МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ПИЛИПА ОРЛИКА

Економічно-технологічний факультет

Кафедра інженерних технологій

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського рівня) вищої освіти

на тему:

«Прикладні аспекти використання нейронних мереж для розпізнавання текстової інформації»

Зі спеціальності 123

«КОМП’ЮТЕРНА ІНЖЕНЕРІЯ»

Виконавець:

Тінкован О.С.

Науковий керівник:

к.т.н., доц. Гайша О.О.

Миколаїв – 2025

ЗМІСТ

ВСТУП

Розпізнавання візуальних образів є однією з важливих задач інтелектуальної обробки даних і полягає у встановленні відповідності між зображенням об’єкта та його суттю. Зокрема, розпізнавання тексту дозволяє поставити у відповідність зображення (графічний документ) та текстову інформацію, яка присутня на цьому зображенні. У найпростішому, але практично важливому випадку зображення може містити всього лише один символ (букву), а програмний продукт на основі аналізу цього графічного файлу видає вердикт, яка саме літера алфавіту там зображена. Така, на перший погляд проста, задача лежить в основі багатьох систем розпізнавання тексту, аж до професійних типу ABBYY FineReader і т.д.

Розв’язувати цю задачу можна, в цілому, різними способами, починаючи від методів математичної статистики, і закінчуючи надсучасними методами теорії нейронних мереж.

Штучні нейронні мережі (ШНМ) будуються по аналогії з біологічними нейронними мережами, що є основою нервової системи тварин, та, зокрема, людини. Клітини нервової системи – нейрони – відрізняються від інших клітин тим, що:

- мають багато входів, по яким до них надходить електро-хімічний сигнал від попередніх нервових клітин;

- часто мають лише один вихід у вигляді дуже довгого відростку (аксону), що прикріплюється до нервових клітин наступного рівня, і по якому сигнал виходить/видається з нейрону;

- мають тіло, в якому відбувається об’єднання вхідних сигналів і вироблення за певними правилами вихідного сигналу.

Вчені 40-х років XX століття спробували відтворити хоча б частково таку надскладну систему, як нервова, ґрунтуючись на штучних нейронах, реалізованих у вигляді математичної моделі (розглянуто у підрозділі 1.2 роботи). Ці дослідження стали початком ери штучних нейронних мереж, хоча

досить швидко про них на деякий час забули. Відновлення інтересу до ШНМ відбулося тільки у 90-і роки, коли з’явилася апаратна база у вигляді обчислювально потужних комп’ютерів, які могли імітувати злагоджену роботу сотень і тисяч штучних нейронів. З того часу інтерес до впровадження нейронних мереж у різні предметні галузі безперервно зростає.

Так, ШНМ є інструментом, що дозволяє успішно вирішувати задачі класифікації, кластеризації, апроксимації, прогнозування, оптимізації та управління. В основному усі ці класи задач більш, чи менш ефективно можуть розв’язуватися традиційними способами, шляхом застосування аналітичних залежностей, статистики, однак, застосування ШНМ для їх вирішення дає надзвичайно відмінні результати. Докладніше задачу розпізнавання символів і розглянемо у даній роботі.

Метою є створення системи розпізнавання візуальної інформації, що дозволятиме проводити аналіз зображень на предмет наявності у них символьних елементів.

Практичне значення роботи полягає у створення робочого програмного продукту для проведення розпізнавання окремих символів латинської (англійської) абетки.

1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА ЗАДАЧІ РОЗПІЗНАВАННЯ

1.1. Особливості задачі розпізнавання образів.

Задача розпізнавання образів — це задача віднесення вихідних даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних.

Розпізнавання образів є однією з найфундаментальніших проблем теорії інтелектуальних систем. З іншого боку, задача розпізнавання образів має величезне практичне значення. Замість терміну "розпізнавання" часто використовується інший термін — "класифікація". Ці два терміни у багатьох випадках розглядаються як синоніми, але не є повністю взаємозамінюваними. Кожний з цих термінів має свої сфери застосування, і інтерпретація обох термінів часто залежить від специфіки конкретної задачі

Розпізнавання образів — це віднесення вихідних даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних.

При постановці задач розпізнавання намагаються користуватися математичною мовою, стараючись, на відміну від теорії штучних нейронних мереж, де основою є одержання результату шляхом експерименту, замінити експеримент логічними міркуваннями й математичними доказами [1].

Найчастіше в задачах розпізнавання образів розглядаються монохромні зображення, що дає можливість розглядати зображення як функцію на площині. Якщо розглянути множину точок на площині, де функція виражає в кожній точці зображення його характеристику — яскравість, прозорість, оптичну щільність, то така функція є формальним записом зображення.

Множина усіх можливих функцій на площині — є моделлю множини всіх зображень . Уводячи поняття подібності між образами можна ставити задачі розпізнавання. Конкретний вид такої постановки залежить від наступних етапів при розпізнаванні відповідно до того або іншого підходу.

1.2. Існуючі методи розпізнавання текстової інформації.

Основним на сьогоднішній день способом розпізнавання текстової інформації є застосування ШНМ. Відповідно, розглянемо їх склад та види.

Як уже було зазначено вище, елементарною складовою ШНМ є один нейрон – рис. 1.1. Це об’єкт, що має певну кількість входів (на рисунку – n входів), які називають дендритами та один вихід, що називається аксоном.

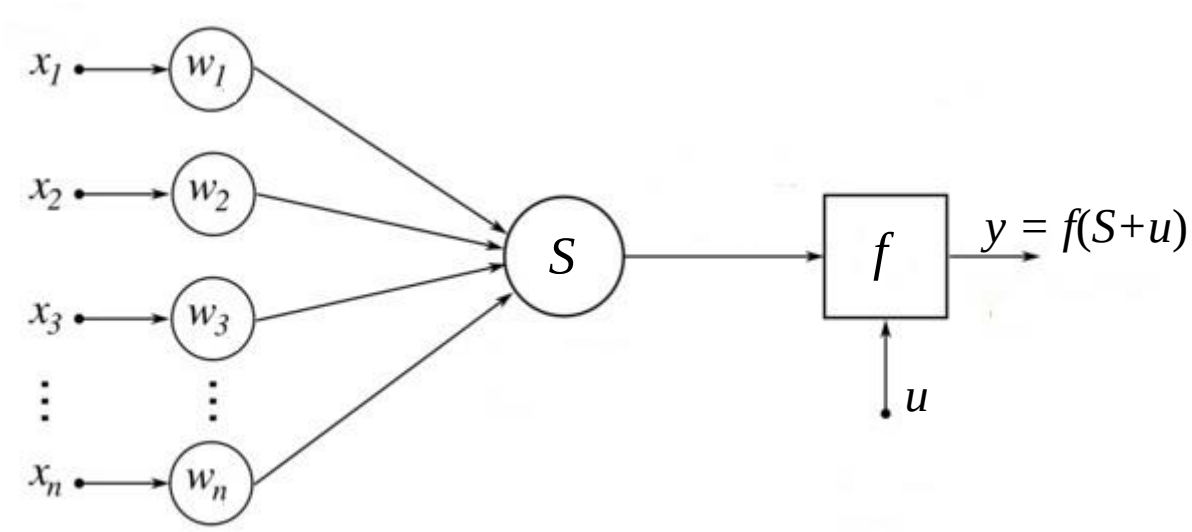


Рис. 1.1 Структура штучного нейрону.

На входи нейрону подаються вхідні сигнали x_1, x_2, …, x_n, що є виходами попередніх нейронів, розташованих раніше по ходу проходження сигналу. Кожен вхідний сигнал x_i множиться на певне число w_i, що називають вагою (або синаптичною вагою) відповідного входу. Усі таким чином зважені вхідні сигнали подаються на суматор, який виробляє зважену суму S виду:

$$S = \sum_{i=1}^{n} w_i x_i \,. \qquad (1.1)$$

До цієї зваженої суми вхідних сигналів може додаватися зсув або зміщення u (у поширеному частинному випадку зміщення відсутнє, тобто $u = 0$) і від цієї суми виробляється певна функція f, що називається активаційною; так і формується вихідний сигнал нейрону y, що передається далі по нейронній мережі:

$$y = f(S + u). \quad (1.2)$$

Відповідно комбінуючи (1) та (2) маємо функцію, що здійснює штучний нейрон:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + u\right). \quad (1.3)$$

При аналізі такого способу завдання функції, що здійснює нейрон, бачимо, що важливу роль для кожного нейрона відіграють три речі (особливо перші дві):

- набір конкретних значень ваг w_i, що описують кожен вхід нейрона;
- вид активаційної функції f даного нейрона;
- наявність зсуву u (який може бути як додатним, так і від'ємним).

Зважаючи на те, що звичайна ШНМ складається як мінімум із кількох нейронів (а частіше – з десятків, чи навіть сотень нейронів), а в кожного з них є багато входів, то задача вибору вагових коефіцієнтів w_i стає дуже серйозною за своєю складністю. Власне, тривалий процес завдання конкретних «правильних» значень цих вагових коефіцієнтів називають процесом навчання нейронної мережі. Існують різні алгоритми навчання нейронних мереж, які укрупнено можна поділити на дві групи: «з учителем» та «без учителя». В цілому, навчання нейронних мереж є настільки складним процесом, що має досліджуватися в окремому великому блоці інформації.

Вид активаційної функції f очевидно також є дуже важливим моментом для функціонування нейронів, хоча й у значно меншій мірі, ніж вибір синаптичних вагових коефіцієнтів. На рис. 1.2 наведено найбільш поширені види функцій активації, що найчастіше використовуються на практиці при роботі з нейронними мережами. З точки зору обчислювальної складності більш кращим є використання порогових функцій, що фактично являють собою константну залежність (або хоча б лінійних).

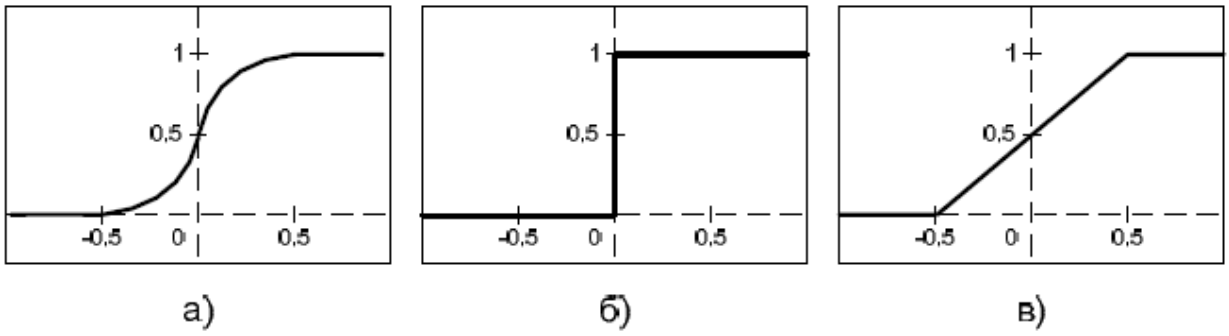


Рис. 1.2. Найбільш поширені типи функцій активації: а – сигмоїдна, б – порогова, в - лінійна.

При побудові цілої нейронної мережі, крім характеристик окремих нейронів, важливим стає також спосіб їх з’єднання (так само, як і їх загальна кількість, організація нейронів у групи, і т.д.).

Найпростішим і в той же час найпоширенішим способом з’єднання нейронів є персептрон (перцептрон), який і буде реалізовуватися на практиці у даній роботі. Однак, крім персептрону, існує ще багато способів організації нейронних мереж, що мають свою специфіку, тому розглянемо їх докладніше.

а) Нейронні мережі прямого поширення (feed forward neural networks, FF або FFNN) і перцептрони (perceptrons, P) – повністю прямолінійні, вони передають інформацію тільки від входу до виходу. Нейронні мережі часто описуються у вигляді багатошарового торта, де кожен шар складається з вхідних (на них поступають сигнали, що є вхідними для нейронної мережі, тобто від зовнішніх джерел), прихованих (тих, що знаходяться між вхідними та вихідними і не є ані тими, ані тими) або вихідних (з яких сигнал передається із нейронної мережі далі на зовнішні споживачі) нейронів. Нейрони одного шару не пов'язані між собою, а сусідні шари зазвичай повністю пов'язані. Найпростіша нейронна мережа (FF типу, або P) має два вхідних нейрони і один вихідний, і може використовуватися у якості моделі логічних вентилів. FFNN зазвичай навчається за методом зворотного поширення помилки, в якому мережа отримує множини вхідних і вихідних

даних. Цей процес називається навчанням з учителем, і він відрізняється від навчання без учителя тим, що в другому випадку множину вихідних даних мережа утворює самостійно. Якщо у мережі є достатня кількість прихованих нейронів, вона теоретично здатна змоделювати будь-яку взаємодію між вхідними і вихідними сигналами. Практично такі мережі використовуються досить часто, особливо у навчальних задачах, але також їх часто комбінують з іншими типами для отримання нових властивостей.

б) Мережі радіально-базисних функцій (radial basis function, RBF) - це FFNN, яка використовує радіальні базисні функції як функції активації. Більше нічим особливим ці ШНМ не виділяються.

в) Нейронна мережа Хопфілда (Hopfield network, HN) - це повнозв’язана нейронна мережа із симетричною матрицею зв'язків. Під час отримання вхідних даних кожен вузол мережі є входом, потім в процесі навчання він стає прихованим, а кінцево – стає виходом. Мережа навчається так: значення нейронів встановлюються відповідно до бажаного шаблону, після чого обчислюються ваги, які в подальшому не змінюються. Після того, як мережа навчилася на одному або декількох шаблонах, вона завжди буде зводитися до одного з них (але не завжди - до бажаного). Вона стабілізується в залежності від загальної «енергії» і «температури» мережі. У кожного нейрона є свій поріг активації, що залежить від «температури», при проходженні якого нейрон приймає одне з двох значень (зазвичай -1 або 1, іноді 0 або 1). Така мережа часто називається мережею з асоціативної пам'яттю; як людина, бачачи половину таблиці, може представити другу половину таблиці, так і ця мережа, отримуючи таблицю, наполовину зашумленною, відновлює її до повної.

г) Ланцюги Маркова (Markov chains, MC або discrete time Markov Chains, DTMC) - це попередники машин Больцмана (BM) і мереж Хопфілда (HN). Їхній зміст можна пояснити так: які шанси потрапити в один з наступних вузлів, якщо перебувати в даному? Кожний наступний стан залежить тільки від попереднього. Насправді ланцюги Маркова не є

нейронними мережами, хоча вони дуже схожі; також ланцюги Маркова не обов'язково є повнозв'язними.

д) Машина Больцмана (Boltzmann machine, BM) дуже схожа на мережу Хопфілда (тобто має повнозв’язану архітектуру), але в ній деякі нейрони позначені як вхідні, а деякі - як приховані. Вхідні нейрони в подальшому стають вихідними. Машина Больцмана - це стохастична мережа. Навчання проходить за методом зворотного поширення помилки або за алгоритмом порівняльної розбіжності. В цілому, процес навчання аналогічний мережі Хопфілда.

е) Обмежена машина Больцмана (restricted Boltzmann machine, RBM) дуже схожа на машину Больцмана і, отже, на мережу Хопфілда. Єдиною різницею є її обмеженість: у ній нейрони одного типу не пов'язані між собою. Обмежену машину Больцмана можна навчати як FFNN, але з одним нюансом: замість прямої передачі даних і зворотного поширення помилки потрібно передавати дані спершу в прямому напрямку, потім в зворотному. Після цього процес навчання проходить за методом прямого і зворотного поширення помилки.

є) Автокодувальник (autoencoder, AE) майже ідентичний до FFNN, і швидше це – інший спосіб використання FFNN, ніж фундаментально нова архітектура. Основною ідеєю є автоматичне кодування (маючи на увазі стиснення, але не шифрування) інформації. Сама мережа по формі нагадує пісочний годинник, тому що в ній приховані шари менше за кількістю нейронів, ніж вхідний і вихідний, причому вона симетрична. Мережу можна навчити методом зворотного поширення помилки, подаючи вхідні дані і задаючи помилку, рівну різниці між входом і виходом.

ж) Розріджений автокодувальник (sparse autoencoder, SAE) - в якомусь сенсі протилежність звичайного. Замість того, щоб навчати мережу відображати інформацію в меншому «обсязі» вузлів прихованих шарів, ми збільшуємо їх кількість. Замість того, щоб звужуватися до центру, мережа там роздувається. Мережі такого типу корисні для роботи з великою

кількістю дрібних властивостей набору даних. Якщо навчати мережу як звичайний автокодувальник, нічого корисного не вийде. Тому крім вхідних даних, подається ще і спеціальний фільтр розрідженості, який пропускає тільки певні помилки.

з) Варіаційні автокодувальники (variational autoencoder, VAE) мають схожу з AE архітектуру, але навчають їх іншому: наближенню імовірнісного розподілу вхідних зразків. У цьому вони беруть початок від машин Больцмана. Проте, вони спираються на Байєсову математику, коли мова йде про імовірнісні висновки і незалежності, які інтуїтивно зрозумілі, але складні в реалізації. Якщо узагальнити, можна сказати що ця мережа бере до уваги впливи нейронів. Якщо щось одне відбувається в одному місці, а щось інше - в іншому, то ці події не обов'язково пов'язані, і це повинно враховуватися.

и) Шумоподавляючі автокодувальники (denoising autoencoder, DAE) - це AE, у які вхідні дані подаються в зашумленому стані. Помилка обчислюється так само, і вихідні дані порівнюються з зашумленими. Завдяки цьому мережа вчиться звертати увагу на більш великі властивості, оскільки маленькі можуть змінюватися разом з шумом.

і) Мережа типу «deep belief» (deep belief networks, DBN) - це назва, яку отримав тип архітектури, в якій мережа складається з декількох з'єднаних RBM або VAE. Такі мережі навчаються по блоках, причому кожному блоку потрібно лише вміти закодувати попередній. Така техніка називається «жадібним навчанням», і полягає у виборі локальних оптимальних рішень, що не гарантують оптимальний кінцевий результат. Також мережу можна навчити (методом зворотного поширення помилки) відображати дані у вигляді ймовірнісної моделі. Якщо використовувати навчання без учителя, стабілізовану модель можна використовувати для генерації нових даних.

й) Згортуючі нейронні мережі (convolutional neural networks, CNN) і глибинні згортуючі нейронні мережі (deep convolutional neural networks, DCNN) сильно відрізняються від інших видів мереж. Зазвичай вони

використовуються для обробки зображень, рідше для аудіо. Типовим способом застосування CNN є класифікація зображень: якщо на зображенні є кішка, мережа видасть «кішка», якщо є собака - «собака». Такі мережі зазвичай використовують «сканер», що не пропускає усі дані за один раз. Наприклад, якщо є зображення 200 × 200, мережа не буде відразу обробляти всі 40 тисяч пікселів. Замість цього мережа буде розглядати квадрати розміру 20 x 20 (зазвичай, починаючи, з лівого верхнього кута), потім зсунеться на 1 піксель і розглядатиме новий квадрат, і т.д. Ці вхідні дані потім передаються через згортуючі шари, в яких не всі вузли з'єднані між собою. Ці шари мають властивість стискуватися з глибиною, причому часто використовуються ступеня двійки: 32, 16, 8, 4, 2, 1. На практиці до кінця CNN прикріплюють FFNN для подальшої обробки даних. Такі мережі називаються глибинними (DCNN).

к) Розгортальні нейронні мережі (deconvolutional networks, DN), що також називаються зворотними графічними мережами, є оберненими до згортуючих нейронних мереж. Наприклад, при передачі мережі слова «кішка», вона генеруватиме картинки з кішками, схожі на реальні зображення котів. DNN теж можна об'єднувати з FFNN. Варто зауважити, що в більшості випадків мережі передається не рядок, а якийсь бінарний вектор: наприклад, <0, 1> - це кішка, <1, 0> - собака, а <1, 1> - і кішка, і собака.

Таким чином, докладно розглянуто існуючі типи нейронних мереж, що активно використовуються у різноманітних задачах, описаних у вступі. Для цілей навчання та вирішення практичних задач у першому наближенні практично завжди підійде мережа прямого поширення типу «перцептрон» - рис. 1.3.

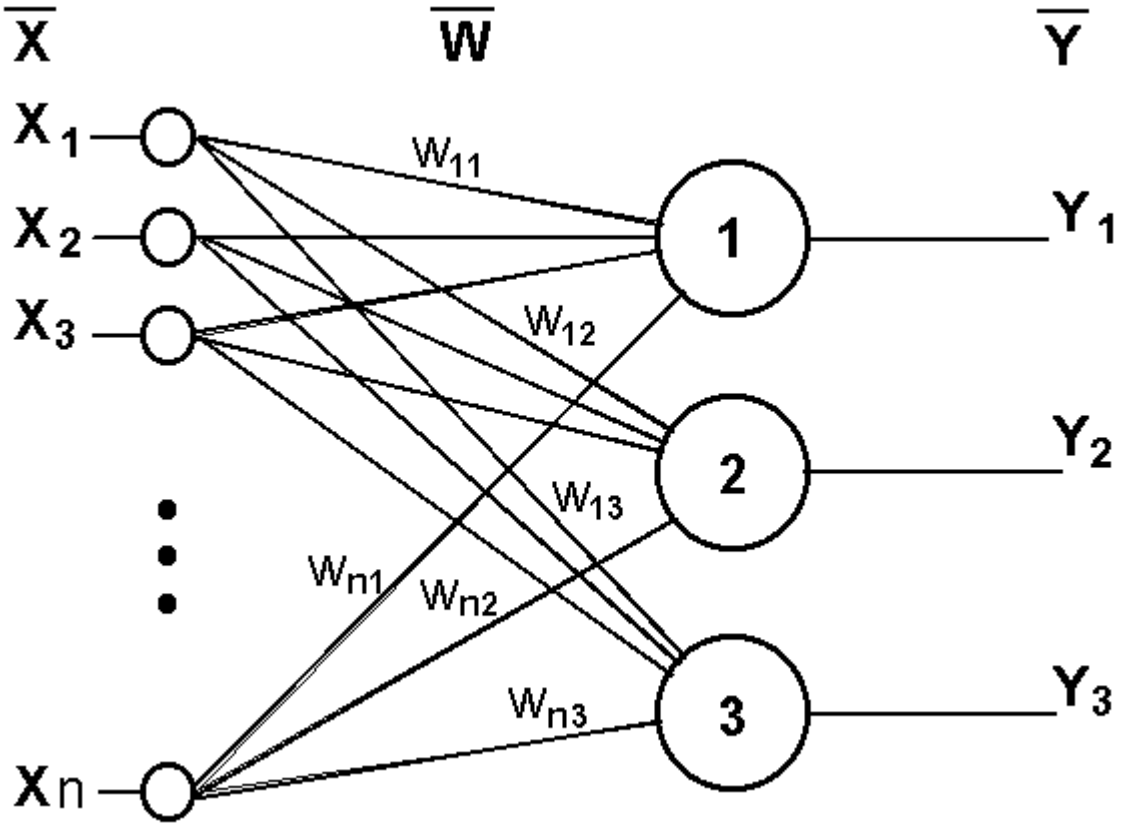


Рис. 1.3. Будова нейронної мережі персептрону (найпростішого, одношарового), можливостей якого достатньо для вирішення великої кількості практичних задач.

Персептрони можуть мати декілька шарів, і саме таку модель обрано для подальших досліджень в рамках виконання дипломної роботи.

Персептро́н, або перцептро́н (англ. perceptron від лат. perceptio - сприйняття; нім. perzeptron) – математична або комп'ютерна модель сприйняття інформації мозком (кібернетична модель мозку), запропонована Френком Розенблатом в 1957 році й реалізована у вигляді електронної машини «Марк-1» у 1960 році. Персептрон став однією з перших моделей нейромереж, а «Марк-1» - першим у світі нейрокомп'ютером. Незважаючи на свою простоту, персептрон здатен навчатися і розв'язувати досить складні завдання. Основна математична задача, з якою він здатний впоратися – це лінійне розділення довільних нелінійних множин, так зване забезпечення лінійної сепарабельності.

Персептрон складається з трьох типів елементів, а саме: сигнали, що надходять від давачів, передаються до асоціативних елементів, а відтак до реагуючих. Таким чином, персептрони дозволяють створити набір «асоціацій» між вхідними стимулами та необхідною реакцією на виході. В біологічному плані це відповідає перетворенню, наприклад, зорової інформації у фізіологічну відповідь рухових нейронів. Відповідно до сучасної термінології, персептрони може бути класифіковано як штучні нейронні мережі:

- з одним прихованим шаром;
- з пороговою або сигмоїдальною передавальною функцією;
- з прямим розповсюдженням сигналу.

Персептрон взятий за основу для системи розпізнавання простих зображень, реалізованої у даній роботі.

Важливими при розпізнаванні образів також є наступні аспекти цієї задачі:

- розмір зображення;
- орієнтація зображення;
- якість зображення.

Якщо перші два аспекти є більш-менш однозначними, то останній – навпаки, найбільш багатогранний і може розглядатися під різними кутами. Перейдемо до докладного аналізу трьох вказаних питань.

Під розміром зображення матимемо на увазі його розмір у пікселах (тобто у кількості точок по горизонталі й вертикалі), а не, наприклад, у метричних одиницях, як для звичайних фотографій. Прикладами типових розмірів зображень є:

640x480, 1024x768, 1600x1200, і т.д.

Звичайно, немає жодної заборони на непропорційні до $4 : 3 = 1{,}333$ розміри зображень на зразок 2048x480 (як для панорамного зображення), або 120x600 (як популярний розмір рекламного баннеру в Інтернет), і т.д.

Відмітимо, що розмір у пікселах може бути обчислений за допомогою метричних розмірів (*W* x *H* сантиметрів, або дюймів) та розширення зображення (задається у dpi – dots per inch, тобто скільки пікселів даного зображення укладається на одному дюймі його довжини). Наприклад, стандартним розширенням для скріншотів, знятих з екрану ПК є 72 точки на дюйм, а для якісного друку рекомендується надавати фотографії з розширенням 300 dpi і вище. Тож ширину зображення у точках можемо отримати множачи ширину у дюймах на dpi цього зображення; аналогічно і для висоти.

Відмітимо, що для задач розпізнавання зображень за допомогою ШНМ абсолютно не важливо яким є dpi, чи реальні розміри зображення у метричних одиницях; тут навпаки єдино важливим є розмір зображення у точках (пікселах).

Тотальна залежність всього процесу розпізнавання від розміру зображення у точках абсолютно природним чином пояснюється відповідністю усієї кількості пікселів у зображенні та числом нейронів вхідного шару ШНМ: для правильної роботи системи вони мають бути ідентичними (якщо говорити точніше, для напівтонових зображень кількість пікселів та кількість вхідних нейронів рівні, а для кольорових – нейронів має бути втричі більше, бо на кожну компоненту кольору R, G, B кожного піксела має бути присутнім окремий нейрон).

Отже, кількість пікселів у висхідному зображенні має бути фіксованою і чітко корелює з числом нейронів вхідного шару (рівна, або нейронів втричі більше), що є постійною величиною, входить до самої структури НС, і яка не може змінюватися. Однак, і дотримання потрібної кількості пікселів не є достатньою умовою адекватної роботи усієї системи, оскільки розподіл цих пікселів по рядках та стовпцях також має бути витриманий.

Наприклад, зображення 320x240 дає 76800 пікселів, які, в загальному випадку, можуть бути розподілені у довільному зображенні різними способами (400x192, 800x96, і т.д.). При обробці ж зображення нейронною

мережею вона зчитує відомості єдиним рядком, тобто зовсім різні для ока людини зображення можуть бути розпізнані мережею, як ідентичні. Наприклад, на рис. 1.4 показано два очевидно різних зображення, які тим не менше, нейронною мережею, яка зчитує усі пікселі в один одновимірний масив, будуть сприйматися як ідентичні.

а) *б*)

Рис. 1.4. Два зображення, різних для людини, але однакових для нейронної мережі, що зчитує усі пікселі зліва-направо згори-донизу в один єдиний вектор: *а* – 9х7, *б* – 7х9.

Для уникнення неправильного розпізнавання необхідно слідкувати за тим, аби пропорції зображення, що подається на розпізнавання були строго витримані відповідно до технічних вимог системи. Для цього із зображення, що, окрім символів, які підлягають розпізнаванню, може містити ще багато додаткової інформації, слід вирізати необхідну ділянку, а при цьому слідкувати за її пропорціями. Якщо наприклад, еталонні зразки літер мають розмір 7х9, то і частини зображення, які містять символи для розпізнавання і вирізаються із більшою картинки, також повинні мати відповідні пропорції (наприклад, 14х18, 21х27, і т.д.). Отже, забезпечувати пропорційність зображення для розпізнавання можна одразу під час його вирізання (локалізації) на висхідній картинці.

Іншим шляхом забезпечення пропорційності є його масштабування лише в одному із напрямків. При цьому слід визначити, який лінійний розмір «не дотягує» до пропорції 7х9 і відповідно змасштабувати лише його.

Наприклад, вирізане із більшої картинки зображення символу має розмір 140х200, а для забезпечення пропорційності ширина має відноситися до висоти як 7:9. Відповідно, можна зменшити вертикальний розмір

зображення у 200/(9*(140/7)) = 200/180 = 1,111 разів і, отже, привести його до розміру 140x180, що відповідає співвідношенню 7:9.

Після такого перетворення, ймовірно отримане пропорційне зображення потрібно буде змасштабувати в меншу сторону, щоби привести кількість пікселів до числа нейронів вхідного шару. Якщо, наприклад, кількість нейронів рівна 63 (7x9), то отримане вище зображення розміром 140x180 потрібно буде зменшити в 20 разів.

Таким чином, для забезпечення першої умови успішного розпізнавання зображення, а саме, відповідності його розміру до виставлених вимог, слід зробити наступні дії:

- вирізати із усієї картинки область, у якій безпосередньо розміщений символ, що підлягає розпізнаванню;

- розтягнути чи стиснути її вздовж одного напрямку, метою чого є приведення розміру зображення до заданого (у інструкції до програми) співвідношення;

- зменшити зображення у необхідну кількість разів, щоби загальна кількість пікселів стала рівною числу нейронів вхідного шару (для чорно-білих зображень) або була втричі меншою (для кольорових).

Наступною вимогою до зображення, що має забезпечувати його успішне розпізнавання, є правильна орієнтація всієї картинки. При цьому можна виділити три варіанти розміщення символу, що підлягає розпізнаванню:

- символ розміщений вірно;

- символ повернутий за чи проти стрілки годинника на 90 градусів, або перевернутий догори ногами;

- символ повернутий на довільний кут відносно осі, перпендикулярної до площини рисунка.

Очевидно, перший випадок не потребує жодних додаткових перетворень.

Перетворення для другого випадку є порівняно простими і можуть виконуватися в автоматичному режимі за формулами:

- для зображення, повернутого на кут 90° проти стрілки годинника необхідно виконати «зворотню» операцію, тобто повернути зображення на цей же кут, але за стрілкою. Для цього потрібно виконати перетворення, схоже на транспонування матриці, складеної із пікселів, у формі:

$$P_2[i][j] = P_1[n - 1 - j][i],$$

де m – кількість рядків у матриці, яка має подаватися на вхід нейронної мережі;

n – кількість стовпців у матриці, яка має подаватися на вхід нейронної мережі.

- для зображення, повернутого на кут 90° за стрілкою годинника необхідно також виконати «зворотню» операцію, тобто повернути зображення на кут 90°, але проти стрілки годинника. Для цього потрібно виконати перетворення матриці, складеної із пікселів, аналогічне до попереднього, але у трохи зміненому виді:

$$P_2[i][j] = P_1[j][n - 1 - i].$$

- для зображення перевернутого «з ніг на голову», а точніше, повернутого на кут 180 градусів слід виконати перетворення:

$$P_2[i][j] = P_1[m - 1 - j][n - 1 - i].$$

Описані перетворення є порівняно простими та можуть виконуватися для будь-якого зображення до його масштабування та приведення до заданих пропорцій, або після цих операцій (конкретний порядок виконання не є суттєво важливим).

Виконання даних операцій можливе як у ручному, так і в автоматичному режимі, причому досить просто з алгоритмічної точки зору – шляхом простого перебору усіх трьох операцій та поданні усіх трьох типів зображень на вхід нейронної мережі. Якщо на виході із високим ступенем впевненості буде діагностована наявність якогось символу, то очевидно, що у

початковому зображенні цей символ і був, але у повернутому виді. Таким чином, можливим є простий перебір зображення у всіх чотирьох його орієнтаціях: висхідного (по-перше, слід спробувати розпізнати зображення без повороту), а потім, в довільному порядку, повернутого на +90°, –90° та 180°. Якщо в жодному з чотирьох варіантів зображення не вдалося розпізнати, високою стає імовірність того, що воно або сильно зашумлене, або взагалі не є символом.

Останній тип повороту, який не був розглянутий вище, це – поворот на довільний кут, який з очевидних причин виявляється набагато складнішою проблемою, ніж попередні, і тому не буде розглядатися в рамках цього дослідження. Відмітимо лише, що поворот прямокутної матриці пікселів на довільний кут не тільки важко описується математично, а й складно приводиться знову до прямокутної форми (яка є потрібною для розпізнавання), та ще й складно детектується в автоматичному режимі (вище описана можливість повного перебору чотирьох варіантів повороту: на 0°, +90°, -90° та 180°, але реалізувати так само перебір довільних кутів не представляється можливим через їх велику кількість – наприклад 360 при кроці в один градус чи навіть 180 при кроці в 2°).

Останнім фактором, який суттєво впливає на результат процесу розпізнавання є якість зображення (яке могло бути попередньо приведено до необхідних пропорцій, смасштабовано та повернуто, а, можливо, подається на вхід мережі в оригінальному вигляді). Під цією характеристикою слід розуміти цілий комплекс факторів, який включає:

- зашумленість усієї картинки окремими пікселами;

- забрудненість зображення плямами, розміри яких лише в рази менше розмірів самого зображення;

- адитивне накладання інших зображень (на зразок водяних знаків, і т.п.);

- викривлення даних про кольорові особливості цілих груп пікселів (особливо таких, інформація про які передається строго послідовно – наприклад про пікселі одного рядка);

- тощо.

Гарним кроком перед подачею не дуже якісного зображення на вхід нейронної мережі було би попереднє покращення його якості (якщо, звичайно, це можна зробити доступними методами та засобами). Розберемо способи, якими це можна виконати.

Безліч підходів до поліпшення зображень розпадається на дві великі категорії: методи обробки в просторовій області (просторові методи) і методи обробки в частотній області (частотні методи). Термін просторова область відноситься до площини зображення як такий, і дана категорія об'єднує підходи, засновані на прямому маніпулюванні пікселями зображення. Методи обробки в частотній області ґрунтуються на модифікації сигналу, що формується шляхом застосування до зображення перетворення Фур'є (причому перетворення Фур'є тут мається на увазі в широкому сенсі, тобто по будь-якій системі ортогональних функцій, а не обов'язково по сімействам косинусів та/або синусів). Разом з цим не є даремними і технології, що базуються на різних комбінаціях методів з цих двох категорій.

Основними серед методів обробки в просторової області є такі:

- зміна контрасту;

- видозміна гістограми;

- зменшення шуму з використанням лінійних і нелінійних методів;

- підкреслення границь.

Всі ці методи направлені на підвищення візуальної якості зображень, причому як об'єктивної, так і суб'єктивної, заснованої на особливостях сприйняття мозку людини. Наприклад, при підкресленні границь суб'єктивно зображення сприймається як зображення з більш високою роздільною здатністю, і т.п.

Частотні методи поліпшення якості зображень подібно до просторових також направлені на підвищення візуальної якості, однак для свого виконання вимагають дуже багато обчислень, оскільки ґрунтуються на двовимірних ортогональних перетвореннях типу перетворення Фур'є, Уолша-Адамара, Карунена-Лоева та інших аналогічних.

Загальної теорії поліпшення зображень не існує. Коли зображення обробляється для візуальної інтерпретації, спостерігач є остаточним суддею того, наскільки добре діє конкретний метод. Візуальне оцінювання якості зображення є украй суб'єктивний процес, що робить тим самим поняття «хорошого зображення» деяким невловимим еталоном, за допомогою якого необхідно порівнювати ефективність алгоритму. Коли метою з'являється обробка зображення для машинної обробки, завдання оцінювання дещо простіше. Наприклад, в завданні розпізнавання символів якнайкращим (залишаючи осторонь інші питання, як обчислювальні вимоги) буде той метод обробки зображень, який дає точніші результати машинного розпізнавання. Проте, навіть за ситуації, коли проблема дозволяє встановити чіткі критерії якості, зазвичай потрібна деяка кількість спроб тестування, поки буде вибраний конкретний підхід до поліпшення зображень.

В більшості практичних випадків якість зображення можна розрахувати мірою близькості двох зображень: ідеального і реального або перетвореного і початкового. Математично у якості цієї міри можна взяти усереднену середньоквадратичну помилку:

$$e^2 = \frac{1}{m \cdot n} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} E(P_1[i][j] - P_2[i][j])^2,$$

де m x n – як і раніше, розміри зображення у пікселах;

$E(\bullet)$ – математичне сподівання;

$P_1[i][j]$ – значення відліків зображення з дефектами;

$P_2[i][j]$ – значення відліків зображення без, або з меншою кількістю дефектів (наприклад, виправленого одним із методів покращення якості).

В експериментах мірою середньоквадратичної помилки служить середнє значення поелементних середньоквадратичних помилок:

$$\bar{e}^2 = \frac{1}{m \cdot n} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (P_1[i][j] - P_2[i][j])^2 .$$

На основі приведених вище залежностей можна визначити відношення сигнал/шум:

$$SNR = 10 \lg \frac{255^2}{\bar{e}^2} .$$

У чисельнику заданий розмах значень відеоданих, що звичайно задаються у виді дискретних відліків, проквантованих на 256 рівнів.

Критерій середньоквадратичної помилки є природньою мірою спотворень з фізичної і математичної точки зору, тобто якраз його зручно застосовувати для попередньої (часто автоматизованої) обробки зображень перед завантаженням у ШНМ для розпізнавання.

Таким чином, до зображення слід застосовувати методи покращення якості, перелічені вище та інші, в т.ч. такі, що лише розробляються, причому слід це робити перед застосуванням інших перетворень, описаних раніше.

Отже, підсумовуючи можна сказати, що, залежно від походження зображень, перед завантаженням їх для розпізнавання на вхід нейронної мережі, їх, можливо, слід обробити одним, чи декількома згаданими тут алгоритмами:

- виділення суттєвої частини зображення із усієї сукупності його пікселів (вирізання символу);

- виконання фільтрації та/або інших операцій по підвищенню якості висхідного зображення;

- виконання повороту зображення за наявності такої необхідності (на один із стандартних кутів: ± 90° або 180°;

- приведення зображення вирізаного символу до необхідного співвідношення висоти і ширини (для цього розтягуємо зображення по одному із напрямків: по висоті, чи по ширині);

- приведення отриманого пропорційного зображення до точних розмірів шляхом пропорційного масштабування (у вихідному зображенні кількість пікселів має бути рівною кількості нейронів вхідного шару нейронної мережі);

- передача отриманого підготовленого зображення на вхід нейромережі.

Усі ці етапи є можливими, але не обов’язковими, якщо, зокрема, зображення уже підготовлене до розпізнавання, тому, зважаючи на конкретну тематику дослідження, в подальшому будемо працювати уже із підготовленими належним чином зображеннями, що будуть подаватися на вхід нейромережі.

РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЦЕСУ РОЗПІЗНАВАННЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ

Відповідно до завдання на дипломну роботу, необхідно розробити програмне забезпечення, що використовує нейронну мережу (порівняно просту – у вигляді персептрону) для виконання типових задач класифікації (а саме, розпізнавання текстових символів). Говорячи простою мовою, слід створити програму для розпізнавання символів, заданих своїм графічним представленням.

У якості символів можна обирати літери англійського алфавіту (латиниці), кириличні символі, або цифри. При розпізнаванні літер у порівнянні з цифрами, кількість нейронів у мережі має бути збільшеною у кілька разів. Відповідно, на підготовку, навчання такої мережі іде у декілька разів більше часу, хоча принципово такі мережі не будуть відрізнятися одна від одної. У якості компромісу оберемо літери латинського алфавіту, яких всього існує 26.

2.1. Вибір методики розпізнавання та опис її особливостей.

Як уже було встановлено у попередньому розділі, за основу системи розпізнавання взято багатошаровий персептрон. Для цілей розпізнавання символів цілком достатньо трьох шарів:

- вхідного шару (це умовний шар, для виходів вузлів якого не застосовуються функції активації, і він необхідний лише для організації можливості подання вхідних сигналів на усі нейрони прихованого шару);
- одного прихованого шару;
- вихідного шару.

Розберемо характеристики цих шарів.

На вхідний шар подаються одновимірні вектори, які формуються на основі зображення, що підлягає розпізнаванню. Будь-яке зображення є

двовимірним, а вхідний вектор – одновимірний, тому має бути зафіксований відповідний механізм переходу між цими структурами даних. Стандартним підходом при цьому є запис по одному рядку, починаючи від верхнього рядка зліва-направо і далі зверху-вниз.

Проектована програма буде працювати з зображеннями формату BMP, причому звичайними, 24-бітними. При цьому на кожен піксель відводиться чотири байти (32-бітне число типу int мови C/C++) – таким є представлення кольору у форматі TColor. Конкретніше, у 16-річній формі формат кольору матиме вигляд:

0x00bbggrr,

де bb – байт, що відповідає за синій кольор, gg – байт зеленого кольору, rr – байт червоного. Наприклад, 0x00FF0000 – відповідає найінтенсивнішому синьому, 0x0000FF00 – найінтенсивніший зелений, 0x000000FF – найінтенсивніший червоний, 0x00FFFFFF – відповідає максимально білому, а 0x00000000 – найтемнішому чорному кольору.

Проектована програма буде працювати з зображеннями, виконаними у відтінках сірого кольору. Усі сірі кольори складаються з однакової кількості всіх трьох компонентів, тобто мають вид:

0x00aaaaaa,

де aa – число від 0 до 255 включно, записане у шістнадцятирічній формі, тобто у межах від 00 до FF. Наприклад, 0x00A9A9A9, або 0x7C7C7C, і т.д.

Відповідно, для більш стабільної роботи програми можна ввести перевірку кожного пікселя, з яким працюватиме програма на те, що у нього саме сірий колір. Для цього реалізована функція IsColourGray(), в коді якої окремо виділяються три байти: синій, зелений та червоний і порівнюються. Якщо всі вони троє не рівні, то функція повертає у місце свого виклику 0 (як ознака невідповідності переданого у якості аргументу кольору до сірого), або 1, якщо переданий колір є сірим.

```
int IsColourGray(TColor col)
{
    char red,green,blue;
    red=col&255;
    green=(col>>8)&255;
    blue=(col>>16)&255;
    if((red!=green)||(green!=blue))
        return 0;
    else
        return 1;
}
```

Доступ до кольорів виконується за допомогою об’єкта типу TBitmap, його властивості Canvas, та її властивості – масиву Pixels[x][y]. Кожен елемент цього масиву – 32-бітний колір у форматі TColor пікселя з координатами x, y. Ці координати відраховуються від верхнього лівого кута зображення по осям x (направлена вправо) та y (направлена вниз).

Розмір зображення (у пікселях) прямим чином впливає на величину нейронної мережі, потрібної для його розпізнавання, що в свою чергу сильно впливає на час навчання мережі. Навчання є обов’язковим етапом експлуатації будь-якої нейронної мережі, і, хоча і виконується лише один раз, при значній тривалості може повністю унеможливити використання мережі. Тому розмір зображення, нажаль, не може сягати навіть десятків пікселів. Реальними для використання значеннями є ширина і висота зображення до десяти пікселів. При цьому слід ураховувати, що, зважаючи на білатеральну симетрію значної кількості літер, ширину зображення краще робити непарною; також слід врахувати, що висота має бути дещо більшою за ширину. Кінцево приймаємо розмір зображення для розпізнавання 7x9 пікселів (ширина 7, висота 9). Відповідно, розмір вхідного вектору дорівнюватиме 63 елементи (як буде показано нижче, навчання навіть такої, порівняно невеликої, мережі займатиме час порядку кількох хвилин, протягом яких користувач програми просто чекатиме).

Таким чином, для заповнення вхідного вектору inputvector використовується наступний цикл:

```
for(int i=0;i<Nneuronsinlayer[0];i++)
{
  if(!IsColourGray(bmp->Canvas->Pixels[i%7][i/7]))
  {
    ShowMessage("Image must be grayscale! Terminating...");
    exit(0);
  }
  gray=(char)(bmp->Canvas->Pixels[i%7][i/7])&255;
  gray=255-gray;
  inputvector[i]=(float)gray/255;   //1 - black, 0 - fully white
}
```

По-перше, за допомогою вищеописаної функції `IsColourGray()` відбувається перевірка того, що колір поточного пікселя є сірим, інакше видається попереджувальне повідомлення та відбувається вихід із програми. Далі, оскільки всі три компоненти однакові, виділяється червона компонента і зберігається у однобайтовій змінній gray. Далі виконується психологічно більш зручний перехід, після якого білий колір стає 0, а чорний – 255. Усі інші відтінки сірого задаються числами від 0 до 255. Останній рядок функції переводить колір у число від 0 до 1 (включно) так, що 0 – є чисто білим, а 1 – повністю чорним, а проміжні сірі кольори задаються дробовими числами від 0 до 1.

Отже, вхідний шар нейронів (умовний, до виходів якого не застосовується функція активації) має 63 одиниці. Вихідний шар можна задати наступним чином: всього існує 26 результатів для розпізнавання, тому і нейронів у вихідному шарі зробимо 26. В процесі навчання на ідеальній вибірці (на еталонних зразках) будемо домагатися, щоби на всіх виходах вихідного шару спостерігалися нулі, крім одного виходу, що відповідає тій літері, яка подається у даний момент на вхід мережі: на цьому виході має бути 1. Після завершення навчання, тобто при звичайному робочому процесі розпізнавання, на кожному виході нейронної мережі формуватиметься певне число, а той вихід, на якому спостерігатиметься найбільший результат, і відповідатиме розпізнаній літері.

Окрім вхідного та вихідного, в мережі можуть існувати також приховані (проміжні) шари нейронів. Загальноприйнято, що одного

прихованого шару достатньо для ефективного вирішення усіх задач, для яких взагалі можна застосовувати персептрони. Впроваджувати більшу кількість прихованих шарів, ніж один, немає ніякого сенсу. Кількість нейронів у прихованому шарі не повинна бути меншою за мінімальну з двох кількостей нейронів: вхідного та вихідного шару (інакше мережа матиме форму пісочного годинника із вузьким місцем у прихованому шарі, і, можливо, може втрачати інформацію).

Отже, кінцево число шарів приймаємо рівним 3 (причому один вхідний шар є, фактично, фіктивним, бо його виходи просто повторюють вхідні сигнали без застосування функції активації), а число нейронів у них буде 63-26-26. Кількості нейронів прописуємо у масиві Nneuronsinlayer:

```
Nneuronsinlayer[0]=63;
Nneuronsinlayer[1]=26;
Nneuronsinlayer[2]=26;
```

Структура отриманої мережі зображена на рис. 2.1.

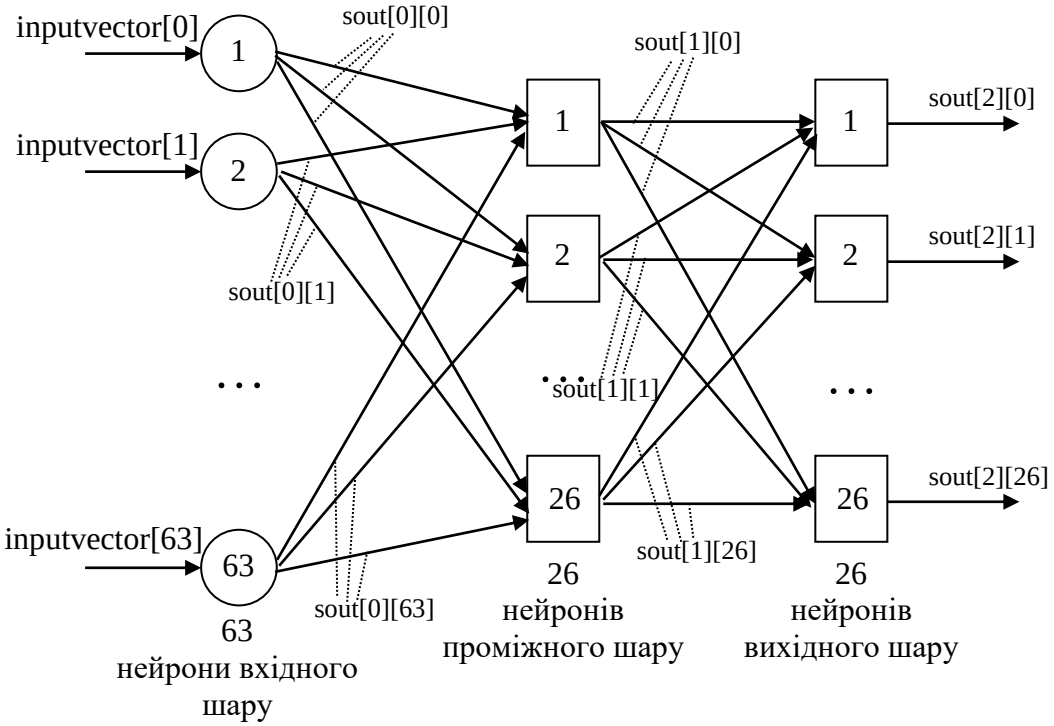


Рис. 2.1. Структура проектованої нейронної мережі із зазначенням сигналів, що спостерігаються на виходах відповідних нейронів.

Кожному зв’язку між нейронами, що показаний на рис. 2.1, відповідає певна вага, що використовується в розрахунках вихідних сигналів sout. Ваги є надзвичайно важливим поняттям, тому проілюструємо правило, за яким формується індексація ваг у розроблюваній програмі – рис. 2.2 (перший індекс – номер шару, з якого виходить зв’язок, другий індекс – номер висхідного нейрону, третій – номер кінцевого нейрону).

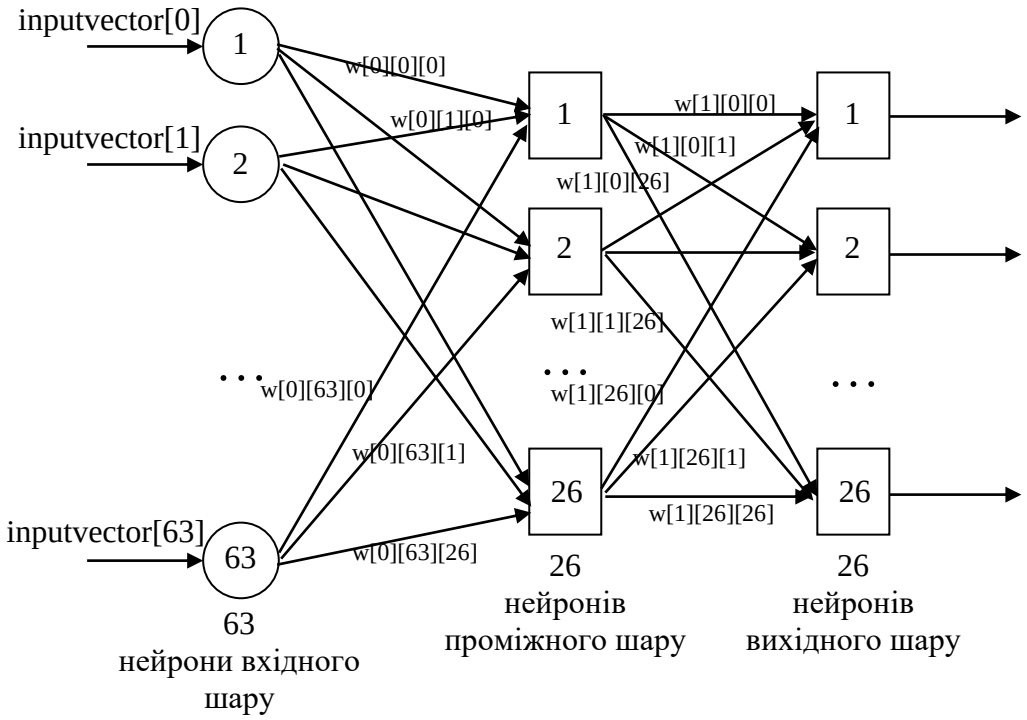


Рис. 2.2. Індексація ваг синаптичних зв’язків між нейронами.

З використанням сигналів sout між нейронами та ваг синаптичних зв’язків w, формуються зважені суми, які стають аргументами функції активації:

$$\text{sout[k+1][i]=Function}\left(\sum_{j=0}^{N_k} \text{sout[k][j]*w[k][j][i]}\right), \qquad (2.1)$$

де N_k – число нейронів у k-тому шарі, тобто Nneuronsinlayer[k];

Function – активаційна функція, яка найчастіше задається як сигмоїдна, що реалізує залежність:

$$f(x) = \frac{1}{1+e^{-\alpha x}}, \qquad (2.2)$$

де α - параметр, за допомогою якого можна управляти роботою нейромережі; приймаємо $\alpha = 0{,}05$.

Залежності (2.2) відповідає функція на мові C/C++:

```
float SigmoidalFunction(float arg)
{
    float res=(float)1/(1+exp(-alfa*arg));
    return res;
}
```

Проведення вхідних сигналів, заданих масивом inputvector, через усю мережу виконується функцією Propagate():

```
void Propagate(float *inputarray)
{
    for(int i=0;i<Nneuronsinlayer[0];i++)
        sout[0][i]=(float)*(inputarray+i*sizeof(float));
    float sum;
    for(int k=0;k<2;k++)
        for(int i=0;i<Nneuronsinlayer[k+1];i++)
        {
            sum=0;
            for(int j=0;j<Nneuronsinlayer[k];j++)
                sum+=sout[k][j]*w[k][j][i];
            sout[k+1][i]=SigmoidalFunction(sum);
        }
}
```

Аргументом цієї функції є стартова адреса inputarray масиву, в якому міститься вхідний вектор. Спочатку у функції зчитуються усі елементи цього float масиву (за допомогою звернення за адресою указника `inputarray` і зміщення `i*sizeof(float)`) і формується перший набір сигналів `sout[0][i]`, що є виходами нейронів вхідного (нульового) шару.

По-друге, задається змінна sum, у якій буде зберігатися зважена сума вхідних сигналів для і-того нейрону. Вихідні сигнали формуються два рази: для прихованого шару та для вихідного шару. Кожного разу функція активації застосовується до зваженої суми усіх сигналів, що підходять до і-того нейрону. Таким чином, в кінці роботи функції Propagate() буде повністю сформованим набір вихідних сигналів для усіх нейронів (що зберігається в масиві sout).

Практичний інтерес представляє набір вихідних сигналів вихідного шару sout[2][i], серед яких слід знайти найбільший за величиною сигнал, що і надає відповідь-рішення нейронної мережі. В ідеалі це число має бути рівним одиниці, а інші виходи мають дорівнювати 0, саме на таку ситуацію навчається нейронна мережа за допомогою еталонних образів. Звичайно, коли на вхід подається не одне з еталонних зображень, а реальне зображення, то більшість виходів міститиме ненульові значення, але рішення приймається за максимальним із них.

Таким чином, алгоритм роботи із підготовленою до роботи мережею виявляється досить простим:

- завантажити зображення 7x9 пікселів у відтінках сірого у програму;

- натиснути кнопку Recognize, після чого сформується вхідний вектор, і запуститься функція Propagate, яка і виконує основну роботу;

- зчитати вихідний результат.

В той же час, алгоритмічно більш складним є підготовка мережі до розпізнавання, а саме проведення її навчання. Воно виконується методом зворотного поширення помилки (back propagation method). Для роботи цього методу необхідний ще один набір даних, у деякому сенсі помилок, нев’язок δ.

Запускає процес навчання натискання на кнопку “Train Neuronet”, після чого завантажується функція-обробник Button1Click, у якій основну частину займає цикл типу do-while, у якому виконується ітеративний процес навчання шляхом багатократного виклику функції TeachEpoch(), яка здійснює один прохід по всім еталонним зразкам, якими є літери від A до Z. Ці еталонні зразки зберігаються у фіксованому каталозі Training, який має бути розташований у тому ж каталозі, що й виконуваний файл програми.

Функція TeachEpoch() повертає найбільшу помилку, що спостерігається серед усіх виходів нейронної мережі. Виклик цієї функції продовжується до тих пір, поки ця максимальна помилка не стане меншою 10%.

Нажаль, з теорії нейронних мереж відомо, що в процесі навчання можливе виникнення небажаних ситуацій, коли мережа «перенавчається» і замість поступового зменшення максимальної помилки, вона починає знову зростати. Це зростання може бути тимчасовим (тоді це нормально і після невеликого збільшення помилки, вона починає знову спадати – процес навчання іде нормально), або стійким (що не нормально і свідчить про «перенавчання» мережі). Для контролю виникнення цього процесу у циклі впроваджено ще одну умову виходу, а саме, якщо протягом останніх 1000 викликів функції TeachEpoch() максимальна помилка невпинно зростає (замість того, щоб зменшуватися), то вважаємо, що мережа «перенавчена». В цьому випадку зупиняємо процес навчання і мережа не стає максимально навченою до самого кінця, але у даних умовах цей стан навчання є найкращим із можливих.

Згадані вище нев'язки приписуються до виходу кожного нейрону: скільки активних нейронів (тобто таких, що вираховують активаційну функцію, а не є простими повторювачами-розгалуджувачами, як нейрони вхідного шару), стільки ж і нев'язок δ. Ці величини обчислюються за двома різними формулами:

- більш простий варіант обчислення δ реалізується для нейронів вихідного шару:

delta[1][i]=sout[2][i]*(1-sout[2][i])*(outputvector[i]-sout[2][i]),

де sout[2][i] – наявний вихід і-того нейрону вихідного шару;

outputvector[i] – бажаний (з еталону) вихід і-того нейрону вихідного шару.

- складніший варіант обчислення δ слід використовувати для нейронів усіх інших шарів (не вихідного), а в проектованій мережі такий шар лише один і його нев'язки будуть:

$$\text{delta}[0][i]=\text{sout}[1][i]*(1-\text{sout}[1][i])*\sum_{m=0}^{N}(\text{delta}[1][m]*w[1][i][m]).$$

Далі на основі отриманих значень нев’язок δ корегуються ваги усіх синаптичних зв’язків за формулою (справедлива для усіх шарів):

w[1][j][i]+=eta*delta[1][i]*sout[1][j],

де величина eta грає роль швидкості навчання і у простому випадку може бути постійною (можливі варіанти і старту з великим eta та поступовим його зменшенням, а також варіант із стартом з малого значення, поступовим збільшенням та фінальним зменшенням до нуля). У даній роботі ефективним значенням прийнято $eta=10^{-5}$.

Таким чином, алгоритми роботи мережі є більш-менш стандартними та добре відомими (пророблениими). Перейдемо до вибору технології та засобів розробки.

2.2. Вибір мови програмування та засобів розробки.

Для вирішення задачі, що розглядається, можна використовувати будь-яку мову програмування загального призначення, причому кожна, як відомо має свої особливості. В той же час найбільш універсальною є мова програмування C/C++. Для програмування цією мовою існує величезна кількість засобів розробки. Для цілей даної роботи було обрано середовище Borland C ++ Builder. Вибір обумовлений великою швидкістю розробки візуального інтерфейсу в даному середовищі. Створення макета вікна програми (форми - по термінології Borland) відбувається в наочному, інтуїтивно зрозумілому режимі, шляхом простого перетягування потрібних компонентів на форму і подальшої їх тонкої настройки за допомогою вікна Object Inspector.

Даний підхід до програмування, застосований також і в середовищі програмування Delphi, був перевірений часом і є найзручнішим. Для обґрунтування цієї тези досить сказати, що найбільш «сучасна» і розрекламована мова програмування від Microsoft - C# (сі-Шарп) створена саме за такою схемою, але пізніше, ніж продукти Borland (фактично

інтерфейс його середовища розробки взято з продуктів Borland). Ймовірно, засоби від Borland (C ++ Builder і Delphi) не стали так популярні серед професіоналів, як зараз C#, через слабку маркетингову політику компанії, на відміну від Microsoft, яка всюди проводить інтеграцію декількох продуктів і продає, таким чином, відразу все (C# з самого початку свого існування був включений в комплект Microsoft Visual Studio і поставлявся разом з іншими мовами програмування типу Visual Basic, Visual C++, Visual J++, і т.д.).

Ще однією перевагою продуктів від Borland є наявність значної кількості компонентів (тобто класів) та їх бібліотек, за допомогою яких можна зручно вирішувати досить високорівневі завдання досить ефективними методами.

І також безперечним плюсом продукту Borland C++ Builder є механізми спрощення деяких простих рутинних операцій, що при виконанні на чистій мові C/C++ потребують багато часу, зокрема роботи з текстовими рядками. Це середовище пропонує зручні класи AnsiString, що порівняно легко інтегруються з указниками мови C, що активно використовуються у проектованому програмному забезпеченні.

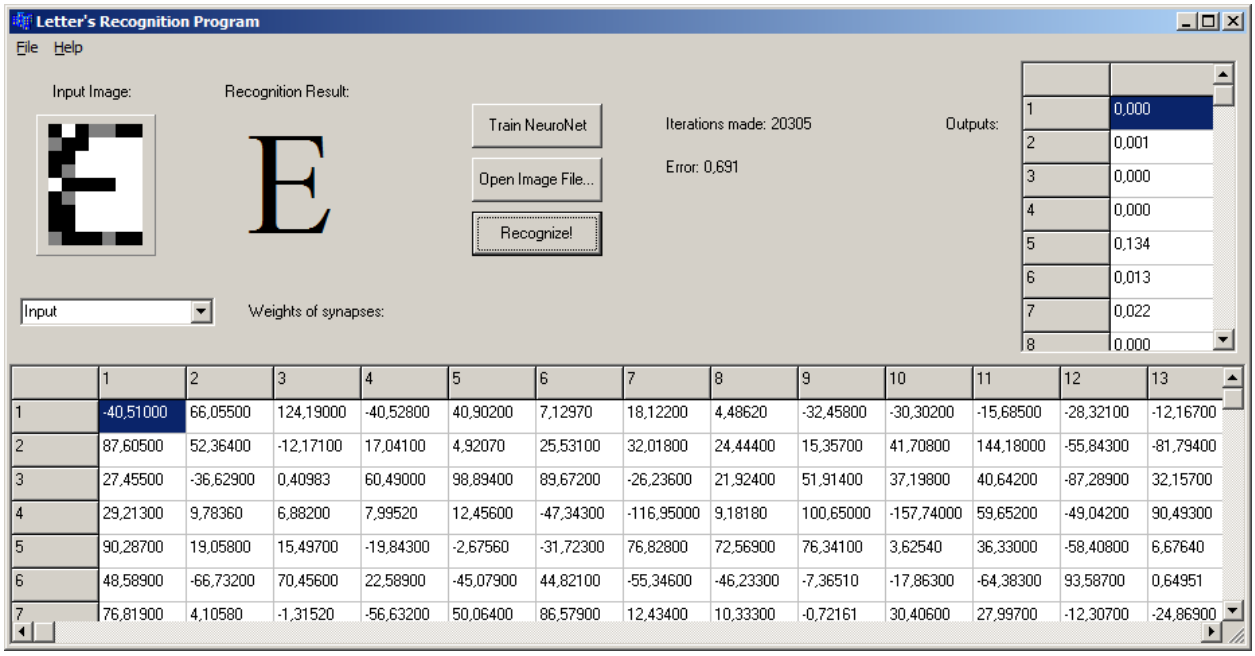
Таким чином, використання середовища розробки C++ Builder дозволяє поєднувати з одного боку швидкість розробки в стилі компонентного програмування (особливо, в частині створення призначеного для користувача інтерфейсу), а з іншого - гнучкі, як завгодно низькорівневі, можливості мови C/C++.

2.3. Проектування програмного забезпечення для розпізнавання текстової інформації.

Першим кроком при проектуванні будь-якого ПЗ є розробка його інтерфейсу користувача.

Зважаючи на перелік деяких стандартних дій, що виконуються при роботі з нейронними мережами (початкове завдання коефіцієнтів-ваг

синапсів, ініціювання процесу навчання, тобто підстройки цих ваг під вирішення конкретних задач розпізнавання за шаблонами, тощо), реалізуємо інтерфейс користувача у наступному вигляді – рис. 2.3.



	1
1	0,000
2	0,001
3	0,000
4	0,000
5	0,134
6	0,013
7	0,022
8	0,000

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-40,51000	66,05500	124,19000	-40,52800	40,90200	7,12970	18,12200	4,48620	-32,45800	-30,30200	-15,68500	-28,32100	-12,16700
2	87,60500	52,36400	-12,17100	17,04100	4,92070	25,53100	32,01800	24,44400	15,35700	41,70800	144,18000	-55,84300	-81,79400
3	27,45500	-36,62900	0,40983	60,49000	98,89400	89,67200	-26,23600	21,92400	51,91400	37,19800	40,64200	-87,28900	32,15700
4	29,21300	9,78360	6,88200	7,99520	12,45600	-47,34300	-116,95000	9,18180	100,65000	-157,74000	59,65200	-49,04200	90,49300
5	90,28700	19,05800	15,49700	-19,84300	-2,67560	-31,72300	76,82800	72,56900	76,34100	3,62540	36,33000	-58,40800	6,67640
6	48,58900	-66,73200	70,45600	22,58900	-45,07900	44,82100	-55,34600	-46,23300	-7,36510	-17,86300	-64,38300	93,58700	0,64951
7	76,81900	4,10580	-1,31520	-56,63200	50,06400	86,57900	12,43400	10,33300	-0,72161	30,40600	27,99700	-12,30700	-24,86900

Рис. 2.3. Загальний вигляд вікна програми для розпізнавання літер.

У нижній частині вікна програми можна переглядати ваги синаптичних зв’язків між усіма нейронами мережі: зверху над таблицею у списку, що випадає можна вибрати шар нейронів, із якого виходять зв’язки, ваги яких відображуються у табличці. Зважаючи на загальну архітектуру мережі, існує два доступних для відображення шари нейронів мережі прямого поширення: вхідного та прихованого (із вихідного шару синаптичних зв’язків уже не виходить).

У верхній середній частині вікна програми розмішено кнопку для приведення системи до робочого стану – “Train Neuronet”. Відмітимо, що початково ваги усіх синапсів задаються випадковим числом від -1 до 1, причому це виконується автоматично при завантаженні програми. Після цього необхідно навчити нейромережу розпізнаванню набору еталонних символів, що зберігаються у папці Training, а сам старт навчання здійснюється по натисненню на кнопку «Train Neuronet».

Процес навчання нейромережі підпадає під категорію «з учителем», та спирається на 26 картинок-еталонів, що зарані були підготовлені у програмі MS Paint. Цією ж програмою рекомендується обробляти файли BMP перед їх відкриттям у проектованій програмі для розпізнавання – рис. 2.4. При цьому зручно використовувати такі функції:

- зміна розміру зображення (має бути 7x9 пікселів);

- безпосереднє рисування літери за допомогою олівця (при цьому слід використовувати тільки сірі кольори).

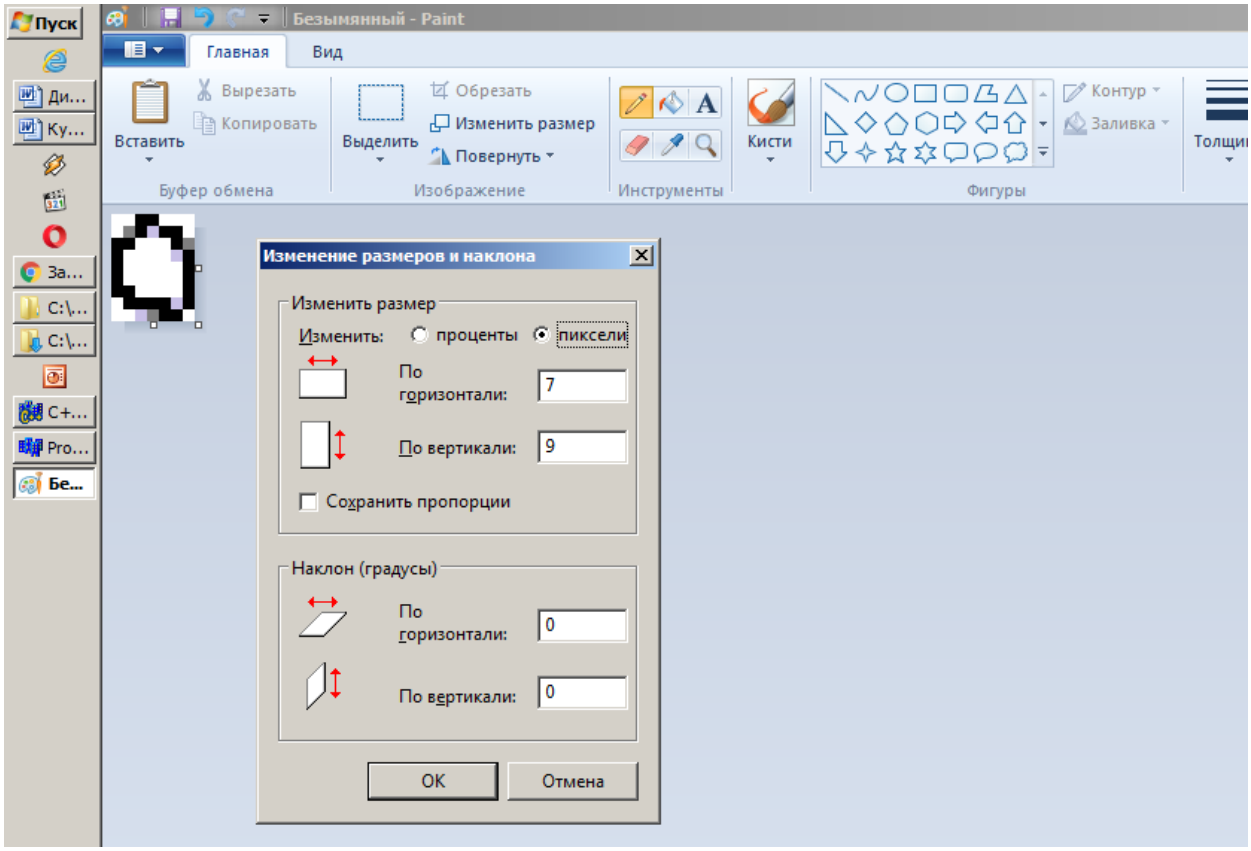


Рис. 2.4. Обробка зображення у програмі MS Paint перед завантаженням у програму розпізнавання.

Після завершення процесу навчання (який може тривати від кількох секунд до кількох хвилин) стає активною кнопка Recognize, тому слід завантажити підготовлений графічний файл (це можна зробити трьома способами: через головне меню «File»->«Open…», або натисканням на кнопку «Open Image File…», або просто подвійним щиглем мишею по місцю,

де розміщується зображення, яке підлягає розпізнаванню), та натиснути цю кнопку.

У верхній правій частині вікна програми після цього можна продивитися значення виходів нейронів вихідного шару та порівняти найбільші їх значення. Основний же варіант відповіді, тобто розпізнаний символ, відображується у верхній лівій частині вікна програми, а саме справа від зображення, що обробляється.

У якості додаткової корисної інформації аналітичного характеру програма відображує кількість епох навчання мережі (наприклад, на рис. 2.3 при навчанні було виконано біля 20 тис. прогонок функції TeachEpoch()). Там же показується найбільша помилка, якої може припускатися система (у відносних одиницях).

Крім інтерфейсу користувача, важливим є також програмний інтерфейс, тобто набір функцій, які реалізовано у висхідному тексті проектованого програмного забезпечення. Серед них:

1) float SigmoidalFunction(float arg) – математична функція для обчислення функції активації нейронів;

2) void StringGridClear(TStringGrid* tsg) – сервісна функція для очищення таблиць (компонентів типу StringGrid);

3) __fastcall TForm1::TForm1(TComponent* Owner) – конструктор головного вікна програми, що виконує усілякі первинні налаштування програми (зокрема, ініціалізує необхідні змінні системи);

4) void __fastcall TForm1::Open1Click(TObject *Sender) – метод, що викликає діалогове вікно відкриття файлу картинки;

5) void __fastcall TForm1::About1Click(TObject *Sender) – метод, що викликає діалогове вікно з інформацією про програму;

6) void __fastcall TForm1::Exit1Click(TObject *Sender) – метод-обробник натискання на пункт «Вихід» у головному меню програми;

7) void __fastcall TForm1::ComboBox1Change(TObject *Sender) – метод обробник зміни стану випадаючого списку, за допомогою якого заповнюється таблиця ваг синапсів мережі;

8) void Propagate(float *inputarray) – функція поширення сигналу через нейронну мережу: на основі вхідного вектору inputarray формує вихідні сигнали масиву sout;

9) int IsColourGray(TColor col) – функція перевірки кольору на відповідність поняттю «сірий»;

10) float TeachEpoch() – найскладніша за своєю логікою функція, що реалізує один прохід по всім еталонним образам методом оберненого поширення помилки;

11) void __fastcall TForm1::Button1Click(TObject *Sender) – обробник натискання на кнопку «Train Neuronet»: містить цикл, в якому багатократно викликається функція TeachEpoch() аж до зменшення помилки, або «перенавчання» мережі;

12) void __fastcall TForm1::Button3Click(TObject *Sender) – головна по суті функція програми, що розпізнає наданий їй графічний файл.

2.4. Результати виконання реалізації програмного забезпечення для розпізнавання текстової інформації.

Після виконання програмної реалізації певний час було витрачено на підбір вільних коефіцієнтів системи, а саме:

- кількості нейронів прихованого шару, яку кінцево встановлено на рівні кількості нейронів вихідного шару, тобто 26;

- швидкості навчання η (eta), яка кінцево встановлена на постійне значення 10^{-5};

- допустимої максимальної помилки серед усіх еталонів, яка кінцево встановлена на рівні 10% (res>0.1);

- кількості викликів функції TeachEpoch(), після якої неперервне зростання помилки вважається твердою ознакою «перенавченості» системи, яка кінцево встановлена на 1000 (dirres<1000);

- параметру α (alfa) сигмоїдальної функції, який кінцево прийнято на рівні 0.05.

Фінально налаштована програма показує непогані результати по розпізнаванню навіть досить зашумлених зображень – рис. 2.5.

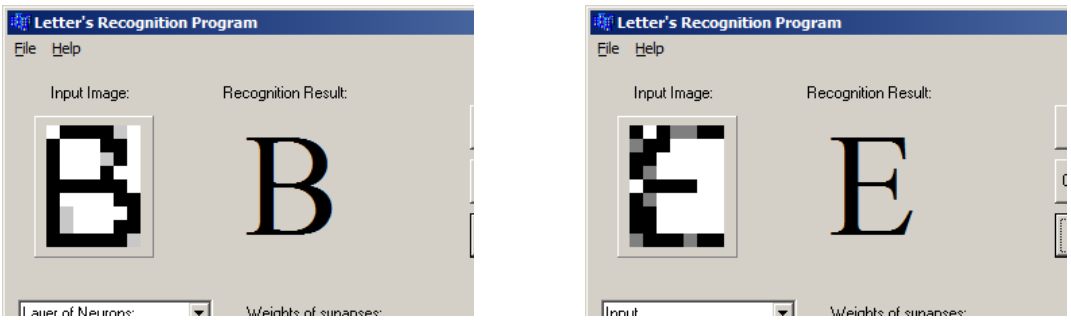


Рис. 2.5. Успішні результати розпізнавання зашумлених зображень.

В той же час навіть і деякі еталонні образи, що дуже схожі між собою, можуть розпізнаватися неправильно – рис. 2.6.

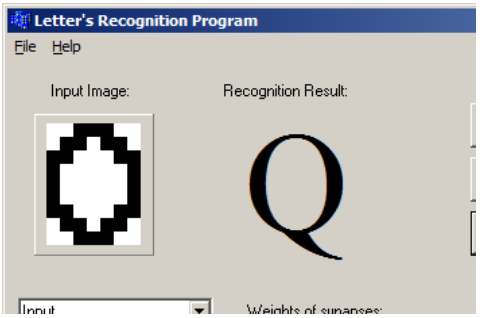


Рис. 2.6. Невірний результат розпізнавання еталонного зображення.

Боротися з цією ситуацією можна підбором еталонних образів схожих літер, що якомога сильніше відрізнялися б між собою.

В цілому, програмне забезпечення працює на задовільному рівні, справляючись з початковим завданням, а наявність помилок розпізнавання існує навіть у професійних програмних продуктів класу OCR.

ВИСНОВКИ

В роботі розглянуто нейронні мережі, які є зручним і ефективним інструментом для вирішення задач деяких специфічних класів, на які, в тому числі націлено і мозок людини. Виконано докладний аналіз таких задач на предмет розв'язування за допомогою нейронних мереж. Також проаналізовано властивості основної структурної одиниці при побудові нейронних мереж - штучного нейрону. Ще виконано аналіз способів об'єднання окремих нейронів у цілі мережі, із зазначенням особливостей мереж різних типів.

У якості найпростішого і в той же час найефективнішого варіанту організації нейронної мережі взято багатошаровий персептрон, на основі якого розроблено програмне забезпечення, що здійснює розпізнавання літер англійського алфавіту. Застосована мова програмування C/C++, середовище розробки Borland C++ Builder (обране в першу чергу через легкість зведення та налаштування інтерфейсу користувача). В результаті досліджень програми встановлено, що у більшості випадків мережа здійснює розпізнавання вірно, але іноді зустрічаються помилки. Очевидно, що для підвищення точності розпізнавання необхідно покращувати якість процесу навчання та комплект еталонних образів, що і має бути зроблене у подальшій роботі за наявності необхідності її удосконалення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bhadeshia H. K. D. H. (1999). Neural Networks in Materials Science. ISIJ International 39 (10): 966–979.
2. M., Bishop, Christopher (1995). Neural networks for pattern recognition. Clarendon Press.
3. Cybenko, G.V. (2006). Approximation by Superpositions of a Sigmoidal function. У van Schuppen, Jan H. Mathematics of Control, Signals, and Systems[en]. Springer International. с. 303–314.
4. Dewdney, A. K. (1997). Yes, we have no neutrons : an eye-opening tour through the twists and turns of bad science. New York: Wiley.
5. Duda, Richard O.; Hart, Peter Elliot; Stork, David G. (2001). Pattern classification (вид. 2). Wiley.
6. Egmont-Petersen, M.; de Ridder, D.; Handels, H. (2002). Image processing with neural networks – a review. Pattern Recognition 35 (10): 2279–2301.
7. Gurney, Kevin (1997). An introduction to neural networks. UCL Press.
8. Haykin, Simon S. (1999). Neural networks : a comprehensive foundation. Prentice Hall.
9. Hertz, J.; Palmer, Richard G.; Krogh, Anders S. (1991). Introduction to the theory of neural computation. Addison-Wesley.
10. Lawrence, Jeanette (1994). Introduction to neural networks : design, theory and applications. California Scientific Software.

ДОДАТОК 1. КОД РОЗРОБЛЕНОГО ОСНОВНОГО МОДУЛЯ

```
#include <vcl.h>
#include <math.h>
#include <stdlib.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"

#pragma package(smart_init)
#pragma resource "*.dfm"

const float alfa=0.05;

TForm1 *Form1;
int Nneuronsinlayer[3];
float w[2][63][63];//Weights:number of layer,neuron FROM, neuron
TO
float sout[3][63];                    //Signal levels on out of
layer
float inputvector[63],outputvector[26];
float delta[2][63];
char alf[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
float eta=1e-5;
float prevres=0;

float SigmoidalFunction(float arg)
{
    float res=(float)1/(1+exp(-alfa*arg));
    return res;
}

void StringGridClear(TStringGrid* tsg)
{
    for(int i=1;i<=tsg->ColCount;i++)
        for(int j=1;j<=tsg->RowCount;j++)
            tsg->Cells[i][j]="";
}

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Nneuronsinlayer[0]=63;
    Nneuronsinlayer[1]=26;
    Nneuronsinlayer[2]=26;
    for(int i=1;i<=63;i++)
    {
        StringGrid1->Cells[0][i]=i;
        StringGrid1->Cells[i][0]=i;
    }
    for(int i=1;i<=26;i++)
        StringGrid2->Cells[0][i]=i;
```

```
    for(int k=0;k<2;k++)
        for(int i=0;i<Nneuronsinlayer[k];i++)
            for(int j=0;j<Nneuronsinlayer[k+1];j++)
                w[k][i][j]=(float)rand()/RAND_MAX*pow(-
1,rand());
}

void __fastcall TForm1::Open1Click(TObject *Sender)
{
    Graphics::TBitmap *bmp = new Graphics::TBitmap;
    if(OpenDialog1->Execute())
    {
        bmp->LoadFromFile(OpenDialog1->FileName);
        if((bmp->Width!=7)||(bmp->Height!=9))
        {
            ShowMessage("Size of the Bitmap must be 7x9
pixels!");
            return;
        }

        Image1->Picture->LoadFromFile(OpenDialog1->FileName);
    }
}

void __fastcall TForm1::About1Click(TObject *Sender)
{
    Form2->ShowModal();
}

void __fastcall TForm1::Exit1Click(TObject *Sender)
{
    exit(0);
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
    StringGridClear(StringGrid1);
    int k=ComboBox1->ItemIndex;
    for(int i=0;i<Nneuronsinlayer[k];i++)
        for(int j=0;j<Nneuronsinlayer[k+1];j++)
            StringGrid1-
>Cells[j+1][i+1]=FloatToStrF(w[k][i][j],ffFixed,5,5);
}

void Propagate(float *inputarray)
{
    for(int i=0;i<Nneuronsinlayer[0];i++)
        sout[0][i]=(float)*(inputarray+i*sizeof(float));
    float sum;
    for(int k=0;k<2;k++)
        for(int i=0;i<Nneuronsinlayer[k+1];i++)
        {
            sum=0;
```

```
            for(int j=0;j<Nneuronsinlayer[k];j++)
                sum+=sout[k][j]*w[k][j][i];
            sout[k+1][i]=SigmoidalFunction(sum);
        }
}

int IsColourGray(TColor col)
{
    char red,green,blue;
    red=col&255;
    green=(col>>8)&255;
    blue=(col>>16)&255;
    if((red!=green)||(green!=blue))
        return 0;
    else
        return 1;
}

float TeachEpoch()
{
    char path[255];
    int gray;
    Graphics::TBitmap *bmp = new Graphics::TBitmap;
    float error=0;
    for(int k=0;k<Nneuronsinlayer[2];k++)
    {
        path[0]=0;
        strcat(path,"Training/");
        StrLCat(path,(char*)alf+k,10);
        strcat(path,".bmp");
        bmp->LoadFromFile(path);
        for(int i=0;i<Nneuronsinlayer[0];i++)
        {
            if(!IsColourGray(bmp->Canvas->Pixels[i%7][i/7]))
            {
                ShowMessage("Image     must     be     grayscale!
Terminating...");
                exit(0);
            }
            gray=(char)(bmp->Canvas->Pixels[i%7][i/7])&255;
            gray=255-gray;
            inputvector[i]=(float)gray/255;          //1 - black,
0 - fully white
        }
        for(int i=0;i<Nneuronsinlayer[2];i++)
        {
            outputvector[i]=((i==k)?1:0);
        }
        Propagate(inputvector);
        for(int i=0;i<Nneuronsinlayer[2];i++)
        {
            for(int j=0;j<Nneuronsinlayer[1];j++)
            {
```

```
                delta[1][i]=sout[2][i]*(1-
sout[2][i])*(outputvector[i]-sout[2][i]);
                w[1][j][i]+=eta*delta[1][i]*sout[1][j];
            }
        }
        for(int i=0;i<Nneuronsinlayer[1];i++)
        {
            float sum1=0;
            for(int m=0;m<Nneuronsinlayer[2];m++)
                sum1+=delta[1][m]*w[1][i][m];
            for(int j=0;j<Nneuronsinlayer[0];j++)
            {
                delta[0][i]=sout[1][i]*(1-sout[1][i])*sum1;
                w[0][j][i]+=eta*delta[0][i]*sout[0][j];
            }
        }
        for(int i=0;i<Nneuronsinlayer[2];i++)
        {
            if(fabs(outputvector[i]-sout[2][i])>error)
                error=fabs(outputvector[i]-sout[2][i]);
        }
    }
    delete bmp;
    return error;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    float res=0,minres=1,dirres=0;
    int i=0;
    eta=1;
    do {
        res=TeachEpoch();
        if(res<minres)
            minres=res;
        if(res>prevres)
            dirres++;
        else
            dirres=0;
        i++;
        prevres=res;
    } while((res>0.1)&&(dirres<1000));
    Label5->Caption="Iterations made: "+IntToStr(i);
    Label6->Caption="Error: "+FloatToStrF(res,ffFixed,3,3);
    Button3->Enabled=true;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
    unsigned char gray;
    for(int i=0;i<Nneuronsinlayer[0];i++)
    {
        if(!IsColourGray(Image1->Canvas->Pixels[i%7][i/7]))
```

```
        {
            ShowMessage("Image must be grayscale!
Terminating...");
            exit(0);
        }
        gray=(char)(Image1->Canvas->Pixels[i%7][i/7])&255;
        gray=255-gray;
        inputvector[i]=(float)gray/255;          //1 - black, 0 -
fully white
    }
    Propagate(inputvector);
    int nummax=0;
    for(int i=0;i<Nneuronsinlayer[2];i++)
        if(sout[2][i]>sout[2][nummax])
            nummax=i;
    Label3->Caption=alf[nummax];
    for(int i=0;i<26;i++)
        StringGrid2-
>Cells[1][i+1]=FloatToStrF(sout[2][i],ffFixed,3,3);
}
```