МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерних наук

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему Використання нейронних мереж із прямим поширенням інформації для розпізнавання зображень

Виконав: студент 2 курсу, групи ІКК 2.1
спеціальності
122 Комп'ютерні науки

Курюкін Ілля Сергійович

Керівник Русу О.П.

Рецензент Григор'єва Т.І.

Одеса – 2023 р

ДОВІДКА

кафедри КН про виконану магістерську роботу

студента 2 курсу ФКПІ та КН групи ІКК 2.1

Курюкін Ілля Сергійович

(прізвище, ім'я та по-батькові)

на тему: Використання нейронних мереж із прямим поширенням інформації для розпізнавання зображень

Висновок нормоконтролера

пояснювальна записка до кваліфікаційної роботи виконана з урахуванням зауважень ДСТУ. оформлено згідно вимог внутрішнього положення МТУ

Нормоконтролер викл. каф ІК (науковий ступінь, вчене звання, посада) 15.12.2023 (підпис, дата) Кліщіна Н.В. (і. б. прізвище)

Висновок відповідального за перевірку на наявність академічного плагіату згідно з сертифікатом ID 1015695367 унікальність роботи підтверджена

Відповідальна особа викл. каф ІК (науковий ступінь, вчене звання, посада) 15.12.2023 (підпис, дата) Кліщіна Н.В. (і. б. прізвище)

Попередня захист магістерської роботи

(бакалаврської роботи чи магістерської роботи)

студ. Курюкіна І.С. (прізвище і.б.) проведена "15" 12 2023р.

Висновки кваліфікаційна робота виконана у повному обсязі. В роботі розроблено використання нейронних мереж із прямим поширенням інформації" для розпізнавання зображень. Кваліфікаційна робота відповідає вимогам до випускних кваліфікаційних робіт зі спеціальності 122 Комп'ютерні науки та рекомендована до захисту.

Члени комісії (підпис) к.т.н., доц. Соловська І.М. (науковий ступінь, вчене звання,посада, прізвище і.б.)

(підпис) к.т.н., доц. Рудь О.П. (науковий ступінь, вчене звання, посада, прізвище і.б.)

(підпис) к.т.н., доц. Розенвассер Д.М. (науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерних наук
Освітній ступінь другий (магістрерський)
Галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
к.т.н., доц.
_____ І.М.Соловська
"25" 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ

1. Тема роботи: Використання нейронних мереж із прямим поширенням інформації для розпізнавання зображень
керівник роботи Русу О. П.
Затверджені наказом закладу вищої освіти від 25.09.2023 р. р. № 1959
2. Строк подання студентом роботи 12.12.2023 р.

3. Вихідні дані до роботи:
1. Використовувати нейроні мережі із прямим поширенням інформації
2. Розмір первинного зображення не менше 20x20 пікселів
3. Розпізнавати не менше 5 класів зображень

4. Зміст розрахунково-пояснювальної записки
Розділ 1: Аналіз предметної області нейронних мереж
Розділ 2: Програмна реалізація модулю нейронної мережі
Розділ 3: Результати дослідження

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Види нейронних мереж
Слайд 2 – Нейронна мережа з прямим поширенням інформації
Слайд 3 – Алгоритм навчання нейронної мережі з прямим поширенням інформації
Слайд 4 – Вибір практичної реалізації
Слайд 5 – Зовнішній вигляд програмного забезпечення
Слайд 6 – Вибірка Fashion MNIST
Слайд 7 – Результати дослідження
Слайд 8 – Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.09.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання	25.09.2023	
2	Види нейронних мереж	26.09.2023 – 5.10.2023	
3	Нейроні мережі з прямим поширенням інформації	6.10.2023 – 15.10.2023	
4	Практична реалізація макету нейронної мережі з прямим поширенням інформації	16.10.2023 – 10.11.2023	
5	Отримання результатів роботи	11.11.2023 – 03.12.2023	
6	Оформлення пояснювальної записки магістерської роботи	04.12.2023 – 11.12.2023	
7			
8			

Здобувач _____ (підпис) І.С. Курюкін

Керівник роботи _____ (підпис) О.П. Русу

ВІДГУК КЕРІВНИКА

на кваліфікаційну роботу другого (магістерського) рівня
здобувача Курюкіна Іллі Сергійовича
зі спеціальності 122 Комп'ютерні науки
на тему: «Використання нейронних мереж із прямим поширенням інформації для розпізнавання зображень»

Використання засобів, що використовують штучний інтелект, вже давно стало об'єктивною реальністю сьогодення, і наразі дуже важливо мати засоби, що дозволяють практично досліджувати засоби штучного інтелекту з метою виявлення їх особливостей та обмежень. Тому кваліфікаційна робота Курюкіна І.С., в якій було створено працездатну багатошарову штучну нейронну мережу із прямим поширенням інформації є актуальною та має велике практичне значення.

Під час проведення дослідження здобувач Курюкін І.С.. самостійно виконав аналіз можливих варіантів побудови штучних нейронних мереж, пошуку навчальних та тестових зразків зображень та розробив програмне забезпечення, в якому активно використовувались технології об'єктно-орієнтованого програмування. Під час роботи здобувач Курюкін І.С. дотримувався графіку консультації та враховував усі рекомендації, що надавалися йому протягом роботи. Поставлене завдання виконано у повному обсязі. Пояснювальна записка та демонстраційні аркуші виконано із дотриманням усіх необхідних вимог.

Під час виконання кваліфікаційної роботи здобувач Курюкін І.С. розібрався з усіма поставленими питаннями та показав уміння користуватись інформаційними джерелами, ставити та розв'язувати дослідницькі задачі.

Кваліфікаційна робота відповідає вимогам до кваліфікаційних робіт другого (магістерського) рівня та заслуговує оцінки «відмінно».

Здобувач Курюкін І.С. заслуговує присвоєння кваліфікації магістр з комп'ютерних наук за заявленою спеціальністю 122 «Комп'ютерні науки».

Керівник
доцент кафедри комп'ютерних наук,
к.т.н., О.П. Русу

РЕЦЕНЗІЯ

на кваліфікаційну роботу другого (магістерського) рівня
здобувача Курюкіна Іллі Сергійовича
зі спеціальності 122 Комп'ютерні науки
на тему: «Використання нейронних мереж із прямим поширенням інформації для розпізнавання зображень»

Кваліфікаційна робота здобувача Курюкіна І.С. присвячена актуальним питанням розвитку систем комп'ютерного зору на основі штучного інтелекту. У роботі створена та протестована штучна нейронна мережа із прямим поширенням інформації, працездатність якої перевірена на тестовій вибірці, яка налічує кілька тисяч зображень. Практична цінність роботи полягає у тому, що створене програмне забезпечення є частиною віртуальної лабораторії, яку можна використовувати в процесі підготовки здобувачів освіти в галузі інформаційних технологій.

Протягом співбесіди здобувач Курюкін І.С. показав глибоке розуміння процесів, що відбуваються в нейронних мережах, що є свідченням високої теоретичної підготовки. Кваліфікаційна робота відповідає завданню, і використовує усі вихідні дані, які були в завданні. Текст роботи послідовний та зрозумілий, оформлення пояснювальної записки та демонстраційних аркушів якісне.

До недоліків роботи слід віднести:

– недостатню увагу приділено питанням вибору функцій активацій нейронів вихідного та прихованих шарів;

– у створеному програмному забезпеченні слід було б реалізувати можливість зміни розміру зображень.

Але названі недоліки не знижують цінності виконаної роботи.

Кваліфікаційна робота Курюкіна І.С. відповідає вимогам до випускних кваліфікаційних робіт здобувачів другого (магістерського) рівня та заслуговує оцінки «відмінно».

Здобувач Курюкіна І.С. заслуговує присвоєння кваліфікації магістр з комп'ютерних наук за заявленою спеціальністю 122 «Комп'ютерні науки».

Рецензент
Завідувач кафедри
інформаційних технологій
к.т.н., доцент Т.І. Григор'єва

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

ID перевірки:
1016010076

Дата перевірки:
15.12.2023 19:38:57 MSK

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.12.2023 19:02:01 MSK

ID користувача:
100001433

Назва документа: ПЗ_Маг_Курюкін І.С

Кількість сторінок: 171 Кількість слів: 19243 Кількість символів: 162993 Розмір файлу: 4.30 MB ID файлу: 1015695767

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

28.8%
Схожість

Найбільша схожість: 14.2% з джерелом з Бібліотеки (ID файлу: 1015695766)

11.7% Джерела з Інтернету	977	Сторінка 173
19.4% Джерела з Бібліотеки	22	Сторінка 180

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	33 сторінки

РЕФЕРАТ

Текстова частина магістерської роботи містить 49 с., 22 рис., 16 табл., 42 джерело , 3 додатки.

РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, ПРЯМЕ ПОШИРЕННЯ ІНФОРМАЦІЇ, МЕТОД ЗВОРОТНЬОГО ПОШИРЕННЯ ПОМИЛКИ.

Об'єкт дослідження – нейронні мережі з прямим поширенням інформації для розпізнавання зображення.

Предмет дослідження – програмне забезпечення для розпізнавання зображень за допомогою нейронних мереж з прямим поширенням інформації.

Мета дослідження – створення програмного модуля для розпізнавання зображень на основі штучної нейронної мережі із прямим поширенням інформації.

Методи дослідження – методи об'єктно-орієнтованого програмування, методи статистичної обробки.

Створено багатошарову нейронну мережу із прямим поширенням інформації, призначену для розпізнавання зображень. Нейронна мережа виконана у вигляді готового програмного модуля, призначеного для використання в універсальній віртуальній лабораторії. Проведено тестування працездатності розробленої нейронної мережі та оцінка ефективності її роботи.

ABSTRACT

The text part of the master's thesis contains 49 pages, 22 figures, 16 tables, 42 sources, 3 appendices.

IMAGE RECOGNITION, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, NEURAL NETWORK, FORWARD INFORMATION PROPAGATION, BACKWARD ERROR PROPAGATION METHOD.

Object of research - neural networks with direct information propagation for image recognition.

Subject of research - software for image recognition using neural networks with direct information dissemination.

The aim of the study is to create a software module for image recognition based on an artificial neural network with direct information dissemination.

Research methods - methods of object-oriented programming, methods of statistical processing.

A multilayer neural network with direct information propagation designed for image recognition was created. The neural network is made in the form of a ready-made program module intended for use in a universal virtual laboratory. The performance of the developed neural network was tested and its efficiency was evaluated.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ANN	(Artificial neural network) – штучна нейронна мережа.
DNN	(Deep neural network) – глибинна нейронна мережа.
DLL	(Dynamic Link Library) – динамічно-зв’язана бібліотека
ФА	функція активації.
ПШ	прихований (проміжний) шар.

ВСТУП

Зростання обчислювальної потужності та доступність великих обсягів даних сприяють активному розвитку глибокого навчання, що визначає нові стандарти у багатьох галузях науки та технології. Зокрема, нейронні мережі, засновані на інформації, отриманій від виявлення та розуміння того, як працює людський мозок, стали ключовими інструментами для вирішення розпізнавання образів та інших складних завдань машинного навчання.

Дана дипломна робота спрямована на вивчення, практичну реалізацію та аналіз навчання глибоких нейронних мереж у контексті розпізнавання образів. У роботі аналізується різновиди нейронних мереж, структуру глибоких моделей та методи навчання, зосереджуючись на методі зворотного розповсюдження помилки як основному механізмі навчання.

Розділ аналізу предметної області включає в себе огляд різних видів нейронних мереж, структуру глибоких архітектур та методи зворотного розповсюдження помилки. Також розглядаються аспекти розпізнавання образів та визначаються труднощі, пов'язані з використанням глибоких моделей.

У другому розділі пропонується програмна реалізація модулю для розпізнавання образів. Вибір програмного забезпечення та мови програмування обгрунтовується, а архітектура нейронної мережі та алгоритм навчання детально описані. Процедури та функції модулю розглядаються в контексті реалізації конкретного завдання.

Розділ результатів дослідження включає в себе опис попередньої обробки датасету, а також опис отриманих результатів.

В заключному розділі формуються висновки на основі отриманих результатів та надаються рекомендації для подальших досліджень у галузі розпізнавання образів та використання глибоких нейронних мереж.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ НЕЙРОНИХ МЕРЕЖ

1.1 Види нейронних мереж

Нейронні мережі є ключовим компонентом у галузі машинного навчання, що імітує структуру людського мозку для розв'язання різних завдань. Нижче наведено огляд основних аспектів нейронних мереж.

Нейрон (або штучний нейрон) – основний будівельний блок нейронних мереж, який моделює роботу нейрона в мозку (рис. 1.1). Він приймає вхідні дані, множить їх на ваги, додає зміщення (bias), застосовує функцію активації та передає результат наступному шару [1].

Також нейрон має аксон – вихідний зв'язок даного нейрона, з якого сигнал (збудження або гальмування) надходить на синапси наступних нейронів.

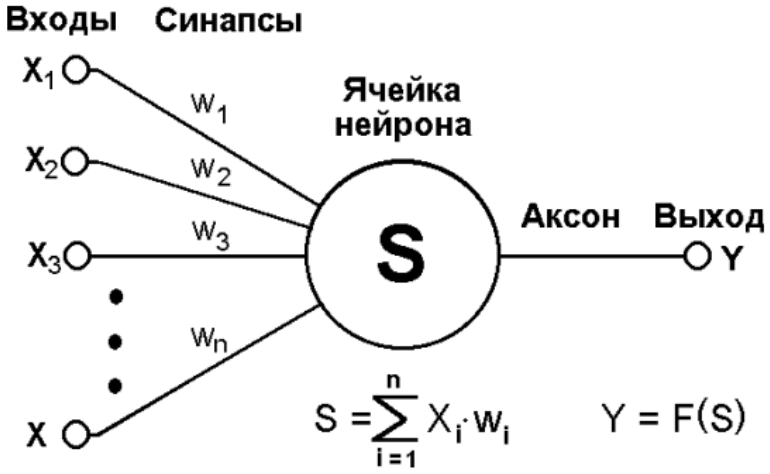


Рисунок 1.1 – Штучний нейрон

Кожен синапс характеризується величиною синаптичного зв'язку, її вагою W_i. Поточний стан нейрона визначається сума його входів за формулою (1.1)[2].

$$S = \sum_{i=1}^{n} x_i * w_i \,, \tag{1.1}$$

де: x_i– поточне значення нейрону, w_i– вага поточного нейрона.

Шар: Нейрони групуються в шари, виходячи з їхніх функцій. Основні типи шарів включають вхідний шар, приховані шари і вихідний шар. Ваги і зміщення: Ваги представляють силу зв'язку між нейронами, а зміщення додаються до зважених входів перед застосуванням ФА [3].

Глибокі нейронні мережі поділяються на кілька типів в залежності від їхньої структури та застосування. Ось кілька типових видів глибоких нейронних мереж [4].

Перцептрон – це найпростіший тип нейронної мережі, що складається лише з вхідного та вихідного шарів. Використовується для бінарної класифікації. Багатошаровий перцептрон має хоча б один ПШ між вхідним та вихідними шарами. Використовується для вирішення більш складних задач, таких як класифікація та регресія [5].

Згорткові нейронні мережі використовуються для обробки зображень і мають спеціальні шари згортки та пулінгу для визначення локальних особливостей та зменшення просторових розмірів даних [6].

Рекурентні нейронні мереж застосовуються для роботи з послідовними даними, такими як мовлення чи часові ряди. Мають зв'язки, які дозволяють передавати інформацію через час.

Довго-короткострокова пам'ять і мережі з забуванням воріт вони є варіантами рекурентних нейронних мереж, спроектованими для вирішення проблеми зниклих градієнтів в довгих послідовностях. Дозволяють моделі більше ефективно вивчати та використовувати контекст в послідовних даних.[1, 3, 4]

Автокодери використовуються для вирішення задач реконструкції та стиснення даних. Складаються з енкодера та декодера, які навчаються витягати та відновлювати вхідні дані.

Глибокі згорткові нейронні мережі аналогічні CNN, але мають більш глибоку структуру з багатьма згортковими та повністю з'єднаними шарами. Застосовуються для більш складних завдань обробки зображень [7].

Трансформери використовують для обробки послідовностей, таких як текст, без використання рекурентних або згорткових шарів. Засновані на механізмі уваги.

Це лише кілька базових типів глибоких нейронних мереж, існує багато інших архітектур та модифікацій, розроблених для вирішення конкретних задач. Вибір конкретної архітектури залежить від характеру даних та вимог. У даному випадку розглядається багатошаровий перцептрон Розенблатта (рис 1.1)[8].

Багатошаровий перцептрон - окремий випадок перцептрона Розенблатта [9], у якому один алгоритм зворотного поширення помилки навчає всі шари.

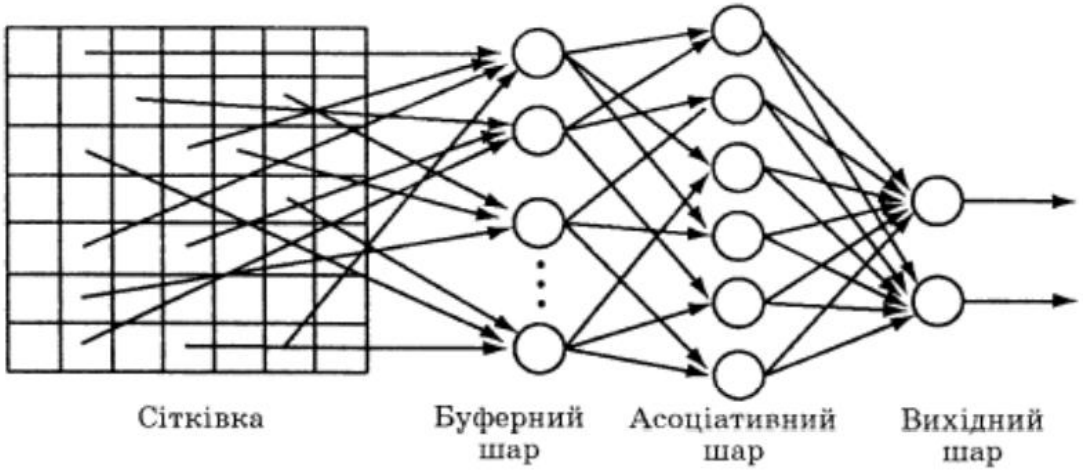


Рисунок 1.1 – Перцептрон Розенблатта

1.2 Короткий огляд структури глибоких нейронних мереж

Глибока нейронна мережа (DNN) є штучною нейронною мережею (ANN) з декількома прихованими шарами між вхідним і вихідним шарами. Подібно звичайній ANN, DNN може моделювати складні нелінійні відносини. DNN архітектури, наприклад, для виявлення об'єкта і синтаксичного аналізу, генерують композиційні моделі, де об'єкт виражається у вигляді шаруватої композиції примітивів зображення [11, 12].

DNN, як правило, виконані у вигляді мереж прямого поширення, але тут також дуже успішно застосовуються рекурентні нейронні мережі.

1.3 Метод зворотного розповсюдження помилки

DNN може бути навчений стандартним алгоритмом зворотного поширення. Згідно з різними джерелами, причини безперервного зворотного поширення були виведені в контексті теорії управління Генрі Джея. Келлі в 1960 році і Артур Е. Брайсон в 1961 році, який застосував принципи динамічного програмування [13]. У 1962 році Стюарт Дрейфус опублікував простіший висновок, заснований на ланцюговому правилі [14]. У 1970 році Сеппо Ліннаінмаа опублікував загальний метод автоматичного диференціювання дискретних з'єднаних мереж з використанням вкладеної диференціації (AD) [15]. Це відповідає поточній версії зворотного розподілу, яка ефективна навіть тоді, коли мережі слабкі. У 1973 році Стюарт Дрейфус використав алгоритм зворотного поширення, щоб налаштувати параметри контролера пропорційно градієнтам помилок [16]. У 1974 році Пол Уербос згадав про можливість застосування цього принципу до штучних нейронних мереж [17], а в 1982 році застосував метод Ліннаінмаа до нейронних мереж, які широко використовуються

сьогодні [18]. У 1986 році Девід Е. Румельхарт, Джеффрі Е. Хінтон і Рональд Дж. Вільямс показали в комп'ютерних експериментах, що цей метод може генерувати корисні внутрішні представлення вхідних даних в прихованих шарах нейронних мереж [19]. У 1993 році Эрик А. Ван був першим, хто виграв міжнародний конкурс для зворотного розпізнавання образів.

1.4 Дослідження розпізнавання образів

Звичайною оцінкою набору для класифікації зображень є набір даних бази даних MNIST. MNIST складається з зображень різноманітного одягу та взуття і включає в себе 60000 прикладів для навчання і 10000 тестових прикладів. Його невеликий розмір дозволяє створювати кілька конфігурацій для тестування. Нині кращий результат на MNIST є коефіцієнт помилки 0,15 відсотків, досягнутий користувачем під найменуванням ajbrock та іншими в 2017 році [20].

1.5 Труднощі з глибокими моделями

Як і в разі ANN, можуть виникнути труднощі з навчанням DNN, якщо вони навчаються примітивними наївними способами. Дві основні проблеми – перенавчання і великий час обчислень [21]. Домінуючий метод для навчання структур DNN – це корекція помилок навчання (наприклад, backpropagation з градієнтним спуском) завдяки простоті реалізації і тенденції сходитися на краще локальних оптимумів, ніж інші методи навчання. Проте, ці методи можуть бути обчислювально дорогими, особливо для DNN (тобто вимагати більшого часу навчання або пам'яті) [22]. Є багато параметрів навчання, які необхідно враховувати при навчанні DNN, такі як розмір (кількість шарів і кількість

одиниць на шар), швидкість навчання та початкових ваг. Охоплення всього простору параметрів для оптимальних параметрів може виявитися неможливим через витрати часу і обчислювальних ресурсів [23] . Було показано, що різні «Трюки», такі як використання міні-пакетування (обчислення градієнта на кількох навчальних прикладах відразу, а не на окремих прикладах), корисні для прискорення обчислень [24]. Велика пропускна здатність обробки графічних процесорів виправила значні прискорення в навчанні в силу векторноматричного характеру обчислень, які тут необхідні і добре виконуються графічними процесорами [25].

1.6 Вибір програмного забезпечення та мови програмування

Delphi — це об’єктно-орієнтована мова програмування, яка використовується для розробки програмного забезпечення для операційних систем Windows.

Якщо ви вирішили використовувати Delphi для програмування нейронних мереж, ось деякі переваги та недоліки мови.

Переваги Delphi для програмування нейронних мереж: легко освоїти – Delphi має простий і прямий синтаксис, який робить легко освоїти для початківців. Це може бути особливо корисно, якщо ви новачок у програмуванні чи нейронних мережах.

Розширені інтерфейси та компоненти: Delphi має повний набір компонентів та інструментів для розробки графічних інтерфейсів, які можуть бути корисними для відображення результатів.

Робота з поширеними форматами даних: Delphi може легко обробляти рядки та інші поширені формати даних, що зручно для завантаження та обробки даних для нейронних мереж [26].

Недоліки Delphi для програмування нейронних мереж:

Обмежена підтримка глибокого навчання: у порівнянні з іншими мовами, такими як Python, Delphi може мати обмежену підтримку бібліотек і інструментів глибокого навчання.

Відсутність великої спільноти та ресурсів: У порівнянні з такими популярними мовами, як Python або Java, Delphi має меншу спільноту та обмежений доступ до ресурсів нейронної мережі.

Застарілі бібліотеки та інструменти: бібліотеки та інструменти глибокого навчання часто оновлюються, і Delphi може відставати від останніх технологій у цій галузі.

Важко керувати великими проектами: Деякі програмісти можуть стверджувати, що Delphi не підтримує великі та складні проекти так само ефективно, як інші мови [26].

Вибираючи мову програмування для нейронної мережі, важливо враховувати не лише переваги, але й конкретні потреби та вимоги проекту.

1.7 Висновки до розділу

В даному розділі було розглянуто загальну інформацію про нейронні мережі, їх структуру та випадки, в яких їх доручно використовувати. Плюсами таких мереж можна вважати навчання без учителя, а також розпізнавання більш глибоких, деколи неочевидних, закономірностей в даних. В мережах з прямим поширенням за кожен рівень ознак відповідає свій шар і вони чудово себе показують в задачах розпізнавання образів на зображенні. Недоліками таких мереж є перенавчання і великий час обчислень.

Зрозуміло, що для вирішення задачі, поставленої в межах дослідження даної роботи, доцільно використовувати нейроні мережі з прямим поширенням інформації, адже вони чудово себе показують в задачах розпізнавання образів на зображеннях.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЮ НЕЙРОННОЇ МЕРЕЖІ

2.1 Архітектура нейронної мережі

Розглянемо основні компоненти архітектури нейронних мереж з прямим поширенням інформації (feedforward neural networks). Ця архітектура є найпростішою формою нейронних мереж і використовується для багатьох задач, включаючи класифікацію. На рис 2.1 зображена модель багатошарового перцептрона. Ця модель складається з чотирьох шарів: вхідного шару, першого прихованого шару, другого прихованого шару і вихідного шару.

У вхідному шарі (Input Layer) – перший шар нейронної мережі, який приймає вхідні дані перебуває сімсот вісімдесят чотири нейрони, оскільки в цьому шарі має зберігатися інформація про кожен піксель зображення. Кількість нейронів у вхідному шарі дорівнює кількості вхідних ознак чи атрибутів, що подаються моделі Оскільки в даному випадку вибрано зображення розміром двадцять вісім пікселів на двадцять вісім пікселів, то й виходить у вхідному шарі нейронів рівно стільки ж, як і пікселів у зображенні [27].

У першому прихованому шарі і в другому прихованому шарі перебувати по двадцять нейронів. Приховані шари (Hidden Layers) нейронні мережі можуть мати один або більше прихованих шарів. Кожен нейрон у прихованому шарі пов'язаний з кожним нейроном попереднього та наступного шару. Ці шари виконують операції лінійного перетворення та застосовують активаційні функції для внесення нелінійності в модель. У випадку задачі з множинними класами використовуватися софтмакс активаційна функція [28].

А ось у вихідному шарі (Output Layer) вже перебувати десять нейронів. Оскільки кількість нейронів вихідного шару має відповідати кількості категорій класів зображень, остільки й у вихідному шарі перебуватимуть саме десять нейронів [21].

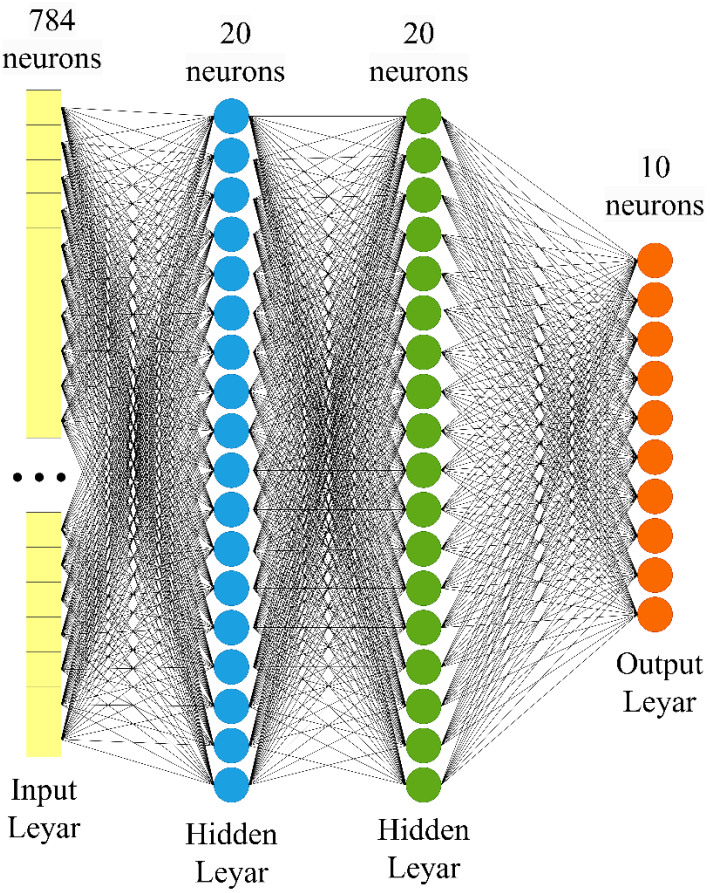


Рисунок 2.1 – Схема багатошарового перцетрона

Зв'язки та ваги (Connections and Weights) – кожен нейрон у шарі пов'язаний з кожним нейроном попереднього та наступного шару. Кожній з цих зв'язків присвоюється вага, яка визначає важливість внеску вхідної інформації. Завдяки вагам, нейрона мережа може навчатись та потім виконувати класифікацію зображень з заданими раніше параметрами [21].

Функції активації (Activation Functions) застосовуються до вагових сум вхідних значень та вирішують, чи повинен активуватися нейрон. В даному випадку використовується сигмоїда за формулою (2.1)[29].

$$\sigma(x) = \frac{1}{1 - e^{-x}} \tag{2.1}$$

2.2 Алгоритм навчання

Алгоритм навчання нейронної мережі зі зворотнім розповсюдженням помилки (Backpropagation) включає кілька кроків (рис. 2.2). Цей алгоритм використовується для оптимізації ваг нейронів у мережі, з метою мінімізації функції втрат [30]. Ось загальний опис кроків алгоритму:

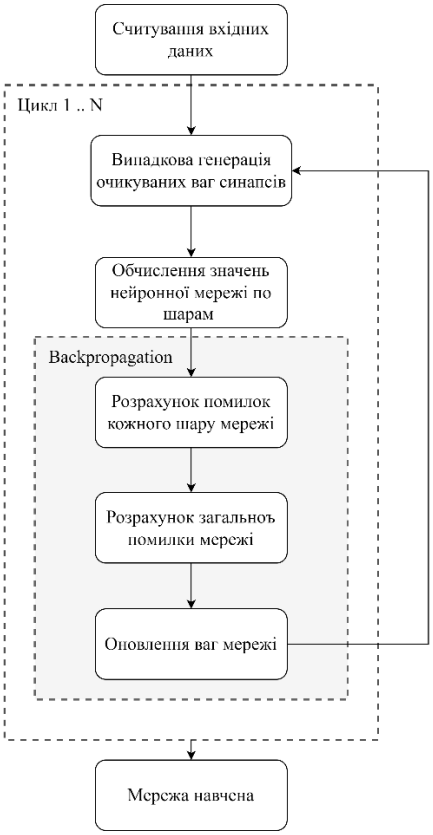


Рисунок 2.2 – Схема алгоритму навчання нейронної мережі зі зворотнім розповсюдженням помилки.

На першому етапі відбувається ініціалізація – ваги нейронної мережі генеруються випадковим чином, в даному випадку від значення мінус один до одного. Це включає установку початкових значень ваг на невеликі випадкові числа [31].

На другому етапі відбувається пряме поширення – вхідні дані подаються в мережу, і прямим поширенням інформації обчислюються виходи мережі. Кожен нейрон у мережі обчислює вагову суму своїх вхідних сигналів та застосовує до цього результату функцію активації, в даному випадку сигмоїду (2.1).

На третьому етапі відбувається розрахунок помилко кожного шару мережі. У обох схованих шарах та вихідному шарі мережі розраховується помилка кожного шару мережі окремо. Це потрібно для подальшого п’ятого кроку для корекції ваг кожного шару.

На четвертому етапі виконується зворотнє поширення помилки – розпочинаючи з вихідного шару, використовуючи помилки кожного шару мережі відносно ваг кожного нейрона, обчислюється загальна помилка мережі. Це виконується за допомогою правила ланцюга та розрахунку помилки від вихідного шару до вхідного.

На п’ятому етапі виконується оновлення ваг за допомогою методів оптимізації, таких як градієнтний спуск. Градієнти використовуються для корекції ваг нейронів так, щоб функція втрат зменшилася.

Останнім етапом виконується повторення – кроки, що були приведені вище, повторюються для кожного пакету даних або для кожного окремого навчального прикладу. Цей процес триває досягнення задовільного рівня точності або мінімізації помилки мережі.

Цей процес триває певну кількість епох, де одна епоха визначається прогоном усієї навчальної вибірки через мережу.

2.3 Опис універсальної віртуальної лабораторії Labs

Віртуальна лабораторія Labs складається із ядра та лабораторних макетів (рис. 2.3). Ядро програми містить користувацький інтерфейс, до складу якого входить інтерактивна схема із функцією масштабування, органи керування

процесом моделювання та параметрами досліджуваної схеми, віртуальні вимірювальні прилади та усі додаткові програмні модулі, що необхідні для проведення досліджень.

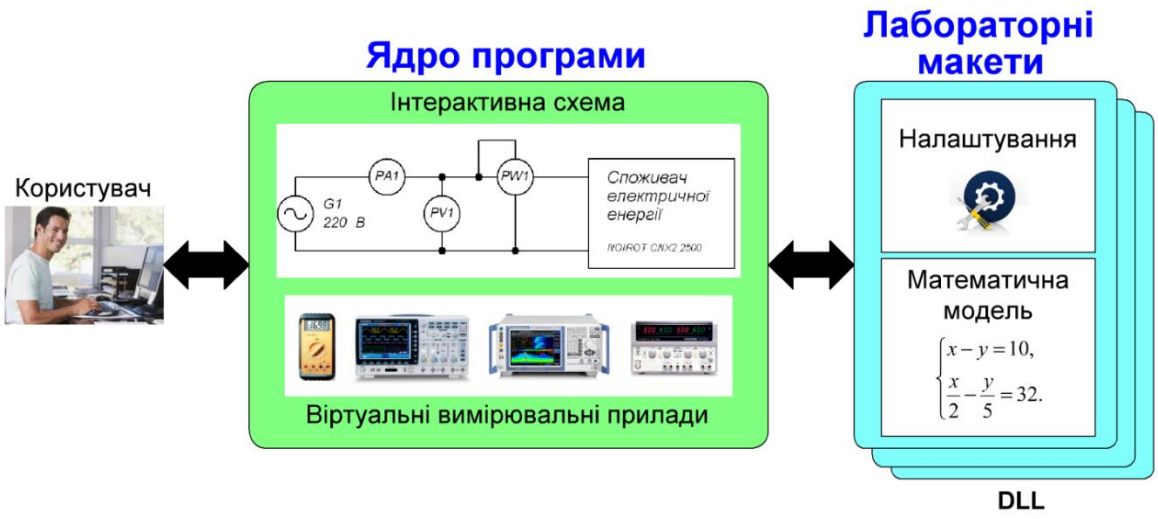


Рисунок 0.3 – Принцип побудови віртуальної лабораторії

Інформацію про параметри досліджуваної схеми (параметри компонен-тів, перелік та налаштування вимірювальних приладів, параметри органів оперативного регулювання) а також математична модель, за допомогою якої відбувається обчислення роботи схеми розташовані у файлах лабораторних макетів, що виконані за технологією динамічно-зв’язаних бібліотек (Dynamic Link Library – DLL).

Після підключення лабораторного макету (файлу DLL) до ядра він фактично стає його невід’ємною частиною, що дозволяє досягти максимальної уніфікації програмного забезпечення та максимальної швидкості проведення обчислень.

Завдяки використанню технології DLL та продуманої системи налаштувань віртуальна лабораторія є гнучкою у використанні, а створення нових лабораторних макетів (створення нових DLL) займає мінімум часу та потребує лише базових знань у області програмування. Крім того, оскільки лабораторний макет фактично є самостійним програмним забезпеченням, а технологію DLL

підтримують майже усі виробники систем автоматизованого проектування для створення програмного забезпечення, то нові лабораторні макети можуть розроблюватися самостійно на будь-якій мові програмування, та будь якій системі розробки програмного забезпечення, що підтримує дану технологію.

Усі лабораторні макети виконані у єдиному стилі (рис. 2.4). Інтерфейс лабораторних макетів є інтуїтивно зрозумілим і потребує мінімум часу на його вивчення. Зміст та призначення органів керування макетом та віртуальними вимірювальними приладами максимально наближені до реальних аналогів.

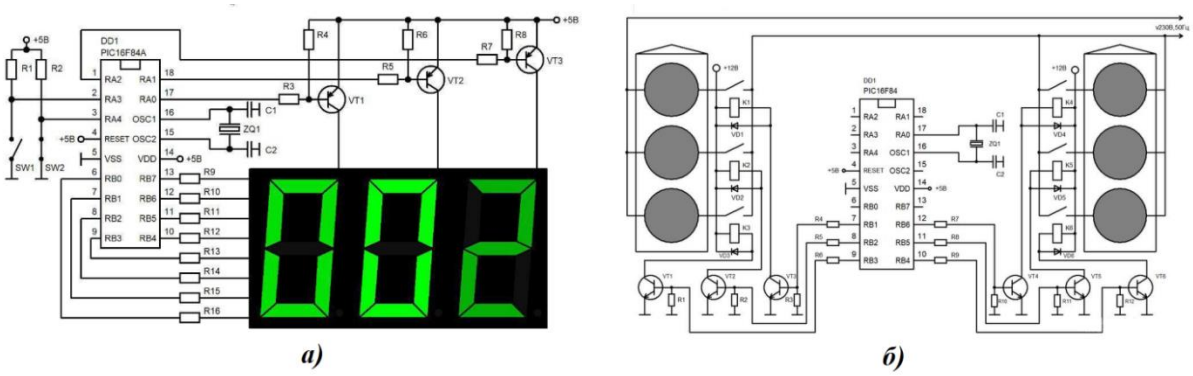


Рисунок 0.4 – Зовнішній вигляд лабораторних макетів, призначеного для програмування мікроконтролерів

Лабораторні макети не потребують значних обчислювальних ресурсів і автоматично підстроюються під параметри конкретного комп'ютера. Вони мають можливість індивідуальної настройки параметрів елементів (для запобігання повторюваності результатів вимірів у різних бригад студентів).

Завдяки використанню технології DLL може бути досягнута висока швидкість обчислень, що у деяких випадках може перевищувати швидкість протікання досліджуваних процесів у реальному часі. При цьому може забезпечуватися максимальна відповідність форми та характеру поведінки досліджуваних сигналів, наприклад, на екрані віртуального осцилографа (рис. 2.5) його реальному аналогу.

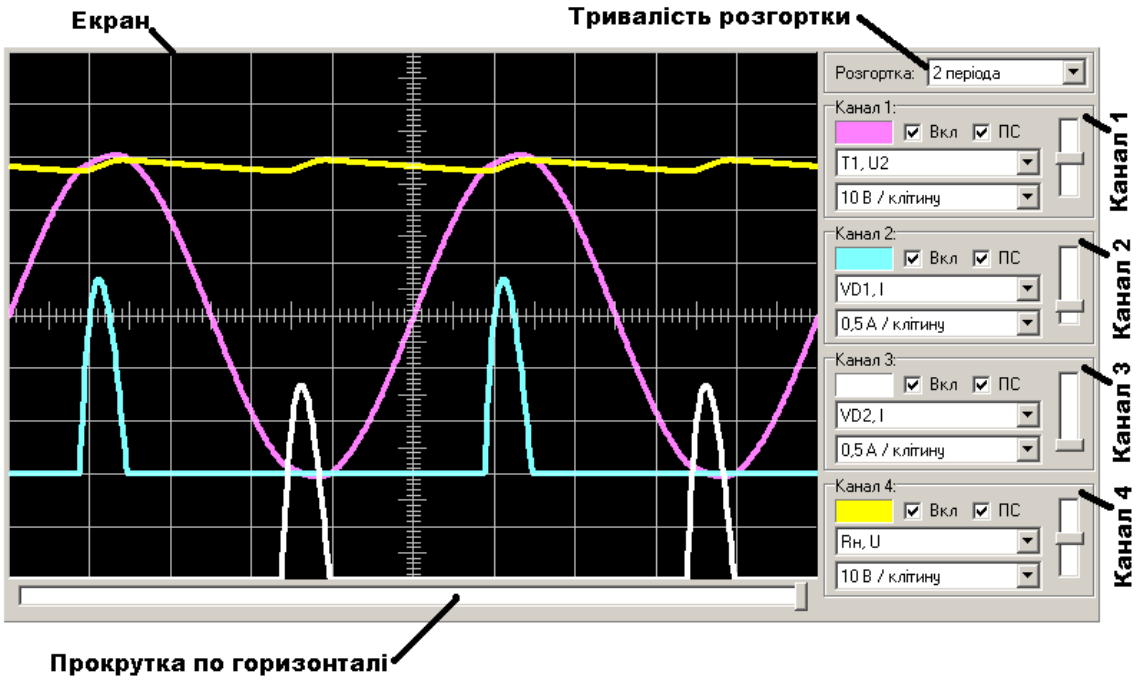


Рисунок 0.5 – Зовнішній вигляд віртуального осцилографа

Головними особливостями віртуальної лабораторії є:

– можливість дослідження широкого кола процесів у різних галузях техніки;

– простота створення нових лабораторних макетів;

– кожен лабораторний макет може містити кілька досліджуваних схем;

– єдиний стиль інтерфейсу для проведення досліджень;

– підтримка кількох мов інтерфейсу;

– широкий набір контрольно-вимірювальних приладів із можливістю гнучкого налаштування;

– висока швидкість та достовірність результатів моделювання;

– можливість зміни параметрів досліджуваної схеми без зупинки моделювання із максимально достовірним відображенням усіх перехідних процесів.

Віртуальний лабораторний практикум, порівняно з аналогічними практикумами має наступні переваги:

– лабораторний практикум містить віртуальні лабораторні макети, які набагато зручніші при встановленні, не потребують спеціального трифазного живлення та подальшого обслуговування;

– підтримка кількох мов інтерфейсу дозволяє використовувати лабораторний практикум для підготовки іноземних студентів;

– лабораторні макети є самостійними програмами і не потребують додаткового програмного забезпечення;

– адаптація програмного забезпечення для використовування мінімуму обчислювальних ресурсів не потребує потужних комп'ютерів;

– інтуїтивно зрозумілий інтерфейс та виконання лабораторних макетів у єдиному стилі мінімізує час освоєння правил роботи з макета-ми;

– велика швидкість моделювання без погіршення його якості зменшує час обчислення перехідних процесів при проведені досліджень – це дозволяє зменшити у цілому час на проведення вимірів;

– один лабораторний макет може містити декілька досліджуваних схем і придатний для проведення декількох лабораторних робіт;

– можливість індивідуального встановлення параметрів елементів для кожного макета запобігає повторюваності результатів вимірів у різних бригад.

Головною сферою застосування віртуальної лабораторії є використання у закладах усіх рівнів освіти для проведення лабораторних робіт та вивчення широкого кола процесів, дослідження яких за допомогою фізичних установок пов’язано із значними матеріальними витратами або має нижчу якість порівняно із дослідженням за допомогою математичних моделей. У першу чергу це теоретичні дисципліни, пов’язані із електричними процесами, спостерігання яких у реальному житті пов’язано із можливістю ураження електричним струмом, споживанням значної кількості енергії, або, через високу швидкість протікання процесів, потребує для дослідження дорогого вимірювального обладнання.

Основними дисциплінами, у яких можна застосувати дану лабораторію є:

– фізика (електричні явища);

– електротехніка;

– теорія електричних кіл;

– теорія електричного зв’язку;

– теорія автоматизованого керування;

– електроживлення та електропостачання;

– енергозберігаючі технології;

– аналогова та цифрова схемотехніка;

– мікроконтролерна техніка;

– пристрої автоматизації;

– робототехніка;

– пристрої Інтернету речей;

– інтелектуальні пристрої.

Віртуальну лабораторію можна використовувати у навчальних закладах практично усіх рівнів освіти:

– школах;

– коледжах;

– закладах вищої освіти;

– центрах підвищення кваліфікації.

Окрім використання для навчання, віртуальну лабораторію можна також використовувати у науковій роботі та під час проектування широкого спектру технічних пристроїв:

– перевірка достовірності математичних моделей;

– розробка та тестування програмного забезпечення;

– проведення наукових досліджень.

2.3 Опис інтерфейсу мережі та функцій

Для взаємодії з нейронною мережею потрібен графічний інтерфейс. Даний графічний інтерфейс можна побачити на рис. 2.6 .

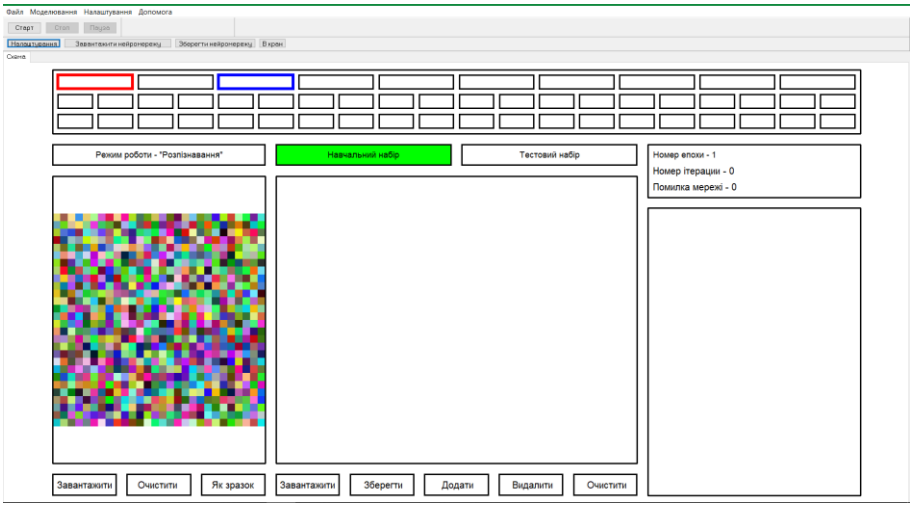


Рисунок 2.6 – Схема макету графічного інтерфейсу нейронної мережі

На макеті, на панелі в верхній лівій частині, присутні елементи налаштування нейронної мережі такі як кнопка налаштування, кнопка завантаження нейронної мережі, кнопка збереження нейронної мережі та кнопка що відповідає за призначення зображення на екран відображення з будь якого нейрона мережі. Також, на панелі присутні кнопки для управління мережею: кнопка старт, кнопка стоп та кнопка пауза (рис. 2.7). Також є можливість налаштувати весь макет, змінивши мову інтерфейсу, змінити макет.

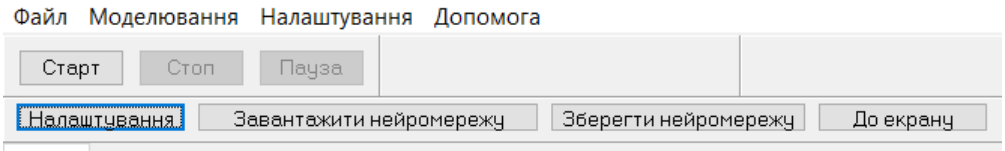


Рисунок 2.7 – Панель налаштування нейронної мережі

У розділі налаштування (рис. 2.8) є можливість налаштувати мережу, вказавши параметри кількості епох від однієї епохи до десяти тисяч епох, мінімального значення помилки мережі. Також є можливість створювати лог файли, для того щоб докладніше подивитись як мережа навчається, які значення набувають нейрони.

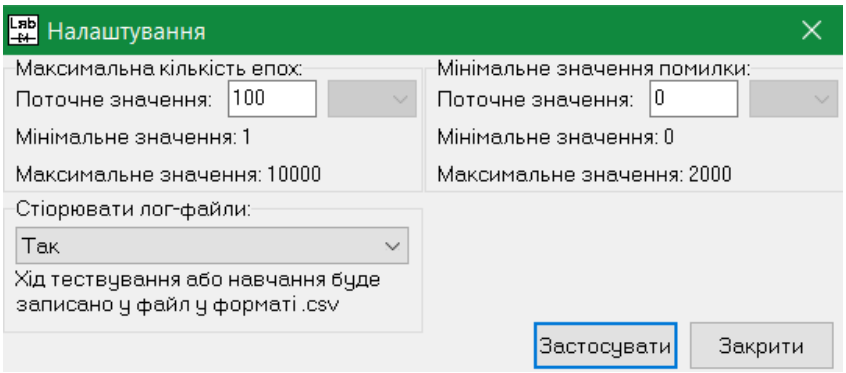


Рисунок 2.8 – Вікно налаштування нейронної мережі

У центрі макету розміщено інтерфейс нейронної мережі. Інтерфейс поділяється на чотири великі блоки.

У першому блоці, що знаходиться у верхній частині макету, знаходяться вихідні нейрони нейронної мережі, у яких призначаються зображення за певною категорією (рис. 2.9).

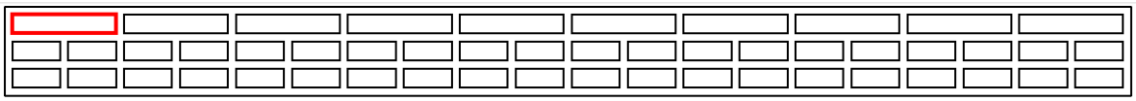


Рисунок 2.9 – Шари нейронної мережі.

У лівій частині розташовано блок два для відображення зображення, яке завантажується з діалогового вікна та кнопка з налаштуванням режиму роботи нейронної мережі: навчання, тестування та розпізнавання (рис. 2.10).

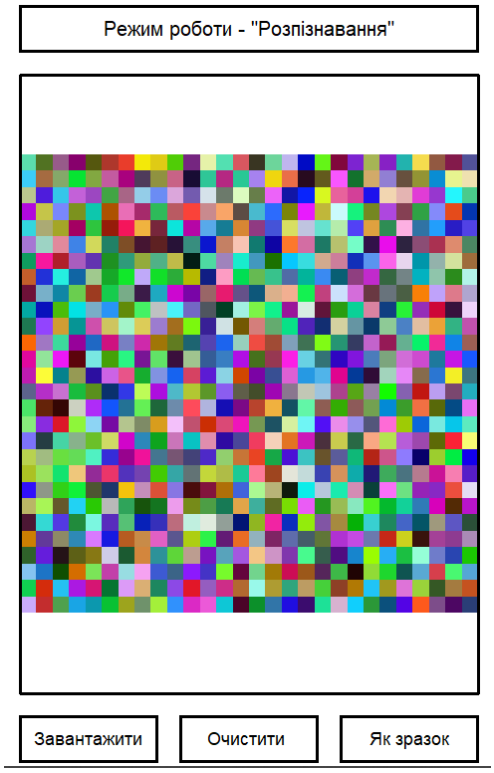


Рисунок 2.10 – Блок інтерфейсу з режимом роботи мережі та завантаженням зображення.

У вікні відображення зображення за замовченням стоїть «заглушка». Через це вікно зображення попадає до нейронної мережі й там обробляється. Також, у блоці є три кнопки: завантажити – відповідає за завантаження зображення у нейронну мережу, очистити – очищає вікно відображення за залишає «заглушку» та кнопка як зразок – відповідає за копіювання завантаженого зображення у список зразків для певного нейрона.

По середині макету розташовано блок три зі списком зображень для навчання та списком зображень для тестування (рис. 2.11).

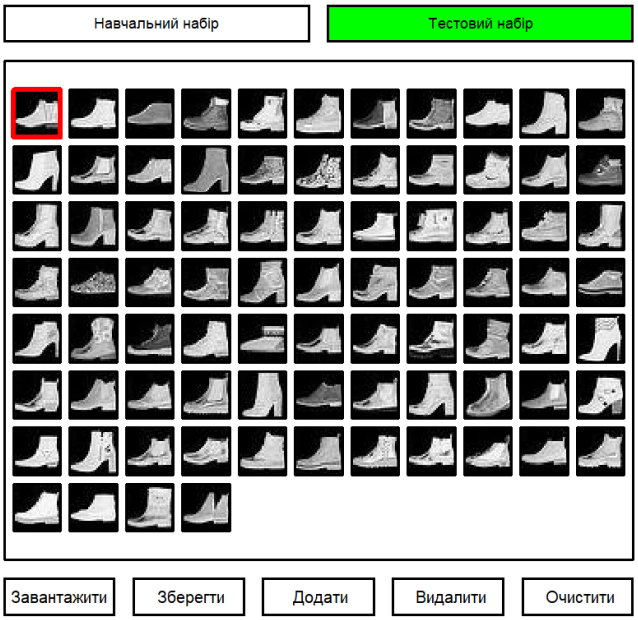


Рисунок 2.11 – Блок інтерфейсу зі списком зображень для нейронної мережі

У верхній частині блоку розташовано дві кнопки для перемикання списків зображень Посередині блоку розташовано список з переліком зображень. У нижній частині розташовані кнопки з маніпуляціями над зразками для нейронної мережі. Кнопка завантажити відповідає за завантаження списку зразків зображення до макету нейронної мережі. Кнопка зберегти відповідає за збереження зразків у списку для подальшої роботи з ними. Кнопка додати відповідає за додавання (копіювання) зразка зображення з екрану відображення у блоці два у список зразків. Кнопка видалити відповідає за видалення зразка зображення зі списку. Кнопка очистити відповідає за видалення даних зображення зі зразка, але місце разка залишається.

У четвертому блоці (рис. 2.12), що знаходиться у правій частині інтерфейсу, відповідає за винесення інформації щодо роботи нейронної мережі. У верхній частині блоку знаходяться дані про кількість епох, що нейронна мережа пройшла за час роботи, номер ітерації – відповідає за кількість зразків, що проходить нейронна мережа за одну епоху, та помилку мережі – на скільки мережа помиляється у розпізнаванні зображення.

Нижче розташовано вікно з відображенням зразка зі списку зразків зображень для детальнішого перегляду зразка зі списку.

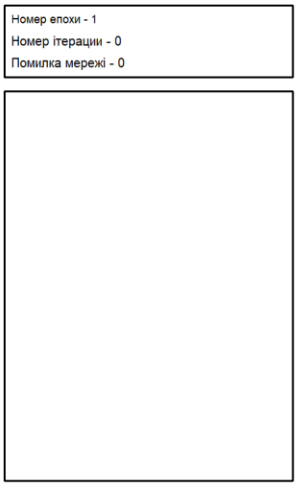


Рисунок 2.12 – Інформативний блок нейронної мережі

На рис 2.13 зображено інтерфейс у режимі навчання. Нейрони при режимі навчання починають блимати зірними відтінками червоного кольору. Насиченість кольору означає вірогідність певного нейрона, що зображення відповідає його критерію ознаки.

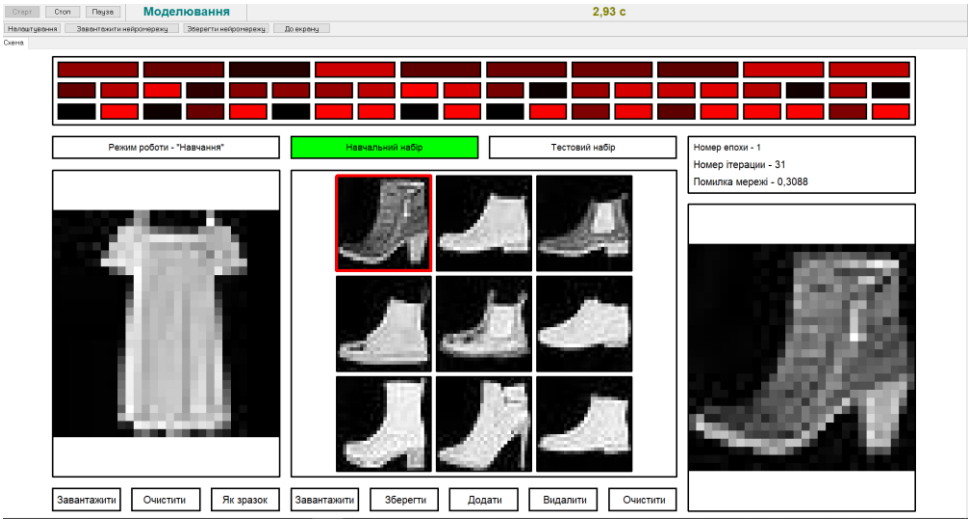


Рисунок 2.13 – Приклад роботи інтерфейсу у режимі навчання.

2.4 Опис класів нейронної мережі та її методів.

Після розробки інтерфейсу реалізуємо логічну або функціональну частину нейронної мережі. Для початку, визначимо яким чином інтерфейс може бути пов’язана з мережею, яким чином зображення буде опинятись у мережі для подальшої обробки даних та навчанню. На рис 2.14 зображена схема архітектури нейронної мережі, як зображення попадає до мережі та як організується мережа.

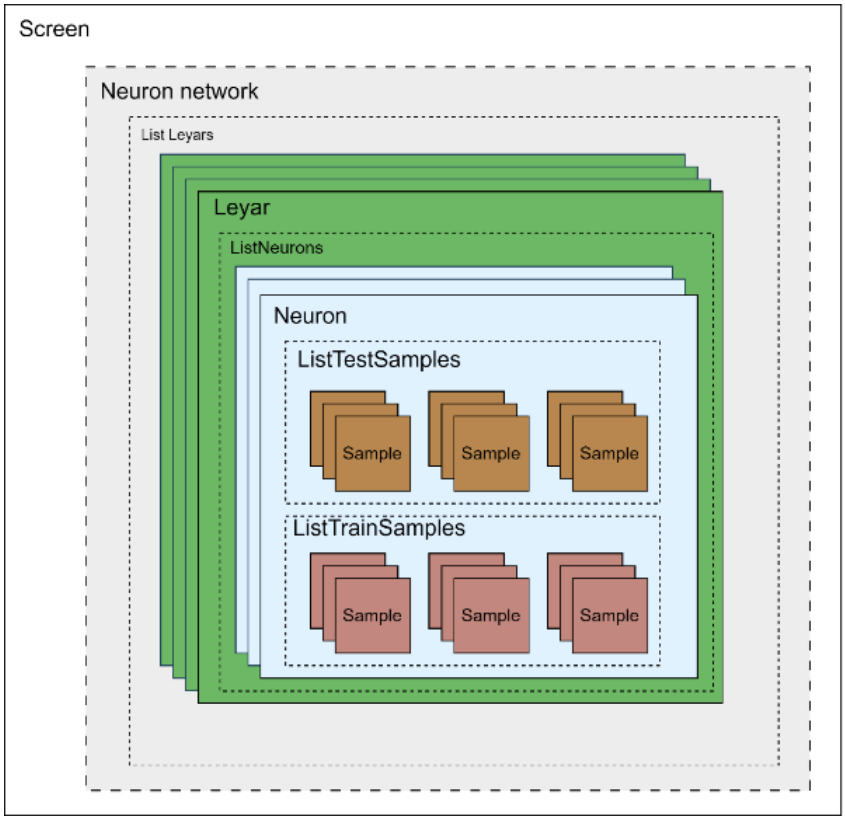


Рисунок 2.14 – Схема архітектури класів нейронної мережі у програмі

Для початку, створимо глобальні змінні для роботи з нейронами мережі, з їх вагами та з параметрами ФА. Створюємо змінні у глобальному оточенні для того, щоб можна було отримати доступ з будь якого класу чи методу. Перелік змінних розміщено у табл. 2.1.

Таблиця 2.1 – Глобальні змінні нейронної мережі.

Назва	Тип	Значення за замовченням	Область видимості	Призначення
TNeuronInputs	array of PDouble	nill	global	вхідні дані для нейрона
TNeuronWeights	array of Double	nill	global	ваги нейрона
TActivationFunction Params	array of Double	nill	global	параметри, що використовуються для активаційної функції нейрона

Створюємо клас TNeuronSample, він призначений для представлення даних зразка в нейроні. Для зберігання даних, створюємо зміні для збереження даних зразка в нейроні (табл. 2.2). Так само створюємо змінну для відстеження фокусу на нейроні.

Таблиця 2.2 – Змінні класу TNeuronSample

Назва	Тип	Значення за замовчуванням	Область видимості	Призначення
FScreen	TBitmap	pfDevice	private	вхідні дані зображення, пов'язане з певним зразком
FFocused	Boolean	False	private	значення яке вказує на те, що перебуває цей зразок у фокусі, чи ні

Для того, щоб зберегти та зчитати завантажене зображення з інтерфейсу, створюємо властивість для зчитування даних зображення та властивість з доступом до фокусу кожного зразка (табл. 2.3).

Таблиця 2.3 – Властивості класу TNeuronSample

Назва	Тип	Читання	Запису	Область видимості	Призначення
Screen	TBitmap	FScreen	–	public	надає доступ до зображення через властивість тільки для читання
Focused	Boolean	FFocused	FFocused	public	надає доступ до стану фокусу через властивість для читання і запису

Реалізуємо метода класу TNeuronSample. Для роботи з класом створюємо конструктор та деструктор для створення об’єкту класу TNeuronSample та знищення його після закінчення роботи. Також, створюємо метод для призначення зображення як зразок для нейронної мережі. Створюємо метод для очищення зразка, метод для встановлення розміру зразка та прописуємо методі для збереження та завантаження зразків у мережу. Перелік методів у табл. 2.4.

Таблиця 2.4 – Методи класу TNeuronSample

Назва	Тип	Параметри	Опис
Create	constructor	–	ініціалізує об'єкт класа
Destroy	destructor	–	викликається при знищенні об'єкта
Assing	procedure	aScreen: TBitmap	присвоює зображення екземпляру класу
Clear	procedure	–	очищає дані зразка

SetDimentions	procedure	aWidth, aHeight: Integer	встановлює розміри (ширину та висоту) зображення
SaveToStream	procedure	F: TFileStream	зберігає дані зразка в потік
LoadFromStream	function	F: TFileStream	завантажує дані зразка з потоку

Наступним кроком створюємо клас TNeuronSampleList, який містить список зразків даних нейрона. У даному класі створюємо змінну для отримання ім’я файлу для завантаження вже готових списків зразків для нейронної мережі та змінну для змінну для кількості сторінок (табл. 2.5). Усім цим змінним призначаємо приватну область видимості.

Таблиця 2.5 – Зміні класу TNeuronSampleList

Назва	Тип	Значення за замовчуванням	Область видимості	Призначення
FFileName	String	‘Default.ann’	private	рядок, що містить ім'я файлу, пов'язаного з певним списком зразків.
FPageNumber	Integer	0	private	число, пов'язане з номером сторінки

Наступним кроком створюємо властивості для доступу до запису чи читання даних. Властивість до індексу кожного зразка, властивість до кількості зразків у нейроні, властивість до читання і запису у файл, властивість до фокусу певного нейрона та номеру сторінки списку (табл. 2.6).

Таблиця 2.6 – Властивості класу TNeuronSampleList

Назва	Тип	Читання	Запису	Призначення
Items	TNeuron Sample	Get	Put	доступ до зразків за індексом
Count	Integer	GetCount	–	кількість зразків у списку
FileName	String	FFileName	FFileName	доступ до імені файлу
Focused Sample	TNeuron Sample	GetFocused Sample	SetFocused Sample	доступ до поточного зразка у фокусі
PageNumber	Integer	FPageNumber	FPageNumber	доступ до номера сторінки

Потрібні також методи для реалізації роботи класу (рис. 2.7). Створюємо методи для отримання першого та останнього зразка у списку. Обов’язково створюємо метод для додавання та знищення певного зразка зі списку зразків у нейроні. Потрібно ще створити методи для збереження списку да завантаження все готового списку зразків. Створюємо методи максимально розміру зразка, метод для видалення усіх зразків із списку, для знищення даних після завершення роботи, метод для збереження списку и потоці для роботи зі списком та метод для фокусу на першому зразку. Обов’язково створюємо метод для створення об’єкту класу.

Таблиця 2.7 – Методи класу TNeuronSampleList

Назва	Тип	Параметри	Опис
First	function	–	повертає перший зразок у списку
Last	function	–	повертає останній зразок у списку
AddNewSample	function	–	додає новий зразок у список і повертає його
LoadFromStream	function	F: TFileStream	завантажує дані списку з

			потоку
LoadFromFile	function	ShowDialog: Boolean	завантажує дані списку з файлу
MaxWidth	function	–	повертає максимальну ширину серед усіх зразків у списку
MaxHeight	function	–	повертає максимальну висоту серед усіх зразків у списку
DeleteAll	procedure	–	видаляє всі зразки зі списку
DeleteSample	procedure	aSample: TNeuronSample	видаляє конкретний зразок зі списку
SaveToStream	procedure	F: TFileStream	зберігає дані списку в потік
SaveToFile	procedure	ShowDialog: Boolean	зберігає дані списку у файл,
FocuseFirst	procedure	–	встановлює фокус на перший зразок у списку
Create	constructor	–	ініціалізує об'єкт

Наступним чином створюємо клас TNeuron, який представляє екземпляр нейрону у нейронній мережі. У цьому класі реалізується обчислення нейронної мережі зразка,

Для початку створимо зміні для роботи з нейроном: змінні для отримання списку тестових та навчальних зразків, що містяться у нейроні, зміну для доступу до кольору зразка, змінну для значення вхідних даних нейрона, змінну ваги кожного нейрона, змінну значення помилки певного нейрона. Також необхідно створити змінну для збереження значення нейрону після розрахунку ФА для кожного нейрона. Всі ці змінні мають приватну область видимості.

Таблиця 2.8 – Приватні змінні класу TNeuron

Назва	Тип	Значення за замовчуванням	Призначення
FTrainingSamples	TNeuronSampleList	nill	список навчальних зразків для нейрона
FTestSamples	TNeuronSampleList	nill	список тестових зразків для нейрона
FColor	TColor	TColors.Red	колір, пов'язаний із нейроном
FInputs	TNeuronInputs	nill	масив покажчиків на вхідні дані нейрона
FWeights	TNeuronWeights	nill	масив ваг нейрона
FActivation Function	TActivationFunction	nill	ФА для нейрона
FActivation FunctionParams	TActivation FunctionParams	nill	параметри ФА
FFocused	Boolean	False	вказує, чи перебуває нейрон у фокусі
FError	Double	nill	значення помилки нейрона

У приватному доступі у класі присутнє ще захищена функція GetColor для отримання кольору.

Наступним кроком створюємо властивості для класу нейрона: властивість до значення помилки нейрона, властивість для отримання списку навчальної та тестової вибірки, властивість для доступу до кольору зраз, властивість до фокусу до нейрона, властивість до кількості входів нейрона, властивості до значення ваги нейрону та кількості ваг та властивість до доступу до ФА, параметрів функції та їх кількості. Кожна властивість представлена у табл. 2.9.

Таблиця 2.9 – Властивості класу TNeuron

Назва	Тип	Читання	Запису	Призначення
Error	Double	FError	FError	доступ до значення помилки нейрона
TrainingSamples	TNeuron SampleList	FTrainingSamples	–	доступ до списку навчальних зразків
TestSamples	TNeuron SampleList	FTestSamples	–	доступ до списку тестових зразків
Color	TColor	FColor	FColor	доступ до кольору нейрона
Current Color	TColor	GetColor	–	доступ до поточного кольору
Focused	Boolean	FFocused	FFocused	доступ до фокусу нейрона
InputCount	Integer	GetInputCount	SetInputCount	доступ до кількості вхідних даних
Inputs	PDouble	GetInput	SetInput	доступ до масиву покажчиків на вхідні дані
Weights	Double	GetWeight	SetWeight	доступ до масиву ваг
WeightCount	Integer	GetWeightCount		доступ до кількості ваг
«Activation Function ParamCount»	Integer	GetActivation–Function–ParamCount	SetActivation–Function–ParamCount	доступ до кількості параметрів ФА
«Activation Function»	TActivation Function	FActivation–Function	FActivation–Function	доступ до ФА
«Activation FunctionParams»	Double	GetActivation–FunctionParams	SetActivation–FunctionParams	доступ до параметрів ФА

Наступним кроком створюємо приватні методи класу: методи для отримання та призначення кількості вхідних даних нейрону, методи для отримання та призначення поточного значення нейрону, методи для отримання

та призначення значень ваг нейрона, методи для отримання та призначення параметрів ФА, методи для отримання та призначення кількості параметрів ФА та метод для отримання кількості ваг поточного нейрона. Методи, описані вище, представлені у Таблиця 2.10.

Таблиця 2.10 – Приватні методи класу TNeuron

Назва	Тип	Параметри	Вихідне значення	Опис
GetInputCount	function		Integer	Отримання кількості входів нейрона
SetInputCount	procedure	const Value: Integer	Integer	Призначення кількості вході нейрона
GetInput	function	Index: Integer	PDouble	Отримання поточного значення нейрону
SetInput	procedure	Index: Integer; const Value: PDouble	–	Призначення потомного значення нейрону
GetWeight	function	Index: Integer	Double	Отримання ваг
SetWeight	procedure	Index: Integer; const Value: Double	–	Призначення ваг
GetActivation FunctionParams	function	Index: Integer	Double	Отримання параметрів ФА нейрона
SetActivation FunctionParams	procedure	Index: Integer; const Value: Double	–	Призначення параметрів ФА нейрона
GetActivation FunctionParam Count	function		Integer	Отримання кількості параметрів ФА нейрона
SetActivation FunctionParam Count	procedure	const Value: Integer	–	Призначення кількості параметрів ФА нейрона
GetWeightCount	function		Integer	Отримання кількості ваг

Після приватних методів, створимо публічні методи для нейрону. За замовчуванням створимо методи ініціалізації да знищення класу нейрона. Наступним кроком пишемо методи для збереження за завантаження даних до нейрону з потоку. Та на останок, створюємо метод для розрахунку реакції нейрона на зразки. Методи, описані раніше, представлені у табл. 2.11.

Таблиця 2.11 – Публічні методи класу TNeuron

Назва	Тип	Параметри	Вихіжне значення	Опис
Create	constructor	–	–	ініціалізує об'єкт класу
Destroy	destructor	–	–	знищення об'єкта класу
Calculate	procedure	aToLog: Boolean	–	обчислює результат роботи нейрона
SaveToStream	procedure	F: TFileStream	–	зберігає дані нейрона в потік
LoadFromStream	function	F: TFileStream	Boolean	завантажує дані нейрона з потоку

Клас по окремому нейрону створено, тепер створюємо клас TNeuronList, який являє собою список нейронів. Цей клас потрібен для збереження нейронів у шарах мережі.

Для початку створюємо методи у приватному оточенні класу: методи для отримання та призначення індексу нейрона у списку, методи для отримання та призначення кількості нейронів у списку та методи для отримання та призначенню значення фокусу на нейроні у списку. Описані методи приведені у Таблиця 2.12.

Таблиця 2.12 – Приватні методи класу TNeuronList

Назва	Тип	Параметри	Вихіжне значення	Опис
Get	function	Index: Integer	TNeuron	Отримання індексу нейрона у списку
Put	procedure	Index: Integer; const Value: TNeuron		Призначення індексу нейрона у списку
GetCount	function		Integer	Отримання кількості нейронів у списку
SetCount	procedure	const Value: Integer		Призначення кількості нейронів у списку
GetFocused	function		TNeuron	Отримання значення фокусу на нейроні у списку
SetFocused	procedure	const Value: TNeuron		Призначення значення фокусу на нейроні

Наступним кроком прописуємо властивості, але вже у публічному оточенні класу. Створюємо властивість для доступу до нейронів за їх індексом, потім потрібна властивість для кількості нейронів у списку, та на останок прописуємо властивість для значення фокусу на певному нейроні. Всі властивості представлені у Таблиця 2.13.

Таблиця 2.13 – Властивості класу TNeuronList

Назва	Тип	Читання	Запису	Призначення
Items	TNeuron	Get	Put	доступ до нейронів за індексом
Count	Integer	GetCount	SetCount	доступ до кількості нейронів у списку
Focused	TNeuron	GetFocused	SetFocused	доступ до нейрона у фокус

Не виходячи з публічного оточення, створюємо наступні методи: методи для отримання першого та останнього нейрона у списку, метод для отримання значення фокусу на певному нейроні, методи для збереження та завантаження списку нейронів у потік.

Таблиця 2.14 – Публічні методи класу TNeuronList

Назва	Тип	Параметри	Опис
First	function	–	повертає перший нейрон у списку
Last	function	–	повертає останній нейрон у списку
FocuseFirst	procedure	–	встановлює фокус на перший нейрон у списку
SaveToStream	procedure	F: TFileStream	зберігає дані списку нейронів у потік
LoadFromStream	function	F: TFileStream	завантажує дані списку нейронів із потоку

Наступним є клас TNeuronLayer, котрий являє собою шар нейронів у нейронній мережі.

Для роботи з цим класом потрібно створити змінні цього класу. Створюємо такі змінні класу: змінна для отриманні списку нейронів з класу TNeuronList та змінна для отриманні значення фокусу шару на певний нейрон. Ці змінні представлені у Таблиця 2.15.

Таблиця 2.15 – Зміні класу TNeuronLayer

Назва	Тип	Значення за замовчуванням	Область видимості	Призначення
FNeurons	TNeuronList		private	список нейронів у шарі
FCanFocus	Boolean	False	private	булеве значення, що вказує, чи може шар фокусуватися на якомусь зі своїх нейронів

Також потрібно прописати властивості для доступу до даних. Створюємо такі властивості: властивість для доступу до списку нейронів з класу TNeuronList, властивість для доступу значення фокусу нейрона (в даний момент) та властивість шару для доступу до його фокусу. Ці властивості представлені у Таблиця 2.16.

Таблиця 2.16 – Властивості класу TNeuronLayer

Назва	Тип	Читання	Запису	Призначення
Neurons	TNeuronList	FNeurons	–	доступ до списку нейронів у шарі
FocusedNeuron	TNeuron	GetFocusedNeuron	–	доступ до нейрона, який на цей момент перебуває у фокусі
CanFocus	Boolean	FCanFocus	FCanFocus	доступ до властивості, що вказує, чи може шар фокусуватися

Також, для роботи з класом шарів нейронної мережі, потрібні методи класу. Створюємо наступні методи: обов’язкові методи для ініціалізації об’єкта класу та його знищення при завершенні роботи, метод для збереження даних гару у потік та метод для завантаження даних у потік. Методи мають публічне оточення. Всі методи представлені у Таблиця 2.17.

Таблиця 2.17 – Публічні методи класу TNeuronLayer

Назва	Тип	Параметри	Вихіжне значення	Опис
Create	constructor	–	–	ініціалізує об'єкт класу
Destroy	destructor	–	–	знищення об'єкта класу
SaveToStream	procedure	F: TFileStream	–	зберігає дані шару нейронів у потік
LoadFromStream	function	F: TFileStream	Boolean	завантажує дані шару нейронів із потоку

Таким чином, але вже у приватній області видимості, створюємо захищених метод GetFocusedNeuron – метод для отримання нейрона, який наразі перебуває у фокусі.

Наступним кроком створюємо клас TNeuronLayerList, який являє собою список шарів нейронів у нейронній мережі. У цьому класі потрібно створити методи у приватному оточенні класу. Створимо такі методи класу: метод для отримання списку шарів у мережі, метод для додавання списку шарів, методи для отримання та призначення кількості шарів у мережі. Методі, описані вище, представлені у Таблиця 2.18.

Таблиця 2.18 – Приватні методи класу TNeuronLayerList

Назва	Тип	Параметри	Вихіжне значення	Опис
Get	function	Index: Integer	TNeuronLayer	Отримання списку шарів
Put	procedure	Index: Integer; const Value: TNeuronLayer	–	Додавання списку шарів
GetCount	function		Integer	Отримання кількості шарів у мережі
SetCount	procedure	const Value: Integer	–	Призначення кількості шарів у мережі.

Створимо властивості: властивість для доступу до шарів за індексом та властивість кількості шарів у списку. Властивості приведені у Таблиця 2.19.

Таблиця 2.19 – Властивості класу TNeuronLayerList

Назва	Тип	Читання	Запису	Призначення
Items	TNeuronLayer	Get	Put	доступ до шарів за індексом
Count	Integer	GetCount	SetCount	доступ до кількості шарів у списку

У публічному просторі створимо методи для роботи з класом: методи для отримання першого та останнього шару мережі та методи для завантаження та зберігання даних шару у потік.

Таблиця 2.20 – Публічні методи класу TNeuronLayerList

Назва	Тип	Параметри	Вихіжне значення	Опис
Last	function	–	TNeuronLayer	повертає останній шар у списку
First	function	–	TNeuronLayer	повертає перший шар у списку
SaveToStream	procedure	F: TFileStream	–	зберігає дані списку шарів нейронів у потік
LoadFromStream	function	F: TFileStream	Boolean	завантажує дані списку шарів нейронів із потоку

Після написання додаткових класів для нейронної мережі, переходимо до створення основного класу нейронної мережі. У цьому класі реалізуються всі інші класи і виконуються основні процеси реалізації логічної частини нейронної мережі.

Клас TNeuronNetwork, який являє собою нейронну мережу, містить в собі такі змінні: зміна для отримання списку шарів нейронної мережі, змінна для отримання зображення з екрану відображення з графічного інтерфейсу мережі, змінна для отримання назви файлу для завантаження зображення у мережу, змінні для отримання назві файлу збереженої мережі, для її завантаження знову, змінна для отримання даних виходів з екрану та змінна для значення базису нейрона. Описані вище змінні представлені у Таблиця 2.21.

Таблиця 2.21 – Зміні класу TNeuronNetwork

Назва	Тип	Значення за замовчуванням	Область видимості	Призначення
FLayers	TNeuron LayerList	nill	private	список шарів нейронів у мережі
FScreen	TBitmap	nill	private	зображення, пов'язане з мережею
FScreenImage FileName	string	nill	private	рядок, що містить ім'я файлу зображення
FFileName	string	‘Default.ann’	private	рядок, що містить ім'я файлу мережі
ScreenOutputs	array of Double	1	public	масив значень виходів екрана
BiasNeuroun	Double	nill	public	значення зміщення нейрона

Також створимо властивості для роботи мережі: властивість для доступу до списку шарів нейронної мережі, властивість для доступу до вихідного шару мережі, властивість для доступу вхідного шару, властивість для отримання імені файлу мережі, якщо вона є, властивість до доступу до значення екрану на який завантажується зображення. Дані властивості прописуємо у публічному оточені. Описані властивості представлені у Таблиці 2.22.

Таблиця 2.22 – Властивості класу TNeuronNetwork

Назва	Тип	Читання	Запису	Призначення
Layers	TNeuronLayerList	FLayers	–	доступ до списку шарів нейронів
OutputLayer	TNeuronLayer	GetOutputLayer	–	доступ до

				вихідного шару мережі
InputLayer	TNeuronLayer	GetInputLayer	–	доступ до вхідного шару мережі
FileName	String	FFileName	FFileName	доступ до імені файлу мережі
Screen	TBitmap	FScreen	–	доступ до зображення мережі

Створюємо методи для роботи з класом нейронної мережі: методи для ініціалізації об’єкту класу мережі та знищення об’єкту класу після роботи з ним, метод для очищення екрану відображення зразків, метод для встановлення розміру зображення для мережі, метод для завантаження зображення з файлу, метод для встановлення виходів з екрану, методів для обчислення реакції мережі, методі для збереження даних мережі у потік або її завантаження та методи для збереження або завантаження файлу нейронної мережі. Всі методі, що були пописані, створюються у публічному оточені. Переглянути методи можна у Таблиця 2.23.

Таблиця 2.23 – Публічні методи класу TNeuronNetwork

Назва	Тип	Параметри	Опис
Create	constructor	–	ініціалізує об'єкт класу
Destroy	destructor	–	знищення об'єкта класу
ClearScreen	procedure	–	очищає зображення мережі
CopyToScreen	procedure	aSource: TBitmap	копіює зображення з екрану в зображення мережі
SetScreenDimentions	procedure	aWidth,	встановлює розміри

		aHeight:Integer	зображення мережі
LoadScreenFromFile	procedure	–	завантажує зображення мережі з файлу
SetScreenOutputs	procedure	–	встановлює значення виходів екрана
Calculate	procedure	–	обчислює результат роботи мережі
SaveToStream	procedure	F: TFileStream	зберігає дані мережі в потік
LoadFromStream	function	F: TFileStream	завантажує дані мережі з потоку
SaveToFile	procedure	ShowDialog: Boolean	зберігає дані мережі у файл
LoadFromFile	function	ShowDialog: Boolean	завантажує дані мережі з файлу

Для захисту даних, прописуємо у приватному ще два метода: метод для отримання вихідного шару та метод для отримання вхідного шару мережі.

2.5 Висновки до розділу

Даний розділ надає детальний огляд алгоритму, архітектури, процедур та функцій, що лежать в основі програмної реалізації нейронної мережі для розпізнавання образів. Ретельний розгляд кожного елементу системи дозволяє зрозуміти не лише структуру, але й логіку взаємодії компонентів для досягнення поставлених цілей.

3 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Попередня обробка датасету

Попередня обробка датасету є критичним етапом в розробці системи розпізнавання одягу на основі нейронних мереж з прямим поширенням інформації. Ефективна обробка даних допомагає підготувати датасет для оптимального навчання моделі, що в свою чергу позитивно впливає на якість та швидкість розпізнавання.

Першим кроком є збір відповідного датасету для задачі розпізнавання одягу. Це може бути набір фотографій одягу, які включають різні типи одягу та різні перспективи зйомки. Крім того, важливо визначити класи (категорії) одягу, які модель повинна розпізнавати, такі як футболки, штани, сукні тощо.

Для оцінки ефективності моделі важливо розділити датасет на дві частини: навчальний та тестовий. Навчальний набір використовується для самого процесу навчання, тоді як тестовий дозволяє визначити загальну ефективність моделі та виявити ознаки перенавчання.

Буде використаний датасет Fashion MNIST [38] фрагмент котрого представлений на рисунку 3.1.

Рисунок 3.1 – Фрагмент датасету Fashion MNIS

3.2 Отримані результати

Нейронна мережа була навчена на обраних зразках першого набору у кількості тридцяти тисяч для навчання та п’яти тисяч для тестування, та другого набору у кількості шістдесяти тисяч зразків для навчання та десяти тисяч зразків для тестування. Навчання було проведено до 100 епох, 200 епох, 300 епох, 400 епох та 500 епох.

Отримані результати свідчать про деякі обмеження та проблеми у процесі тренування моделі для класифікації одягу. Проблемою став час навчання нейронної мережі, чим більший набір зразків зображень, тим більше часу мережа навчалась. Але є зворотна сторона проблеми, якщо забагато зменшити набір є можливість до недостатнього навчання мережі та зниження ефективності розпізнавання зображення.

Не навчена нейронна мережа показує значення ефективності розпізнавання зображень на рівні 4,83 відсотка відсотків, що показує, що мережа не знає нічого.

На першому наборі для навчання нейронна мережа показує такі результати як 42 відсотки. Результати представлені в Таблиця 3.1.

Таблиця 3.2 – Розпізнавання нейронної мережі на другому наборі зразків

Кількість епох	Навчання	Тестування
50	41,32	39,32
100	42,78	40,78
200	43,49	41,49
300	43,57	41,57
400	43,63	41,63
500	44,02	42,02

На другому наборі для навчання результати нейронної мережі значно покращуються і становлять 87 відсотків. Кількість часу, витраченого на навчання нейронної мережі, становила 15 годин і 25 хвилин. Результати

представлено в Таблиці 3.2 та графік правильного розпізнавання мережі на рис.3.3 та значення помилки мережі на рис. 3.4.

Таблиця 3.2 – Розпізнавання нейронної мережі на другому наборі зразків

Кількість епох	Навчання	Тестування
50	78,88	78,95
100	84,52	83,57
200	86,99	85,83
300	88,29	86,73
400	88,86	86,92
500	89,87	87,6

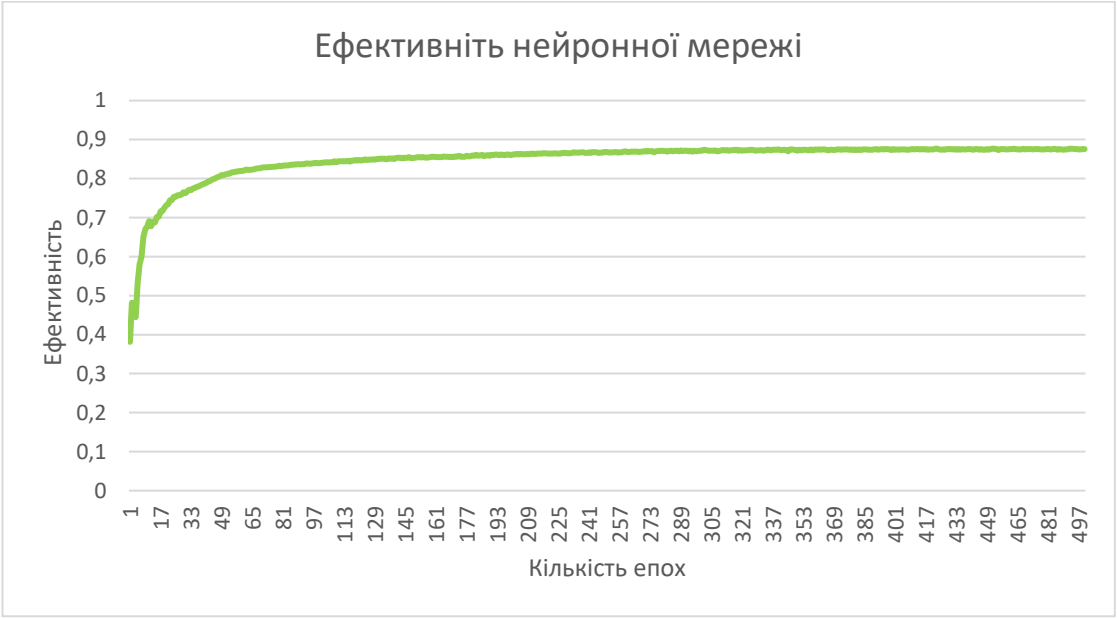


Рисунок 3.2 – Графік ефективності навчання нейронної мережі

Рисунок 3.3 – Графік ефективності навчання нейронної мережі

3.3 Висновки до розділу

Згідно з проведеним дослідженням, ефективність моделі розпізнавання одягу, що базується на нейронних мережах з прямим поширенням інформації , залежить від кількох ключових факторів.

Першим із них є обсяг та якість даних, які використовуються для тренування моделі. Якщо датасет містить недостатню кількість зображень або вони не є достатньо різноманітними, то модель може неправильно класифікувати одяг або навіть не зможе його розпізнати. Другим важливим фактором є час навчання. При великому наборі зображень час навчання мережі може збільшитися в десятки чи у сотні рази.

Оскільки ефективність моделі залежить від цих двох факторів, необхідно провести подальші дослідження з метою оптимізації моделі. Зокрема, можна використати алгоритми оптимізації для швидкого оброблення інформації.

Нарешті, можна переглянути архітектуру моделі, щоб визначити, чи не можна зробити якісь зміни, які збільшать її ефективність. Наприклад, можна додати додаткові шари або змінити конфігурацію існуючих шарів.

Загалом, ефективність моделі розпізнавання одягу на основі мереж з прямим поширенням інформації залежить від кількох ключових факторів, і для досягнення кращих результатів необхідно проводити подальші дослідження та оптимізацію моделі [41].

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

Головними результатами роботи є наступні пункти:

1. Проведено аналіз видів нейронних мереж для розпізнавання зображень з метою щоб обрати вид нейронної мережі для реалізації програмного забезпечення. За результатами проведеного аналізу та розгляду характеристик різних типів нейронних мереж, було прийнято рішення вибрати нейронну мережу з прямим поширенням інформації.

2. В розділу було проведено аналіз детальний аналіз в результаті чого було обрано багатошаровий перцептрон з двома схованими шарами. Кожний схований шар містить по 20 нейронів шар.

3. За даними аналізу дана модель є ефективною, але є недолік у часі роботи мережі. Ефективність моделі на першому наборі склала 42 відсотки, а вже на другому наборі склала 87 відсотки.

Результати даної роботи доцільно рекомендувати для удосконалення і практичного застосування у наступних напрямках.

1. При створенні програмного забезпечення функція котрого полягає в розпізнаванні об’єктів завчасно продумуючи сверу використання нейронної мережі.

2. Під час підготовки фахівців в галузі інформаційних технологій, інженерії програмного забезпечення, комп’ютерних науках, комп’ютерній інженерії і кібербезпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Каллан Р. Основные концепции нейронных сетей The Essence of Neural Networks First Edition. 1-ше. «Вильямс», 2011. 288 с. ISBN 5-8459-0210-X.

2. Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – К. : НТУУ «КПІ» Вид-во «Політехніка», 2015. – 396 с. – Бібліогр. : с. 385–387. – 55 пр. ISBN 978-966-622-691-7

3. Субботін, С. О. Нейронні мережі: навчальний посібник / С. О. Субботін, А. О. Олійник; під заг. ред. проф. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2014. – 132 с

4. Jeff Heaton, Tracy Heaton: «Artificial Intelligence for Humans, Volume 3: Neural Networks and Deep Learning December» – 2015 – Heaton Research, Inc on ISBN: 978-1505714340

5. Rummelhart D. E., Hinton G. E., Williams R. J. Learning Representations of Back-Propagation Errors // Nature. — 1986. — Vol. 323. — P. 533—536.

6. G. Huang, Z. Liu, and K. Q. Weinberger, Densely connected convolutional networks, arXiv preprint arXiv:1608.06993, Aug. 2016. URL: https://arxiv.org/abs/1608.06993v1

7. T. Xiao, J. Zhang et al., Error-driven incremental learning in deep convolutional neural network for large-scale image classification, in International Conference on Multimedia, no. 22. ACM, 2014, pp. 177– 186.

8. М.А. Новотарський, Б.Б. Нестеренко. Штучні нейронні мережі: обчислення // Праці Інституту математики НАН України. – Т50. – Київ: Ін-т математики НАН України, 2004. – 408 с.

9. Neural M. Networks and Deep Learning – M.Nielsen – Determination Press. – 2015. – P. 224.

10. Umberto Michelucci: «Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks» – toelt.ai – Dübendorf, Switzerland –

Copyright 2018 by Umberto Michelucci – ISBN-13 (pbk): 978-1-4842-3789-2 URL: https://doi.org/10.1007/978-1-4842-3790-8

11. D. C. Ciresan, U. Meier, J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. IEEE Conf. on Computer Vision and Pattern Recognition CVPR 2012.

12. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L.D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, Winter 1989. Vol 1(4): P. 541-551.

13. Charu C. «Aggarwal Neural Networks and Deep Learning» IBM T. J. Watson Research Center International Business Machines Yorktown Heights, NY, USA, 2018, ISBN 978-3-319-94462-3, ISBN 978-3-319-94463-0 (eBook), URL: https://doi.org/10.1007/978-3-319-94463-0

14. Новотарський М. А., Нестеренко Б. Б. Штучні нейронні мережі: обчислення // Праці Інституту математики НАН України. – Т.51.– Київ: Ін-т математики НАН України, 2004. –408 с.

15. Li, Y.; Fu, Y.; Li, H.; Zhang, S. W. (1 June 2019). "The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate". 2019 International Conference on Computational Intelligence and Natural Computing. Vol. 1. pp. 73–76. doi:10.1109/CINC.2019.111. ISBN 978-0-7695-3645-3. S2CID 10557754.

16. Jospin, Laurent Valentin; Laga, Hamid; Boussaid, Farid; Buntine, Wray; Bennamoun, Mohammed (2022). "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users". IEEE Computational Intelligence Magazine. Vol. 17, no. 2. pp. 29–48.

17. Fashion-MNIST [Електронний ресурс] – Режим доступу до ресурсу URL: https://github.com/zalandoresearch/fashion-mnist

18. Aggarwal Charu C. Neural Networks and Deep Learning — Springer, 2023. — 541 p. — ISBN 978-3-031-29642-0.

19. Umberto Michelucci: «Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks» – toelt.ai – Dübendorf, Switzerland –

Copyright 2018 by Umberto Michelucci – ISBN-13 (pbk): 978-1-4842-3789-2 URL: https://doi.org/10.1007/978-1-4842-3790-8

20. A. G. Howard, Some improvements on deep convolutional neural network based image classification, arXiv preprint arXiv:1312.5402, Dec. 2013. URL: https://arxiv.org/abs/1312.5402

21. N. Srivastava, G. E. Hinton et al., Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014. URL: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

22. Williams, Ronald J.; Zipser, D. (1 February 2013). "Gradient-based learning algorithms for recurrent networks and their computational complexity". In Chauvin, Yves; Rumelhart, David E. (eds.). Backpropagation: Theory, Architectures, and Applications. Psychology Press. ISBN 978-1-134-77581-1.

23. Delphi Documentation [Електронний ресурс] – Режим доступу до ресурсу URL: https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Documentation

24. Глибовец М.М., Олецький О.В. Штучний інтелект. – К.: Академія, 2002.

25. Трушевський В. , Шинкаренко Г. , Щербина Н. Метод скінченних елементів і штучні нейронні мережі: теоретичні аспекти та застосування: монографія. – Львів: ЛНУ ім. І. Франка, 2014. – 396 с.

26 . Rumelhart D. E. Learning Internal Representations by Error Propagation / D. E. Rumelhart, G. E. Hinton, R. J. Williams // In Parallel Distributed Processing. – Vol. 1. – Cambridge, MA, MIT Press. 2016. – P. 318–362.

27. Palchevsky E.V., Khristodulo O.I. Developing a self-learning method for a spiking neural network to protect against DDoS attacks. Software & Systems. 2019, vol. 32, no. 3, pp. 419–432. DOI: 10.15827/0236-235X.127.419-432.

28. J. Ortigosa-Hernández, I. Inza, and J. A. Lozano, Towards competitive classifiers for unbalanced classification problems: A study on the performance scores, arXiv preprint arXiv:1608.08984, Aug. 2016. URL: https://arxiv.org/abs/1608.08984

29. Теоретична оцінка ефективності застосування нейронних мереж в галузі захисту програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/7653871/page:3/

30. J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization” Journal of Machine Learning Research, vol. 13, no. Feb, pp. 281–305, 2012. URL: http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf

31. Hiai, Fumio; Lin, Minghua “On an eigenvalue inequality involving the Hadamard product”, 2017.

32. S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167, Feb. 2015. URL: https://arxiv.org/abs/1502.03167

33. S. J. Nowlan and G. E. Hinton, Simplifying neural networks by soft weight-sharing, Neural computation, vol. 4, no. 4, pp. 473–493, 2015. URL: https://www.cs.toronto.edu/~hinton/absps/sunspots.pdf

34. Chumachenko E. I. Building a system of simulation modeling for spatially-distributed processes / E. I. Chumachenko, A. Y. Luzhetskyi // Electronics and Control Systems, N 1(39) – Kyiv: NAU, 2014. – pp. 108–113.

35. Madan M. Gupta Statistic and Dynamic Neural Networks: From Fundamentals to Advanced Theory / M. Gupta Madan, Jin Liang, Homma Noriyasu. – USA: IEEE Press, 2013. – 722 p.

36. Anil K. Jain, Jianchang Mao, Mohiuddin K.M. Artificial Neural Networks – Anil K. Jain, Jianchang Mao, K.M. Mohiuddin. – Tutorial, Computer, Vol.29, No.3, March. – 2016.

37. Khaustov P.A. Algorithms for handwritten character recognition based on constructing structural models – P.A. Khaustov – Computer Optics. – 2017. – № 41 (1). – P. 67–78.

38. Добровська Л. М. Штучний інтелект. Основи теорії нейронних мереж: метод. вказівки до практ. занять для студ. спец. «Інформаційні управляючі системи та технології» міжуніверситетськ. медико-інженер. ф-ту / Уклад. : Л. М. Добровська, І. А. Добровська. – Київ: НТУУ «КПІ», 2009. – 180 с.

39. Sejnowski, Terrence J. (2018). The Deep Learning Revolution. MIT Press. p. 47. ISBN 978-0-262-03803-4.

40. Mohri, Mehryar; Rostamizadeh, Afshin (2013). "Perceptron Mistake Bounds". arXiv:1305.0208 [cs.LG]. https://en.wikipedia.org/wiki/Perceptron

41. Juergen Schmidhuber; Neural and Evolutionary Computing; Machine Learning (cs.LG) – Neural Networks, Vol 61, pp 85-117, Jan 2015

42. Alaloul, Wesam Salah; Qureshi, Abdul Hannan (2019). "Data Processing Using Artificial Neural Networks". Dynamic Data Assimilation - Beating the Uncertainties. doi:10.5772/intechopen.91935. ISBN 978-1-83968-083-0. S2CID 219735060

ДОДАТОК А
ПЕРЕЛІК КОПІЙ ДЕМОНСТРАЦІЙНОГО МАТЕРІАЛУ

Магістерська робота

На тему: «Використання нейронних мереж з прямим поширенням інформації для розпізнавання зображення»

Виконав: здобувач ступеню магістра групи ІКК 2.1
Курюкін Ілля Сергійович
Керівник: к.н.т., доц. Русу О.П.

ВСТУП

В сучасному світі непередбачуваного росту технологій та зв'язку, кожною секундою генерується надзвичайно велика кількість даних.

З урахуванням цього потоку даних стає критично важливим розвивати та вдосконалювати методи обробки інформації.

Однією з перспективних галузей розвитку є сфера комп'ютерного зору та розпізнавання об'єктів на зображеннях. Завдяки великій кількості доступної візуальної інформації, виникає необхідність у високоефективних методах автоматизованого розпізнавання та обробки зображень.

Основні характеристики роботи

- Об’єкт дослідження - нейронні мережі для розпізнавання зображення
- Предмет дослідження - програмне забезпечення для розпізнавання зображення за допомогою нейронних мереж
- Мета роботи – розробка програмного модуля для розпізнавання зображення за допомогою нейронних мереж
- Методи дослідження – аналіз нейронних мереж, експериментальний вибір архітектури

Види нейромереж

Прямого поширення (Feedforward Neural Networks, FNN) - Інформація переміщується лише в одному напрямку, від вхідних вузлів до вихідних.

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) - Має здатність зберігати інформацію в попередніх входах, використовуючи зв'язки між нейронами.

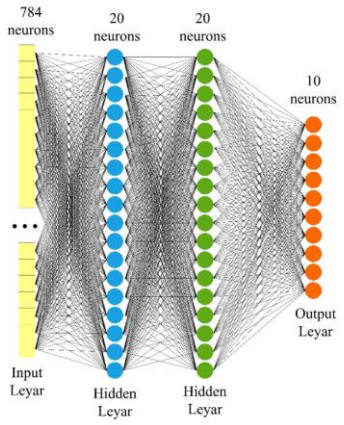
Згорткові нейронні мережі (Convolutional Neural Networks, CNN) - Використовують згортку для зменшення кількості параметрів та спільного використання ваг.

Мережі довготривалої короткотривалої пам'яті (Long Short-Term Memory Networks, LSTM) - Вирішують проблему зниклих градієнтів у рекурентних мережах і дозволяють враховувати довгострокові залежності у даних.

Рекурсивна нейронна мережа (Recursive Neural Network, Recursive NN) - Використовується для обробки структурованих даних, таких як дерева або графи.

Багатошаровий перцептрон

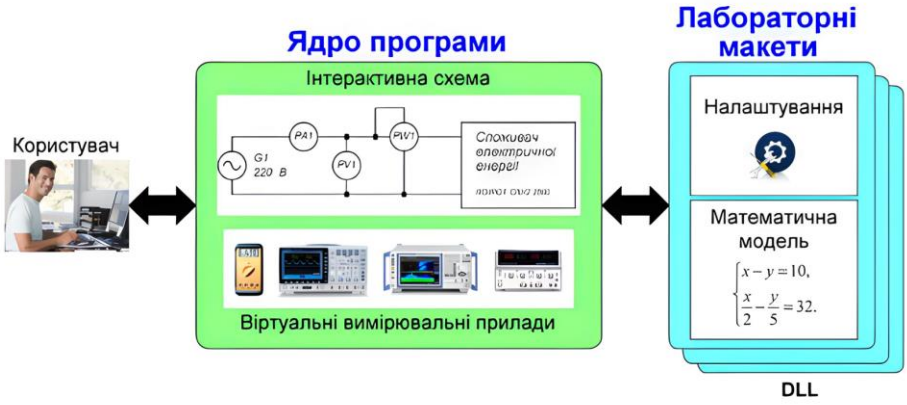
- Вхідні дані
- Перший схований шар
- Другий схований шар
- Шари виводу



Метод зворотного поширення помилки

- Вхідні дані
- Випадкова генерація ваг
- Обчислення мережі по шарам
- Розрахунок помилки шару
- Розрахунок загальної помилки мережі
- Оновлення ваг мережі

Вибір практичної реалізації

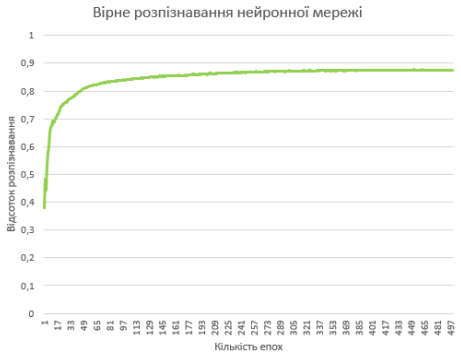


Зовнішній вигляд програмного забезпечення

Датасет Fashion MNIST

- Розмір зображень 28x28 пікселів
- 10 типів одягу
- 60 000 зображень
- Чорно-білого формату

Результати дослідження



Висновки

Головними результатами роботи є наступні пункти:

1. Проведено аналіз видів нейронних мереж для розпізнавання зображень та було прийнято рішення вибрати нейронну мережу з прямим поширенням інформації.

2. Було проведено детальний аналіз в результаті чого було обрано модель багатошарового перцептрана з двома прихованими шарами. У кожному прихованому шарі знаходиться по 20 неронів.

3. За даними аналізу дана модель є ефективною, оскільки точність набуває значення 87,6% при навчанні на збірці зображень кількістю 60 000 зображень. Проведення тестування на першому наборі даних, який включає 30 000 зразків одягу, призводить до помітного покращення ніж на другому.

Результати даної роботи доцільно рекомендувати для удосконалення і практичного застосування у наступних напрямках.

1. При створенні програмного забезпечення функція котрого полягає в розпізнаванні об’єктів завчасно продумуючи сверу використання нейронної мережі.

2. Під час підготовки фахівців в галузі інформаційних технологій, інженерії програмного забезпечення, комп’ютерних науках, комп’ютерній інженерії і кібербезпеки.

Дякую за увагу!

Курюкін Ілля Сергійович

ДОДАТОК Б
ЛІСТИНГ ПРОГРАМИ

```
library AI_99;

uses
  Winapi.Windows,
  System.Classes,
  System.SysUtils,
  System.UITypes,
  System.Math,
  System.DateUtils,
  VCL.Graphics,
  VCL.Forms,
  VCL.Dialogs,
  Vcl.StdCtrls,
  uNeuronNetwork in '..\Common\uNeuronNetwork.pas',
  uDLLTools in '..\Common\uDLLTools.pas',
  uPaintComponents in '..\Common\uPaintComponents.pas';

{$R *.res}

const
  LOG_FILE_SEPARATOR = ';';

  NEURON_ACTIVE_VALUE = 0.98;
  NEURON_UNACTIVE_VALUE = 0.02;

  RUN_MODE_RECOGNITION = 0;
  RUN_MODE_TRAINING = 1;
  RUN_MODE_TESTING = 2;

  SHOW_TRAINING_SAMPLES = 0;
  SHOW_TEST_SAMPLES = 1;

  SCHEM_CIRCUIT = 0;
  SCHEM_WIDTH = 38;
  SCHEM_HEIGHT = 20;
```

```
        BUTTON_HEIGHT = 1;
        DEFAULT_MAGRIN = 0.5;

        NEURON_AREA_TOP = 0;
        NEURON_AREA_LEFT = 0;
        NEURON_AREA_HEIGHT = 3;
        NEURON_AREA_WIDTH = SCHEM_WIDTH;
        NEURON_MARGIN = 0.25;

        SCREEN_AREA_TOP = NEURON_AREA_TOP + NEURON_AREA_HEIGHT +
DEFAULT_MAGRIN
         + BUTTON_HEIGHT + DEFAULT_MAGRIN;
        SCREEN_AREA_LEFT = 0;
        SCREEN_AREA_HEIGHT = SCHEM_HEIGHT - NEURON_AREA_HEIGHT - 2 *
BUTTON_HEIGHT
         - 3 * DEFAULT_MAGRIN;
        SCREEN_AREA_WIDTH = 10;

        INFO_AREA_WIDTH = 10;
        INFO_AREA_LEFT = SCHEM_WIDTH - INFO_AREA_WIDTH;
        INFO_AREA_TOP = NEURON_AREA_TOP + NEURON_AREA_HEIGHT +
DEFAULT_MAGRIN;
        INFO_AREA_HEIGHT = 2.5;

        SAMPLE_AREA_WIDTH = 10;
        SAMPLE_AREA_LEFT = SCHEM_WIDTH - SAMPLE_AREA_WIDTH;
        SAMPLE_AREA_TOP = INFO_AREA_TOP + INFO_AREA_HEIGHT + DEFAULT_MAGRIN;
        SAMPLE_AREA_HEIGHT = SCHEM_HEIGHT - NEURON_AREA_HEIGHT -
INFO_AREA_HEIGHT
         - 2 * DEFAULT_MAGRIN;

        SAMPLES_AREA_TOP = NEURON_AREA_TOP + NEURON_AREA_HEIGHT +
DEFAULT_MAGRIN
         + BUTTON_HEIGHT + DEFAULT_MAGRIN;
        SAMPLES_AREA_LEFT = SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH +
DEFAULT_MAGRIN;
        SAMPLES_AREA_HEIGHT = SCHEM_HEIGHT - NEURON_AREA_HEIGHT - 2 *
BUTTON_HEIGHT
```

```
    - 3 * DEFAULT_MAGRIN;
    SAMPLES_AREA_WIDTH = SCHEM_WIDTH - SCREEN_AREA_WIDTH -
SAMPLE_AREA_WIDTH - 1;
    SAMPLES_MARGIN = 0.25;

    DEBUG_MODE = False;
  var
    gNeuronUnderCursor: TNeuron = nil;
    gSampleUnderCursor: TNeuronSample = nil;

    // кнопки
    gFocusedNeuronButton: PComponentInfo = nil;
    gFocusedSampleButton: PComponentInfo = nil;

    gLoadScreenButton: PComponentInfo = nil;
    gClearScreenButton: PComponentInfo = nil;
    gScreenInSampleButton: PComponentInfo = nil;

    gAddSampleButton: PComponentInfo = nil;
    gDeleteSampleButton: PComponentInfo = nil;
    gClearSampleButton: PComponentInfo = nil;
    gSaveSamplesButton: PComponentInfo = nil;
    gLoadSamplesButton: PComponentInfo = nil;

    gShowTrainingSamplesButton: PComponentInfo = nil;
    gShowTestSamplesButton: PComponentInfo = nil;
    gShowModeButton: PComponentInfo = nil;

    // настройки
    gMaxEpocheCount: PDouble;
    gMinErrorValue: PDouble;
    gCreateLog: PDouble;

    // нейронная сеть
    gNetwork: TNeuronNetwork = nil;

    // переменные
    gRunMode: Integer = RUN_MODE_RECOGNITION;
```

```
  gShowMode: Integer = SHOW_TRAINING_SAMPLES;

  gEpocheNumber: Integer;

  gIterationNumber: Integer;
  gTotalEpocheError: Double;
  gTotalEpocheErrorSum: Double;

  gNeuronIndex: Integer;
  gSampleIndex: Integer;
  gLastCalcTime: TDateTime;

  gLogFileName: string;

  // ###########################################################################

procedure AddToLog(aFileName, S: String);
var
  F: TextFile;
begin
//  if Round(gCreateLog^) <> 1 then
    Exit;

  if S = '' then
    Exit;

  AssignFile(F, aFileName);
  try
    try
      if FileExists(aFileName) then Append(F)
        else Rewrite(F);

      Writeln(F, S);
    except

    end;
  finally
    CloseFile(F);
```

```
  end;
end;

// ############################################################################

function GetFocusedNeuron: TNeuron;
begin
  if Assigned(gNetwork.OutputLayer) then
    Result := gNetwork.OutputLayer.FocusedNeuron
  else
    Result := nil;
end;

// ############################################################################

function GetActiveSampleList: TNeuronSampleList;
var
  ActiveNeuron: TNeuron;
begin
  ActiveNeuron := GetFocusedNeuron;

  if Assigned(ActiveNeuron) then
    case gShowMode of
      SHOW_TRAINING_SAMPLES: Result := ActiveNeuron.TrainingSamples;
      else Result := ActiveNeuron.TestSamples;
    end
  else
    Result := nil;
end;

// ############################################################################

function GetFocusedSample: TNeuronSample;
var
  ActiveList: TNeuronSampleList;
begin
  ActiveList := GetActiveSampleList;
```

```
  if Assigned(ActiveList) then
    Result := ActiveList.FocusedSample
  else
    Result := nil;
end;

// ############################################################################
//                  Конфигурирование нейронной сети
// ############################################################################

procedure ConfigurateNetwork();
var
  i, j: Integer;
  ActiveNeuron: TNeuron;
begin
  // установка размеров рабочей области ----------------------------------------
  gNetwork.SetScreenDimentions(28, 28);

  // устанвливаем количество слоев ---------------------------------------------
  gNetwork.Layers.Count := 4;

  //---------------------------- входной слой ---------------------------------

  // количество нейронов
  gNetwork.Layers[0].Neurons.Count := Length(gNetwork.ScreenOutputs);

  // настройка нейронов
  for i := 0 to gNetwork.Layers[0].Neurons.Count - 1 do
  begin
    ActiveNeuron := gNetwork.Layers[0].Neurons[i];

    // создаем входы
    ActiveNeuron.InputCount := 1;

    // подключаем вход
    ActiveNeuron.Inputs[0] := @gNetwork.ScreenOutputs[i];

    // иницализируем весовые коэффициенты
```

```
  ActiveNeuron.Weights[0] := 1;

  // подключем функцию активации
  ActiveNeuron.ActivationFunction := @LinearActivationFunction;
end;

// ------------------------ промежуточный слой 1 --------------------------

// количество нейронов
gNetwork.Layers[1].Neurons.Count := 20;

// настройка нейронов
for i := 0 to gNetwork.Layers[1].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[1].Neurons[i];

  // создаем входы
  ActiveNeuron.InputCount := gNetwork.Layers[0].Neurons.Count;

  // подключаем входы
  for j := 0 to ActiveNeuron.InputCount - 1 do
  begin
    ActiveNeuron.Inputs[j] := @gNetwork.Layers[0].Neurons[j].Output;

    // иницализируем весовые коэффициенты
    ActiveNeuron.Weights[j] := (Random(201) - 100) / 100;
   end;

  // подключем функцию активации
  ActiveNeuron.ActivationFunction := @SigmaActivationFunction;
end;

//----------------------- промежуточный слой 2 ---------------------------------

// количество нейронов
gNetwork.Layers[2].Neurons.Count := 20;

// настройка нейронов
```

```
for i := 0 to gNetwork.Layers[2].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[2].Neurons[i];

  // создаем входы
  ActiveNeuron.InputCount := gNetwork.Layers[1].Neurons.Count;

  // подключаем входы
  for j := 0 to ActiveNeuron.InputCount - 1 do
  begin
    ActiveNeuron.Inputs[j] := @gNetwork.Layers[1].Neurons[j].Output;

    // инициализируем весовые коэффициенты
    ActiveNeuron.Weights[j] := (Random(201) - 100) / 100;
  end;

  // подключем функцию активации
  ActiveNeuron.ActivationFunction := @SigmaActivationFunction;
end;

//------------------------------ выходной слой ----------------------------

// количество нейронов
gNetwork.Layers[3].Neurons.Count := 10;

// разрешение на редактирование
gNetwork.Layers[3].CanFocus := True;

// настройка нейронов
for i := 0 to gNetwork.Layers[3].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[3].Neurons[i];

  // создаем входы
  ActiveNeuron.InputCount := gNetwork.Layers[2].Neurons.Count;

  // подключаем входы
  for j := 0 to ActiveNeuron.InputCount - 1 do
```

```
    begin
      ActiveNeuron.Inputs[j] := @gNetwork.Layers[2].Neurons[j].Output;

      // инициализируем весовые коэффициенты
      ActiveNeuron.Weights[j] := (Random(201) - 100) / 100;
    end;

    // подключем функцию активации
    ActiveNeuron.ActivationFunction := @SigmaActivationFunction;
  end;

  // загрузка параметров по умолчанию
  gNetwork.LoadFromFile(False);

  // подготовка к отображению
  gNetwork.OutputLayer.Neurons.FocuseFirst;
  if Assigned(gNetwork.OutputLayer.FocusedNeuron) then
  begin
    gNetwork.OutputLayer.FocusedNeuron.TrainingSamples.FocuseFirst;
    gNetwork.OutputLayer.FocusedNeuron.TestSamples.FocuseFirst;
  end;
end;

// ##############################################################################
//                    Настройка параметров макета
// ##############################################################################

procedure SetLayoutGlobals(LG: PLayoutGlobals); stdcall;
begin
  // проверка указателя
  if not Assigned(LG)
    then Exit;

  // установка параметров
  Application.Handle := LG.ApplicationHandle;
  gVariantNumber := LG.VariantNumber;
  gLanguage := LG.Language;
  gGraphicParams := LG.GraphicParams;
```

```
  LG.DebugMode := DEBUG_MODE;
  gDebugMode := DEBUG_MODE;

  // создание нейронной сети
  if not Assigned(gNetwork) then
  begin
    gNetwork := TNeuronNetwork.Create;
    ConfigurateNetwork;
  end;
end;

// ############################################################################

function GetLayoutInfo(LayoutInfo: PLayoutInfo;
  aLanguage: TLanguage): Boolean; stdcall;
begin
  Result := False;

  // проверка указателя
  if not Assigned(LayoutInfo)
    then Exit;

  // Штучний інтелект
  case aLanguage of
    lgEng: LayoutInfo.Subject := 'AI';
    lgRus: LayoutInfo.Subject := 'ИИ';
    else LayoutInfo.Subject := 'ШІ';
  end;

  case aLanguage of
    lgEng: LayoutInfo.Caption := 'Layout xx';
    lgRus: LayoutInfo.Caption := 'Макет xx';
    else LayoutInfo.Caption := 'Макет xx';
  end;

  case aLanguage of
    lgEng: LayoutInfo.FullCaption := 'Name of Layout';
```

```
   lgRus: LayoutInfo.FullCaption := 'Название макета';
   else LayoutInfo.FullCaption := 'Назва макету';
  end;

  case aLanguage of
   lgEng: LayoutInfo.Decription := 'Description of Layout';
   lgRus: LayoutInfo.Decription := 'Описание макета';
   else LayoutInfo.Decription := 'Опис макету';
  end;

  LayoutInfo.SchemIndexDefault := SCHEM_CIRCUIT;

  Result := True;
end;

// ############################################################################
//                  Настройка параметров схемы
// ############################################################################

function SetSchemInfo(SchemInfo: PSchemInfo; const SchemIndex: Integer): Boolean; stdcall;
begin
  Result := False;

  // проверка указателя
  if not Assigned(SchemInfo)
   then Exit;

  // установка параметров (в зависимости от индекса схемы)
  case SchemIndex of
   SCHEM_CIRCUIT: begin
     Result := True;

     case gLanguage of
      lgEng: SchemInfo.Caption := 'Neuron Network';
      lgRus: SchemInfo.Caption := 'Нейронная сеть';
      else SchemInfo.Caption := 'Нейронна мережа';
     end;
```

```
    SchemInfo.Width := SCHEM_WIDTH;
    SchemInfo.Height := SCHEM_HEIGHT;
    SchemInfo.UpdateInterval := 0;//0.25;
    SchemInfo.UpdateOnStop := True;
   end;
  end;
end;

// ############################################################################
//                      Настройка компонентов
// ############################################################################

function CheckSimulation: Boolean;
var
  S: String;
begin
  Result := not gInSimulation;

  if Result then
    Exit;

  case gLanguage of
    lgEng: S := 'Stop simulation!';
    lgRus: S := 'Остановите моделирование!';
    else S := 'Зупиніть моделювання!';
  end;

  ShowMessage(S);
end;

// ############################################################################

function LoadNetworkFromFile(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin

  if CheckSimulation then
    gNetwork.LoadFromFile(True);
```

```
  Result := True;
end;

// ############################################################################

function SaveNetworkToFile(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  if CheckSimulation then
    gNetwork.SaveToFile(True);

  Result := True;
end;

// ############################################################################

function ChangeRunMode(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  if CheckSimulation then
    case gRunMode of
      RUN_MODE_RECOGNITION: gRunMode := RUN_MODE_TRAINING;
      RUN_MODE_TRAINING: gRunMode := RUN_MODE_TESTING;
      else gRunMode := RUN_MODE_RECOGNITION;
    end;

  Result := True;
end;

// ############################################################################

function ShowTrainingSamples(Language: TLanguage; ComponentInfo: PComponentInfo;
```

```
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  gShowMode := SHOW_TRAINING_SAMPLES;

  Result := True;
end;

// ############################################################################

function ShowTestSamples(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  gShowMode := SHOW_TEST_SAMPLES;

  Result := True;
end;

// ############################################################################

function SaveSamplesToFile(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;
```

```
  ActiveSampleList := GetActiveSampleList;

  if Assigned(ActiveSampleList) then
    ActiveSampleList.SaveToFile(True);

  Result := True;
end;

// ############################################################################

function LoadSamplesFromFile(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  ActiveSampleList := GetActiveSampleList;

  if CheckSimulation and Assigned(ActiveSampleList) then
    ActiveSampleList.LoadFromFile(True);

  Result := True;
end;

// ############################################################################

function AddSample(Language: TLanguage; ComponentInfo: PComponentInfo;
  Params: TList; PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
begin
  Result := False;
```

```
  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  ActiveSampleList := GetActiveSampleList;

  if CheckSimulation and Assigned(ActiveSampleList) then
    ActiveSampleList.AddNewSample.Assing(gNetwork.Screen);

  Result := True;
end;

// ############################################################################

function ClearSample(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSample: TNeuronSample;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  ActiveSample := GetFocusedSample;

  if CheckSimulation and Assigned(ActiveSample) then
    ActiveSample.Clear;

  Result := True;
end;

// ############################################################################

function DeleteSample(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
  ActiveSample: TNeuronSample;
```

```
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  ActiveSample := GetFocusedSample;

  if Assigned(ActiveSample) then
  begin
    ActiveSampleList := GetActiveSampleList;

    if CheckSimulation and Assigned(ActiveSampleList) then
      ActiveSampleList.DeleteSample(ActiveSample)
  end;

  Result := True;
end;

// ############################################################################

function SetSampleFocus(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  ActiveSampleList := GetActiveSampleList;

  if Assigned(ActiveSampleList) then
    ActiveSampleList.FocusedSample := gSampleUnderCursor;

  Result := True;
end;
```

```
// ###########################################################################

function SetNueronFocus(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  if Assigned(gNetwork.OutputLayer) then
    gNetwork.OutputLayer.Neurons.Focused := gNeuronUnderCursor;

  Result := True;
end;

// ###########################################################################

function LoadScreenFromFile(Language: TLanguage; ComponentInfo: PComponentInfo; Params:
TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  if CheckSimulation then
    gNetwork.LoadScreenFromFile;

  Result := True;
end;

// ###########################################################################

function ClearScreen(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
```

```
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  if CheckSimulation then
    gNetwork.ClearScreen;

  if Assigned (PaintSchemProc) then
    PaintSchemProc;



  Result := True;
end;

// ############################################################################

function ScreenInSample(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSampleList: TNeuronSampleList;
begin
  Result := False;

  if EditMode <> EM_MOUSE_DOWN then
    Exit;

  if CheckSimulation then
  begin
    ActiveSampleList := GetActiveSampleList;

    if Assigned(ActiveSampleList) then
      ActiveSampleList.AddNewSample.Assing(gNetwork.Screen);
  end;

  Result := True;
end;
```

```
function SampleToScreen(Language: TLanguage; ComponentInfo: PComponentInfo; Params: TList;
  PaintSchemProc: TPaintSchemProc; EditMode: Integer): Boolean;
var
  ActiveSample: TNeuronSample;
begin
  Result := False;

  if EditMode <> EM_BTN_CLICK then
    Exit;

  ActiveSample := GetFocusedSample;

  if Assigned(ActiveSample) then
    gNetwork.CopyToScreen(ActiveSample.Screen);

  if Assigned (PaintSchemProc) then
    PaintSchemProc;

  Result := True;
end;


// ############################################################################

function SetComponentInfo(Component: PComponentInfo; const SchemIndex,
  ComponentIndex: Integer): Boolean; stdcall;
begin
  Result := False;

  // проверка указателя
  if not Assigned(Component)
    then Exit;

  case ComponentIndex of
    // плавающая кнопка для выбора найерона
    0: begin
      Result := True;
```

```
  gFocusedNeuronButton := Component;
  Component.EditParamsProc := SetNueronFocus;
  Component.ShowButtonOntoolBar := False;
end;

// плавающая кнопка для выбора образца
1: begin
  Result := True;

  gFocusedSampleButton := Component;
  Component.EditParamsProc := SetSampleFocus;
  Component.ShowButtonOntoolBar := False;
end;

// настройка нейронной сети
2: begin
  Result := True;

  case gLanguage of
    lgEng: Component.Caption := 'Settings';
    lgRus: Component.Caption := 'Настройки';
    else Component.Caption := 'Налаштування';
  end;

  Component.ShowButtonOntoolBar := True;
end;

// кнопка "Загрузить изображение"
3: begin
  Result := True;

  gLoadScreenButton := Component;
  Component.EditParamsProc := LoadScreenFromFile;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SCREEN_AREA_LEFT,
```

```
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5,
    SCREEN_AREA_LEFT + 3,
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5);
end;

// Кнопка "Очистить изображение"
4: begin
  Result := True;

  gClearScreenButton := Component;
  Component.EditParamsProc := ClearScreen;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SCREEN_AREA_LEFT + 3.5,
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5,
    SCREEN_AREA_LEFT + 6.5,
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5);
end;

 // Кнопка "Изображение как образец"
5: begin
  Result := True;

  gScreenInSampleButton := Component;
  Component.EditParamsProc := ScreenInSample;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH - 3,
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5,
    SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH,
    SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5);
end;

// Кнопка "Загрузить образцы"
6: begin
  Result := True;
```

```
  gLoadSamplesButton := Component;
  Component.EditParamsProc := LoadSamplesFromFile;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 0,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5,
    SAMPLES_AREA_LEFT + 3,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5);
end;

// Кнопка "Сохранить образцы"
7: begin
  Result := True;

  gSaveSamplesButton := Component;
  Component.EditParamsProc := SaveSamplesToFile;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 3.5,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5,
    SAMPLES_AREA_LEFT + 6.5,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5);
end;

// Кнопка "Добавить образец"
8: begin
  Result := True;

  gAddSampleButton := Component;
  Component.EditParamsProc := AddSample;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 7,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5,
```

```
    SAMPLES_AREA_LEFT + 10,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5);
end;

// Кнопка "Удалить образец"
9: begin
  Result := True;

  gDeleteSampleButton := Component;
  Component.EditParamsProc := DeleteSample;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 10.5,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5,
    SAMPLES_AREA_LEFT + 13.5,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5);
end;

// Кнопка "Очистить образец"
10: begin
  Result := True;

  gClearSampleButton := Component;
  Component.EditParamsProc := ClearSample;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 14,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5,
    SAMPLES_AREA_LEFT + 17,
    SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5);
end;

// Кнопка "Отобразить тренировочный набор"
11: begin
  Result := True;
```

```
  gShowTrainingSamplesButton := Component;
  Component.EditParamsProc := ShowTrainingSamples;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT,
    SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN,
    SAMPLES_AREA_LEFT + 0.5 * SAMPLES_AREA_WIDTH - 0.5 * DEFAULT_MAGRIN,
    SAMPLES_AREA_TOP - DEFAULT_MAGRIN);
end;

// Кнопка "Отобразить тестовый набор"
12: begin
  Result := True;

  gShowTestSamplesButton := Component;
  Component.EditParamsProc := ShowTestSamples;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SAMPLES_AREA_LEFT + 0.5 * SAMPLES_AREA_WIDTH + 0.5 * DEFAULT_MAGRIN,
    SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN,
    SAMPLES_AREA_LEFT + SAMPLES_AREA_WIDTH,
    SAMPLES_AREA_TOP - DEFAULT_MAGRIN);
end;

// Кнопка "Режим работы"
13: begin
  Result := True;

  gShowModeButton := Component;
  Component.EditParamsProc := ChangeRunMode;
  Component.ShowButtonOntoolBar := False;

  Component.ActiveRect := ActiveRect(
    SCREEN_AREA_LEFT,
    SCREEN_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN,
    SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH,
```

```
    SCREEN_AREA_TOP - DEFAULT_MAGRIN);
end;

// Кнопка "Загрузить нейросеть"
14: begin
  Result := True;

  case gLanguage of
    lgEng: Component.Caption := 'Load neural network';
    lgRus: Component.Caption := 'Загрузить нейросеть';
    else Component.Caption := 'Завантажити нейромережу';
  end;

  Component.ShowButtonOntoolBar := True;
  Component.EditParamsProc := LoadNetworkFromFile;
end;

// Кнопка "Сохранить нейросеть"
15: begin
  Result := True;

  case gLanguage of
    lgEng: Component.Caption := 'Save neural network';
    lgRus: Component.Caption := 'Сохранить нейросеть';
    else Component.Caption := 'Зберегти нейромережу';
  end;

  Component.ShowButtonOntoolBar := True;
  Component.EditParamsProc := SaveNetworkToFile;
end;

// Кнопка "Сохранить нейросеть"
16: begin
  Result := True;

  case gLanguage of
    lgEng: Component.Caption := 'In the screen';
    lgRus: Component.Caption := 'В экран';
```

```
      else Component.Caption := 'До екрану';
    end;

    Component.ShowButtonOntoolBar := True;
    Component.EditParamsProc := SampleToScreen;
   end;
  end;
end;

// #############################################################################

function SetComponentParamInfo(Param: PComponentParamInfo;
  const SchemIndex, ComponentIndex, ParamIndex: Integer): Boolean; stdcall;
begin
  Result := False;

  // проверка указателя
  if not Assigned(Param)
    then Exit;

  case ComponentIndex of
    // настройки
    2: case ParamIndex of
      0: begin
        Result := True;

        case gLanguage of
          lgEng: Param.FullCaption := 'Maximum epoche count';
          lgRus: Param.FullCaption := 'Максимальное количнство эпох';
          else Param.FullCaption := 'Максимальна кількість епох';
        end;

        Param.ShortCaption := '';
        Param.UnitCaption := '';

        Param.Minimum := 1;
        Param.Maximum := 10000;
        Param.Value := 100;
```

```
  gMaxEpocheCount := @Param.Value;
end;

1: begin
  Result := True;

  case gLanguage of
    lgEng: Param.FullCaption := 'Minimum error value';
    lgRus: Param.FullCaption := 'Минимальное значение ошибки';
    else Param.FullCaption := 'Мінімальне значення помилки';
  end;

  Param.ShortCaption := '';
  Param.UnitCaption := '';

  Param.Minimum := 0;
  Param.Maximum := 2000;
  Param.Value := 0;

  gMinErrorValue := @Param.Value;
end;

2: begin
  Result := True;

  case gLanguage of
    lgEng: Param.FullCaption := 'Create log-file';
    lgRus: Param.FullCaption := 'Создавать лог-файлы';
    else Param.FullCaption := 'Стіорювати лог-файли';
  end;

  case gLanguage of
    lgEng: Param.ListValues := '"No","Да"';
    lgRus: Param.ListValues := '"Нет","Да"';
    else Param.ListValues := '"Ні","Так"';
  end;
```

```
      case gLanguage of
       lgEng: Param.ListDescription := '"No files will be created"'
         + '"The progress of testing or training will be recorded in a file in .csv format"';
       lgRus: Param.ListDescription := '"Файлы создаваться не будут",'
         + '"Ход тестирования или обучения будет записан в файл в формате .csv"';
       else Param.ListDescription := '"Файли створюватися не будуть",'
         + '"Хід тествування або навчання буде записано у файл у форматі .csv"';
      end;

      Param.Value := 1;

      gCreateLog := @Param.Value;
     end;
    end;
  end;
end;

// ############################################################################
//                          Графика
// ############################################################################

function PointInRegion(X, Y, aLeft, aTop, aRight, aBottom: Double): Boolean;
begin
  Result := (X >= aLeft) and (X <= aRight) and (Y >= aTop) and (Y <= aBottom);
end;

// ############################################################################

function CalculateSamplesPixelSize(SamplesList: TNeuronSampleList;
  var NumberOfColumns, NumberOfRows: Integer): Double;
var
  aPixelSize: Double;
  ActiveWidth, ActiveHeight: Double;
  MaxSampleWidth, MaxSampleHeight: Integer;
  NumberOfImageInColumn, NumberOfImageInRow: Integer;
begin
  Result := 0;
  NumberOfColumns := 0;
```

```
NumberOfRows := 0;

if not Assigned(SamplesList) then
  Exit;

MaxSampleWidth := SamplesList.MaxWidth;
MaxSampleHeight := SamplesList.MaxHeight;

if (MaxSampleWidth = 0) or (MaxSampleHeight = 0) then
  Exit;

// пробуем разместить по горизонтали
for NumberOfImageInRow := SamplesList.Count downto 1 do
begin
  NumberOfImageInColumn := SamplesList.Count div NumberOfImageInRow;

  if SamplesList.Count mod NumberOfImageInRow > 0 then
    Inc(NumberOfImageInColumn);

  ActiveWidth := SAMPLES_AREA_WIDTH - 2 * SAMPLES_MARGIN
    - (NumberOfImageInRow - 1) * SAMPLES_MARGIN;
  ActiveHeight := SAMPLES_AREA_HEIGHT - 2 * SAMPLES_MARGIN
    - (NumberOfImageInColumn - 1) * SAMPLES_MARGIN;

  aPixelSize := Min(ActiveWidth / (NumberOfImageInRow * MaxSampleWidth),
    ActiveHeight / (NumberOfImageInColumn * MaxSampleHeight));

  if aPixelSize > Result then
  begin
    Result := aPixelSize;
    NumberOfRows := NumberOfImageInColumn;
    NumberOfColumns := NumberOfImageInRow;
  end;
end;

// пробуем разместить по вертикали
for NumberOfImageInColumn := SamplesList.Count downto 1 do
begin
```

```
    NumberOfImageInRow := SamplesList.Count div NumberOfImageInColumn;

    if SamplesList.Count mod NumberOfImageInColumn > 0 then
      Inc(NumberOfImageInRow);

    ActiveWidth := SAMPLES_AREA_WIDTH - 2 * SAMPLES_MARGIN
      - (NumberOfImageInRow - 1) * SAMPLES_MARGIN;
    ActiveHeight := SAMPLES_AREA_HEIGHT - 2 * SAMPLES_MARGIN
      - (NumberOfImageInColumn - 1) * SAMPLES_MARGIN;

    aPixelSize := Min(ActiveWidth / (NumberOfImageInRow * MaxSampleWidth),
      ActiveHeight / (NumberOfImageInColumn * MaxSampleHeight));

    if aPixelSize > Result then
    begin
      Result := aPixelSize;
      NumberOfRows := NumberOfImageInColumn;
      NumberOfColumns := NumberOfImageInRow;
    end;
  end;
end;

// #############################################################################

procedure PaintCircuit(C: TCanvas; const InSimulation: Boolean);
const
  DEFAULT_SCALE_COEFFICIENT = 7;
var
 i: Integer;
 j: Integer;

 S: String;

 CursorX, CursorY: Double;
 aWidth, aHeight, aPixelSize: Double;
 aLeft, aRight, aTop, aBottom: Double;
 aNumberOfRows, aNumberOfColumns: Integer;
 aRect: TRect;
```

```
 aFreeHorizontalSize, aFreeVerticalSize: Double;

 aActiveList: TNeuronSampleList;
 aActiveSample: TNeuronSample;
begin
 if not Assigned(gNetwork) then
   Exit;

 SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);

 // определение позиции курсора в условных координатах схемы ------------------
 CursorX := XT(gGraphicParams.CursorX);
 CursorY := YT(gGraphicParams.CursorY);

 // область отображения образцов нейрона ---------------------------------------
 aActiveList := GetActiveSampleList;
 aActiveSample := GetFocusedSample;

 if Assigned(aActiveList) then
 begin
   aPixelSize := CalculateSamplesPixelSize(aActiveList, aNumberOfColumns,
     aNumberOfRows);

   if (aPixelSize > 0) and (aActiveList.Count > 0) then
   begin
     aWidth := aPixelSize * aActiveList.MaxWidth;
     aHeight := aPixelSize * aActiveList.MaxHeight;

     aFreeHorizontalSize := SAMPLES_AREA_WIDTH - aWidth * aNumberOfColumns
       - 2 * SAMPLES_MARGIN - (aNumberOfColumns - 1) * SAMPLES_MARGIN;

     aFreeVerticalSize := SAMPLES_AREA_HEIGHT - aHeight * aNumberOfRows
       - 2 * SAMPLES_MARGIN - (aNumberOfRows - 1) * SAMPLES_MARGIN;

     aLeft := SAMPLES_AREA_LEFT + SAMPLES_MARGIN + aFreeHorizontalSize / 2 ;
     aTop := SAMPLES_AREA_TOP + SAMPLES_MARGIN + aFreeVerticalSize / 2;

     aRight := aLeft + aWidth;
```

```
aBottom := aTop + aHeight;

for i := 0 to aActiveList.Count - 1 do
begin
 aRect.Left := TX(aLeft);
 aRect.Top := TY(aTop);
 aRect.Right := TX(aLeft + aWidth);
 aRect.Bottom := TY(aTop + aHeight);

 C.StretchDraw(aRect, aActiveList[i].Screen);

 SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);

 if aActiveList[i] = aActiveList.FocusedSample then
 begin
  C.Pen.Color := TColors.Red;
  C.Pen.Width := 2 * GetDefaultLineWidth(DEFAULT_SCALE_COEFFICIENT);
 end;

 if PointInRegion(CursorX, CursorY, aLeft, aTop, aRight, aBottom) then
 begin
  C.Pen.Color := TColors.Blue;
  C.Pen.Width := 2 * GetDefaultLineWidth(DEFAULT_SCALE_COEFFICIENT);

  gSampleUnderCursor := aActiveList[i];

  if Assigned(gFocusedSampleButton) then
    gFocusedSampleButton.ActiveRect := ActiveRect(aLeft, aTop, aRight, aBottom);
 end;

 C.Polyline([TP(aLeft, aTop),
          TP(aLeft + aWidth, aTop),
          TP(aLeft + aWidth, aTop + aHeight),
          TP(aLeft, aTop + aHeight),
          TP(aLeft, aTop)]);

 aLeft := aLeft + aWidth + SAMPLES_MARGIN;
 if (aLeft + aWidth) > (SAMPLES_AREA_LEFT + SAMPLES_AREA_WIDTH) then
```

```
          begin
            aLeft := SAMPLES_AREA_LEFT + SAMPLES_MARGIN + aFreeHorizontalSize / 2;
            aTop := aTop + aHeight + SAMPLES_MARGIN;
          end;

          aRight := aLeft + aWidth;
          aBottom := aTop + aHeight;
        end;
      end;
    end;

    SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);
    C.Brush.Color := TColors.White;

    C.Polyline([TP(SAMPLES_AREA_LEFT, SAMPLES_AREA_TOP),
                TP(SAMPLES_AREA_LEFT + SAMPLES_AREA_WIDTH, SAMPLES_AREA_TOP),
                TP(SAMPLES_AREA_LEFT + SAMPLES_AREA_WIDTH, SAMPLES_AREA_TOP +
SAMPLES_AREA_HEIGHT),
                TP(SAMPLES_AREA_LEFT, SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT),
                TP(SAMPLES_AREA_LEFT, SAMPLES_AREA_TOP)]);


    // область отображения информации ---------------------------------------------
    SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);
    C.Brush.Color := TColors.White;

    C.Polyline([TP(INFO_AREA_LEFT, INFO_AREA_TOP),
                TP(INFO_AREA_LEFT + INFO_AREA_WIDTH, INFO_AREA_TOP),
                TP(INFO_AREA_LEFT + INFO_AREA_WIDTH, INFO_AREA_TOP +
INFO_AREA_HEIGHT),
                TP(INFO_AREA_LEFT, INFO_AREA_TOP + INFO_AREA_HEIGHT),
                TP(INFO_AREA_LEFT, INFO_AREA_TOP)]);


    case gLanguage of
      lgEng: S := 'Epoche number - ' + IntToStr(gEpocheNumber);
      lgRus: S := 'Номер эпохи - '  + IntToStr(gEpocheNumber);
      else   S := 'Номер епохи - '  + IntToStr(gEpocheNumber);
```

```
end;

PaintText(C, S, False,
  INFO_AREA_LEFT + 0.25,
  INFO_AREA_TOP + 0.25,
  INFO_AREA_WIDTH - 0.5, 0.5, haLeft, vaCenter);

case gLanguage of
  lgEng: S := 'Iteration number - ' + IntToStr(gIterationNumber);
  lgRus: S := 'Номер итерации - '  + IntToStr(gIterationNumber);
  else   S := 'Номер ітерации - '  + IntToStr(gIterationNumber);
end;

PaintText(C, S, False,
  INFO_AREA_LEFT + 0.25,
  INFO_AREA_TOP + 0.25 + 0.75,
  INFO_AREA_WIDTH - 0.5, 0.5, haLeft, vaCenter);

case gLanguage of
  lgEng: S := 'Total error - ' + FloatToStr(Round(gTotalEpocheError * 10000)/10000);
  lgRus: S := 'Ошибка сети - '  + FloatToStr(Round(gTotalEpocheError * 10000)/10000);
  else   S := 'Помилка мережі - '  + FloatToStr(Round(gTotalEpocheError * 10000)/10000);
end;

PaintText(C, S, False,
  INFO_AREA_LEFT + 0.25,
  INFO_AREA_TOP + 0.25 + 1.5,
  INFO_AREA_WIDTH - 0.5, 0.5, haLeft, vaCenter);

// область отображения активного образца ------------------------------------
if Assigned(aActiveSample) then
begin
  if (aActiveSample.Screen.Width > 0) and (aActiveSample.Screen.Height > 0) then
  begin
    aPixelSize := Min(SAMPLE_AREA_WIDTH / aActiveSample.Screen.Width,
      SAMPLE_AREA_HEIGHT / aActiveSample.Screen.Height);

    aWidth := aPixelSize * aActiveSample.Screen.Width;
```

```
        aHeight := aPixelSize * aActiveSample.Screen.Height;

        aRect.Left := TX(SAMPLE_AREA_LEFT + (SAMPLE_AREA_WIDTH - aWidth) / 2);
        aRect.Right := TX(SAMPLE_AREA_LEFT + (SAMPLE_AREA_WIDTH - aWidth) / 2 +
aWidth);
        aRect.Top := TY(SAMPLE_AREA_TOP + (SAMPLE_AREA_HEIGHT - aHeight) / 2);
        aRect.Bottom := TY(SAMPLE_AREA_TOP + (SAMPLE_AREA_HEIGHT - aHeight) / 2 +
aHeight);

        C.StretchDraw(aRect, aActiveSample.Screen);
      end;
    end;

    SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);
    C.Brush.Color := TColors.White;

    C.Polyline([TP(SAMPLE_AREA_LEFT, SAMPLE_AREA_TOP),
          TP(SAMPLE_AREA_LEFT + SAMPLE_AREA_WIDTH, SAMPLE_AREA_TOP),
          TP(SAMPLE_AREA_LEFT + SAMPLE_AREA_WIDTH, SAMPLE_AREA_TOP +
SAMPLE_AREA_HEIGHT),
          TP(SAMPLE_AREA_LEFT, SAMPLE_AREA_TOP + SAMPLE_AREA_HEIGHT),
          TP(SAMPLE_AREA_LEFT, SAMPLE_AREA_TOP)]);



    // окно изображения ------------------------------------------------------
    if (gNetwork.Screen.Width > 0) and ((gNetwork.Screen.Height > 0)) then
    begin
      aPixelSize := Min(SCREEN_AREA_WIDTH / gNetwork.Screen.Width,
        SCREEN_AREA_HEIGHT / gNetwork.Screen.Height);

      aWidth := aPixelSize * gNetwork.Screen.Width;
      aHeight := aPixelSize * gNetwork.Screen.Height;

      aRect.Left := TX(SCREEN_AREA_LEFT + (SCREEN_AREA_WIDTH - aWidth) / 2);
      aRect.Right := TX(SCREEN_AREA_LEFT + (SCREEN_AREA_WIDTH - aWidth) / 2 + aWidth);
      aRect.Top := TY(SCREEN_AREA_TOP + (SCREEN_AREA_HEIGHT - aHeight) / 2);
      aRect.Bottom := TY(SCREEN_AREA_TOP + (SCREEN_AREA_HEIGHT - aHeight) / 2 +
aHeight);
```

```
  C.StretchDraw(aRect, gNetwork.Screen);
 end;

 SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);
 C.Brush.Color := TColors.White;

 C.Polyline([TP(SCREEN_AREA_LEFT, SCREEN_AREA_TOP),
        TP(SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH, SCREEN_AREA_TOP),
        TP(SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH, SCREEN_AREA_TOP +
SCREEN_AREA_HEIGHT),
        TP(SCREEN_AREA_LEFT, SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT),
        TP(SCREEN_AREA_LEFT, SCREEN_AREA_TOP)]);

 // отображение нейронов ----------------------------------------------------
 // отобраются только слои с индексом больше 0
 // причем в обратном поряде (вверху последний слой)
 if gNetwork.Layers.Count > 1 then
 begin
  aHeight := (NEURON_AREA_HEIGHT - NEURON_MARGIN * (gNetwork.Layers.Count))
   / (gNetwork.Layers.Count - 1);

  for i := 1 to gNetwork.Layers.Count - 1 do
  begin
   if gNetwork.Layers[i].Neurons.Count = 0 then
    Continue;

   aWidth := (NEURON_AREA_WIDTH - NEURON_MARGIN *
(gNetwork.Layers[i].Neurons.Count + 1))
    / gNetwork.Layers[i].Neurons.Count;

   for j := 0 to gNetwork.Layers[i].Neurons.Count - 1 do
   begin
    if gNetwork.Layers[i].Neurons[j].Focused and (not gInSimulation) then
    begin
     C.Pen.Color := TColors.Red;
     C.Pen.Width := 2 * GetDefaultLineWidth(DEFAULT_SCALE_COEFFICIENT);
    end else
```

```
          begin
            C.Pen.Color := TColors.Black;
            C.Pen.Width := GetDefaultLineWidth(DEFAULT_SCALE_COEFFICIENT);
          end;

          // координаты нейрона
          aLeft := NEURON_AREA_LEFT + NEURON_MARGIN + j * (NEURON_MARGIN +
aWidth);
          aTop := NEURON_AREA_TOP + NEURON_MARGIN + (gNetwork.Layers.Count - i - 1)
            * (NEURON_MARGIN + aHeight);
          aRight := aLeft + aWidth;
          aBottom := aTop + aHeight;

          if gInSimulation then
            C.Brush.Color := gNetwork.Layers[i].Neurons[j].CurrentColor
          else
            C.Brush.Color := TColors.White;

          if gNetwork.Layers[i].CanFocus
            and PointInRegion(CursorX, CursorY, aLeft, aTop, aRight, aBottom) then
          begin
            C.Pen.Color := TColors.Blue;
            C.Pen.Width := 2 * GetDefaultLineWidth(DEFAULT_SCALE_COEFFICIENT);

            gNeuronUnderCursor := gNetwork.Layers[i].Neurons[j];

            if Assigned(gFocusedNeuronButton) then
              gFocusedNeuronButton.ActiveRect := ActiveRect(aLeft, aTop, aRight, aBottom);
          end;

          C.Rectangle(TX(aLeft), TY(aTop), TX(aRight), TY(aBottom));
        end;
      end;
    end;

    SetPenAndText(C, False, DEFAULT_SCALE_COEFFICIENT);

    C.Polyline([TP(NEURON_AREA_LEFT, NEURON_AREA_LEFT),
```

```
            TP(NEURON_AREA_LEFT + NEURON_AREA_WIDTH, NEURON_AREA_TOP),
            TP(NEURON_AREA_LEFT + NEURON_AREA_WIDTH, NEURON_AREA_TOP +
NEURON_AREA_HEIGHT),
            TP(NEURON_AREA_LEFT, NEURON_AREA_TOP + NEURON_AREA_HEIGHT),
            TP(NEURON_AREA_LEFT, NEURON_AREA_TOP)]);

        // кнопка "Загрузить изображение" ------------------------------------------
        SetPenAndText(C, gLoadScreenButton.Selected, DEFAULT_SCALE_COEFFICIENT);

        if gLoadScreenButton.Selected then
          C.Brush.Color := TColors.Yellow
        else
          C.Brush.Color := TColors.White;

        C.Rectangle(TX(SCREEN_AREA_LEFT),
            TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5),
            TX(SCREEN_AREA_LEFT + 3),
            TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5));

        case gLanguage of
          lgEng: S := '   Load    ';
          lgRus: S := ' Загрузить ';
          else   S := 'Завантажити';
        end;

        PaintText(C, S, gLoadScreenButton.Selected,
          SCREEN_AREA_LEFT + 0.1,
          SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.75,
          2.8, 0.5, haCenter, vaCenter);

        // кнопка "Очистить изображение" -------------------------------------------
        SetPenAndText(C, gClearScreenButton.Selected, DEFAULT_SCALE_COEFFICIENT);

        if gClearScreenButton.Selected then
          C.Brush.Color := TColors.Yellow
        else
          C.Brush.Color := TColors.White;
```

```
C.Rectangle(TX(SCREEN_AREA_LEFT + 3.5),
        TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5),
        TX(SCREEN_AREA_LEFT + 6.5),
        TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := '   Clear   ';
  lgRus: S := ' Очистить  ';
  else   S := ' Очистити  ';
end;

PaintText(C, S, gClearScreenButton.Selected,
  SCREEN_AREA_LEFT + 3.6,
  SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.75,
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Как образец" ------------------------------------------
SetPenAndText(C, gScreenInSampleButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gScreenInSampleButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SCREEN_AREA_LEFT + 7),
        TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.5),
        TX(SCREEN_AREA_LEFT + 10),
        TY(SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := ' As Sample ';
  lgRus: S := 'Как образец';
  else   S := ' Як зразок ';
end;

PaintText(C, S, gScreenInSampleButton.Selected,
  SCREEN_AREA_LEFT + 7.1,
  SCREEN_AREA_TOP + SCREEN_AREA_HEIGHT +  0.75,
```

```
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Загрузить образцы" ------------------------------------------
SetPenAndText(C, gLoadSamplesButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gLoadSamplesButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 0),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5),
          TX(SAMPLES_AREA_LEFT + 3),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := '    Load   ';
  lgRus: S := ' Загрузить ';
  else   S := 'Завантажити';
end;

PaintText(C, S, gLoadSamplesButton.Selected,
  SAMPLES_AREA_LEFT + 0 + 0.1,
  SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5 + 0.25,
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Сохранить образцы" ------------------------------------------
SetPenAndText(C, gSaveSamplesButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gSaveSamplesButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 3.5),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5),
          TX(SAMPLES_AREA_LEFT + 6.5),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5));
```

```
case gLanguage of
  lgEng: S := '    Save   ';
  lgRus: S := ' Сохранить ';
  else   S := '  Зберегти ';
end;

PaintText(C, S, gSaveSamplesButton.Selected,
  SAMPLES_AREA_LEFT + 3.5 + 0.1,
  SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5 + 0.25,
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Добавить образец" ------------------------------------------
SetPenAndText(C, gAddSampleButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gAddSampleButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 7),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5),
          TX(SAMPLES_AREA_LEFT + 10),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := '     Add   ';
  lgRus: S := '  Добавить ';
  else   S := '   Додати  ';
end;

PaintText(C, S, gAddSampleButton.Selected,
  SAMPLES_AREA_LEFT + 7 + 0.1,
  SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5 + 0.25,
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Удалить образец" -------------------------------------------
SetPenAndText(C, gDeleteSampleButton.Selected, DEFAULT_SCALE_COEFFICIENT);
```

```
if gDeleteSampleButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 10.5),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5),
          TX(SAMPLES_AREA_LEFT + 13.5),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := '   Delete  ';
  lgRus: S := '  Удалить  ';
  else   S := '  Видалити ';
end;

PaintText(C, S, gDeleteSampleButton.Selected,
  SAMPLES_AREA_LEFT + 10.5 + 0.1,
  SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5 + 0.25,
  2.8, 0.5, haCenter, vaCenter);

// кнопка "Очистить образец" ------------------------------------------
SetPenAndText(C, gClearSampleButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gClearSampleButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 14),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5),
          TX(SAMPLES_AREA_LEFT + 17),
          TY(SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 1.5));

case gLanguage of
  lgEng: S := '    Clear  ';
  lgRus: S := '  Очистить ';
```

```
  else   S := '  Очистити ';
 end;

 PaintText(C, S, gClearSampleButton.Selected,
  SAMPLES_AREA_LEFT + 14 + 0.1,
  SAMPLES_AREA_TOP + SAMPLES_AREA_HEIGHT + 0.5 + 0.25,
  2.8, 0.5, haCenter, vaCenter);

 // кнопка отобразить тренировочную выборку ---------------------------------

 SetPenAndText(C, gShowTrainingSamplesButton.Selected, DEFAULT_SCALE_COEFFICIENT);

 if gShowTrainingSamplesButton.Selected then
 begin
  C.Brush.Color := TColors.Yellow
 end else
 begin
  if gShowMode = SHOW_TRAINING_SAMPLES then
   C.Brush.Color := TColors.Lime
  else
   C.Brush.Color := TColors.White;
 end;

 C.Rectangle(TX(SAMPLES_AREA_LEFT),
        TY(SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN),
        TX(SAMPLES_AREA_LEFT + 0.5 * SAMPLES_AREA_WIDTH - 0.5 *
DEFAULT_MAGRIN),
        TY(SAMPLES_AREA_TOP - DEFAULT_MAGRIN));

  case gLanguage of
   lgEng: S := 'Training set';
   lgRus: S := 'Учебный набір';
   else   S := 'Навчальний набір';
  end;

 PaintText(C, S, gShowTrainingSamplesButton.Selected,
  SAMPLES_AREA_LEFT + 0.1,
  SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN + 0.25,
```

```
  0.5 * SAMPLES_AREA_WIDTH - 0.5 * DEFAULT_MAGRIN - 0.2, 0.5, haCenter, vaCenter);

// кнопка отобразить тестовую выборку ----------------------------------------

SetPenAndText(C, gShowTestSamplesButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gShowTestSamplesButton.Selected then
begin
  C.Brush.Color := TColors.Yellow
end else
begin
  if gShowMode = SHOW_TEST_SAMPLES then
    C.Brush.Color := TColors.Lime
  else
    C.Brush.Color := TColors.White;
end;

C.Rectangle(TX(SAMPLES_AREA_LEFT + 0.5 * SAMPLES_AREA_WIDTH + 0.5 *
DEFAULT_MAGRIN),
        TY(SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN),
        TX(SAMPLES_AREA_LEFT + SAMPLES_AREA_WIDTH),
        TY(SAMPLES_AREA_TOP - DEFAULT_MAGRIN));

 case gLanguage of
    lgEng: S := 'Test set';
    lgRus: S := 'Тестовый набор';
    else   S := 'Тестовий набір';
 end;

PaintText(C, S, gShowTestSamplesButton.Selected,
  SAMPLES_AREA_LEFT + 0.5 * SAMPLES_AREA_WIDTH + 0.5 * DEFAULT_MAGRIN +
0.1,
  SAMPLES_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN + 0.25,
  0.5 * SAMPLES_AREA_WIDTH - 0.5 * DEFAULT_MAGRIN - 0.2, 0.5, haCenter, vaCenter);


// кнопка "Режим работы" ----------------------------------------------------
```

```
SetPenAndText(C, gShowModeButton.Selected, DEFAULT_SCALE_COEFFICIENT);

if gShowModeButton.Selected then
  C.Brush.Color := TColors.Yellow
else
  C.Brush.Color := TColors.White;

C.Rectangle(TX(SCREEN_AREA_LEFT),
            TY(SCREEN_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN),
            TX(SCREEN_AREA_LEFT + SCREEN_AREA_WIDTH),
            TY(SCREEN_AREA_TOP - DEFAULT_MAGRIN));

 case gRunMode of
   RUN_MODE_TRAINING:
     case gLanguage of
       lgEng: S := 'Run mode - "Training"';
       lgRus: S := 'Режим работы - "Обучение"';
       else   S := 'Режим роботи - "Навчання"';
     end;

   RUN_MODE_TESTING:
     case gLanguage of
       lgEng: S := 'Run mode - "Test"';
       lgRus: S := 'Режим работы - "Тестирование"';
       else   S := 'Режим роботи - "Тестування"';
     end;

   else
     case gLanguage of
       lgEng: S := 'Run mode - "Recognition"';
       lgRus: S := 'Режим работы - "Распознавание"';
       else   S := 'Режим роботи - "Розпізнавання"';
     end;
 end;

PaintText(C, S, gShowModeButton.Selected,
  SCREEN_AREA_LEFT + 0.1,
  SCREEN_AREA_TOP - BUTTON_HEIGHT - DEFAULT_MAGRIN + 0.25,
```

```
    SCREEN_AREA_WIDTH - 0.2,
    0.5, haCenter, vaCenter);
end;

// ############################################################################

procedure PaintSchem(const SchemIndex: Integer; const InSimulation: Boolean); stdcall;
var
  C: TCanvas;
begin
  gInSimulation := InSimulation;

  if not Assigned(gGraphicParams)
    then Exit;

  C := TCanvas.Create;

  C.Brush.Color := clWhite;
  C.Brush.Style := bsSolid;

  C.Pen.Color := clBlack;
  C.Pen.Width := 2;

  C.Font.Name := 'Arial';
  C.Font.Style := [];

  try
    C.Handle := gGraphicParams.Handle;

    case SchemIndex  of
      0: PaintCircuit(C, InSimulation);
    end;
  finally
    C.Handle := 0;
    C.Free;
  end;
end;
```

```
// ##############################################################################
//                    Математическая модель
// ##############################################################################

function GetLogFileName: String;
var
  Y, M, D, HH, MM, SS, MSS: Word;
begin
  case gRunMode of
    RUN_MODE_TRAINING: Result := 'Training_';
    RUN_MODE_TESTING: Result := 'Testing_';
    else Result := '';
  end;

  if Result = '' then
    Exit;

  DecodeDateTime(Now, Y, M, D, HH, MM, SS, mSS);

  Result := Result + IntToStr(Y);

  if M < 10
    then Result := Result + '0' + IntToStr(M)
    else Result := Result + IntToStr(M);

  if D < 10
    then Result := Result + '0' + IntToStr(D)
    else Result := Result + IntToStr(D);

  if HH < 10
    then Result := Result + '_0' + IntToStr(HH)
    else Result := Result + '_' + IntToStr(HH);

  if MM < 10
    then Result := Result + '0' + IntToStr(MM)
    else Result := Result + IntToStr(MM);

  if SS < 10
```

```
    then Result := Result + '0' + IntToStr(SS)
    else Result := Result + IntToStr(SS);

  Result := Result + '.csv';
end;

// ###########################################################################

procedure ResetWeights;
var
  i, j, k: Integer;
begin
  for i := 0 to gNetwork.Layers.Count - 1 do
    for j := 0 to gNetwork.Layers[i].Neurons.Count - 1 do
      for k  := 0 to gNetwork.Layers[i].Neurons[j].InputCount - 1 do
        if i = 0 then
          gNetwork.Layers[i].Neurons[j].Weights[k] := 1
        else
          gNetwork.Layers[i].Neurons[j].Weights[k] := (Random(201) - 100) / 100;
end;

// ###########################################################################

procedure ResetSchem(); stdcall;
var
  S: String;
  i: Integer;
begin
  gNeuronIndex := 0;
  gSampleIndex := 0;

  gEpocheNumber := 1;
  gIterationNumber := 0;

  gTotalEpocheError := 0;
  gTotalEpocheErrorSum := 0;

  gLastCalcTime := Now;
```

```
  if gRunMode = RUN_MODE_TRAINING then
  begin
    gShowMode := SHOW_TRAINING_SAMPLES;
    ResetWeights;
  end;

  if gRunMode = RUN_MODE_TESTING then
    gShowMode := SHOW_TEST_SAMPLES;

  gLogFileName := GetLogFileName;

  if gLogFileName <> '' then
  begin
    S := 'EpocheNumber' + LOG_FILE_SEPARATOR
      + 'IterationNumder' + LOG_FILE_SEPARATOR
      + 'NeuronIndex' + LOG_FILE_SEPARATOR
      + 'SampleIndex' + LOG_FILE_SEPARATOR
      + 'NetworkError' + LOG_FILE_SEPARATOR;

    if Assigned(gNetwork.OutputLayer) then
      for i := 0 to gNetwork.OutputLayer.Neurons.Count - 1 do
        S := S + 'Neuron_' + IntToStr(i) + LOG_FILE_SEPARATOR;

    AddToLog(gLogFileName, S);
  end;

end;

// ############################################################################

function MyFloatToStr(A: Double): String;
begin
  Result := FloatToStr(Round(A * 10000) / 10000);
end;

// ############################################################################
```

```
function GetLogMessage(aNetworkError: Double): String;
var
  i: Integer;
begin
  Result := IntToStr(gEpocheNumber) + LOG_FILE_SEPARATOR
        + IntToStr(gIterationNumber) + LOG_FILE_SEPARATOR
        + IntToStr(gNeuronIndex) + LOG_FILE_SEPARATOR
        + IntToStr(gSampleIndex) + LOG_FILE_SEPARATOR
        + MyFloatToStr(aNetworkError) + LOG_FILE_SEPARATOR;

  if Assigned(gNetwork.InputLayer) then
    for i := 0 to gNetwork.Layers[1].Neurons.Count - 1 do
      Result := Result + MyFloatToStr(gNetwork.Layers[1].Neurons[i].Output)
        + LOG_FILE_SEPARATOR;

end;

// ############################################################################

function DifSigma(X: Single): Single; //производная (дифференциал) сигмоиды
begin
  Result := X * (1 - X);
end;

procedure ChangeNetworkWeights(IndexOfActiveNueron: Integer);
var
  i, j: Integer;
  Etalon, S: Double;
  ActiveNeuron: TNeuron;
begin
  for i := 0 to gNetwork.Layers[3].Neurons.Count - 1 do
  begin
    ActiveNeuron := gNetwork.Layers[3].Neurons[i];

    if i = IndexOfActiveNueron then
      Etalon := NEURON_ACTIVE_VALUE
    else
      Etalon := NEURON_UNACTIVE_VALUE;
```

```
  ActiveNeuron.Error := (Etalon - ActiveNeuron.Output)
    * DifSigma(ActiveNeuron.Output);
end;

for i := 0 to gNetwork.Layers[2].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[2].Neurons[i];

  S := 0;

  for j := 0 to gNetwork.Layers[3].Neurons.Count - 1 do
   S := S + gNetwork.Layers[3].Neurons[j].Weights[i] * gNetwork.Layers[3].Neurons[j].Error;

  ActiveNeuron.Error := S * DifSigma(ActiveNeuron.Output);
end;

for i := 0 to gNetwork.Layers[1].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[1].Neurons[i];

  S := 0;

  for j := 0 to gNetwork.Layers[2].Neurons.Count - 1 do
    S := S + gNetwork.Layers[2].Neurons[j].Weights[i] * gNetwork.Layers[2].Neurons[j].Error;

  ActiveNeuron.Error := S * DifSigma(ActiveNeuron.Output);
end;

for i := 0 to gNetwork.Layers[1].Neurons.Count - 1 do
begin
  ActiveNeuron := gNetwork.Layers[1].Neurons[i];

  for j := 0 to ActiveNeuron.WeightCount - 1 do
    ActiveNeuron.Weights[j] := ActiveNeuron.Weights[j]
      + 0.01 * ActiveNeuron.Inputs[j]^ * ActiveNeuron.Error;
end;
```

```
  for i := 0 to gNetwork.Layers[2].Neurons.Count - 1 do
  begin
    ActiveNeuron := gNetwork.Layers[2].Neurons[i];

    for j := 0 to ActiveNeuron.WeightCount - 1 do
      ActiveNeuron.Weights[j] := ActiveNeuron.Weights[j]
        + 0.01 * ActiveNeuron.Inputs[j]^ * ActiveNeuron.Error;
  end;

   for i := 0 to gNetwork.Layers[3].Neurons.Count - 1 do
  begin
    ActiveNeuron := gNetwork.Layers[3].Neurons[i];

    for j := 0 to ActiveNeuron.WeightCount - 1 do
      ActiveNeuron.Weights[j] := ActiveNeuron.Weights[j]
        + 0.01 * ActiveNeuron.Inputs[j]^ * ActiveNeuron.Error;
  end;

end;

// ############################################################################

// расчет средней ошибки нейронной сети
function CalculateNetworkError(IndexOfActiveNeuron: Integer): Double;
var
  i: Integer;
  Etalon: Double;
begin
  Result := 0;

  for i := 0 to gNetwork.OutputLayer.Neurons.Count - 1 do
  begin
    if i = IndexOfActiveNeuron then
      Etalon := NEURON_ACTIVE_VALUE
    else
      Etalon := NEURON_UNACTIVE_VALUE;

    Result := Result + Sqr(gNetwork.OutputLayer.Neurons[i].Output - Etalon);
```

```
  end;

  if gNetwork.OutputLayer.Neurons.Count > 0 then
    Result := Result / gNetwork.OutputLayer.Neurons.Count;
end;


// ############################################################################

procedure NetworkToLog(FN: String);
var
  F: TextFile;
  i, j, k: Integer;
  S: String;
begin
  AssignFile(F, FN);
  try
    try
      Rewrite(F);

      for i := 1 to gNetwork.Layers.Count - 1 do
      begin
        for j := 0 to gNetwork.Layers[i].Neurons.Count - 1 do
        begin
          Writeln(F, 'Нейрон ' + IntToStr(j));

          for k  := 0 to gNetwork.Layers[i].Neurons[j].InputCount - 1 do
          begin
            S := 'Слой ' + IntToStr(i);
            S := S + ' Нейрон ' + IntToStr(j);
            S := S + ' Output = ' + FloatToStr(gNetwork.Layers[i].Neurons[j].Output);
            S := S + ' W_' + IntToStr(k) + ' = ' + FloatToStr(gNetwork.Layers[i].Neurons[j].Weights[k]);
            S := S + ' Input_' + IntToStr(k) + ' = ' + FloatToStr(gNetwork.Layers[i].Neurons[j].Inputs[k]^);
            Writeln(F, S);
          end;

          Writeln(F, ' ################################################');
        end;
```

```
      end;

    except

    end;
  finally
    CloseFile(F);
  end;


  sleep(10);
end;

procedure LearnNN;
var
  O: TNeuronLayer;
  NetworkError: Double;
  i, j: Integer;
  ActiveNeuron: TNeuron;
begin
  gNeuronIndex := 0;
  gSampleIndex := 0;

  gEpocheNumber := 1;
  gIterationNumber := 0;

  gTotalEpocheError := 0;
  gTotalEpocheErrorSum := 0;

  ResetWeights;

  while gNeuronIndex >= 0 do
  begin

    O := gNetwork.OutputLayer;

    if gNeuronIndex >= O.Neurons.Count then
      gNeuronIndex := -1;
```

```
if gNeuronIndex > -1 then
begin
 if gSampleIndex < O.Neurons[gNeuronIndex].TrainingSamples.Count then
 begin
  Inc(gIterationNumber);

  // копирование изображения
  gNetwork.CopyToScreen(O.Neurons[gNeuronIndex].TrainingSamples[gSampleIndex].Screen);

  // расчет реакции сети
  gNetwork.Calculate;

  // изменение весовых коеффциентов
  ChangeNetworkWeights(gNeuronIndex);

  // расчет ошибки сети
  NetworkError := CalculateNetworkError(gNeuronIndex);
  gTotalEpocheErrorSum := gTotalEpocheErrorSum + NetworkError;
  gTotalEpocheError := gTotalEpocheErrorSum / gIterationNumber;

  Inc(gSampleIndex);
 end else
 begin
  repeat
   gSampleIndex := -1;
   Inc(gNeuronIndex);

   if gNeuronIndex < O.Neurons.Count then
   begin
    if O.Neurons[gNeuronIndex].TrainingSamples.Count > 0 then
     gSampleIndex := 0;
   end else
   begin
    gNeuronIndex := -1;
   end;
  until (gNeuronIndex < 0) or (gSampleIndex >= 0);
 end;
```

```
    end;

    // конец епохи
    if gNeuronIndex = -1 then
    begin
      if gEpocheNumber < Round(gMaxEpocheCount^) then
      begin
        Inc(gEpocheNumber);

        gIterationNumber := 0;
        gTotalEpocheErrorSum := 0;

        gNeuronIndex := 0;
        gSampleIndex := 0;
      end;
    end;
  end;

end;

// ############################################################################

procedure CalcNextIteration(SchemIndex: Integer; var CurTime: Double;
  var ResetSimulation, StopSumilation, UpdateSchem: Boolean); stdcall;
var
  O: TNeuronLayer;
  NetworkError: Double;
  i, j: Integer;
  ActiveNeuron: TNeuron;
begin
  // проверка правильности настройки сети ----------------------------------
  if not Assigned(gNetwork.OutputLayer) then
  begin
    StopSumilation := True;
    Exit;
  end;

  if gNetwork.OutputLayer.Neurons.Count = 0 then
```

```
begin
  StopSumilation := True;
  Exit;
end;

// время ---------------------------------------------------------------
CurTime := CurTime + MilliSecondsBetween(Now, gLastCalcTime) * 0.001;
gLastCalcTime := Now;

// режим распознавания -------------------------------------------------
if gRunMode = RUN_MODE_RECOGNITION then
  gNetwork.Calculate;

// режим тестирования --------------------------------------------------
if gRunMode = RUN_MODE_TESTING then
begin
  O := gNetwork.OutputLayer;

  if gNeuronIndex >= O.Neurons.Count then
    gNeuronIndex := -1;

  if gNeuronIndex > -1 then
  begin
    if gSampleIndex < O.Neurons[gNeuronIndex].TestSamples.Count then
    begin
      Inc(gIterationNumber);

      // копирование изображения из образца в екран
      gNetwork.CopyToScreen(O.Neurons[gNeuronIndex].TestSamples[gSampleIndex].Screen);

      // расчет реакции сети
      gNetwork.Calculate;

      // расчет ошибки сети
      NetworkError := CalculateNetworkError(gNeuronIndex);
      gTotalEpocheErrorSum := gTotalEpocheErrorSum + NetworkError;
      gTotalEpocheError := gTotalEpocheErrorSum / gIterationNumber;
```

```
      AddToLog(gLogFileName, GetLogMessage(NetworkError));

      Inc(gSampleIndex);
    end else
    begin
      repeat
        gSampleIndex := -1;
        Inc(gNeuronIndex);

        if gNeuronIndex < O.Neurons.Count then
        begin
          if O.Neurons[gNeuronIndex].TestSamples.Count > 0 then
            gSampleIndex := 0;
        end else
        begin
          gNeuronIndex := -1;
        end;
      until (gNeuronIndex < 0) or (gSampleIndex >= 0);
    end;
  end;

  StopSumilation := gNeuronIndex < 0;
end;

// режим обучения ------------------------------------------------------------
if gRunMode = RUN_MODE_TRAINING then
begin
  O := gNetwork.OutputLayer;

  if gNeuronIndex >= O.Neurons.Count then
    gNeuronIndex := -1;

  if gNeuronIndex > -1 then
  begin
    if gSampleIndex < O.Neurons[gNeuronIndex].TrainingSamples.Count then
    begin
      Inc(gIterationNumber);
```

```
    // копирование изображения
    gNetwork.CopyToScreen(O.Neurons[gNeuronIndex].TrainingSamples[gSampleIndex].Screen);

    // расчет реакции сети
    gNetwork.Calculate;

    // изменение весовых коеффциентов
    ChangeNetworkWeights(gNeuronIndex);

    // расчет ошибки сети
    NetworkError := CalculateNetworkError(gNeuronIndex);
    gTotalEpocheErrorSum := gTotalEpocheErrorSum + NetworkError;
    gTotalEpocheError := gTotalEpocheErrorSum / gIterationNumber;

    Inc(gSampleIndex);
  end else
  begin
    repeat
      gSampleIndex := -1;
      Inc(gNeuronIndex);

      if gNeuronIndex < O.Neurons.Count then
      begin
        if O.Neurons[gNeuronIndex].TrainingSamples.Count > 0 then
          gSampleIndex := 0;
      end else
      begin
        gNeuronIndex := -1;
      end;
    until (gNeuronIndex < 0) or (gSampleIndex >= 0);
  end;
end;

// конец епохи
if gNeuronIndex = -1 then
begin
  if gEpocheNumber < Round(gMaxEpocheCount^) then
  begin
```

```
      Inc(gEpocheNumber);

      gIterationNumber := 0;
      gTotalEpocheErrorSum := 0;

      gNeuronIndex := 0;
      gSampleIndex := 0;
    end;
  end;

  StopSumilation := gNeuronIndex < 0;
 end;

 UpdateSchem := True;
end;

// ###########################################################################

procedure FreeLayout(); stdcall;
begin
 // удаление компонентов
 FreeAndNil(gNetwork);
end;

// ###########################################################################

exports
 SetLayoutGlobals,
 GetLayoutInfo,
 SetSchemInfo,
 SetComponentInfo,
 SetComponentParamInfo,
 PaintSchem,
 ResetSchem,
 CalcNextIteration,
 FreeLayout;

begin
```

```
end.
```

Лістинг файлу uNeuronNetwork

```
unit uNeuronNetwork;

interface

uses
  Winapi.Windows,
  System.Types,
  System.SysUtils,
  System.Classes,
  System.Math,
  System.UITypes,
  VCL.Graphics,
  VCL.Dialogs,
  VCL.Forms,
  VCL.Imaging.jpeg,
  VCL.Imaging.GIFImg,
  VCL.Imaging.pngimage,
  uDLLTools;

type
  TNeuronInputs = array of PDouble;
  TNeuronWeights = array of Double;
  TActivationFunctionParams = array of Double;

  TActivationFunction = function(S: Double; Params: TActivationFunctionParams): Double;

  TNeuronSample = class(TObject)
  private
    FScreen: TBitmap;
    FFocused: Boolean;
  public
    property Screen: TBitmap read FScreen;
    property Focused: Boolean read FFocused write FFocused;

    constructor Create;
    destructor Destroy; override;
```

```
    procedure Assing(aScreen: TBitmap);
    procedure Clear;
    procedure SetDimentions(aWidth, aHeight: Integer);

    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;
  end;

// ############################################################################

  TNeuronSampleList = class(TList)
  private
    FFileName: String;
    FPageNumber: Integer;

    function Get(Index: Integer): TNeuronSample;
    procedure Put(Index: Integer; const Value: TNeuronSample);
    function GetCount: Integer;
    function GetFocusedSample: TNeuronSample;
    procedure SetFocusedSample(const Value: TNeuronSample);
  public
    property Items[Index: Integer]: TNeuronSample read Get write Put; default;
    property Count: Integer read GetCount;
    property FileName: String read FFileName write FFileName;
    property FocusedSample: TNeuronSample read GetFocusedSample
      write SetFocusedSample;
    property PageNumber: Integer read FPageNumber write FPageNumber;

    function First: TNeuronSample;
    function Last: TNeuronSample;

    procedure DeleteAll;
    procedure DeleteSample(aSample: TNeuronSample);

    function AddNewSample: TNeuronSample;

    procedure SaveToStream(F: TFileStream);
```

```
    function LoadFromStream(F: TFileStream): Boolean;

    procedure SaveToFile(ShowDialog: Boolean);
    function LoadFromFile(ShowDialog: Boolean): Boolean;

    procedure FocuseFirst;
    function MaxWidth: Integer;
    function MaxHeight: Integer;

    constructor Create;
  end;

// ###########################################################################

  TNeuron = class(TObject)
  private
    FTrainingSamples: TNeuronSampleList;
    FTestSamples: TNeuronSampleList;

    FColor: TColor;
    FInputs: TNeuronInputs;
    FWeights: TNeuronWeights;

    FActivationFunction: TActivationFunction;
    FActivationFunctionParams: TActivationFunctionParams;

    FFocused: Boolean;
    FError: Double;

    function GetInputCount: Integer;
    procedure SetInputCount(const Value: Integer);
    function GetInput(Index: Integer): PDouble;
    procedure SetInput(Index: Integer; const Value: PDouble);
    function GetWeight(Index: Integer): Double;
    procedure SetWeight(Index: Integer; const Value: Double);
    function GetActivationFunctionParams(Index: Integer): Double;
    procedure SetActivationFunctionParams(Index: Integer; const Value: Double);
    function GetActivationFunctionParamCount: Integer;
```

```
    procedure SetActivationFunctionParamCount(const Value: Integer);
    function GetWeightCount: Integer;
  protected
    function GetColor: TColor;
  public
    Output: Double;
    property Error: Double read FError write FError;

    property TrainingSamples: TNeuronSampleList read FTrainingSamples;
    property TestSamples: TNeuronSampleList read FTestSamples;

    property Color: TColor read FColor write FColor;
    property CurrentColor: TColor read GetColor;
    property Focused: Boolean read FFocused write FFocused;

    property InputCount: Integer read GetInputCount write SetInputCount;
    property Inputs[Index: Integer]: PDouble read GetInput write SetInput;
    property Weights[Index: Integer]: Double read GetWeight write SetWeight;
    property WeightCount: Integer read GetWeightCount;

    property ActivationFunctionParamCount: Integer
      read GetActivationFunctionParamCount write SetActivationFunctionParamCount;
    property ActivationFunction: TActivationFunction read FActivationFunction
      write FActivationFunction;
    property ActivationFunctionParams[Index: Integer]: Double
      read GetActivationFunctionParams write SetActivationFunctionParams;

    constructor Create;
    destructor Destroy; override;

    procedure Calculate(aToLog: Boolean);

    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;
  end;

// ###########################################################################
```

```
 TNeuronList = class(TList)
 private
   function Get(Index: Integer): TNeuron;
   procedure Put(Index: Integer; const Value: TNeuron);
   function GetCount: Integer;
   procedure SetCount(const Value: Integer);
   function GetFocused: TNeuron;
   procedure SetFocused(const Value: TNeuron);
 public
   property Items[Index: Integer]: TNeuron read Get write Put; default;
   property Count: Integer read GetCount write SetCount;
   property Focused: TNeuron read GetFocused write SetFocused;

   function First: TNeuron;
   function Last: TNeuron;

   procedure FocuseFirst;

   procedure SaveToStream(F: TFileStream);
   function LoadFromStream(F: TFileStream): Boolean;
 end;

// ###########################################################################

 TNeuronLayer = class(TObject)
 private
   FNeurons: TNeuronList;
   FCanFocus: Boolean;

   function GetFocusedNeuron: TNeuron;
 public
   property Neurons: TNeuronList read FNeurons;
   property FocusedNeuron: TNeuron read GetFocusedNeuron;
   property CanFocus: Boolean read FCanFocus write FCanFocus;

   constructor Create;
   destructor Destroy; override;
```

```
    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;
  end;

// ############################################################################

  TNeuronLayerList = class(TList)
  private
    function Get(Index: Integer): TNeuronLayer;
    procedure Put(Index: Integer; const Value: TNeuronLayer);
    function GetCount: Integer;
    procedure SetCount(const Value: Integer);
  public
    property Items[Index: Integer]: TNeuronLayer read Get write Put; default;
    property Count: Integer read GetCount write SetCount;

    function Last: TNeuronLayer;
    function First: TNeuronLayer;

    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;
  end;

// ############################################################################

  TNeuronNetwork = class(TObject)
  private
    FLayers: TNeuronLayerList;
    FScreen: TBitmap;

    FScreenImageFileName: string;
    FFileName: String;

    function GetOutputLayer: TNeuronLayer;
    function GetInputLayer: TNeuronLayer;
  public
    // выходы екрана
    ScreenOutputs: array of Double;
```

```
    BiasNeuroun: Double;

    property Layers: TNeuronLayerList read FLayers;
    property OutputLayer: TNeuronLayer read GetOutputLayer;
    property InputLayer:  TNeuronLayer read GetInputLayer;
    property FileName: String read FFileName write FFileName;

    property Screen: TBitmap read FScreen;

    constructor Create;
    destructor Destroy; override;

    procedure ClearScreen;
    procedure CopyToScreen(aSource: TBitmap);

    procedure SetScreenDimentions(aWidth, aHeight: Integer);
    procedure LoadScreenFromFile;

    procedure SetScreenOutputs;
    procedure Calculate;

    procedure SaveToStream(F: TFileStream);
    function LoadFromStream(F: TFileStream): Boolean;

    procedure SaveToFile(ShowDialog: Boolean);
    function LoadFromFile(ShowDialog: Boolean): Boolean;
  end;

// ###########################################################################

  // функции активации нейронов
  function SigmaActivationFunction(X: Double; Params: TActivationFunctionParams): Double;
  function LinearActivationFunction(S: Double; Params: TActivationFunctionParams): Double;
  function StepperActivationFunction(S: Double; Params: TActivationFunctionParams): Double;
  function PseudoRectifieActivationFunction(S: Double; Params: TActivationFunctionParams):
Double;

implementation
```

```
const
  MAX_BYTE: Byte = 255;
  SAMPLE_SET_FILE_EXTENTION = '.smpl';
  NETWORK_SETTING_FILE_EXTENTION = '.ann';

// ###########################################################################

procedure AddToLog(aFileName, S: String);
var
  F: TextFile;
begin
//  if Round(gCreateLog^) <> 1 then
//    Exit;

  if S = '' then
    Exit;

  AssignFile(F, aFileName);
  try
    try
      if FileExists(aFileName) then Append(F)
        else Rewrite(F);

      Writeln(F, S);
    except

    end;
  finally
    CloseFile(F);
  end;
end;


function RandomColor: TColor;
var
  R, G, B: Byte;
begin
```

```
  R := RandomRange(0, MAX_BYTE);
  G := RandomRange(0, MAX_BYTE);
  B := RandomRange(0, MAX_BYTE);

  Result := RGB(R, G, B);
end;

// ##############################################################################
//              Некоторые стандартные функции активации нейронов
// ##############################################################################

// Сігмоїда
function SigmaActivationFunction(X: Double;
  Params: TActivationFunctionParams): Double;
begin
  Result := 1 / (1 + Exp(-X));
end;

// ##############################################################################

// линейная
function LinearActivationFunction(S: Double;
  Params: TActivationFunctionParams): Double;
begin
  Result := S;
end;

// ##############################################################################

// пороговая
function StepperActivationFunction(S: Double;
  Params: TActivationFunctionParams): Double;
begin
  Result := 0;

  if Length(Params) > 0 then
    if S > Params[0]  then
      Result := 1;
```

```
end;

// ############################################################################

// псевдовыпрямленая
function PseudoRectifieActivationFunction(S: Double;
  Params: TActivationFunctionParams): Double;
begin
  if S >= 0 then
    Result := S
  else
    Result := 0;
end;

// ############################################################################

{ TNeuornNetwork }

constructor TNeuronNetwork.Create;
begin
  BiasNeuroun := 1;
  ScreenOutputs := nil;

  FLayers := TNeuronLayerList.Create;

  FScreen := TBitmap.Create;
  FScreen.PixelFormat := pf24bit;

  FFileName := 'Default' + NETWORK_SETTING_FILE_EXTENTION;
end;

destructor TNeuronNetwork.Destroy;
begin
  FScreen.Free;

  FLayers.Count := 0;
  FLayers.Free;
```

```
  ScreenOutputs := nil;

  inherited;
end;

function TNeuronNetwork.GetInputLayer: TNeuronLayer;
begin
  if FLayers.Count > 0 then
    Result := FLayers.First
  else
    Result := nil;
end;

function TNeuronNetwork.GetOutputLayer: TNeuronLayer;
begin
  if FLayers.Count > 0 then
    Result := FLayers.Last
  else
    Result := nil;
end;

function TNeuronNetwork.LoadFromFile(ShowDialog: Boolean): Boolean;
var
  S: String;
  D: TOpenDialog;
  F: TFileStream;
  CanShowMessage: Boolean;
begin
  Result := True;
  CanShowMessage := ShowDialog;

  if ShowDialog then
  begin
    D := TOpenDialog.Create(nil);
    try
      if FFileName <> '' then
        D.FileName := FFileName;
```

```
    case gLanguage of
      lgEng: S := 'Neural network settings';
      lgRus: S := 'Настройки нейросети';
      else S := 'Налаштування нейромережі';
    end;

    D.Filter := S + ' (*' + NETWORK_SETTING_FILE_EXTENTION + ') |*'
      + NETWORK_SETTING_FILE_EXTENTION;

    if D.Execute(Application.Handle) then
    begin
      FFileName := D.FileName;

      if ExtractFileExt(FFileName) = '' then
        FFileName := FFileName + NETWORK_SETTING_FILE_EXTENTION;
    end else
    begin
      Result := False;
      CanShowMessage := False;
    end;
  finally
    D.Free;
  end;
end;

if Result then
  Result := FileExists(FFileName);

if Result then
begin
  F := TFileStream.Create(FFileName, fmOpenRead);
  try
    Result := LoadFromStream(F);
  finally
    F.Free;
  end;
end;
```

```
  if CanShowMessage and (not Result) then
  begin
    case gLanguage of
      lgEng: S := 'The file is missing or damaged';
      lgRus: S := 'Файл отсуствует или поврежден';
      else S := 'Файл вітсутній або пошкоджений';
    end;

    ShowMessage(S);
  end;
end;

function TNeuronNetwork.LoadFromStream(F: TFileStream): Boolean;
begin
  Result := Assigned(F);

  if Result then
    Result := FLayers.LoadFromStream(F);
end;

procedure TNeuronNetwork.LoadScreenFromFile;
var
  D: TOpenDialog;
  S: String;
  aGraphic: TGraphic;
  aRect: TRect;
begin
  aGraphic := nil;

  D := TOpenDialog.Create(nil);
  try
    if FileExists(FScreenImageFileName) then
      D.FileName := FScreenImageFileName;

    case gLanguage of
      lgEng: S := 'Image files';
      lgRus: S := 'Файлы изображений';
      else S := 'Файли зображень';
```

```
end;

D.Filter := S + ' (*.bmp; *.jpg; *.jpeg; *.png) |*.bmp; *.jpg; *.jpeg; *.png';

if D.Execute(Application.Handle) then
begin
 FScreenImageFileName := D.FileName;

 if FileExists(FScreenImageFileName) then
 begin
  try
   S := AnsiUpperCase(ExtractFileExt(FScreenImageFileName));

   if S = '.BMP' then
    aGraphic := TBitmap.Create;

   if S = '.JPG' then
    aGraphic := TJPEGImage.Create;

   if S = '.JPEG' then
    aGraphic := TJPEGImage.Create;

   if S = '.PNG' then
    aGraphic := TPngImage.Create;

   if Assigned(aGraphic) then
   begin
    aGraphic.LoadFromFile(FScreenImageFileName);

    aRect.Left := 0;
    aRect.Top := 0;
    aRect.Right := FScreen.Width;
    aRect.Bottom := FScreen.Height;

    FScreen.Canvas.StretchDraw(aRect, aGraphic);
   end;
  except
   on E: Exception do
```

```
          begin
            case gLanguage of
              lgEng: S := 'Error opening file:' + #13 + E.Message;
              lgRus: S := 'Ошибка при открытии файла:' + #13 + E.Message;
              else S := 'Помилка під час відкриття файлу:' + #13 + E.Message;
            end;
           ShowMessage(S);
          end;
        end;
      end;
    end;
  finally
    D.Free;

    if Assigned(aGraphic) then
      aGraphic.Free;
  end;
end;

procedure TNeuronNetwork.Calculate;
var
  i, j: Integer;
begin
  SetScreenOutputs;
  for i := 0 to FLayers.Count - 1 do
  begin
    for j := 0 to FLayers[i].Neurons.Count - 1 do
    begin
      FLayers[i].Neurons[j].Calculate(i > 0);
     end;
  end;
end;

procedure TNeuronNetwork.ClearScreen;
var
  X, Y: Integer;
begin
  for Y := 0 to FScreen.Height - 1 do
```

```
    for X := 0 to FScreen.Width - 1 do
      FScreen.Canvas.Pixels[X, Y] := RandomColor;
end;

procedure TNeuronNetwork.CopyToScreen(aSource: TBitmap);
var
  aRect: TRect;
begin
  if not Assigned(aSource) then
    Exit;

  aRect.Left := 0;
  aRect.Top := 0;
  aRect.Right := FScreen.Width;
  aRect.Bottom := FScreen.Height;

  FScreen.Canvas.StretchDraw(aRect, aSource);
end;

procedure TNeuronNetwork.SaveToFile(ShowDialog: Boolean);
var
  S: String;
  D: TSaveDialog;
  F: TFileStream;
  CanSaveNetwork: Boolean;
begin
  CanSaveNetwork := True;

  if ShowDialog then
  begin
    D := TSaveDialog.Create(nil);
    try
      D.Options := D.Options + [ofOverwritePrompt];

      if FileExists(FFileName) then
        D.FileName := FFileName;

      case gLanguage of
```

```
      lgEng: S := 'Neural network settings';
      lgRus: S := 'Настройки нейросети';
      else S := 'Налаштування нейромережі';
    end;

    D.Filter := S + ' (*' + NETWORK_SETTING_FILE_EXTENTION + ') |*'
      + NETWORK_SETTING_FILE_EXTENTION;

    if D.Execute(Application.Handle) then
    begin
      FFileName := D.FileName;

      if ExtractFileExt(FFileName) = '' then
        FFileName := FFileName + NETWORK_SETTING_FILE_EXTENTION;
    end else
      CanSaveNetwork := False;
  finally
    D.Free;
  end;
 end;

 if CanSaveNetwork then
 begin
  F := TFileStream.Create(FFileName, fmCreate);
  try
   SaveToStream(F);
  finally
   F.Free;
  end;
 end;
end;

procedure TNeuronNetwork.SaveToStream(F: TFileStream);
begin
 if not Assigned(F) then
  Exit;

 FLayers.SaveToStream(F);
```

```
end;

procedure TNeuronNetwork.SetScreenDimentions(aWidth, aHeight: Integer);
begin
  if (aWidth = FScreen.Width) and (aHeight = FScreen.Height) then
    Exit;

  if (aWidth < 0) or (aHeight < 0) then
    Exit;

  FScreen.Width := aWidth;
  FScreen.Height := aHeight;
  ClearScreen;

  SetLength(ScreenOutputs, aWidth * aHeight);
end;

procedure TNeuronNetwork.SetScreenOutputs;
var
  X, Y, Index: Integer;
  R, G, B: Byte;
begin
  // преобразует картинку в черно-белое изображение
  // выходной сигнал - 0..1 - пропорционален яркости
  Index := 0;

  for Y := 0 to FScreen.Height - 1 do
    for X := 0 to FScreen.Width - 1 do
    begin
      R := GetRValue(FScreen.Canvas.Pixels[X, Y]);
      G := GetGValue(FScreen.Canvas.Pixels[X, Y]);
      B := GetBValue(FScreen.Canvas.Pixels[X, Y]);

      if Index <= High(ScreenOutputs) then
        ScreenOutputs[Index] := (R + G + B) / (3 * MAX_BYTE);

      Inc(Index);
    end;
```

```
end;

{ TNeuron }

procedure TNeuron.Calculate (aToLog: Boolean);
var
  i: Integer;
  S: Double;
begin
  Output := 0;

  S := 0;
  for i := 0 to High(FInputs) do
    if Assigned(FInputs[i]) then
      S := S + FInputs[i]^ * FWeights[i];

  if Assigned(FActivationFunction) then
    Output := FActivationFunction(S, FActivationFunctionParams);


//   if aToLog then
//     AddToLog('1.txt', 'W = ' + FloatToStr());

end;

constructor TNeuron.Create;
begin
  FTrainingSamples := TNeuronSampleList.Create;
  FTestSamples := TNeuronSampleList.Create;

  FColor := TColors.Red;
  Output := 0;
  FFocused := False;

  FInputs := nil;
  FWeights := nil;
  FActivationFunctionParams := nil;
  FActivationFunction := @LinearActivationFunction;
```

```
end;

destructor TNeuron.Destroy;
begin
  FTrainingSamples.DeleteAll;
  FTrainingSamples.Free;

  FTestSamples.DeleteAll;
  FTestSamples.Free;

  FInputs := nil;
  FWeights := nil;
  FActivationFunctionParams := nil;

  inherited;
end;

procedure TNeuron.SetWeight(Index: Integer; const Value: Double);
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    FWeights[Index] := Value;
end;

function TNeuron.GetActivationFunctionParamCount: Integer;
begin
  Result := Length(FActivationFunctionParams);
end;

function TNeuron.GetActivationFunctionParams(Index: Integer): Double;
begin
  if (Index >= 0) and (Index < Length(FActivationFunctionParams)) then
    Result := FActivationFunctionParams[Index]
  else
    Result := 0;
end;

function TNeuron.GetColor: TColor;
const
```

```
  MIN_VALUE = 0;
  MAX_VALUE = 1;
var
  R, G, B: Byte;
begin
  R := Round(GetRValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));
  G := Round(GetGValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));
  B := Round(GetBValue(FColor) * Output / (MAX_VALUE - MIN_VALUE));

  Result := RGB(R, G, B);
end;

function TNeuron.GetInput(Index: Integer): PDouble;
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    Result := FInputs[Index]
  else
    Result := nil;
end;

function TNeuron.GetInputCount: Integer;
begin
  Result := Length(FInputs);
end;

function TNeuron.GetWeight(Index: Integer): Double;
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    Result := FWeights[Index]
  else
    Result := 0;
end;

function TNeuron.GetWeightCount: Integer;
begin
  Result := Length((FWeights));
end;
```

```
function TNeuron.LoadFromStream(F: TFileStream): Boolean;
var
  i, aCount, aReadByteCount: Integer;
begin
  Result := Assigned(F);

  // цвет
  if Result then
  begin
    aReadByteCount := F.Read(FColor, SizeOf(FColor));
    Result := aReadByteCount = SizeOf(FColor);
  end;

  // количество весовых коэффициентов
  if Result then
  begin
    aReadByteCount := F.Read(aCount, SizeOf(aCount));
    Result := aReadByteCount = SizeOf(aCount);
  end;

  if Result then
    Result := aCount = Length(FWeights);

  // весовые коеффициенты
  if Result then
    for i := 0 to aCount - 1 do
    begin
      aReadByteCount := F.Read(FWeights[i], SizeOf(FWeights[i]));
      Result := aReadByteCount = SizeOf(FWeights[i]);

      if not Result then
        Break;
    end;

  // учебная выборка
  if Result then
    Result := FTrainingSamples.LoadFromStream(F);
```

```
  // тестовая выборка
  if Result then
    Result := FTestSamples.LoadFromStream(F);
end;

procedure TNeuron.SaveToStream(F: TFileStream);
var
  i, aCount: Integer;
begin
  if not Assigned(F) then
    Exit;

  // Цвет
  F.Write(FColor, SizeOf(FColor));

  // количество весовых коэффициентов
  aCount := Length(FWeights);
  F.Write(aCount, SizeOf(aCount));

  // весовые коэффициенты
  for i := 0 to aCount - 1 do
    F.Write(FWeights[i], SizeOf(FWeights[i]));

  // учебная выборка
  FTrainingSamples.SaveToStream(F);

  // тестовая выборка
  FTestSamples.SaveToStream(F);
end;

procedure TNeuron.SetActivationFunctionParamCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Length(FActivationFunctionParams) then
    Exit;

  if Value >= 0 then
```

```
  begin
    SetLength(FActivationFunctionParams, Value);

    for i := 0 to High(FActivationFunctionParams) do
      FActivationFunctionParams[i] := 0;
  end;
end;

procedure TNeuron.SetActivationFunctionParams(Index: Integer;
  const Value: Double);
begin
  if (Index >= 0) and (Index < Length(FActivationFunctionParams)) then
    FActivationFunctionParams[Index] := Value;
end;

procedure TNeuron.SetInput(Index: Integer; const Value: PDouble);
begin
  if (Index >= 0) and (Index < Length(FInputs)) then
    FInputs[Index] := Value;
end;

procedure TNeuron.SetInputCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Length(FInputs) then
    Exit;

  if Value >= 0 then
  begin
    SetLength(FInputs, Value);
    SetLength(FWeights, Value);
  end;

  for i := 0 to High(FInputs) do
  begin
    FInputs[i] := nil;
    FWeights[i] := 0;
```

```
  end;
end;

{ TNeuronSample }

procedure TNeuronSample.Assing(aScreen: TBitmap);
var
  aRect: TRect;
begin
  FScreen.Width := aScreen.Width;
  FScreen.Height := aScreen.Height;

  aRect.Left := 0;
  aRect.Top := 0;
  aRect.Width := FScreen.Width;
  aRect.Height := FScreen.Height;

  FScreen.Canvas.StretchDraw(aRect, aScreen);
end;

constructor TNeuronSample.Create;
begin
  FScreen := TBitmap.Create;
  FScreen.PixelFormat := pf24bit;

  FFocused := False;
end;

destructor TNeuronSample.Destroy;
begin
  FScreen.Free;

  inherited;
end;

function TNeuronSample.LoadFromStream(F: TFileStream): Boolean;
var
  X, Y, aWidth, aHeight, RealByteRead: Integer;
```

```
  Pixel: TColor;
begin
  Result := Assigned(F);

  // размеры изображения
  if Result then
  begin
    RealByteRead := F.Read(aWidth, SizeOf(aWidth));
    Result := RealByteRead = SizeOf(aWidth);
  end;

  if Result then
  begin
    RealByteRead := F.Read(aHeight, SizeOf(aHeight));
    Result := RealByteRead = SizeOf(aHeight);
  end;

  // пиксели
  if Result then
  begin
    FScreen.Width := aWidth;
    FScreen.Height := aHeight;

    for Y := 0 to aHeight - 1 do
      for X := 0 to aWidth - 1 do
        if Result then
        begin
          RealByteRead := F.Read(Pixel, SizeOf(Pixel));
          Result := RealByteRead = SizeOf(Pixel);

          FScreen.Canvas.Pixels[X, Y] := Pixel;
        end;
  end;
end;

procedure TNeuronSample.Clear;
var
  X, Y: Integer;
```

```
begin
  for Y := 0 to FScreen.Height - 1 do
    for X := 0 to FScreen.Width - 1 do
      FScreen.Canvas.Pixels[X, Y] := RandomColor;
end;

procedure TNeuronSample.SaveToStream(F: TFileStream);
var
  X, Y, aWidth, aHeight: Integer;
  Pixel: TColor;
begin
  if not Assigned(F) then
    Exit;

  // размеры изображения
  aWidth := FScreen.Width;
  F.Write(aWidth, SizeOf(aWidth));

  aHeight := FScreen.Height;
  F.Write(aHeight, SizeOf(aHeight));

  // пиксели
  for Y := 0 to aHeight - 1 do
    for X := 0 to aWidth - 1 do
    begin
      Pixel := FScreen.Canvas.Pixels[X, Y];
      F.Write(Pixel, SizeOf(Pixel));
    end;
end;

procedure TNeuronSample.SetDimentions(aWidth, aHeight: Integer);
begin
  if (aWidth = FScreen.Width) and (aHeight = FScreen.Height) then
    Exit;

  FScreen.Width := aWidth;
  FScreen.Height := aHeight;
  Clear;
```

```
end;

{ TNeuronLayerList }

function TNeuronLayerList.First: TNeuronLayer;
begin
  if Count > 0 then
    Result := inherited First
  else
    Result := nil;
end;

function TNeuronLayerList.Get(Index: Integer): TNeuronLayer;
begin
  Result := inherited Get(Index);
end;

function TNeuronLayerList.GetCount: Integer;
begin
  Result := inherited Count;
end;

function TNeuronLayerList.Last: TNeuronLayer;
begin
  if Count > 0 then
    Result := inherited Last
  else
    Result := nil;
end;

function TNeuronLayerList.LoadFromStream(F: TFileStream): Boolean;
var
  i, aCount, aReadByteCount: Integer;
begin
  Result := Assigned(F);

  // количество слоев
  if Result then
```

```
  begin
    aReadByteCount := F.Read(aCount, SizeOf(aCount));
    Result := aReadByteCount = SizeOf(aCount);
  end;

  if Result then
    Result := aCount = Count;

  // слои
  if Result then
    for i := 0 to aCount - 1 do
    begin
      Result := Items[i].LoadFromStream(F);

      if not Result then
        Break;
    end;
end;

procedure TNeuronLayerList.Put(Index: Integer; const Value: TNeuronLayer);
begin
  inherited Put(Index, Value);
end;

procedure TNeuronLayerList.SaveToStream(F: TFileStream);
var
  aCount, i: Integer;
begin
  if not Assigned(F) then
    Exit;

  // количество слоев
  aCount := Count;
  F.Write(aCount, SizeOf(aCount));

  // слои
  for i := 0 to aCount - 1 do
    Items[i].SaveToStream(F);
```

```
end;

procedure TNeuronLayerList.SetCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Count then
    Exit;

  for i := 0 to Count - 1 do
    Items[i].Free;

  for i := 0 to Value - 1 do
    Add(TNeuronLayer.Create);
end;

{ TNeuronList }

function TNeuronList.First: TNeuron;
begin
  if Count > 0 then
    Result := inherited First
  else
    Result := nil;
end;

procedure TNeuronList.FocuseFirst;
begin
  SetFocused(First);
end;

function TNeuronList.Get(Index: Integer): TNeuron;
begin
  Result := inherited Get(Index);
end;

function TNeuronList.GetCount: Integer;
begin
```

```
  Result := inherited Count;
end;

function TNeuronList.GetFocused: TNeuron;
var
  i: Integer;
begin
  Result := nil;

  for i := 0 to Count - 1 do
    if Items[i].Focused then
      Result := Items[i];
end;

function TNeuronList.Last: TNeuron;
begin
  if Count > 0 then
    Result := inherited Last
  else
    Result := nil;
end;

function TNeuronList.LoadFromStream(F: TFileStream): Boolean;
var
  i, aCount, aReadByteCount: Integer;
begin
  Result := Assigned(F);

  // количество нейронов
  if Result then
  begin
    aReadByteCount := F.Read(aCount, SizeOf(aCount));
    Result := aReadByteCount = SizeOf(aCount);
  end;

  if Result then
    Result := aCount = Count;
```

```
  // нейроны
  if Result then
    for i := 0 to aCount - 1 do
    begin
      Result := Items[i].LoadFromStream(F);

      if not Result then
        Break;
    end;
end;

procedure TNeuronList.Put(Index: Integer; const Value: TNeuron);
begin
  inherited Put(Index, Value);
end;

procedure TNeuronList.SaveToStream(F: TFileStream);
var
  aCount, i: Integer;
begin
  if not Assigned(F) then
    Exit;

  // количество нейронов
  aCount := Count;
  F.Write(aCount, SizeOf(aCount));

  // нейроны
  for i := 0 to aCount - 1 do
    Items[i].SaveToStream(F);
end;

procedure TNeuronList.SetCount(const Value: Integer);
var
  i: Integer;
begin
  if Value = Count then
    Exit;
```

```
  for i := 0 to Count - 1 do
    Items[i].Free;

  Clear;

  for i := 0 to Value - 1 do
    Add(TNeuron.Create);
end;

procedure TNeuronList.SetFocused(const Value: TNeuron);
var
  i: Integer;
begin
  for i := 0 to Count - 1 do
    Items[i].Focused := Items[i] = Value;
end;

{ TNeuronLayer }

constructor TNeuronLayer.Create;
begin
  FNeurons := TNeuronList.Create;
  FCanFocus := False;
end;

destructor TNeuronLayer.Destroy;
begin
  FNeurons.Count := 0;
  FNeurons.Free;

  inherited;
end;

function TNeuronLayer.GetFocusedNeuron: TNeuron;
var
  i: Integer;
begin
```

```
  Result := nil;

  for i := 0 to FNeurons.Count - 1 do
    if FNeurons[i].Focused then
    begin
      Result := FNeurons[i];
      Break;
    end;
end;

function TNeuronLayer.LoadFromStream(F: TFileStream): Boolean;
begin
  Result := Assigned(F);

  if Result then
    Result := FNeurons.LoadFromStream(F);
end;

procedure TNeuronLayer.SaveToStream(F: TFileStream);
begin
  if not Assigned(F) then
    Exit;

  FNeurons.SaveToStream(F);
end;

{ TNeuronSampleList }

function TNeuronSampleList.AddNewSample: TNeuronSample;
begin
  Result := TNeuronSample.Create;
  inherited Add(Result);

  SetFocusedSample(Result);
end;

constructor TNeuronSampleList.Create;
begin
```

```
  inherited;

  FFileName := '';
  FPageNumber := 0;
end;

procedure TNeuronSampleList.DeleteAll;
var
  I: Integer;
begin
  for I := 0 to Count - 1 do
    Items[i].Free;

  Clear;
end;

procedure TNeuronSampleList.DeleteSample(aSample: TNeuronSample);
var
  SampleIndex: Integer;
begin
  SampleIndex := IndexOf(aSample);

  if SampleIndex >= 0 then
  begin
    inherited Remove(aSample);

    aSample.Free;

    while SampleIndex >= Count do
      Dec(SampleIndex);

    if SampleIndex >= 0 then
      SetFocusedSample(Items[SampleIndex])
    else
      SetFocusedSample(nil);
  end;
end;
```

```
function TNeuronSampleList.First: TNeuronSample;
begin
  if Count > 0 then
    Result := inherited First
  else
    Result := nil;
end;

procedure TNeuronSampleList.FocuseFirst;
begin
  SetFocusedSample(First);
end;

function TNeuronSampleList.Get(Index: Integer): TNeuronSample;
begin
  Result := inherited Get(Index);
end;

function TNeuronSampleList.GetCount: Integer;
begin
  Result := inherited Count;
end;

function TNeuronSampleList.GetFocusedSample: TNeuronSample;
var
  i: Integer;
begin
  Result := nil;

  for i := 0 to Count - 1 do
    if Items[i].Focused then
      Result := Items[i];
end;

function TNeuronSampleList.Last: TNeuronSample;
begin
  if Count > 0 then
    Result := inherited Last
```

```
  else
    Result := nil;
end;

function TNeuronSampleList.LoadFromFile(ShowDialog: Boolean): Boolean;
var
  D: TOpenDialog;
  S: String;
  F: TFileStream;
  CanShowMessage: Boolean;
begin
  Result := True;
  CanShowMessage := ShowDialog;

  if ShowDialog then
  begin
    D := TOpenDialog.Create(nil);
    try
      if FileExists(FFileName) then
        D.FileName := FFileName;

      case gLanguage of
        lgEng: S := 'Sample sets';
        lgRus: S := 'Наборы образцов';
        else S := 'Набори зразків';
      end;

      D.Filter := S + ' (*' + SAMPLE_SET_FILE_EXTENTION + ') |*'
        + SAMPLE_SET_FILE_EXTENTION;

      if D.Execute(Application.Handle) then
      begin
        FFileName := D.FileName
      end else
      begin
        Result := False;
        CanShowMessage := False;
      end;
```

```
    finally
      D.Free;
    end;
  end;

  if Result then
    Result := FileExists(FFileName);

  if Result then
  begin
    F := TFileStream.Create(FFileName, fmOpenRead);
    try
      Result := LoadFromStream(F);
    finally
      F.Free;
    end;
  end;

  if CanShowMessage and (not Result) then
  begin
    case gLanguage of
      lgEng: S := 'The file is missing or damaged';
      lgRus: S := 'Файл отсуствует или поврежден';
      else S := 'Файл вітсутній або пошкоджений';
    end;

    ShowMessage(S);
  end;
end;

function TNeuronSampleList.LoadFromStream(F: TFileStream): Boolean;
var
  i, SamplesCount, RealByteRead: Integer;
  aSample: TNeuronSample;
begin
  Result := False;

  if not Assigned(F) then
```

```
    Exit;

  // количество семплов
  SamplesCount := 0;
  RealByteRead := F.Read(SamplesCount, SizeOf(SamplesCount));
  Result := RealByteRead = SizeOf(SamplesCount);

  if Result then
  begin
    DeleteAll;

    aSample := TNeuronSample.Create;
    try
      for i := 0 to SamplesCount - 1 do
      begin
        Result := aSample.LoadFromStream(F);

        if Result then
          AddNewSample.Assing(aSample.Screen)
        else
          Break;
      end;
    finally
      aSample.Free;
    end;

    SetFocusedSample(First);
  end;
end;

function TNeuronSampleList.MaxHeight: Integer;
var
  i: Integer;
begin
  Result := 0;
  for i := 0 to Count - 1 do
    Result := Max(Result, Items[i].Screen.Height);
end;
```

```
function TNeuronSampleList.MaxWidth: Integer;
var
  i: Integer;
begin
  Result := 0;
  for i := 0 to Count - 1 do
    Result := Max(Result, Items[i].Screen.Width);
end;

procedure TNeuronSampleList.Put(Index: Integer; const Value: TNeuronSample);
begin
  inherited Put(Index, Value);
end;

procedure TNeuronSampleList.SaveToFile(ShowDialog: Boolean);
var
  S: String;
  D: TSaveDialog;
  F: TFileStream;
  CanSaveNetwork: Boolean;
begin
  CanSaveNetwork := True;

  if ShowDialog then
  begin
    D := TSaveDialog.Create(nil);
    try
      D.Options := D.Options + [ofOverwritePrompt];

      if FileExists(FFileName) then
        D.FileName := FFileName;

      case gLanguage of
        lgEng: S := 'Sample sets';
        lgRus: S := 'Наборы образцов';
        else S := 'Набори зразків';
      end;
```

```
      D.Filter := S + ' (*' + SAMPLE_SET_FILE_EXTENTION + ') |*'
        + SAMPLE_SET_FILE_EXTENTION;

      if D.Execute(Application.Handle) then
      begin
        FFileName := D.FileName;

        if ExtractFileExt(FFileName) = '' then
          FFileName := FFileName + SAMPLE_SET_FILE_EXTENTION;
      end else
        CanSaveNetwork := False;
    finally
      D.Free;
    end;
  end;

  if CanSaveNetwork then
  begin
    F := TFileStream.Create(FFileName, fmCreate);
    try
      SaveToStream(F);
    finally
      F.Free;
    end;
  end;
end;

procedure TNeuronSampleList.SaveToStream(F: TFileStream);
var
  i, aSamplesCount: Integer;
begin
  if not Assigned(F) then
    Exit;

  // количество семплов
  aSamplesCount := Count;
  F.Write(aSamplesCount, SizeOf(aSamplesCount));
```

```
  for i := 0 to aSamplesCount - 1 do
    Items[i].SaveToStream(F);
end;

procedure TNeuronSampleList.SetFocusedSample(const Value: TNeuronSample);
var
  i: Integer;
begin
  for i := 0 to Count - 1 do
    Items[i].Focused := Items[i] = Value;
end;

end.
```