виході; $w_с$ – попереднє значення цієї ваги; α – коефіцієнт швидкості навчання, що може варіюватися в процесі навчання.

Кожна вага, пов'язаний з нейроном Кохонена, що дає одиницю на виході, змінюється пропорційно різниці між його величиною й величиною входу, до якого він приєднаний. Напрямок зміни мінімізує різниця між вагою і його входом.

На рисунку 3.15 цей процес показаний геометрично у двовимірному виді. Спочатку знаходиться вектор $X - W_с$. Потім цей вектор коротшає множенням його на скалярну величину α, меншу одиниці, у результаті чого виходить вектор зміни δ. Остаточно новий ваговий вектор $W_н$ є відрізком, спрямованим з початку координат у кінець вектора δ. Звідси можна бачити, що ефект навчання

складається в обертанні вагового вектора в напрямку вхідного вектора без істотної зміни його довжини.

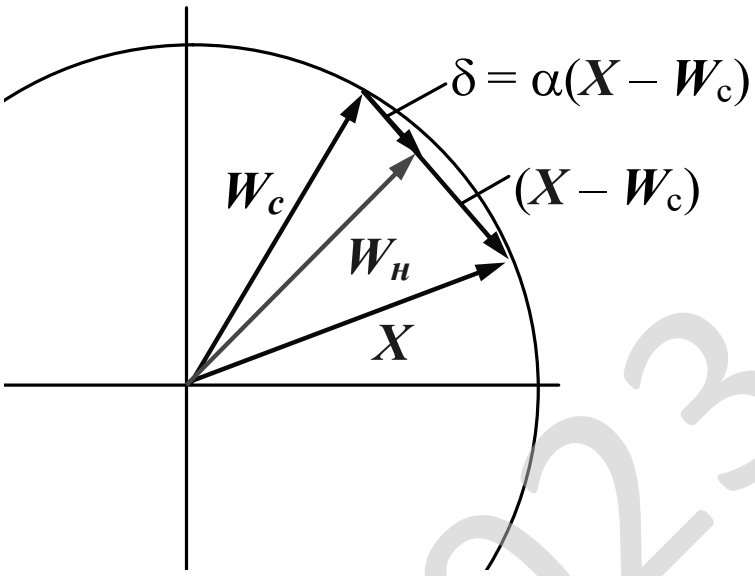

Рисунок 3.15 – Обертання вагового вектора в процесі навчання

Коефіцієнтом швидкості навчання спочатку звичайно дорівнює 0,7 і може поступово зменшуватися в процесі навчання. Це дозволяє робити більші початкові кроки для швидкого грубого навчання й менші кроки при підході до остаточної величини.

Якби з кожним нейроном Кохонена асоціювався один вхідний вектор, то шар Кохонена міг би бути навчений за допомогою одного обчислення на вагу. Ваги нейрона з одиницею на виході прирівнювалися б до компонентів навчального вектора ($\alpha = 1$). Як правило, множина що навчає, включає багато подібних між собою вхідних векторів, і мережа повинна бути навчена активувати той самий нейрон Кохонена для кожного з них. У цьому випадку ваги цього нейрона повинні виходити усередненням вхідних векторів, які повинні його активувати. Поступове зменшення величини (зменшує вплив кожного навчального кроку, так що остаточне значення буде середньою величиною від вхідних векторів, на яких відбувається навчання. Таким чином, ваги, асоційовані

з нейроном, приймуть значення поблизу «центра» вхідних векторів, для яких даний нейрон дає одиницю на виході.

Вибір початкових значень вагових векторів

Всім вагам мережі перед початком навчання варто додати початкові значення. Загальноприйнятою практикою при роботі з нейронними мережами є присвоювання вагам невеликих випадкових значень. При навчанні шару Кохонена випадково обрані вагарні вектори варто нормалізувати. Остаточні значення вагових векторів після навчання збігаються з нормалізованими вхідними векторами. Тому нормалізація перед початком навчання наближає вагові вектори до їхніх остаточних значень, скорочуючи, таким чином, що навчає процес.

Рандомизація ваг шару Кохонена може породити проблему в процесі навчання, так як в результаті її вагові вектори розподіляються рівномірно по поверхні гіперсфери. Через те, що вхідні вектори, як правило, розподілені нерівномірно й мають тенденцію групуватися на відносно малій частині поверхні гіперсфери, більшість вагових векторів будуть так вилучені від будь-якого вхідного вектора, що вони ніколи не будуть давати найкращої відповідності. Ці нейрони Кохонена будуть завжди мати нульовий вихід і виявляться марними. Більше того, якщо початкова щільність вагових векторів в околиці навчальних векторів занадто мала, то може виявитися неможливим розділити подібні класи через те, що не буде достатньої кількості вагових векторів на поверхні, що цікавить, гіперсфери, щоб приписати по одному з них кожному класу вхідних векторів. У теж час, якщо кілька вхідних векторів отримані незначними змінами з того самого зразка й повинні бути об'єднані в один клас, то вони повинні включати той самий нейрон Кохонена. Якщо ж щільність вагових векторів дуже висока поблизу групи злегка різних вхідних векторів, то кожний вхідний вектор може активувати окремий нейрон Кохонена. Це не приводить до проблеми, оскільки шар Гроссберга може відобразити різні нейрони Кохонена в той самий вихід, але це марнотратна витрата нейронів Кохонена.

Найбільш бажане рішення полягає в тому, щоб розподіляти вагові вектори відповідно до щільності вхідних векторів, які повинні бути розділені, поміщаючи тим самим більше вагових векторів в околиці великої кількості вхідних векторів. На практиці це нездійсненно, однак існує кілька методів наближеного досягнення тих же цілей.

Одне з рішень, відоме за назвою методу опуклої комбінації (convex combination method), полягає в тому, що всі ваги прирівнюються однієї й тій же величині:

$$w_i = 1/\sqrt{m},$$

де m – число входів і, отже, число компонентів кожного вагового вектора. Завдяки цьому всі вагові вектори збігаються й мають одиничну довжину. Кожному ж елементу вхідного вектора X надається значення:

$$x_i = \alpha x_i + (1-\alpha)/\sqrt{m},$$

де m – число входів. На початку α дуже мало, внаслідок чого всі вхідні вектори мають довжину, близьку до $1/\sqrt{m}$, і майже збігаються з векторами ваг. У процесі навчання мережі (поступово зростає, наближаючись до одиниці. Це дозволяє розділяти вхідні вектори й остаточно приписує їм їхні щирі значення. Вагові вектори відслідковують один або невелика група вхідних векторів і наприкінці навчання дають необхідну картину виходів. Метод опуклої комбінації добре працює, але сповільнює процес навчання, так як вагові вектори підбудовуються до мети, що змінюється.

Інший підхід складається в додаванні шуму до вхідних векторів. Тим самим вони піддаються випадковим змінам, схоплюючи зрештою ваговий вектор. Цей метод також працездатний, але ще більше повільний, ніж метод опуклої комбінації.

Третій метод починає з випадкових ваг, але на початковій стадії навчального процесу підлаштовує всі ваги, а не тільки пов'язані з нейроном Кохонена, що дає на виході одиницю. Тим самим вагові вектори переміщаються

ближче до області вхідних векторів. У процесі навчання корекція ваг починає вироблятися лише для найближчих до нейрона з виходом рівним одиниці. Цей радіус корекції поступово зменшується, так що зрештою коректуються тільки ваги, пов'язані з нейроном який має одиницю на виході.

Ще один метод розділяє час навчання. Якщо нейрон Кохонена частіше інших дає одиницю на виході (більше 1/*n* раз, де *n* – число нейронів Кохонена), він тимчасово збільшує свій поріг, що зменшує його шанси мати одиницю на виході, даючи тим самим можливість навчатися й іншим нейронам.

У багатьох додатках точність результату істотно залежить від розподілу ваг. На жаль, ефективність різних рішень вичерпним образом не оцінена й залишається проблемою.

Навчання шару Гроссберга

Шар Гроссберга навчається відносно просто. Вхідний вектор, що є виходом шару Кохонена, подається на шар нейронів Гроссберга, і виходи шару Гроссберга обчислюються, як при нормальному функціонуванні. Далі, кожна вага коректується лише в тому випадку, якщо вона з'єднана з нейроном Кохонена, що має ненульовий вихід. Величина корекції ваги пропорційна різниці між вагою й необхідним виходом нейрона Гроссберга, з яким він з'єднаний. У символьному записі

$$vijн = vijc + \beta(OUTj - vijc)NETi, \qquad (3.22)$$

де *NETi* – вихід *i-го* нейрона Кохонена (тільки для одного нейрона Кохонена він відмінний від нуля); *OUTj – j-а* компонента вектора бажаних виходів, *vijc* – старе значення ваги, що набудовується, *vijн* – нове значення ваги.

Спочатку β береться рівним близько 0,1 і потім поступово зменшується в процесі навчання.

З (3.22) видно, що ваги шару Гроссберга будуть сходитися до середніх величин від бажаних виходів, тоді як ваги шару Кохонена навчаються на середніх значеннях входів. Навчання шару Гроссберга – це навчання із вчителем, алгоритм має у своєму розпорядженні бажаний вихід, по якому він навчається. Навчаємий

без вчителя, що самоорганізується шар Кохонена дає виходи в недетермінованих позиціях. Вони відображаються в бажані виходи шаром Гроссберга.

На рисунку 3.16 зображена структурна схема системи. На наведеній структурній схемі зазначені основні класи об'єктів ядра системи і їхня взаємодія. Стрілками показані потоки даних при роботі системи. Кожному з основних блоків управляючої системи відповідає свій блок у системі. Чотири блоки складають управляючу систему:

– апарат формування й розпізнавання образів;

– база знань;

– блок оцінки стану;

– блок прийняття рішень.

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 нейронної мережі 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.

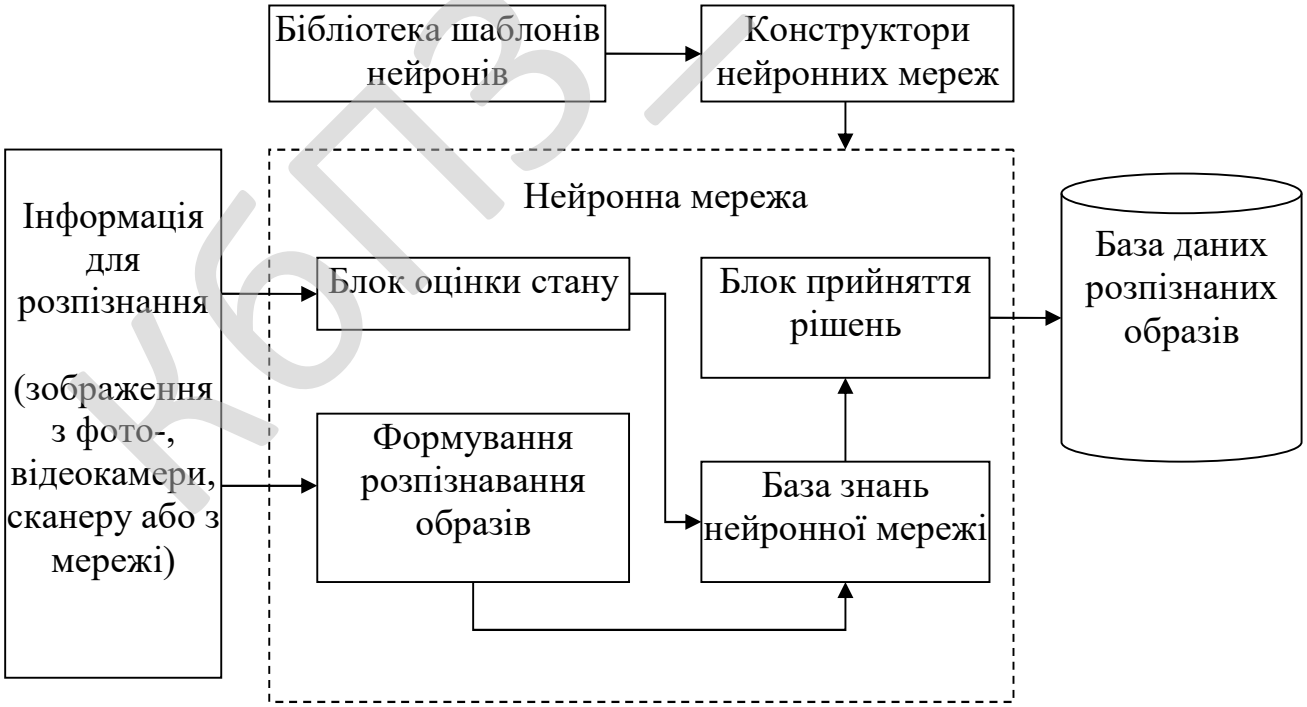

Рисунок 3.16 – Структурна схема системи

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 управляючої системи 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 управляючої системи, вміст окремих блоків може помінятися, можливо стати більш складним і ієрархічним, при цьому поводження системи не зміниться, якщо новий вміст забезпечує структурність 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 нейронну мережу, 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 нейронної мережі 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 графічного інтерфейсу користувача файли специфікації мереж; ці конструктори в ядро системи розробки нейронних мереж 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 п 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 пу 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 (у змісті визначення 3.1–3.5). Ми опишемо тільки реалізацію ядра 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 графічного інтерфейсу користувача. 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 бази знань, також ми вважаємо, знадобиться окремий інструмент для конструювання самих баз знань, а, можливо, при певному рівні складності блоків управляючої системи, і для кожного з них по окремому інструменті, які б ураховували повною мірою специфіку блоків управляючої системи.

3.3 Розробка функціональної схеми

На рисунку 3.17 зображена функціональна схема системи. Нижче розглянемо її більш докладно. Функціонально система, що розроблена в даному магістерському проекті, складається із двох систем:

– Система розпізнавання осіб.

– Система контролю дорожнього руху.

Система розпізнавання осіб:

– Виділення особи людей у відеопотоці й побудова їх 3D-моделі.

– Розпізнавання особи, порівнюючи побудовані 3D-моделі з еталонними моделями або звичайними фотографіями.

– Попередження оператора про збіг.

– Збереження в архіві всіх фотографій й 3D-моделі виділених осіб.

– Дозвіл вести пошук в архіві по базі збережених осіб.

– Дозвіл переглядати зображення з камер у реальному часі й транслювати їх по мережі.

Рисунок 3.17 – Функціональна схема системи

Система контролю дорожнього руху:

– Розпізнавання номера автомобілів.

– Детектування порушення ПДР:

1. Перевищення швидкості.

2. Перевищення середньої швидкості на ділянці дороги.

3. Проїзд на червоний сигнал світлофора/виїзд за стоп-лінію.

4. Перетинання суцільної лінії.

5. Виїзд на зустрічну смугу.

6. Стоянка й зупинка в неналежному місці.

7. Порушення при проїзді перехресть і переїздів.

– Ведення автоматичного пошуку по базах розшуку й інших баз даних.

– Збирання статистики для керування дорожнім рухом.

Сфери застосування системи розпізнавання осіб

Забезпечення безпеки:

– Транспортні вузли: аеропорти, вокзали, автостанції, метро.

– Місця масового перебування людей: стадіони, розважальні центри, бізнес-центри, кінотеатри.

Посилення контролю:

– Прикордонні паспортно-візові контрольні пункти.

– Прохідні й КПП на режимних, стратегічних і комерційних об'єктах.

Безконтактний контроль стану людини:

– Диспетчерів і охоронців.

– Пілотів, машиністів, водіїв.

Реклама й маркетингові дослідження:

– Інтерактивні сервіси, що реагують на міміку.

– Інтерактивна реклама.

– Оцінка задоволеності споживача.

Індустрія розваг:

– Створення 3D-фільмів.

– Моделювання віртуальної реальності.

3D-медицина:

– Навчання на 3D-моделях.

– 3D-телемедицина.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерського проектування, наведена на рисунку 3.18.

Після початку роботи розробленого ПЗ ми потрапляємо до головного вікна ПЗ далі можна перейти до нейронної мережі та навчання нейронної мережі чи з головного меню провести звертання вхідного зображення в матрицю, формування масиву еталонних зразків символів, розпізнавання образів та виведення результатів.

Крім цього з головного вікна ПЗ можна провести моніторинг нейронної мережі шляхом визначення швидкості розпізнавання образів, оцінку ефективності роботи мережі з виведенням результатів на єкран.

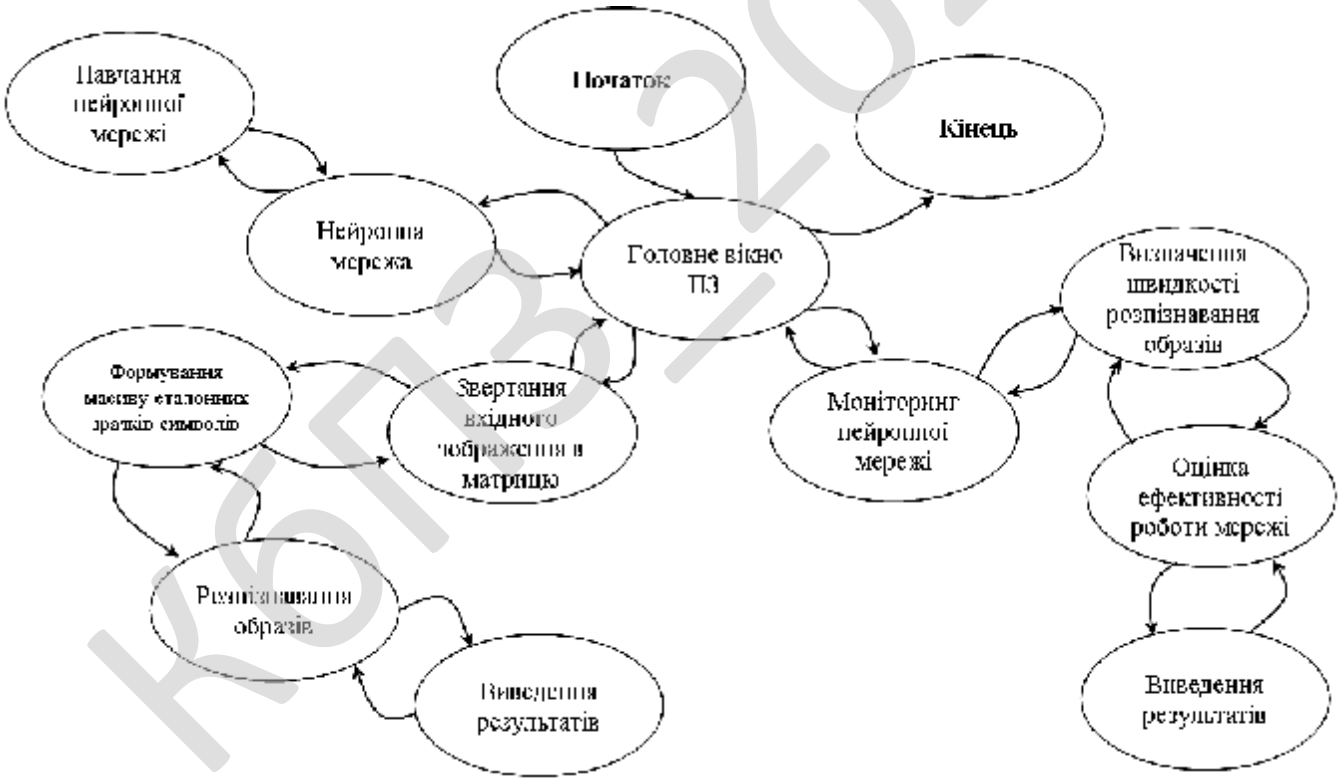

Рисунок 3.18 – Діаграма взаємодії процесів

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

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків:

– Виділення пам'яті програми.

– Організація черги повідомлень та передача управління ПЗ.

– Ініціалізація призначених для користувача типів даних.

– Завантаження файлу, запуск та перевірка нейронної мережі.

– Запит – НМ завантажено.

– Завантаження бази знань НМ.

– Запит – базу знань завантажено.

– Виведення головного вікна ПЗ.

– Очікування запиту користувача.

– Запит – розпізнавання образів.

– Підпрограма конвертації зображення.

– Оновлення шаблонів.

– Підпрограма розпізнавання образів.

– Навчання нейронної мережі.

– Запит – налаштування НМ.

– Встановлення нових параметрів НМ.

– Сигнал WM_CLOSE (запит).

– Звільнення виділених динамічних ресурсів користувача.

– Звільнення призначених для користувача типів даних.

– Завершення роботи ОС.

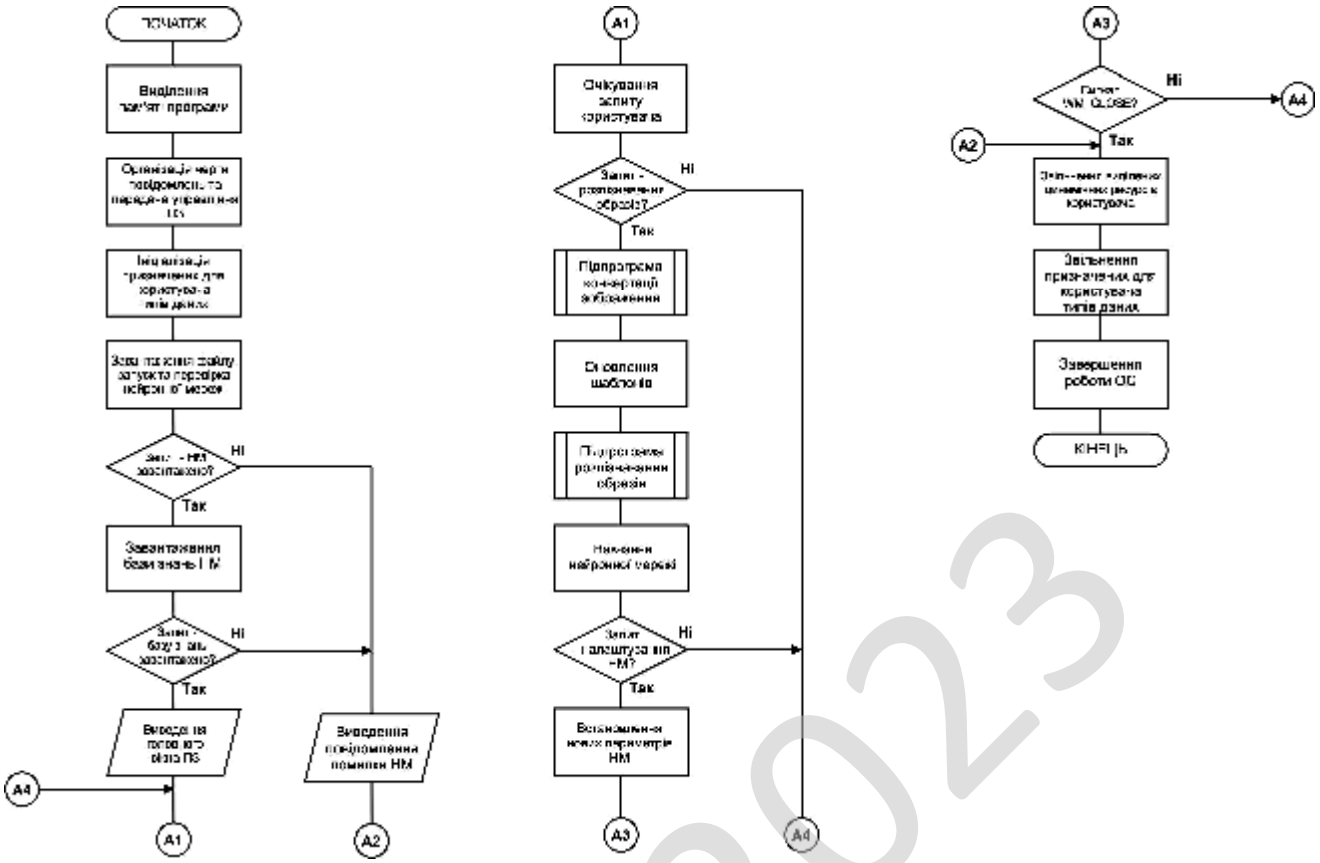

Рисунок 4.1 – Блок-схема основної програми

На рисунку 4.2 наведено блок-схему підпрограми конвертації зображення. Її робота складається з виконання наступних кроків:

– Завантаження вхідного зображення.

– Конвертація формату Jpg у Bmp.

– Перехід у палітру 256 відтінків сірого.

– Звертання розміру зображення.

– Створення динамічного масиву.

– Зчитування рядка пікселів у масив.

– Пошук та видалення полу тонів у рядку.

– Додавання рядку до матриці зображення.

– Всі рядки оброблені (запит).

– Пошук та видалення полу тонів у рядку.

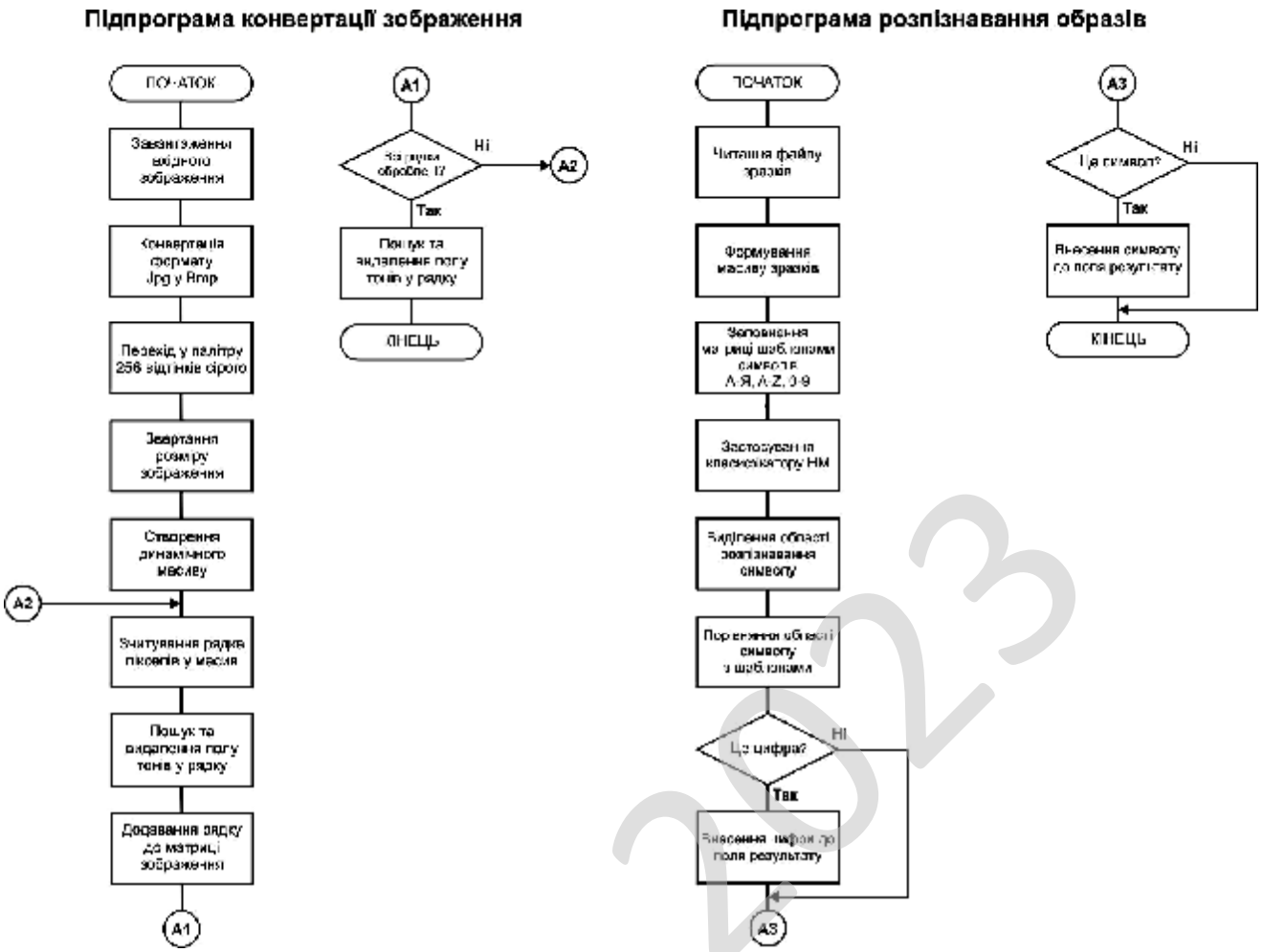

Рисунок 4.2 – Блок-схема підпрограм

На рисунку 4.2 також наведено блок-схему підпрограму розпізнавання образів. Її робота складається з виконання наступних кроків:

– Читання файлу зразків.

– Формування масиву зразків.

– Заповнення матриці шаблонами символів А-Я, A-Z, 0-9.

– Застосування класифікатору НМ.

– Виділення області розпізнавання символу.

– Порівняння області символу з шаблонами.

– Це цифра (запит).

– Внесення цифри до поля результату.

– Це символ (запит).

– Внесення символу до поля результату.

Опис алгоритмів функціонування системи

Розглянемо розроблений мною у дипломі основний алгоритм розпізнавання образів. Спочатку була створена канва для малювання й формування її образа в пам'яті.

У якості канви використовувався клас Tbitmap (для простоти роботи з бітмапом використовуємо режим 1 байт на піксель, тобто Tbitmap.Pixelformat:=pf8bit), віалізуем його на Tpaintbox, відображаємо в пам'яті за допомогою структури:

```
type
  Masx = Pbytearray;
var
Masy : array of Masx // масив пікселей,
{ де Masy[ y-координата][ x- координата] = номер кольору в палітрі цвіту ( при 8 біт/піксель). Відображення здійснюємо з використанням Tbitmap.Scanline (швидко й просто) }
  Setlength(Masy, Tbitmap.Height);
  for j := 0 to Tbitmap.Height - 1 do
    Masy[j] := Tbitmap.Scanline[j];
```

Після формування канви був створений і формований масив еталонних зразків символів.

Еталонні зразки будемо формувалися на основі матриці розміром 16x16. Для цього була розроблена процедура генерації такої матриці по довільному зображенню еталона.

Процедура function Create_16x16(Img : Tbitmap): Tmas16x16 отримує в якості параметра посилання на картинку, на якій намальований еталон символу, повертає приведену матрицю розміром 16x16.

Коротко поясню роботу процедури.

1. Одержуємо посилання на бітмап і здійснюємо його відображення в пам'яті.

2. Обчислюємо координати границь (описаного прямокутника) образа (еталонного або розпізнаваного) шляхом сканування рядків/стовпців.

При цьому тут і при подальшому аналізі зображення передбачається, що символ намальований чорним кольором (№0 у палітрі квітів) і відповідно всі значущі пікселі мають значення 0.

```
for j := 0 to Img.Height - 1 do// Top
  begin
    for i := 0 to Img.Width - 1 do
      if Masy[j][i] = 0 then
      begin ytop := j; break; end;
    if ytop = j then break;
  end;
  for j := Img.Height - 1 downto 0 do
  begin
    for i := 0 to Img.Width - 1 do
      if Masy[j][i] = 0 then
      begin ybottom := j + 1; break; end;
    if ybottom = j + 1 then break;
  end;
  for i := 0 to Img.Width - 1 do
  begin
    for j := 0 to Img.Height - 1 do
      if Masy[j][i] = 0 then begin xleft := i; break; end;
    if xleft = i then break;
  end;
 for i := Img.Width - 1 downto 0 do
  begin
    for j := 0 to Img.Height - 1 do
if Masy[j][i] = 0 then begin xright := i + 1; break; end;
    if xright = i + 1 then break;
  end;
```

3. Для подальшого аналізу буде потрібно якийсь критерій, по якім буде проводитися згортка вихідного зображення символу в матрицю 16x16.

Таким критерієм був обраний загальний відсоток заповнення – відношення кількості значущих пікселей (з яких полягає символ) до загальної кількості пікселей в описаному навколо вихідного зображення прямокутнику.

Даний параметр може впливати на якість розпізнавання, причому якщо він більше 1 для розпізнаваного символу буде відповідати менша кількість можливих альтернатив, при значенні меншому 1 – навпаки.

У нашому випадку коефіцієнт виправлення прийняте рівним 0,99.

```
nsymbol := 0;
for j := ytop to ybottom do
for i := xleft to xright do
 if Masy[j][i] = 0 then inc(nsymbol);
Percent := nsymbol / ((ybottom - ytop)*(xright - xleft));
Percent := 0.99*Percent;
```

4. Далі розбиваємо прямокутник із зображенням символу на 16x16 комірок шляхом розподілу сторін нової комірки на 2. Запам'ятовуємо відносні координати кожної комірки й приступаємо до заповнення матриці 16x16. Приймаємо в якості критерію спільний відсоток заповнення. Якщо в аналізованій комірці відсоток заповнення більше, ніж загальний відсоток – відповідний елемент матриці 16x16 установлюється в 1, а якщо ні, то – в 0.

5. Інша частина алгоритму торкається питань малювання на Tbitmap букв або цифр (у циклі), запам'ятовування в масиві матриць 16x16, що відповідають кожному еталонному символу.

Після цього для розпізнавання здійснюємо порівняння матриці 16x16 шляхом розпізнаваного символу з матрицею еталона (шляхом перебору). Порівняння робимо поєлементно за допомогою оператора XOR. Результат – матриця 16x16, що містить одиниці в місцях розбіжностей тест – символу й еталона. Шляхом підрахунку кількості розбіжностей формуємо вектор, що містить цю інформацію для кожного еталонного символу, і робимо сортування його елементів по зростанню кількості розбіжностей.

Параметр (1 – Result[i]/256)*100%, де Result[i] – кіллу розбіжностей для і – го символу, показує "імовірність" відповідності образа конкретному символу.

Для підвищення точності розпізнавання окремих символів, необхідно здійснювати додатковий аналіз признаків, наприклад симетричність образа (горизонтальна, вертикальна), наявність замкнених областей (ПРО, В, Д, Р и ін.), кількість відрізків і дуг, їх взаємне розташування й орієнтація (потрібно векторізация зображення).

Далі розглянемо завантаження шаблонів скриптов у магістерській програмі. При запуску програми в масив Data: array of array [0..9] of Tbitmap; записуються цифри від 0 до 9, написані наступними шрифтами: Arial, Century Gothic, Courier New Cyr, Goudy Old Style і Times New Roman – усього п'ять комплектів (можна легко збільшити).

```
Procedure Loaddata;
var i,j:integer; path:string;
begin
Setlength(Data,5);
for i := 0 to 4 do
begin
path := Extractfilepath(Application.Exename)+'\fonts\';
case i of
  0: path  := path + 'Arial\';
  1: path  := path + 'Century Gothic\';
  2: path  := path + 'Courier New Cyr\';
  3: path  := path + 'Goudy Old Style\';
  4: path  := path + 'Times New Roman\';
 end;
 for j := 0 to 9 do
  begin
  Data[i,j] := Tbitmap.Create;
Data[i,j].Loadfromfile(path + Inttostr(j) + '.bmp');
  end;
 end;
end;
```

Після завантаження еталонних зображень програма очікує одержання даних на обробку. При одержанні цих даних вираховуються відстані від розпізнаваного малюнка до кожного з еталонних (відстань по Хэммингу).

```
function Compare( b1,b2:Tbitmap):integer;
var i,j,count:integer;
begin
count := 0;
for i := 0 to 49 do
 for j := 0 to 69 do
if b1.Canvas.Pixels[i,j] <> b2.Canvas.Pixels[i,j] then
   inc(count);
   Result := count;
   end;
```

Знаючи цю відстань, легко обчислити потенціал, створюваний кожним еталонним малюнком у точці, що відповідає намальованому користувачем зображенню.

Потенціали, створювані нулями всіх накреслень, підсумовуються в p[0], одиницями – у p[1] і так далі.

```
for i := 0 to 4 do
  for j := 0 to 9 do
   begin
   r := Compare(Image1.Picture.Bitmap,Data[i,j]);
   p[j] := p[j] + 1000000/(1+r*r);
   end;
```

Після всього цього залишається знайти, якому образу відповідає найбільший потенціал.

Запропонований метод дає досить непогані результати як на машинописних, так і на рукописних цифрах.

Алгоритм легко може бути переучений для розрізнення інших символів (латинських/руських/ українських букв, розділових знаків і т.п.).

Для підвищення надійності розпізнавання можна передбачити еталони, що сильно різняться між собою (для того, щоб якнайсильніше розкидати еталонні точки в межах компактної множини). Для розпізнавання цифр я ще використала такий прийом: вихідний малюнок розбивався на 7 ділянок по вертикалі й 5 по горизонталі, ділянку цілком заливався білим або чорним залежно від того, яких пікселей ділянка містила більше, після чого робота з кожною ділянкою відбувалася як з пікселем.

Опис експортованих функцій. Робота з диском. Робота зі сховищем.

1. 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.

2. 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.

3. 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.

Робота з файлами

4. 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.

5. 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.

6. 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).

7. 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.

8. Записати дані на диск.

CFIO_WriteToFile (hFile, lpData, dwSize);

hFile – покажчик отриманий від Open.

lpData – покажчик на зберігаються данные, що.

dwSize – число зберігаємих байт.

Повертає число записаних байт.

9. 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.

10. 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.

11. 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.

12. Скинути кешовані дані на диск.

CFIO_FlushFile(hFile);

hFile – дескриптор, отриманий від Open;

Робота з пам'яттю

13. 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.

14. Перевиділити блок пам'ять.

CFIO_ReAllocMemory (hMemory, dwSize, dwFlag);

hMemory – дескриптор блоку пам'яті.

dwSize – реаллокуємий розмір.

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.

15. Звільнити блок пам'яті.

CFIO_FreeMemory(hMem);

hMem – звільняється память, що.

16. 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.

17. Відкріпити глобальний об'єкт.

CFIO_UnlockMemory(hMem);

hMem – покажчик на глобальний об'єкт.

18. 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.

19. 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.

20. 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.

21. 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.

Приведемо опис нотифікації ряду функцій, які використовуються у розробленій, у результаті виконання магістерського проекту, системі проектування нейронної мережі.

Розглянемо контейнер збереження даних.

Опис реалізованих функцій:

1. Почати роботу з бібліотекою.

CFIO_Init(Word16 wHeightCode, hStorage);

WHeightCode – старші два байти коду повернення.

HStorage – дескриптор поточного сховища даних нейронної мережі, NULL для даного модуля NULL.

Значення, що повертається, TRUE – у випадку успіху.

2. Завершити роботу з бібліотекою.

CFIO_Done();

Значення, що повертається, TRUE – у випадку успіху.

3. Одержати код помилки.

CFIO_GetReturnCode();

Значення, що повертається – код помилки. Нуль – помилок немає.

4. Одержати рядок опису помилки.

CFIO_GetReturnString(dwError);

DwError – код помилки.

5. Одержати покажчик експортованої функції або дані.

CFIO_GetExportData (dwType, pData);

dwType – число, номер експортованої функції або даних.

pData – покажчик на експортовану функцію або дані.

Значення, що повертається, TRUE – якщо опції встановлені, FALSE – якщо опції помилкові або функція не реалізована.

Виклик CFIO_GetReturnCode() може приймати значення:

– NO_ERROR – помилок немає.

– FUNCTION_NOT_IMPLEMENT – функція не реалізована.

6. Передати покажчик імпортованої функції або дані.

CFIO_SetImportData (dwType, pData)

dwType – число, номер імпортованої функції або даних.

pData – покажчик на імпортовану функцію або дані.

Значення, що повертається, TRUE – якщо опції встановлені, FALSE – якщо опції помилкові або функція не реалізована.

Виклик CFIO_GetReturnCode() може приймати значення:

– ERR_NONE – помилок немає.

– CFIO_ERR_NOT_IMPLEMENT – функція не реалізована.

Опис імпортованих даних:

1. CFIO_PCHAR_TEMPORARY_FOLDER – папка для тимчасових файлів.
2. CFIO_PCHAR_STORAGE_FOLDER – папка для створення сховища даних нейронної мережі за замовчуванням.
3. CFIO_PCHAR_FILE_FOLDER – папка для створення файлів за замовчуванням.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Sinople – симетричний блоковий криптоалгоритм, побудований на основі незбалансованої «мережі Фейстеля». Алгоритм розроблено у 2003 році.

Основні вимоги до алгоритму при його розробці:

– Можливість програмної і апаратної реалізації.
– Висока швидкість.
– Простота.
– Низькі вимоги до пам'яті.
– Високий рівень безпеки.

Алгоритм заснований на 32-розрядних операціях і має 64 раунду, серед яких два типи – C і D. D раунди спроектовані для досягнення максимальної дифузії, C раунди – для досягнення перемішування. F-функція D раунду використовує один з елементів блоку даних (D[3]) та поточного з'єднання (K[r]) для трансформації 3-х елементів блоку даних. F-функція C раунду, навпаки, використовує перші три елемента блоку даних і поточний з'єднання (K[r]) для трансформації останнього елемента блоку даних (D[3]). Раунди D-типу виконуються до раундів C-типу. Додавання ключів з даними проводиться тільки через таблиці замін. Операції XOR (додавання по модулю 2) обов'язково поєднуються з операціями ADD (додавання по модулю 2^{32}).

Таблиці замін спочатку запозичені з алгоритму MARS і містять 512 32-розрядних елементів, проте були жорстко проаналізовано на предмет посилення.

Ключове розклад було спроектовано з урахуванням вимог:

– Простота

– Використовується та ж процедура, що і при шифруванні та розшифрування

– Установка ключа займає менше часу, ніж зашифрування

– Виключення еквівалентних ключів

– Виключення слабких ключів

Алгоритм, згідно із заявою авторів, стійкий до лінійного і диференціального аналізу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ, головне вікно якого зображено на рисунку 5.1. Як можна побачити з рисунку у вікні проходе чекання рукописного введення даних на канву.

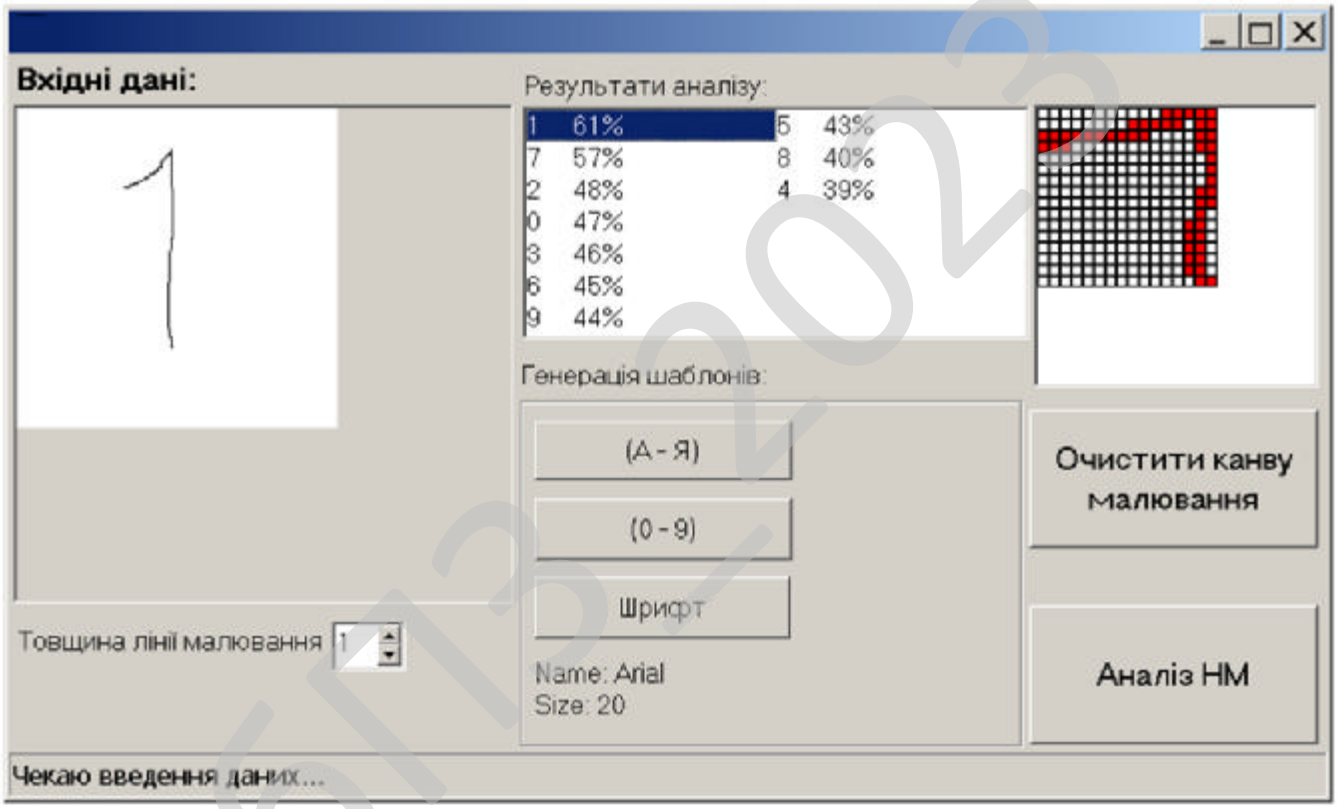

Рисунок 5.1 – Головне вікно програми

Перед тим як ПЗ почне працювати необхідно згенерувати шаблони символів А-Я та цифр 0-9 для коректної роботи ПЗ.

Під час генерування вибирається тип шрифту (по замовчанню це Arial він більш всього схожий на рукописні прямі букви) та розмір (по замовчанню 30 пт.). Після вдалого генерування на екран виводиться вікно яке зображено на рисунку 5.2.

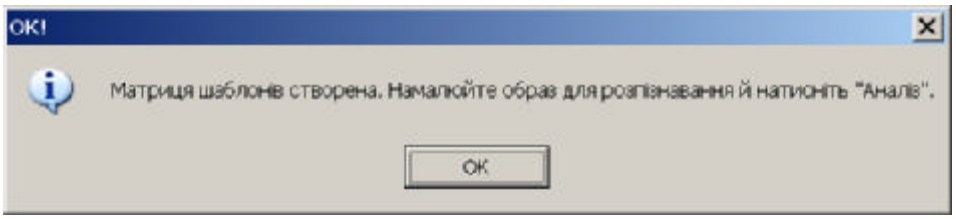

Рисунок 5.2 – Вікно інформування генерації матриці шаблонів

Після запуску ПЗ, та генерування шаблонів починається чекання на отримання зображення.

На рисунку 5.3 зображено форму авторського права. Була вибрана ліцензія Freeware. При розробці ПЗ була створена форма авторського права з ліцензій на використання програмного забезпечення. Ліцензія на використання програмного забезпечення це вид ліцензії, що визначає умови використання виробу, яким є комп'ютерне програмне забезпечення. Ліцензія може надавати дозвіл робити з ним речі, які були б інакше заборонені законом про авторське право.

Наприклад, ліцензія на використання програмного забезпечення може дати дозвіл робити копії програмного забезпечення. Власник авторського права може запропонувати ліцензію на використання ПЗ односторонньо, або як частину ліцензійної угоди на використання ПЗ з іншою стороною.

Рисунок 5.3 – Довідка

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи створення та навчання нейронної мережі для розпізнавання образів.

Метою розробки є дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів.

Об'єктом дослідження є процес створення та навчання нейронної мережі для розпізнавання образів.

Предметом дослідження є методи створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання образів.

– Розроблено вітчизняний продукт створення та навчання нейронної мережі для розпізнавання 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 48 днів (два місяці). В магістерській роботі було проведено дослідження та виконана програмна реалізація системи створення та навчання нейронної мережі для розпізнавання 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	17
3. Запланований термін розробки, днів	Fpq	48 (2 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	–	7
8. Кількість форм вихідної інформації.	–	8
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	2
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	1
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)	–	3
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	1
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	1
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)	–	1
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	3
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	3
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)	–	1
29. Досвід роботи з програмними інструментами розробки (1-6)	–	2
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	3
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	17000
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.

$$B = 1{,}01+0{,}001(3{,}24+3{,}64+3{,}38+4{,}94+2{,}73) = 1{,}028.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}028} = 6{,}8 \text{ люд-міс.}$$

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

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

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

$$T_{уточн} = 6{,}8 \cdot (0{,}88 \cdot 1 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}89 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1 \cdot 1{,}1 \cdot 1 \cdot$$

$$\cdot 1{,}12 \cdot 1{,}22 \cdot 1{,}12 \cdot 1 \cdot 1{,}1) = 8{,}38 \text{ люд-міс.}$$

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)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 8{,}38^{0{,}33+0{,}2(1{,}028-1{,}01)} \cdot 53 = 86 \text{ люд/день.}$$

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	15	Д7
Робочий проект	86	Ф 7.1-7.4
Впровадження	17	Д13
Всього	137	–

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.

$$Ч = \frac{137 \cdot 1}{48\text{-}5} = 3,2 \text{ ставки.}$$

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	8	720	12
Монітор	60	8	480	8
Клавіатура	30	8	240	4
Маніпулятор «мишка»	30	8	240	4
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	8	240	4
Кабельні господарства ЛВС на 1 м. п.	2,5	180	450	7,5
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	45,16

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{45 \cdot 2}{1,2} = 75 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 75/(48 \cdot 8) = 0{,}2 \text{ ставки.}$$

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 2022, 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)	1	22937,5	45875
Продакт-менеджер	0,25	16000	8000
Інженер-програміст	3,2	21000	134400
Інженер-електронщик	0,2	16000	6400
Інженер-системотехнік	0,25	16000	8000
Адміністратор мережі	0,5	16000	16000
Системний програміст	0,25	16000	8000
Дизайнер WEB	0,25	16000	8000
Інженер-верстальник	0,25	16000	8000
Бухгалтер-економіст	0,25	16000	8000
Всього за період розробки	$R_{сп}$ = 6,4	-	$Ф_{роб}$ = 250675

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.

$$З_{сд} = \frac{250675}{6,4 \cdot 48} = 816 \; грн.$$

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 400…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:

$$Б_{уд} = 9 \cdot 8 \cdot 20000 = 1440000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 144000 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.

$$І_{нв} = 9 \cdot 3500 = 31500 \text{ грн.}$$

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 supercomp.kiev.ua за 22.10.23 – джерело http: //supercomp.kiev.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		12721
Системний блок		7721
Процесор	AMD Ryzen 3 2200G PRO (YD220BC5M4MFB) AM4, 4 ядра, 4 потоки, 3.5 GHz, 3.7 GHz, TDP – 65 Вт, 14nm, L1: 384KB, L2: 2MB, L3: 4MB, Radeon Vega 8, Zen, Tray	-
Системна плата	Biostar A520MH 3.0, AM4, DDR4, 64 ГБ, 4400 MHz, LAN – 1 Гбіт/с, D-Sub (VGA), HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Відеокарта	Інтегрована Radeon Vega 8	-
Жорсткий диск	480 GB SSD	-
Оперативна пам’ять	DDR4 8GB 2666 MHZ KINGSTON (KVR26N19S8/8)	-
Система охолодження	Vinga CL3011 Для процесорів – INTEL, AMD, сокет – 1151, 1150, 1155, AM2, AM3, AM3+, AM4, 775, 1156	-
Корпус	Vinga CS104B-500W Класіка, Miditower, ATX, Micro – ATX, Mini – ITX, PSU – 500 Вт	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	220
інше	Клавіатура, мишка	Подару-нок
Монітор	Монітор BenQ GL2450HM Black	3600

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

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

Для визначення необхідної кількості капітальних вкладень складем 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	9	12721	11448,9	125937,9
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	147569,4

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	1440000	-	-
2. Передавальні пристрої	144000	-	-
Всього по групі	1584000	5	79200
Група 4			
3. Обчислювальна техніка	147570	-	-
Всього по групі	147570	40	59028
Нематеріальні активи			
4. Нематеріальні активи	17000	10	1700
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	121875	20	24375
7. Господарський інвентар	31500	25	7875
Всього по групі	162406	-	34507,75
Разом	$К_р$ = 1910976		$А_р$ = 174435,75

Примітка: вартість автомобіля взята по даним електронного ресурсу https://auto.ria.com/uk/auto_daewoo_lanos_33592444.html та складає 121875 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.

$$З_о = 816 \cdot 137/17 = 6579 \text{ грн.}$$

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, %.

$$З_д = 6579 \cdot 10 \cdot 0{,}01 = 658 \text{ грн.}$$

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, %.

$$С_{оц} = 0{,}01 \cdot 22(6579+658) = 1592 \text{ грн.}$$

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, %.

$$Г_{осп} = 6579 \cdot 15 \cdot 0{,}01 = 987 \text{ грн.}$$

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 5 пачек паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=206 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 206 \cdot 5 \cdot 1 = 1030 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 17):

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 35,6 грн./шт., DVD-R box – 55 грн./шт.

$$З_{М2} = 17 \cdot 55 = 935 \text{ грн.}$$

Згідно виданих 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.

$$З_{М3} = 500+508+1155 = 2163 \text{ грн.}$$

$$З_М = (1030 + 935 + 2163)/17 = 243 \text{ грн.}$$

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, %.

$$О_п = 6579 \cdot 15 \cdot 0{,}01 = 987 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 17 прим.):

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

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

$$A_м = 174436 \cdot 2/(17 \cdot 12) = 1710 \text{ грн.}$$

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	$З_о$	6579
2. Додаткова зарплата виконавців	$З_д$	658
3. Відрахування на соціальні потреби	$С_{оц}$	1592
4. Загальногосподарські витрати	$Г_{осп}$	987
5. Витрати на матеріали	$З_М$	243
6. Освоєння нових операційних систем, мов програмування	$О_п$	987
7. Амортизація основних фондів	$А_м$	1710
8. Повна собівартість програмного забезпечення	$С_п$	12756
9. Плановий прибуток	$П_р$	6378
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	19134
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	3826,8
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	22960,8

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

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

$$C_п = 6579+658+1592+987+243+987+1710 = 12756 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 12756 = 6378 \text{ грн.}$$

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.

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

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	22961
Всього капітальних витрат	–	22961

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 обслуговування	$З_р$	24156	9664
2. Витрати на електроенергію	$З_{ел}$	114	46
3. Витрати на амортизацію	$З_{ам}$	0	5740
Всього витрат за рік	$І$	24270	15450

Витрати на обслуговування системи:

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

де: $Т_р$ – кількість годин обслуговування за рік, *год.*;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 200 годин на рік до 80 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\,баз} = 200 \cdot 90 \cdot 1{,}1 \cdot 1{,}22 = 24156 \text{ грн},$$

до:

$$З_{р\,нов} = 80 \cdot 90 \cdot 1{,}1 \cdot 1{,}22 = 9664 \text{ грн}.$$

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 ($Ц_{ел}$):

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

$$З_{ел\,баз} = 0{,}15 \cdot 200 \cdot 3{,}8 = 114 \text{ грн}.$$

$$З_{ел\,нов} = 0{,}15 \cdot 80 \cdot 3{,}8 = 46 \text{ грн}.$$

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	–	22961	–	5740,25
Всього відрахувань	-	–	22961	–	5740,25

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:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.24)$$

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

$$Ев = (19134 - 12756) \cdot 17 - (0,05 \cdot 1584000 + 0,4 \cdot 147570 + \\ + 0,2 \cdot 121875 + 0,25 \cdot 40531 + 0,1 \cdot 17000) \cdot 2/12 = 79353 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.25)$$

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

$$T_в = \frac{326976}{(19134 - 12756) \cdot 17 \cdot 12 / 2} = 0,5 \text{ років}.$$

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.	17
2. Повна собівартість розробленої програми	Грн.	12756
3. Ціна розробленої програми	Грн.	19134
4. Плановий прибуток від реалізації розробленої програми	Грн.	6378
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1910976
7. Загальний прибуток від реалізації програмної продукції	Грн.	108426
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	79353
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,5
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	22961
11.Величина економічного ефекту у користувача програмної продукції	Грн.	3080
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	2,6

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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.

$$E_{сп} = (24270\text{-}15450)\text{-}0{,}25 \cdot 22961 = 3080 \text{ грн.}$$

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

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{22961}{24270 - 15450} = 2{,}6 \text{ року}.$$

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.

Законом України "Про охорону праці" [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207 [4], який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2].

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням [2]. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами:

«Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [2], та «Правила охорони праці під час експлуатації електронно-обчислювальних машин» НПАОП 0.00-1.28-10,

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста,

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Це важливе питання, адже робота з комп'ютером може впливати на здров'я людини. За ДСанПіН 3.3.2.007-98 шкідливі і небезпечні фактори при роботі з комп'ютером можуть бути такі:

– Електромагнітне випромінювання – це випромінювання, яке створюється комп'ютером і його периферійними пристроями, такими як монітор, пртер, сканер тощо. Це випромінювання може викликати головний біль, запаморчення, розлади сну, зниження імунітету та інші негативні ефекти.

– Електростатичне поле – це поле, яке утворюється внаслідок накопичення електричних зарядів на поверхні комп'ютера і його частин. Це поле може спричиняти сухість шкіри, свербіж, подразнення очей, алергічні реакції та інші проблеми.

– Нервово-емоційне напруження – це напруження, яке виникає внаслідок трвлої концентрації уваги, високої вимогливості до результату роботи, нестабільності програмного забезпечення, конфліктних ситуацій тощо. Це

напруження може призводити до стресу, депресії, роздратування, погіршення пам’яті та інших порушень.

– Навантаження на органи зору – це навантаження, яке виникає внаслідок тривалого перебування перед монітором, низької якості зображення, недостнго освітлення приміщення, поганої ергономіки робочого місця тощо. Це навантаження може спричиняти зниження гостроти зору, появу блимавок і кругів перед очима, синдром сухого ока та інші захворювання.

– Дрібні стереостатичні рухи кінцівок – це рухи, які пов’язані з керуванням клавіатурою і мишею. Ці рухи можуть призводити до перевантаження і запалення суглобів і сухожиль рук і пальців, а також до тендовагінітів і синдрому зап’ясткового каналу.

Таким чином, робота з комп’ютером не така безпечна, як може здатися на перший погляд. Тому дуже важливо дотримуватися правил охорони праці і гігієни при роботі з комп’ютером.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	6,2
Довжина	6,8
Висота	3,4

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6
Обсяг, V	$м^3$	не менше 20.0	19,2

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин) [2].

У зазначеному приміщенні працє 7 людей. За даними, які наведено у табл. 8.1 та табл. 8.2, можна зробити висновок, що площа приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам (Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], а об'єм – НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови Головного державного санітарного лікаря України [5], робота, яка виконується в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 кКал. у годину. Вологість повітря у приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	21,8-23	40-60	0,1
Тепла	23-25	50-70	0,1	23-25	52-70	0,11

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер HP Laser 107a, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5 – 28 – 2006 р можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

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 значень.

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

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).

8.5 Розрахункова частина

Завдання: розрахувати *штучне освітлення робочого приміщення.*

Початкові дані: ширина *робочого* приміщення: 6,2 м.; довжина –6,8 м.; висота – 3,4 м.

Розрахунок штучного освітлення проводемо за методом коефіцієнта використання світлового потоку.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S=6,2 х 6,8 = 42,2* м^2);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50% *[6]*.

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: *S* – площа приміщення, *S = 42,2* м^2;

h – розрахункова висота підвісу, *h = 3,4* м;

A – ширина приміщення, А = 6,2 м;

B – довжина приміщення, В = *6,8* м.

Підставимо всі значення у формулу та визначемо індекса приміщення: *i=0,57.*

Знаючи індекс приміщення, за знаходимо *n = 0.29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу).* Підставимо всі значення у формулу, визначемо світловий потік: *F=45368 Лм.*

Для штучного освітлення приміщення використовуються *LED панель MAXUS ASSISTANCE PRO 80W 5000K WHITE (M1052480531)*, світловий потік яких $F_л = 8000$ Лм.

Число світильників визначається по формулі:

$$N=F/F_л$$

де: *F* – світловий потік,

$F_л$ – світловий потік одного *світильника*.

$$N= 45368/ 8000=5,7 \text{ шт.}$$

Приймаємо необхідну кількість світильників 6 шт.

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому. 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. Виконано розрахунок захисного штучного освітлення. Розроблено заходи з охорони праці.

Список використаних джерел інформації

1. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

2. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2.007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

3. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

4. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

5. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

6. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький: ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення: 16.06.2023).

7. Методичні рекомендації до виконання розділу «Заходи з охорони праці та техніки безпеки» у магістерській дисертації / Л.Д. Третякова; М-во освіти і науки України, Національний технічний університет україни «Київський політехнічний інститут» – Київ, КПІ, 2014. – 26 с. – Режим доступу до ресурсу: http://surl.li/dhulo (дата звернення: 16.06.2023).

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання образів.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем створення та навчання нейронної мережі для розпізнавання образів.

– Досліджена система створення та навчання нейронної мережі для розпізнавання образів.

– На основі отриманих результатів досліджень створена програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання створення та навчання нейронної мережі для розпізнавання 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 Sinople.

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 3080 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 2,6 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Михайлов Д.В. Дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.
2. Foreman J.W. Data Smart: Using Data Science to Transform Information into Insight 1st Edition. – Wiley, 2013. – 432 p.
3. Hurbans R. Grokking Artificial Intelligence Algorithms. – Manning, 2020. – 631 p.
4. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.
5. Kotu V., Deshpande B. Data Science: Concepts and Practice. – Elsevier Science, 2018. – 953 p.
6. Knowledge Base A Complete Guide – 2021 Edition // The Art of Service – Knowledge Base Publishing, 2020. – 306 p.
7. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.
8. Mattmann C. Machine Learning with TensorFlow, Second Edition. – Manning, 2020. – 1124 p.
9. Mueller J.P., Massaron L. Machine Learning For Dummies. – Wiley, 2016. – 714 p.
10. Teofili T. Deep Learning for Search. – Manning, 2019. – 695 p.
11. Rungta K. TensorFlow in 1 Day: Make your own Neural Network. – Publishdrive, 2019. – 587 p.
12. Weidman S. Deep Learning from Scratch: Building with Python from First Principles. – O'Reilly. 2019. – 252 p.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

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

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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).

37. 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.

38. 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.

39. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». Сучасні інформаційні системи. 2021. Т. 5, № 4. С. 79-95

40. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

41. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

42. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

43. Смірнов, С.А., Смірнова, Т.В., Минайленко, Р.М., Доренський, О.П., Сисоєнко С.В. «Хмарна автоматизована система інтелектуальної підтримки прийняття рішень для технологічних процесів». Вісник Черкаського державного технологічного університету. Технічні науки. №4, 2020, С. 84-92.

44. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

45. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

46. О.А. Смірнов, Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура № 1 (11). с. 48-57, 2019.

47. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

48. Смірнова Т.В., Дрєєв О.М., Смірнов О.А. Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням. Системи управління, навігації та зв'язку, № 2 (54). С. 149-154, 2019.

49. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

50. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

51. Смірнов О.А., Коваленко О.В. Використання псевдобулевих методів бівалентного програмування для управління ризиками розробки програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103

52. Смірнов О.А., Лисенко І.А. Формалізація процесу проектування тестових наборів. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 3 (48). – Харків: ХУПС. – 2016. – С.96-100.

53. Смірнов О.А., Лисенко І.А. Удосконалення методу перевірки коректності таблиць рішень для подання тестових наборів. Збірник наукових праць "Системи обробки інформації". – Випуск 8 (145). – Х.: ХУПС – 2016. – С. 77-80.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи створення та навчання нейронної мережі для розпізнавання 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. № 33-13 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 створення та навчання нейронної мережі для розпізнавання 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 створення та навчання нейронної мережі для розпізнавання 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 повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

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 – 140 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 21.12.2023 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

_____ Петренюк В.І.

Дослідження та програмна реалізація системи створення та навчання нейронної мережі для розпізнавання образів

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 63

Літера: РП

Кропивницький – 2023 року

NeuralNetExtend.pas – навчання мережі

```
unit NeuralNetExtend;

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
  TfrmNeuralNetExtend = 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 NeuralNetExtendedEpochPassed(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 NeuralNetExtendedAfterTeach(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
  frmNeuralNetExtend: TfrmNeuralNetExtend;
implementation

{$R *.DFM}

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

// Вихід із програми
procedure TfrmNeuralNetExtend.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 TfrmNeuralNetExtend.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 TfrmNeuralNetExtend.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 TfrmNeuralNetExtend.ChangePage;
begin
  case PageControl.ActivePage.PageIndex of
    1: OpenFile;
    2: LoadFile;
    3: Tune;
    4: CreateNet;
    6: TestNet;
  end;
end;

// Вибрати файл - джерело даних
procedure TfrmNeuralNetExtend.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 TfrmNeuralNetExtend.btnOpenFileClick(Sender: TObject);
begin
  OpenDialog.Execute;
  Caption := FormCaption + ' - ' + ExtractFileName(OpenDialog.FileName);
  edtFileName.Text := OpenDialog.FileName;
end;

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

// Настроювання параметрів мережі
procedure TfrmNeuralNetExtend.Tune;
var
  i: integer;
begin
```

```
  speLayers.Value := NeuralNetExtended.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(NeuralNetExtended.Layers[i +
1].NeuronCount);
  end;

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

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

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

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

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

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

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

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

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

// Змінити тип поля
procedure TfrmNeuralNetExtend.rdgFieldTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
end;

// Змінити тип нормалізації
procedure TfrmNeuralNetExtend.rdgNormTypeClick(Sender: TObject);
```

```
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
end;

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

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

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

// Створення мережі обраної топології
procedure TfrmNeuralNetExtend.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 NeuralNetExtended 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 TfrmNeuralNetExtend.edtMomentumChange(Sender: TObject);
begin
  NeuralNetExtended.Momentum := StrToFloat(edtMomentum.Text);
end;

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

// Дія по проходженню однієї епохи навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedEpochPassed(Sender: TObject);
begin
  sttMaxTestError.Caption := '';
  sttMidTestError.Caption := '';
  with NeuralNetExtended do
  begin
    sttMaxTeachError.Caption := FloatToStr(MaxTeachResidual); // Показати макс.
помилку на навчальній множині
    sttMidTeachError.Caption := FloatToStr(MidTeachResidual); // Показати серед.
помилку на навчальній множині
    if NeuralNetExtended.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 TfrmNeuralNetExtend.btnContinueTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := true; // Скинути прапор - "Продовжити
навчання"
  RunTeach;  // Запуск
end;

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

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

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

procedure TfrmNeuralNetExtend.cbxEpochClick(Sender: TObject);
begin
  NeuralNetExtended.Epoch := cbxEpoch.Checked;
end;

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

procedure TfrmNeuralNetExtend.cbxMaxTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachError := cbxMaxTeachError.Checked;
end;

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

procedure TfrmNeuralNetExtend.cbxMidTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTeachError := cbxMidTeachError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMidTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTeachErrorValue :=
StrToFloat(edtMidTeachErrorValue.Text);
end;
```

```
procedure TfrmNeuralNetExtend.cbxTeachIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TeachIdent := cbxTeachIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTeachIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTeachIdentValue.Value;
end;

procedure TfrmNeuralNetExtend.cbxMaxTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTestError := cbxMaxTestError.Checked;
end;

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

procedure TfrmNeuralNetExtend.cbxMidTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTestError := cbxMidTestError.Checked;
end;

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

procedure TfrmNeuralNetExtend.cbxTestIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TestIdent := cbxTestIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTestIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTestIdentValue.Value;
end;

// Зупинка навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedAfterTeach(Sender: TObject);
begin
  btnBack.Enabled := true;
  btnNext.Enabled := true;
  btnCancel.Enabled := true;
  btnContinueTeach.Caption := 'Навчити';
  NeuralNetExtended.StopTeach := true;
end;

procedure TfrmNeuralNetExtend.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;
    NeuralNetExtended.StopTeach := false;
    btnContinueTeach.Caption := 'Зупинити навчання';
    NotSaved := true;
    NeuralNetExtended.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';
    NeuralNetExtended.StopTeach := true; // Зупинити навчання
  end;
end;

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

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

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

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

// Зберегти навчену нейромережу
procedure TfrmNeuralNetExtend.btnSaveClick(Sender: TObject);
begin
  SaveDialog.InitialDir := ExtractFilePath(NeuralNetExtended.FileName);
  SaveDialog.FileName := ExtractFileName(NeuralNetExtended.FileName);
  if SaveDialog.Execute then
```

```
  begin
    NeuralNetExtended.NnwFile := TIniFile.Create(SaveDialog.FileName);
    NeuralNetExtended.SavePhase1;
    NeuralNetExtended.SavePhase2;
    NeuralNetExtended.SavePhase4;
    NeuralNetExtended.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;
    NeuralNetHopf: TNeuralNetHopf;
    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
  NeuralNetHopf.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
  NeuralNetHopf.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;


  NeuralNetHopf.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
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := 1
      else
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := -1;

  // Запуск процесу розпізнавання
  NeuralNetHopf.Calc;

  // Перетворення виходів мережі до рядка
  xString := '';
  for i := 1 to stgOutput.RowCount * stgOutput.ColCount do
    if NeuralNetHopf.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;
  xNeuralNetExtended: TNeuralNetExtended;
  frmNeuroFields: TfrmNeuroFields;
  xCurrent: integer;
begin
  frmNeuroFields := TfrmNeuroFields.Create(nil);
  try
    with frmNeuroFields do
    begin
      xNeuralNetExtended := (GetComponent(0) as TNeuralNetExtended);
      for i := 0 to xNeuralNetExtended.AvailableFieldsCount - 1 do
        ltbFieldName.Items.Add(xNeuralNetExtended.Fields[i].Name);
      xCurrent := 0;
      rdgFieldType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].Kind;
      rdgNormType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].NormType;
      edtAlpha.Text := FloatToStr(xNeuralNetExtended.Fields[xCurrent].Alpha);
      sttMin.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMin);
      sttMax.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMax);
      NeuralNetExtended := xNeuralNetExtended;
      ShowModal;
    end;
```

```
  except
    frmNeuroFields.Free;
  end;
end;

function TOptionsFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetExtended) 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 TNeuralNetBP) 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;
  xNeuralNetBP: TNeuralNetBP;
  frmNeuronsInLayer: TfrmNeuronsInLayer;
  xChangesMade: boolean;
begin
  xChangesMade := False;
  frmNeuronsInLayer := TfrmNeuronsInLayer.Create(nil);
  try
    with frmNeuronsInLayer do
    begin
      xNeuralNetBP := (GetComponent(0) as TNeuralNetBP);
      speLayers.Value := xNeuralNetBP.LayerCount;
      stgNeuronsInLayer.RowCount := xNeuralNetBP.LayerCount + 1;
      for i := 0 to xNeuralNetBP.LayerCount - 1 do
      begin
        stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
        stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(xNeuralNetBP.LayersBP[i].NeuronCount);
      end;
      if ShowModal = mrOk then
      begin
        xNeuralNetBP.ResetLayers;
```

```
        if speLayers.Value = 0 then
        begin
          xNeuralNetBP.ResetLayers;
          Exit;
        end;
        if xNeuralNetBP.LayerCount <> speLayers.Value then
          xChangesMade := True
        else
          for i := 0 to xNeuralNetBP.LayerCount - 1 do
            if xNeuralNetBP.LayersBP[i].NeuronCount <>
StrToInt(stgNeuronsInLayer.Cells[1, i + 1]) then
            begin
              xChangesMade := True;
              Break;
            end;
        if xChangesMade then
        begin
          if xNeuralNetBP.LayerCount = speLayers.Value then
            for i := 0 to speLayers.Value - 1 do
              xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1]
          else
            if xNeuralNetBP.LayerCount < speLayers.Value then
            begin
              for i := 0 to xNeuralNetBP.LayerCount - 1 do
                xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1];
              for i := xNeuralNetBP.LayerCount to speLayers.Value - 1 do
                xNeuralNetBP.AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i +
1]));
            end
            else
            begin
              if speLayers.Value > 0 then
                for i := 0 to speLayers.Value - 1 do
                  xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i
+ 1];
              xPreviousCount := xNeuralNetBP.LayerCount -  speLayers.Value;
              for i := 1 to xPreviousCount do
                xNeuralNetBP.DeleteLayer(xNeuralNetBP.LayerCount - 1);
            end;
          if not xNeuralNetBP.AutoInit then
            xNeuralNetBP.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 TNeuralNetExtended).FileName := xOpenDialog.FileName;
    if UpperCase(GetName) = 'SOURCEFILENAME' then
      (GetComponent(0) as TNeuralNetExtended).SourceFileName :=
xOpenDialog.FileName;
  end;
  xOpenDialog.Free;
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(TStrings), TNeuralNetBP, 'NeuronsInLayer',
TNeuronsInLayerFieldsProperty);
  RegisterPropertyEditor(TypeInfo(TFileName), TNeuralNetExtended, '',
TFileNameFieldsProperty);
  RegisterPropertyEditor(TypeInfo(string), TNeuralNetExtended, '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
    NeuralNetExtended: TNeuralNetExtended;
    { Public declarations }
  end;

var
  frmNeuroFields: TfrmNeuroFields;

implementation

{$R *.DFM}

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

procedure TfrmNeuroFields.rdgFieldTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.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
        NeuralNetExtended.Fields[i].Kind := rdgFieldType.ItemIndex;
end;

procedure TfrmNeuroFields.rdgNormTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.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
        NeuralNetExtended.Fields[i].NormType := rdgNormType.ItemIndex;
end;

procedure TfrmNeuroFields.edtAlphaChange(Sender: TObject);
begin
  NeuralNetExtended.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
  TNeuralNet = 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
  TNeuralNetHopf = class(TNeuralNet)
  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
  TNeuralNetBP = class(TNeuralNet)
  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 TNeuralNetExtended }
  TNeuralNetExtended = class(TNeuralNetBP)
```

```
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;

{ TNeuralNet }

destructor TNeuralNet.Destroy;
begin
  Clear;
  SetLength(FPatternsInput, 0, 0);
  FPatternsInput := nil;
  SetLength(FPatternsOutput, 0 ,0);
  FPatternsOutput := nil;
  FLayers := nil;
  inherited;
end;

procedure TNeuralNet.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 TNeuralNet.GetInputNeuronCount: integer;
```

```
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNet.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNet.GetLayers(Index: integer): TLayer;
begin
  Result := FLayers[Index];
end;

function TNeuralNet.GetOutputNeuronCount: integer;
begin
  Result := Layers[LayerCount - 1].NeuronCount;
end;

function TNeuralNet.GetPatternsInput(PatternIndex: integer; InputIndex:
integer): double;
begin
  Result := FPatternsInput[PatternIndex, InputIndex];
end;

procedure TNeuralNet.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 TNeuralNet.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 TNeuralNet.ResetPatterns;
begin
  FPatternCount := DefaultPatternCount;
  ResizeInputDim;
```

```
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternsOutput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsOutput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.SetPatternsInput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsInput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.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 TNeuralNet.ResizeInputDim;
begin
  SetLength(FPatternsInput, FPatternCount, InputNeuronCount)
end;

procedure TNeuralNet.ResizeOutputDim;
begin
  SetLength(FPatternsOutput, FPatternCount, OutputNeuronCount)
end;

procedure TNeuralNet.SetLayerCount(Value: integer);
begin
  SetLength(FLayers, Value);
end;

procedure TNeuralNet.SetLayers(Index: integer; Value: TLayer);
begin
  try
    FLayers[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;
```

```
{  TNeuralNetHopf }

constructor TNeuralNetHopf.Create(AOwner: TComponent);
begin
  inherited;
  PatternCount := DefaultPatternCount;
  InputNeuronCount := DefaultNeuronCount;
  MaxIterCount := DefaultMaxIterCount;
  AutoInit := False;
end;

destructor TNeuralNetHopf.Destroy;
begin
  FOnAfterInit := nil;
  FOnBeforeInit := nil;
  FOnPatternRecognized := nil;
  SetLength(FPatterns, 0, 0);
  FPatterns := nil;
  inherited;
end;

function TNeuralNetHopf.GetInput(Index: integer): double;
begin
  Result := Layers[1].Neurons[Index].Output;
end;

function TNeuralNetHopf.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNetHopf.GetLayerCount: integer;
begin
  Result := DefaultHopfLayerCount;
end;

function TNeuralNetHopf.GetPatterns(InputIndex: integer; PatternIndex: integer):
integer;
begin
  Result := FPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetHopf.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 TNeuralNetHopf.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 TNeuralNetHopf.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 TNeuralNetHopf.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 TNeuralNetHopf.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 TNeuralNetHopf.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 TNeuralNetHopf.ResetPatterns;
begin
  PatternCount := DefaultPatternCount;
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.ResizePatternsDim;
begin
  SetLength(FPatterns, FPatternCount, FInputNeuronCount);
end;

procedure TNeuralNetHopf.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 TNeuralNetHopf.SetInputNeuronCount(Value: integer);
begin
  if Value > DefaultNeuronCount then
    FInputNeuronCount := Value
  else
    FInputNeuronCount := DefaultNeuronCount;
  ResizePatternsDim;
  Init;
end;

procedure TNeuralNetHopf.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
end;

procedure TNeuralNetHopf.SetPatterns(InputIndex: integer; PatternIndex: integer;
Value: integer);
begin
  FPatterns[InputIndex, PatternIndex] := Value;
end;

{ TNeuralNetBP }

constructor TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.GetLayersBP(Index: integer): TLayerBP;
begin
  Result := FLayers[Index] as TLayerBP;
end;

function TNeuralNetBP.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNetBP.GetDesiredOut(Index: integer): double;
begin
  Result := FDesiredOut[Index];
end;

function TNeuralNetBP.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 TNeuralNet.GetPatternsOutput(PatternIndex: integer; OutputIndex:
integer): double;
begin
  Result := FPatternsOutput[PatternIndex, OutputIndex];
end;

function TNeuralNetBP.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 TNeuralNetBP.Activation(Value: double): double;
begin
  // Активаційна функція - сигмоїд
  Result := 1/( 1 + exp(-FAlpha * Value) )
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Похідна сигмоїди
  Result := FAlpha * Value * (1 - Value)
end;

function TNeuralNetBP.GetTestSetPatterns(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetBP.GetTestSetPatternsOut(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatternsOut[InputIndex, PatternIndex];
end;

procedure TNeuralNetBP.AddLayer(ANeurons: integer);
begin
  if ANeurons < DefaultNeuronCount then
    NeuronCountError
  else
    NeuronsInLayer.Add(IntToStr(ANeurons));
end;

procedure TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.DoOnAfterInit;
begin
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetBP.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 TNeuralNetBP.DoOnAfterTeach;
begin
  if Assigned(FOnAfterTeach) then
    FOnAfterTeach(Self);
end;

procedure TNeuralNetBP.DoOnBeforeInit;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
end;

procedure TNeuralNetBP.DoOnBeforeTeach;
begin
  if Assigned(FOnBeforeTeach) then
    FOnBeforeTeach(Self);
end;

procedure TNeuralNetBP.DoOnEpochPassed;
begin
  if Assigned(FOnEpochPassed) then
    FOnEpochPassed(Self);
end;

procedure TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.LoadPatternsInput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to InputNeuronCount - 1 do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  PatternsInput[APatternIndex,
i];
end;

procedure TNeuralNetBP.LoadPatternsOutput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to OutputNeuronCount - 1 do
    DesiredOut[i] := PatternsOutput[APatternIndex, i];
end;

procedure TNeuralNetBP.NeuronsInLayerChange(Sender: TObject);
begin
  if AutoInit then
```

```
    Init;
end;

procedure TNeuralNetBP.NeuronCountError;
begin
  raise ENeuronCountError.Create(SNeuronCount)
end;

procedure TNeuralNetBP.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 TNeuralNetBP.ResetLayers;
begin
  Clear;
  FNeuronsInLayer.Clear;
end;

procedure TNeuralNetBP.SetDesiredOut(Index: integer; Value: double);
begin
  FDesiredOut[Index] := Value;
end;

procedure TNeuralNetBP.SetLayersBP(Index: integer; Value: TLayerBP);
begin
  FLayers[Index] := Value as TLayerBP;
end;

procedure TNeuralNetBP.SetAlpha(Value: double);
begin
  if (Value > 10) or (Value < 0.01) then
    FAlpha := DefaultAlpha
  else
    FAlpha := Value;
end;

procedure TNeuralNetBP.SetTeachRate(Value: double);
begin
  if (Value > 1) or (Value <= 0) then
    FTeachRate := DefaultTeachRate
  else
    FTeachRate := Value;
end;

procedure TNeuralNetBP.SetTestSetPatterns(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatterns[InputIndex, PatternIndex] := Value;
end;
```

```
procedure TNeuralNetBP.SetTestSetPatternsOut(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatternsOut[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternCount(const Value: integer);
begin
  FTestSetPatternCount := Value;
  SetLength(FTestSetPatterns, FTestSetPatternCount, InputNeuronCount);
  SetLength(FTestSetPatternsOut, FTestSetPatternCount, OutputNeuronCount);
end;

procedure TNeuralNetBP.SetMomentum(Value: double);
begin
  if (Value > 1) or (Value < 0) then
    FMomentum := DefaultMomentum
  else
    FMomentum := Value;
end;

procedure TNeuralNetBP.SetEpochCount(Value: integer);
begin
  if Value < 1 then
    FEpochCount := 1
  else
    FEpochCount := Value;
end;

procedure TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.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 TNeuralNetBP.SetPatternCount(const Value: integer);
begin
  FPatternCount := Value;
  inherited;
end;

procedure TNeuralNetBP.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Alpha := DefaultAlpha;
  ContinueTeach := False;
  Epoch := True;
```

```
  EpochCount := DefaultEpochCount;
  Momentum := DefaultMomentum;
  TeachRate := DefaultTeachRate;
  ResizeInputDim;
  ResizeOutputDim;
end;

{ TNeuralNetExtended }

constructor TNeuralNetExtended.Create(AOwner: TComponent);
begin
  inherited;
  SetDefaultProperties;
end;

destructor TNeuralNetExtended.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 TNeuralNetExtended.GetFields(Index: integer): TNeuroField;
begin
  Result := FFields[Index];
end;

function TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.GetRealInputIndex(Index: integer): integer;
begin
  Result := FRealInputIndex[Index];
end;

function TNeuralNetExtended.GetRealOutputIndex(Index: integer): integer;
begin
  Result := FRealOutputIndex[Index];
end;

procedure TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.LoadPhase1;
begin
  FSourceFileName := FNnwFile.ReadString('Phase1', 'LearnSampleFileName', '');
  FNeuroDataSource.Name := FSourceFileName;
end;

procedure TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.SavePhase1;
begin
  FNnwFile.WriteString('Phase1', 'LearnSampleFileName', FNeuroDataSource.Name);
end;

procedure TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.SetFields(Index: integer; Value: TNeuroField);
begin
  try
    FFields[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SFieldIndexRange, [Index])
  end;
end;

procedure TNeuralNetExtended.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 TNeuralNetExtended.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 TNeuralNetExtended.SetTeachIdentCount(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FTeachIdentCount := DefaultTeachIdentCount
  else
    FTeachIdentCount := Value;
end;

procedure TNeuralNetExtended.SetUseForTeach(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FUseForTeach := DefaultUseForTeach
  else
    FUseForTeach := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndex(Index: integer; const Value:
integer);
begin
  FRealOutputIndex[Index] := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndexCount(const Value: integer);
begin
  SetLength(FRealOutputIndex, Value)
end;
procedure TNeuralNetExtended.SetRealInputIndex(Index: integer; const Value:
integer);
begin
  FRealInputIndex[Index] := Value;
end;
procedure TNeuralNetExtended.SetRealInputIndexCount(const Value: integer);
begin
  SetLength(FRealInputIndex, Value)
end;
procedure Register;
begin
  RegisterComponents('Neural_network_', [TNeuralNetHopf, TNeuralNetBP,
TNeuralNetExtended]);
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.
```