МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
в.о. завідувача кафедри
_____ Рязанцев О.І.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Система штучного інтелекту пошуку зображень за певними ознаками

Освітній рівень"Магістр"
Спеціальність 123 "Комп'ютерна інженерія"

Науковий керівник роботи: _____ (підпис) Г.Ф.Кривуля (ініціали, прізвище)

Консультант з охорони праці: _____ (підпис) Я.О.Критська (ініціали, прізвище)

Студент: _____ (підпис) Д.О. Ботнар (ініціали, прізвище)

Група: КІ-19дм

Сєвєродонецьк 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітній рівень магістр
Напрям підготовки _____
(шифр і назва)
Спеціальність 123 "Комп'ютерна інженерія"
(шифр і назва)

ЗАТВЕРДЖУЮ:

Т.в.о. завідувача кафедри _____
_____ В.С.Кардашук
«_____» _____20____р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Ботнару Денису Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система штучного інтелекту пошуку зображень за певними ознакам

керівник проекту (роботи) Кривуля Генннадій Федорович, д.т.н., проф.
(прізвище, м.'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «5» 10 2020 р. № 140/15.15
2. Строк подання студентом роботи 10.01.2021
3. Вихідні дані до роботи Матеріали науково-дослідної практики, математичні моделі знаходження об'єктів на зображенні, а саме автомобільних номерів; перелік використовуваних програмних засобів: мова програмування Python, бібліотеки програмного забезпечення Keras, TensorFlow, OpenCV; теоретичні відомості про методи штучні нейронні мережі
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Огляд основних методів вирішення задачі знаходження місця розташування автомобільних номерів, опис математичного забезпечення згорткової нейронної мережі, реалізація системи штучного інтелекту, охорона праці та безпека в надзвичайних ситуаціях, висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та безпека в надзвичайних ситуаціях	Критська Я.О. ст. викл. кафедри КНІ		

7. Дата видачі завдання 14.10.2020

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Розробка технічного завдання	02.09.2020-15.09.2020	
2	Критичний аналіз літератури з досліджуваної проблеми	16.09.2020-22.09.2020	
3	Розробка нейронної мережі	23.09.2020-25.09.2020	
4	Реалізація системи	26.09.2020-06.10.2020	
5	Аналіз результатів дослідження	07.10.2020-25.11.2020	
6	Розробка частини проекту "Охорона праці та безпеки в надзвичайних ситуаціях"	26.11.2020-1.12.2020	
7	Оформлення пояснювальної записки, автореферату та презентації	2.12.2020-09.01.2021	

Студент _____ Ботнар Д.О.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Кривуля Г.Ф.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ботнар Д.О. Система штучного інтелекту пошуку зображень за певними ознаками.

Метою роботи є розробка методів, що базуються на використанні нейронної мережі, які дозволяють детектувати розташування автомобільних номерів на зображенні.

Об’єктом дослідження є послідовність зображень з автомобілями та їх номерами.

Використано метод згорткових мереж для знаходженні розташування номерів. Проведено дослідження видів штучних нейронних мереж та їх використання для знаходження об’єктів на зображенні. Отримані оцінки точності знаходження автомобільних номерів за допомогою загорткових нейронних мереж.

У результаті роботи здійснена програмна реалізація системи для пошуку автомобільних номерів на зображенні.

Ключові слова: нейронні мережі, згорткові мережі, детектування зображень, види нейронних мереж.

ABSTRACT

Botnar D.O. An artificial intelligence system for searching images by certain features.

The purpose of the work is to develop methods based on the use of a neural network that can detect the location of car license plate on an image.

The object of the study is a sequence of images of cars and their license plates.

The convolution network method was used to find the location of license plates. Types of artificial neural networks have been investigated and used to locate objects on an image. Estimates of accuracy of finding of car license plates by means of convolutional neural networks were obtained.

As a result, the software implementation of the system to search for car license plates on an image.

Keywords: neural networks, conventional networks, image detection, types of neural networks.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ГА – генетичний алгоритм

ПЕОМ – персональна електронно-обчислювальна машина

ПЗ – програмне забезпечення

ФП – функція приналежності

ШІ – штучний інтелект

ШНМ – штучна нейронна мережа

ВСТУП

На фоні стрімкого розвитку комп'ютерно-інтегрованих технологій і автоматизації багатьох технологічних процесів широкого поширення набула задача розпізнавання символів, які можуть належати до різного роду шрифтів. Отже, реалізація методів розпізнавання місця розташування автомобільних номерів є актуальною для практичного вирішення таких задач як пошук та ідентифікації номерних написів, що може бути використано службами стеження на пропускних пунктах, полицею для пошуку вкрадених машин, а також при роботі інтелектуальних систем відеоспостереження.

Задача ідентифікації місця розташування числового ряду передбачає вирішення таких підзадач, як визначення положення ряду на зображенні, виділення окремих символів, класифікація даних. При ідентифікації необхідно враховувати перешкоди на зображенні, специфічний шрифт написання цифр, колір фону.

Метою роботи є дослідження та програмна розробка ШНМ для знаходження місця розташування автомобільних номерів на зображенні. Об’єктом дослідження є процес знаходження місця розташування автомобільних номерів. Предметом дослідження є ШНМ для знаходження місця розташування автомобільних номерів на зображенні.

Для досягнення поставленої мети виникла необхідність вирішення наступних завдань:

1) Створення програмного модуля завантаження та обробки зображення з відокремленням робочою області зображення.
2) Вибір типу ШНМ.
3) Структурна ідентифікація топології ШНМ.
4) Навчання ШНМ.
5) Оцінка якості функціонування системи розпізнавання місця розташування автомобільних номерів з проведенням експериментів для тестової вибірки різнорідних зображень.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ ЗНАХОДЖЕННЯ МІСЦЯ РОЗТАШУВАННЯ АВТОМОБІЛЬНИХ НОМЕРІВ

1.1 Опис методів обробки зображення

При комп'ютерній обробці вхідні дані графічного зображення зберігаються в цифровому вигляді згідно колірній моделі RGB. Модель базується на поєднанні трьох кольорів – червоного, зеленого і синього. У зображенні точкам належить відповідна пропорція змішування цих кольорів. Для кожного кольору виділяється фіксована пам'ять обсягом вісім біт.

Будь-який колірний простір RGB має бути пов'язаний з еталонним колірним простором CIE XYZ [9]. Для CIE XYZ відома залежність між значенням коефіцієнтів (x, y, z) пікселя і фактичним значенням яскравості еталонних джерел світла, змішенням яких домагаються отримання потрібного сприйманого кольору. Для кожного RGB-зображення перетворення в XYZ має бути явно або неявно визначене, інакше коректне відображення можливе тільки на тому моніторі, на якому створювалося зображення. Дана інформація називається колірним профілем зображення. Якщо монітор має такий профіль, то шляхом перетворення RGBimage→XYZimage з наступним перетворенням XYZimage→RGBdisplay можна отримати на іншому моніторі кольори, що відповідають початковим (з урахуванням обмежень перенесення кольорів).

У просторі XYZ координата Y за визначенням відповідає сприйманій яскравості кольору. Щоб отримати з повнокольорового зображення монохромне, необхідно перетворити кожен піксель в XYZ і взяти компоненту Y як результат. Відомо, що перетворення між будь-якими адитивними колірними системами лінійне (в силу лінійності сприйняття кольору людиною), отже, може бути описане матрицею $M_{3\times3}$, такою, що:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1.1)$$

.

Переважна більшість цифрових зображень відповідають стандарту sRGB. Так само необхідно виконувати гамма-перетворення [9]. Таким чином, перетворення з кольорового sRGB- зображення в монохромне здійснюється згідно співвідношень:

$$Y = 255(0.21r + 0.72g + 0.07b)^{1/2.2}, \quad (1.2)$$

$$r = (R/255)^{2.2}, g = (G/255)^{2.2}, b = (B/255)^{2.2}. \quad (1.3)$$

де коефіцієнти (0.21, 0.72, 0.07) – округлений до другого знаку рядок із матриці перетворення M sRGB з системи sRGB в XYZ.

Передбачається, що на зображенні з цифрами, які підлягають розпізнаванню, колір цих цифр суттєво відрізняється (виділяється) на фоні іншого кольору: наприклад, на сторінках книги фон - білий, цифра - чорна; на маркуванні вантажного вагону фон - темний, цифра (маркування) - біла. Тому доцільно працювати з монохромним зображенням (кольори зображення знаходяться в діапазоні [0, 255]), тобто з градацією сірого «0» - чорний, «255» - білий.

Запропоновано використати «критичне» значення кольору (градації сірого) для цифри Yзд=[0, 255], по якому відбуватиметься відсічення фону і виділення тільки тієї частини зображення, де знаходиться цифра. Значення Yзд задається залежно від передбачуваного кольору цифр і фону. Якщо фон темніший за цифри, то відсічення частини зображення з фоном відбуватиметься при Y<Yзд, за умови світлішого фону – за цифри Y>Yзд, де Y – значення поточного пікселя.

З урахуванням вищезазначеного, розроблено алгоритм виділення цифри на монохромному зображенні (рис. 1.1), який складається з наступних етапів:

– завдання значення Yзд в діапазоні [0, 255], що відповідає градації сірого;

– попіксельне зчитування зображення;

– фіксація координати «крайніх» (ліворуч і праворуч, вгорі і внизу) пікселів, значення яких Y>Yзд (фон темніший за цифри);

– обрізка зображення по отриманим «крайніх» точок – вилучення області зображення з цифрою.

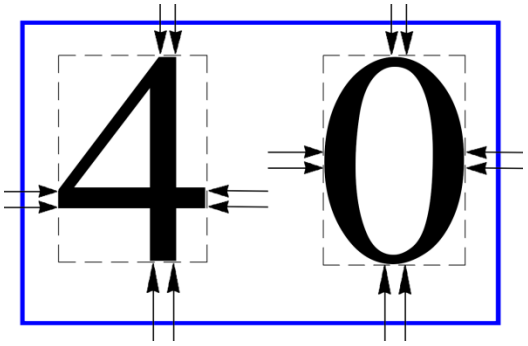


Рисунок 1.1 – Виділення цифри на зображенні

В системі повинна бути реалізована можливість отримання зображення для ідентифікації чисел із зовнішнього пристрою – веб-камери. Проблема ідентифікації такого зображення полягає в тому, що неможливо передбачити якість освітлення, при якому був зроблений знімок, відблиски на знімку та інші дефекти. В процесі конвертації зображення в монохромне всі ці недоліки можуть призвести до того, що зображення буде занадто темним або світлим. Тобто задане за умовчанням значення Yзд, по якому виділяється область з цифрою, буде неактуальне і програма не зможе відокремити, отже, і розпізнати, цифру.

Для вирішення цієї проблеми запропоновано використання медіанної фільтрації зображення [10]. При такому підході відбувається послідовна обробка кожної точки зображення, внаслідок чого утворюється послідовність оцінок. Дана фільтрація дозволяє виключити викиди, тобто значення пікселів, які суттєво відрізняються від інших. При цьому ці викиди замінюються сусідніми значеннями тієї ж послідовності. Кількість значень початкових пікселів, що залучаються до розгляду, називається апертурою фільтру. При апертурі 2m+1 можна виключити m викидів, що йдуть поспіль. При медіанній фільтрації використовується двомірне вікно (апертура фільтру - рис. 1.2), що зазвичай має центральну симетрію, при цьому його центр розташовується в поточній точці фільтрації. Точки зображення, які виявилися в межах вікна, утворюють робочу вибірку поточного кроку.

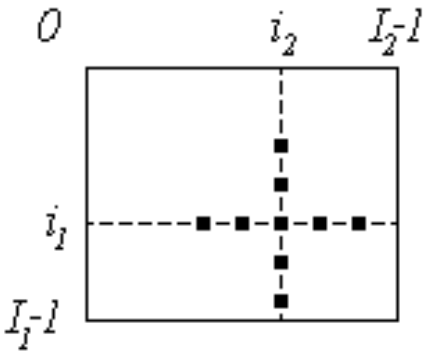


Рисунок 1.2 – Вікно при медіанній фільтрації

Робоча вибірка позначається у вигляді одномірного масиву $Y = \{y_1, y_2, \ldots, y_n\}$, число елементів якого дорівнює розміру вікна, а їх розташування є довільним. Якщо упорядкувати послідовність $\{y_i, i = \overrightarrow{1,n}\}$ за збільшенням, то її медіаною буде той елемент вибірки, який займає центральне положення в цій впорядкованій послідовності. Отримане таким чином число є продуктом фільтрації для поточної точки зображення. Медіанна фільтрація позначається як:

$$x^* = med(y_1, y_2, \cdots, y_n). \tag{1.4}$$

Для того, щоб зробити алгоритм розпізнавання універсальним – здатним працювати з цифрами різних шрифтів і розмірів, виникає потреба враховувати той факт, що кожне зображення цифри, яка ідентифікується, може мати свою, відмінну від використовуваної в навчальній вибірці, роздільність. Для вирішення цієї проблеми було запропоновано розділяти зображення з цифрою на прямокутні області (кількість областей задається програмно) та обчислювати концентрацію пікселів певного кольору в кожній з областей, в результаті чого отримано вхідний вектор параметрів.

Значення (кількість пікселів) ширини w і висоти h оброблюваного зображення може не відповідати заданій кількості частин n і m, на які розподіляється зображення з цифрою, отже при обчисленні ширини і висоти для областей буде отримано значення дійсного типу, що неприпустимо при роботі з пікселями. У такому разі при розділенні зображення на n по вертикалі і m по горизонталі областей для «крайніх» областей ширина і висота відрізнятимуться від інших.

Розбиття зображення (рис. 1.3) на nxm областей відбувається за наступним алгоритмом:

1) Обчислюється ширина w і висота h обробленого зображення з цифрою;

2) Обчислюється ширина wt і висота ht прямокутної області на які ділиться зображення відповідно формулам:

$$\begin{cases} wt = \dfrac{w}{n}, \\ ht = \dfrac{h}{m}. \end{cases} \tag{1.5}$$

3) Якщо ширина *w* і висота *h* не кратні кількості частин *n* і *m*, то *wt* і *ht* округлюють до меншого – це ширина і висота для всіх областей, окрім «крайніх»;

4) Обчислюється остаточна ширина і висота для «крайніх» областей по формулам:

$$\begin{cases} wc = w - wct * n, \\ hc = h - hct * m. \end{cases} \tag{1.6}$$

де *wct* - закруглене до меншого значення *wt*, *hct* - закруглене до меншого значення *ht*.

5) Обчислюється площа кожної області, яка використовується для подальшого знаходження відсотка пікселів певного кольору градації сірого в області, згідно рівнянь:

$$\begin{cases} st = wt * ht, \\ sc = wc * hc, \\ scw = wc * ht, \\ sch = wt * hc. \end{cases} \quad (1.7)$$

де *sc*, *scw*, *sch* – площа «крайніх» областей, *st* – площа інших областей.

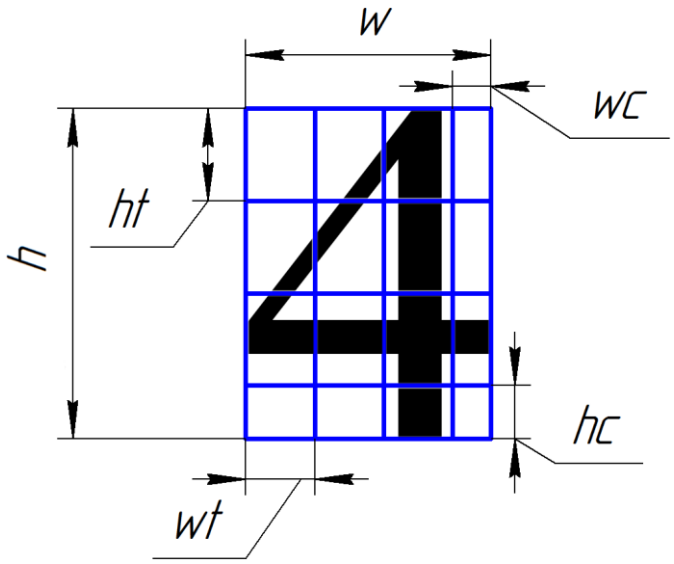


Рисунок 1.3 – Розподілення зображення на області

1.2 Опис методів визначення розташування об’єктів на основі нейронних мереж

З поширенням нейронних мереж відкрилися можливості використання обчислень в сфері розпізнавання автомобільних номерів. В основі нейроінтелекта лежить нейронна організація штучних систем, яка має біологічні передумови. Здатність біологічних систем до навчання, самоорганізації і адаптації має більшу перевагу в порівнянні з сучасними обчислювальними системами. Перші кроки в області штучних нейронних мереж зробили в 1943 р В.Мак-Калох і В.Пітс [1]. Вони показали, що за допомогою порогових нейронних елементів можна реалізувати обчислення будь-яких логічних функцій. У 1949 р Хебб запропонував правило навчання, яке стало математичною основою для навчання ряду нейронних мереж. У 1957-1962 рр [1]. Ф. Розенблатт запропонував і дослідив модель нейронної мережі, яку він назвав персептроном. У 1959 р В. Відроу і М. Хофф запропонували процедуру навчання для лінійного адаптивного елемента – ADALINE [1]. Процедура навчання отримала назву "дельта правило". У 1969 р М.

Мінський і С. Пайперт опублікували монографію "Персептрони", в якій було дано математичний аналіз персептрона, і показані властиві йому обмеження. У 80-ті роки значно розширюються дослідження в області нейронних мереж. Д. Хопфилд в 1982 р. дав аналіз стійкості нейронних мереж із зворотними зв'язками і запропонував використовувати їх для рішень завдань оптимізації. Т.Кохонен розробив і дослідив самоорганізовані нейронні мережі [1]. Ряд авторів запропонував алгоритм зворотного поширення помилки, який став потужним засобом для навчання багатошарових нейронних мереж. В даний час розроблено велику кількість нейросистем, що застосовуються в різних областях: прогнозування, управлінні, діагностиці в медицині і техніці, розпізнаванні образів тощо.

Нейронна мережа - сукупність нейронних елементів та зв'язків між ними. Основний елемент нейронної мережі - це формальний нейрон, що здійснює операцію нелінійного перетворення суми здобутків вхідних сигналів на вагові коефіцієнти.

На вхід штучного нейрона надходить деяка множина сигналів x_1, x_2, ..., x_n, кожен з яких є виходом іншого нейрона. Кожен вхід множиться на відповідну вагу w_1, w_2, ..., w_n, і надходить на підсумовуючий блок, позначений Σ, який складає зважені входи алгебраїчно, створюючи вихід – NET [1].

Найпростішою нейронною мережею вважається персептрон (рис. 1.4). За своєю структурою він аналогічний нейрону, але на його виходах знаходиться деякий аналізатор. І в залежності від значення сформованої суми, вихідне значення аналізатора дорівнюватиме "1", якщо сума більше порогового значення і "0", якщо менше.

Персептрон є однією з найпопулярніших реалізацій нейронних мереж. Причиною його популярності є відносна простота реалізації на тлі універсальності і широкого кола завдань, які можуть вирішувати персептрони.

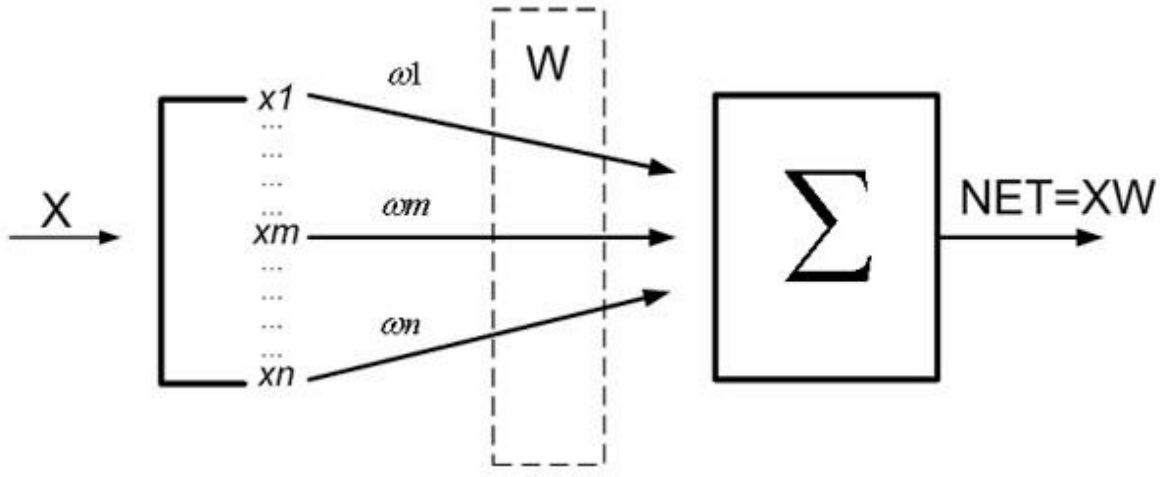


Рисунок 1.4 – Загальний вигляд нейронної мережі

Для вирішення більш складних завдань використовують багатошарові нейронні мережі [1]. Але для навчання багатошарових нейронних мереж потрібні більш складні алгоритми навчання нейронних мереж. Штучна нейронна мережа навчається за допомогою деякого процесу, що модифікує її ваги. Якщо навчання успішне, то передача мережі множини вхідних сигналів призводить до появи бажаної множини вихідних сигналів.

Навчання нейронної мережі полягає в рішенні задачі багатомірної оптимізації, яка уявляє собою здачу зменшення суми квадратів похибок між експериментальним та модельним значенням:

$$\Sigma\left(Y_e - Y_m\right)^2 \to \min . \tag{1.8}$$

Є два класи навчальних методів: детерміністський і стохастичний. Детерміністський метод навчання крок за кроком здійснює процедуру корекції ваги мережі, засновану на використанні їх поточних значень, а також величин входів, фактичних виходів і бажаних виходів. Стохастичні методи навчання виконують псевдовипадкові зміни величин ваг, зберігаючи ті зміни, які ведуть до покращення результату.

Для навчання мережі використовуються різні алгоритми навчання та їх модифікації. Дуже важко визначити, який навчальний алгоритм буде найшвидшим при вирішенні того чи іншого завдання. Існує алгоритм зворотного поширення помилки, який є ефективним засобом для навчання багатошарових нейронних мереж прямого поширення. Алгоритм мінімізує середньоквадратичну помилку нейронної мережі. Для цього з метою налаштування синаптичних зв'язків використовується метод градієнтного спуску в просторі вагових коефіцієнтів і порогів нейронної мережі. Слід зазначити, що для налаштування синаптичних зв'язків мережі використовується не тільки метод градієнтного спуску, а й методи сполучених градієнтів, Ньютона, квазіньютоновський метод. Для прискорення процедури навчання замість постійного кроку навчання запропоновано використовувати адаптивний крок навчання h (t). Алгоритм з адаптивним кроком навчання працює в 4 рази швидше. На кожному етапі навчання мережі він вибирається таким, щоб мінімізувати середньоквадратичну помилку мережі.

Для систем на базі нейронних мереж найкращі якості показує гетерогенна мережа, що складається з прихованих шарів з нелінійною функцією активації нейронних елементів та вихідного лінійного нейрона. Недоліком більшості розглянутих нелінійних функцій активації є те, що область вихідних значень їх обмежена відрізком [0,1] або [-1,1]. Це призводить до необхідності масштабування даних, якщо вони не належать зазначеним вище діапазонами значень.

Аналіз різних типів нейронних мереж показав, що нейронна мережа може вирішувати завдання додавання, віднімання десяткових чисел, завдання лінійного авторегресійного аналізу і прогнозування часових рядів з використанням методу «ковзного вікна».

Проведений аналіз багатошарових нейронних мереж і алгоритмів їх навчання дозволив виявити ряд недоліків і проблем, що виникають:

– невизначеність у виборі числа шарів і кількості нейронних елементів в шарі;

– повільна збіжність градієнтного методу з постійним кроком навчання;

– складність вибору відповідної швидкості навчання. Так як маленька швидкість навчання призводить до скочування нейронної мережі в локальний мінімум, а велика швидкість навчання може привести до пропуску глобального мінімуму і зробити процес навчання розбіжним;

– неможливість визначення точок локального і глобального мінімуму, так як градієнтний метод їх не розрізняє;

– вплив випадкової ініціалізації вагових коефіцієнтів нейронної мережі на пошук мінімуму функції середньоквадратичної помилки.

Велику роль для ефективності навчання мережі грає архітектура. За допомогою тришарової нейронної мережі можна апроксимувати будь-яку функцію зі як завгодно заданою точністю. Точність визначається числом нейронів в прихованому шарі, але при занадто великий розмірності прихованого шару може наступити явище, зване перетренуванням мережі. Для усунення цього недоліку необхідно, щоб число нейронів в проміжному шарі було значно менше, ніж число тренувальних образів. З іншого боку, при дуже маленькій розмірності прихованого шару можна потрапити в небажаний локальний мінімум.

1.3 Принципи визначення розташування об’єктів на основі нечіткої логіки

У разі розпізнавання об’єктів, розробка математичної моделі яких утруднена або неможлива, застосовуються методи розпізнавання, засновані на використанні нечіткої логіки (фазі-логіки).

Нечітка логіка передбачає відмову від основного твердження класичної теорії множин про те, що певний елемент x може або належати, або не належати до певної множини X. При цьому вводиться спеціальна характеристична функція множини - функція приналежності $\mu_X(x)$. Ця функція визначає ступінь приналежності елемента множини в формі реального

чисельного значення в діапазоні між 0 і 1. Ступінь приналежності елемента множини називають ступенем істинності даної величини, яку формально можна позначити наступним чином [6]:

$$X = \frac{\mu_1}{x_1} + \frac{\mu_2}{x_2} + ... + \frac{\mu_i}{x_i} + ... \quad . \tag{1.9}$$

В теорії нечітких множин центральну роль грає поняття «Лінгвістичної величини», за допомогою якої діапазони вимірювання фізичних величин окреслюються неоднозначно, тобто описуються лише якісно за допомогою таких понять, як «мало», «середньо», «багато», замість конкретних чисельних значень [6-9].

Процедура фазифікації [7] параметрів процесу полягає в переведенні поточних значень вхідних змінних в лінгвістичні величини істинності. Кожна така величина інтерпретується як фаззі-множина і описується функцією приналежності. Цим якісне висловлювання переводиться в кількісну величину таким чином, що вона для кожного поточного чисельного значення змінної величини процесу повідомляє ступінь приналежності до тієї підмножини, яка символізує конкретну лінгвістичну величину. Оскільки функції приналежності, як правило, перекривають одна одну, то для однієї і тієї ж змінної процесу кілька функцій приналежності можуть повідомляти різні величини істинності, що відрізняються від нуля.

Для формування логічного рішення застосовується лінгвістичне правило «якщо – то». Частина «якщо» (передумова) може при цьому означати сполучення будь-якої складності логічних операцій. Частина «то» (рішення, висновок) представляють собою зазвичай просте значення лінгвістичної величини для вихідного параметра [10-11].

Найбільш часто застосовується «мінімаксний» метод логічного висновку. Його можна описати рівнянням [6-9]:

$$\mu^{(y)} = \max_{k=1,..,n} \{\min[\mu_{e,k}, \mu_{T,k}(y)]\}, \tag{1.10}$$

де n – число правил, $\mu_{e,k}$ – величина істинності частини «якщо» к *k*-му правилі, $\mu_{T,k}(y)$ – функція приналежності частини «то» *k*-ого правила.

Загальними засобами представлення знань в системах з нечіткою логікою є [5]:

- правила;
- семантичні мережі;
- фрейми.

Модуль вилучення знань є найбільш трудомістким. Правила забезпечують формальний спосіб представлення рекомендацій, знань або стратегій. Вони частіше підходять в тих випадках, коли предметні знання виникають з емпіричних асоціацій, накопичених за роки роботи з вирішення завдань в даній області.

В системах, заснованих на правилах «якщо-то», предметні знання представляються набором правил, які перевіряють на групі фактів і знань поточну ситуацію (вхідну інформацію). Коли частина правила «якщо» задовольняє фактам, то дії, зазначені в частині «то», виконуються. Коли це відбувається, то правило спрацьовує. Інтерпретатор правил зіставляє частини правил «якщо» з фактами і виконує те правило, частина «якщо» якого сходиться з фактами, тобто інтерпретатор правил працює в циклі "Зіставити - виконати", формуючи послідовність дій.

Нечітким логічним висновком є апроксимація залежності між вхідними та вихідними даними на основі лінгвістичних висловлювань у формі «якщо-то» і логічних операцій над нечіткими множинами [10-11].

Структурна схема процесу нечіткого логічного виведення наведена на рисунку 1.5. Нечітка база знань, позначена на рисунку 1.5 містить інформацію про залежність Y(X) у вигляді лінгвістичних правил «якщо-то». Функції приналежності використовуються для представлення лінгвістичних термів у вигляді нечітких множин. Машина нечіткого логічного висновка на основі правил з бази знань визначає значення вихідної змінної у вигляді нечіткої множини Y*, яка відповідає нечітким значенням вхідних змінних X*. Дефазифікатор виконує перетворення вихідної нечіткої множини Y* в чітке число, яке є результатом роботи системи [7-9].

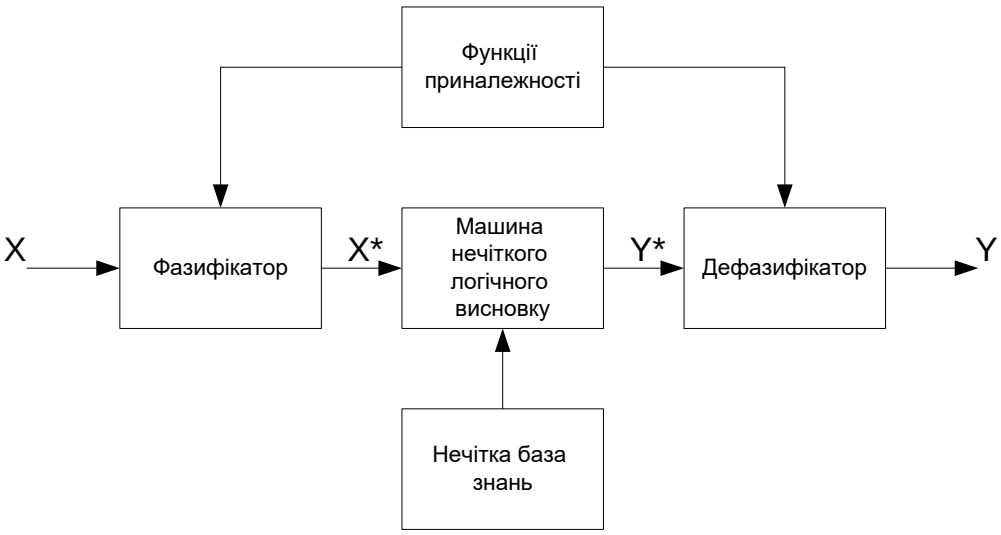


Рисунок 1.5 – Схема процесу нечіткого логічного висновку Мамдані

При використанні нечіткого логічного висновку Мамдані для визначення приналежності вихідної множини нечітким термам здійснюється операція агрегування за функціями приналежності нечітких множин, проілюстрована на рисунку 1.6 [6].

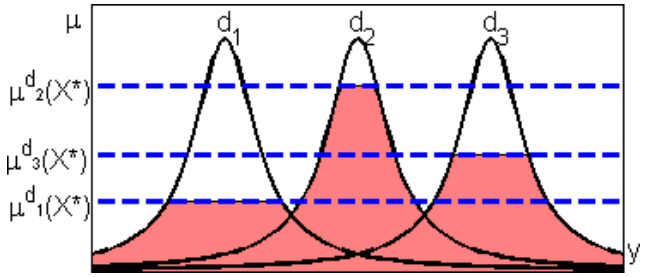


Рисунок 1.6 – Ілюстрація операції агрегування нечіткої множини

Дефазифікацію вихідного параметра можна здійснити методом центру ваги за рівнянням [7]:

$$Y = \frac{\sum_{i=1}^{k} y_i \mu(y_i)}{\sum_{i=1}^{k} \mu(y_i)} \tag{1.11}$$

Вибір типу нечіткого висновку Мамдані обумовлений використанням формалізованої компактної бази знань, яка може бути сформована як на основі експертних оцінок, так і за допомоги вилучення знань шляхом використання методів інтелектуального аналізу даних [11-14]. Нечіткий логічний висновок Мамдані на відміну від інших видів нечіткого логічного висновку спрощує процес змістовної інтерпретації отриманих результатів.

Модель NEFCLASS (NEuro Fuzzy CLASSifier) є нейро-нечітким класифікатором, принцип роботи якого заснований на отриманні нечітких правил з сукупності даних, які можна розділити на кілька непересічних класів [4]. Завдання NEFCLASS полягає в тому, щоб визначити належність вхідного зразка з деяким набором параметрів до певного класу. Нечіткість пов’язана з недосконалістю або неповнотою вимірювань тих властивостей об'єктів, за якими здійснюється процес класифікації.

Система NEFCLASS має послідовну архітектуру та складається з трьох шарів нейронів, як показано на рисунку 1.6.

Перший шар містить вхідні нейрони. Другий, прихований шар, містить продукційні правила, кожне з яких наведено в наступному вигляді:

якщо x_1 належить до μ_1 І x_2 належить до μ_2 І … І x_n належить до μ_n , ТО зразок з вектором параметрів $\overline{x}$ належить до класу c_i, де $\mu_{1...}$ μ_n – нечіткі множини, $x_{1...}$ x_n – вхідні параметри класифікатора.

Третій шар складається з вихідних нейронів, кожен з яких відповідає певному класу. Активація нейрону правила R в мережі з n вхідними нейронами обчислюється наступним чином:

$$a_R = \bigwedge_{i=1}^{n} W(x_i, R),$$

(1.12)

де $W(x_i, R)$ – ваговий коефіцієнт з'єднання вхідного нейрона x_i з нейроном правила R.

Активація нейрону вихідного шару, пов'язаного з m нейронами правил прихованого шару обчислюється наступним чином:

$$a_c = \bigvee_{i=1}^{m} a_{Ri}.$$

(1.13)

Співвідношення (1.12) та (1.13) визначаються шляхом заміни нечітких термів, які якісно характеризують значення вхідних параметрів, на функції належності кожного вхідного нейрона кожній з нечітких множин, а також заміни логічних операцій «І» та «АБО» на функції t-норми і s -норми (t-конорми), наприклад, на операції мінімуму та максимуму [4].

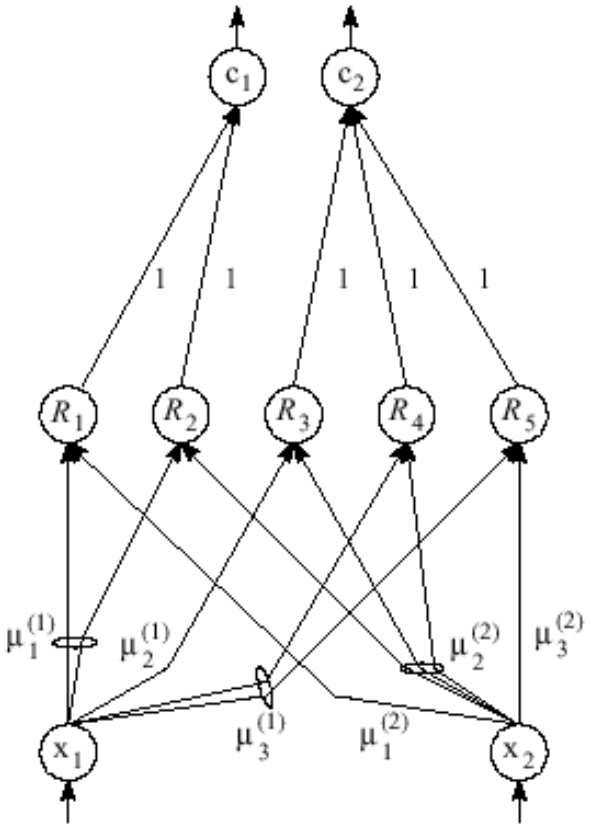


Рисунок 1.6 – Приклад системи NEFCLASS з двома вхідними і двома вихідними нейронами

Система NEFCLASS може бути побудована в умовах нестачі достовірної інформації про об'єкти, які підлягають класифікації. Потрібно визначити кількість початкових нечітких множин і задати максимальну кількість нейронів правил, які можуть бути створені в прихованому шарі. Для навчання може бути використана Гаусова функція належності [5].

При генерації продукційних правил для системи NEFCLASS з n вхідними нейронами x_1 ... x_n, та m вихідними нейронами c_1 ... c_m, використовується навчальна вибірка зображень, яка представлена сукупністю зразків з вхідними параметрами x_1 ... x_n і бажаним вихідним параметром c_i, що характеризує клас, до якого належить даний зразок навчальної вибірки. Навчальний алгоритм, мета якого полягає у формуванні прихованого шару нейронів правил системи NEFCLASS, складається з наступних етапів [4]:

1) Отримання і обробка поточного зображення з навчальної вибірки.

2) Знаходження для кожного вхідного нейрона x_i такої функції належності, яка відповідає співвідношенню:

$$\mu_i^R = \bigvee_{j=1}^{p} \mu_{ij},$$

(1.14)

де μ_{ij}- значення функції належності вхідного нейрона x_i до нечіткої множини A_j,

p – кількість нечітких множин.

3) Якщо поточна кількість нейронів правил менше максимально можливої та не існує нейрона правила R з вектором вагових коефіцієнтів, що задовольняє співвідношенню:

$$\overline{W(x,R)} = \overline{\mu^R},$$

(1.15)

то необхідно створити такий вузол та з'єднати його з вихідним вузлом с, відповідним бажаному вихідному нейрону.

4) Якщо в навчальній вибірці залишилися необроблені зображення, то здійснюється перехід до кроку 1, інакше – перехід до кроку 5.

5) Результуюча база продукційних правил визначається згідно з процедурою "кращого" навчання правил, в процесі якого при обробці зразків навчальної вибірки відбувається накопичення активації кожного нейрона правил для кожного класу зразків, які було розглянуто. Якщо нейрон правила R показує більше накопичення активації для класу c_j, ніж для класу c_R, який був специфікований для поточного правила, тоді слід змінити наслідок правила R на c_j, тобто з'єднати нейрон прихованого шару R з нейроном вихідного шару c_j. Обробка зразків з навчальної вибірки проводиться для кожного нейрона прихованого шару R шляхом обчислення накопичення активації згідно співвідношення:

$$V_R \sum_{i=1}^{N} a_R^i * q_i,$$

(1.16)

де q_i=1 в тому випадку, якщо *i*-й зразок класифіковано вірно, або q_i=-1 в тому випадку, якщо *i*-й зразок класифіковано невірно.

Після навчання в складі нечіткої базі продукційних правил залишається певна кількість нейронів з найвищими значеннями функції накопичення, розрахованої за рівнянням (1.16).

У модифікованій системі нечіткої класифікації NEFCLASSM нечіткі множини мають Гаусову функцію належності, яка описується відповідно до рівняння:

$$\mu(x) = e^{-\frac{(x-a)^2}{2b^2}},$$

(1.17)

де a – параметр функції належності, який є коефіцієнтом максимуму, b – параметр функції належності, який є коефіцієнтом концентрації.

Задача налаштування параметрів функцій належності зводиться до вирішення задачі мінімізації функції середньоквадратичної помилки на навчальній вибірці:

$$F(\bar{a},\bar{b})=\frac{1}{N}\sum_{p=1}^{N}\left\|a_c^{(p)}-a_c^{(p)*}\right\|^2\to\min,$$

(1.18)

де N – кількість зразків навчальної вибірки, $a_c^{(p)}$ – вектор активації нейронів вихідного шару для навчального зразка p, $a_c^{(p)*}$ – цільової вектор активації нейронів вихідного шару для навчального зразка p, $\bar{a},\bar{b}$ – вектори параметрів функцій належності.

Координати цільового вектору для зразка p_i визначаються згідно виразу:

$$a_c^{(p)*}=\begin{cases}0,\ i\neq j,\\ 1,i=j;\end{cases}$$

(1.19)

де j – номер класу, до якого належить даний зразок.

Аргументом чисельної оптимізації функції (1.18) є сукупний вектор параметрів функцій належності всіх нечітких множин. Для вирішення даної задачі застосовується метод сполучених градієнтів [6-8].

1.4 Опис методів для визначення розташування об’єктів з використанням генетичного алгоритму

Генетичні алгоритми - клас методів оптимізації, заснованих на імітації процесів, які протікають в природі, зокрема природного відбору - концепції, озвученої в еволюційній теорії Чарльза Дарвіна [11]. У відповідністю з теорією Дарвіна в природному середовищі перевагу до виживання і розмноження мають особи, найбільш пристосовані до умов конкретного середовища проживання.

Для визначення структури нейронної мережі використано методи генетичного програмування, що викликано наступними факторами:

1) Оскільки нейронні мережі знаходять своє застосування в багатьох областях діяльності суспільства, то очевидно, що топологія мережі буде сильно змінюватися при вирішенні різних видів завдань, оскільки для вирішення спеціалізованих завдань потрібні різні варіанти топологій.

2) Ручне створення нейронної мережі - досить трудомістке завдання і створення систем для їх автоматичної генерації дозволяє зробити застосування нейронних мереж економічно виправданим кроком для більш широкого кола завдань.

3) Алгоритми генерації нейронних мереж, розроблені для мереж із заздалегідь передбаченою топологією, передбачають вирішення лише вузькоспеціалізованих завдань. Незважаючи на те, що використання таких алгоритмів дозволяє вирішувати деякі завдання, хотілося б мати спільне рішення для множини завдань.

4) Генетичне програмування дозволяє вирішувати найрізноманітніші проблеми уніфікованим чином, що дозволяє оптимізувати процес розробки рішення і отримати однакові результати для кожного конкретного завдання.

Даний підхід дозволяє не тільки створювати нейронні мережі, які вирішують конкретні задачі, але і знаходити для цих завдань оптимальну топологію штучних нейронних мереж. Незважаючи на те, що генетичне програмування має ряд недоліків, на сьогоднішній день ці недоліки багато в чому можна обійти за рахунок використання різних стратегій як при реалізації різних генетичних операторів, так і всього генетичного алгоритму в цілому. Урахування недоліків нейроеволюційних алгоритмів при створенні програми дозволить знизити як ймовірність попадання в локальний оптимум, так і вимоги до обчислювальних ресурсів.

На даний момент, існує кілька головних недоліків використання генетичного алгоритму і генетичного програмування. Нижче представлений список основних недоліків даного підходу:

1) Повторна оцінка функції пристосованості (фітнес-функції) для складних проблем часто є чинником, що обмежує використання алгоритмів штучної еволюції. Пошук оптимального рішення для складної задачі високої розмірності часто вимагає дуже витратної оцінки функції пристосованості.

2) Генетичні алгоритми погано масштабуються під складність вирішуваної проблеми.

3) Рішення є більш придатним лише в порівнянні з іншими рішеннями. В результаті умова зупинки алгоритму може бути передчасною для певної проблеми.

4) У багатьох задачах генетичні алгоритми мають тенденцію сходитися до локального оптимуму або навіть до опірних точок, замість глобального оптимуму для даного завдання [11].

Однак, багато з цих недоліків можуть бути виправлені. Так, наприклад, для запобігання передчасної збіжності слід правильно підбирати такі параметри генетичного алгоритму як розмір популяції і відсоток особин, які піддаються мутації. Крім того, постійно розробляються нові варіанти генетичних операторів. Додаткових витрат, пов'язаних з повторним обчисленням значення функції пристосованості можна уникнути, за допомогою використання прапорів для випадків коли значення функції пристосованості не змінюється з часом.

Однак, крім вищезазначених недоліків, генетичні алгоритми мають і досить багато переваг:

1) Масштабованість - генетичні алгоритми легко можна пристосувати для багатопотокового і багатопроцесорного програмування, таким чином завдяки особливостям цього підходу в значній мірі знижуються відповідні накладні витрати.

2) Універсальність - генетичні алгоритми не вимагають ніякої інформації про поверхні відповіді, вони працюють з практично будь-якими завданнями.

3) Можливе застосування генетичних алгоритмів для тих задач, в яких значення функції пристосованості змінюється з часу або ж залежить від різних перемінних чинників.

4) Навіть у тих випадках, для яких добре працюють існуючі методики, можна досягти цікавих результатів поєднуючи їх з генетичними алгоритмами, використовуючи їх в якості доповнення до перевірених методів.

Таким чином, з огляду на всі переваги і недоліки генетичних алгоритмів, можна отримати досить універсальну систему вирішення необхідних завдань [11] і, зокрема, для задач оптимізації нейронної мережі.

Нейроеволюційний підхід багато в чому відрізняється від звичайних генетичних алгоритмів оскільки йому доводиться оперувати з набагато більш складними структурами. Крім того, нейронні мережі вимагають великих витрат обчислювальних ресурсів для оцінки трудомісткості отриманих рішень. Крім цього, нейронні мережі потрібно навчити.

Таким чином для застосування нейроеволюційних алгоритмів потрібно вирішити такі завдання:

– вибрати метод кодування нейронних мереж;

– реалізувати з урахуванням особливостей нейронних мереж генетичні оператори.

Як відзначають більшість дослідників, центральною точкою будь-якого методу еволюційної побудови нейронних мереж є вибір генетичного представлення. Вибір представлення визначає клас мереж, які можуть бути побудовані за допомогою даного методу. Крім того, від них залежить ефективність методу за всіма параметрами.

Загальні властивості представлень, за якими можна оцінювати і порівнювати різні методи, виділялися багатьма авторами. Найчастіше пропонують використовувати такі

властивості: повнота, замкнутість, компактність, масштабованість, множинність, пристосованість, модульність, надмірність та складність.

Способи кодування умовно можна розділити на два класи:

– пряме кодування - генетичний алгоритм працює з програмою в явному вигляді;

– непряме кодування - генетичний алгоритм працює не з самим кодом програми, а з правилами його побудови.

Запропоновано кодувати структуру нейронної мережі за допомогою матриці суміжності. Вона використовується для запису тільки багатошарових нейронних мереж без зворотних зв'язків. Кожному можливому прямому зв'язку нейрона і та не вхідного нейрона j відповідає елемент матриці з координатами (i, j). Якщо значення цього елемента дорівнює 1, зв'язок є; якщо 0 - зв'язку немає. Для зсувів кожного нейрона виділений окремий стовпець. Таким чином, нейронній мережі з N нейронів відповідає матриця розмірності $N * (N + 1)$.

При декодуванні отриманого генома назад в нейронну мережу всі зворотні зв'язки ігноруються. Всі нейрони, до яких не веде жодний зв'язок, або від яких не виходить жодного зв'язку видаляються. Таке представлення досить погано масштабуються, через те що довжина отриманого генома пропорційна квадрату числа нейронів в мережі. Тому його можна ефективно застосовувати тільки для побудови достатньо невеликих за розміром нейронних мереж.

У своїх дослідженнях автори виділили ряд ключових проблем, властивих прямому кодуванню:

1) Конкуруючі представлення - один і той же фенотип ШНМ може бути по-різному представлений в генотипі навіть в рамках одного способу кодування.

2) Незахищені інновації - при нейроеволюції зміни здійснюються додаванням або видаленням груп нейронів. І часто додавання нової структури в ШНМ веде до того, що значення її фітнес-функції знижується.

3) 3. Початковий розмір і топологічні інновації - у багатьох методиках нейроеволюції початкова популяція є набором випадкових топологій. Крім того, що доводиться витрачати певний час на відсіювання спочатку нежиттєздатних мереж, такі популяції мають тенденцію до передчасної збіжності до рішень, розмір яких не оптимальний.

Запропоновані методики рішення базується на біологічному понятті аллелю, а також на існуванні в природі процесу синапсиса - вирівнювання аллелю перед кросинговером.

У методиці передбачається, що два гени (різні особини) є гомологічними, якщо вони виникли в результаті однієї і тієї ж мутації в минулому. Іншими словами, за будь-якої структурної мутації, новому гену присвоюється унікальний номер, який потім не змінюється в процесі еволюції.

Використання історичних маркерів покладено в основу рішення всіх описаних вище завдань, за рахунок:

1) Виконання кросинговеру тільки між гомологічними генами.

2) Захисту інновацій за рахунок введення особин, які мають близькі топологічні структури, відсіваються, таким чином залишаючи місце для «новачків».

3) Мінімізації розмірності за рахунок послідовного зростання від мінімального розміру.

Найвідоміший підхід до реалізації ідеї непрямого кодування базується на системах Ліндемайра (L-системах). Граматика L-системи складається з набору правил, що використовуються для генерації рядка-опису структури. Процес застосування правил називають переписуванням рядків - до початкового символу S послідовно застосовуються правила переписування рядків, поки не отримано рядок тільки з термінальних символів.

Існує методика кодування нейронів їх координатами в двовимірному просторі. Кожна пара координат в генотипі відповідає одному нейрону. Крайні нейрони зліва вважаються вхідними, а крайні праворуч - вихідними. Після декодування всіх нейронів і розміщення їх в просторі, з кожного нейрона починає будуватися дерево зв'язків. Довжини сегментів-гілок дерева, і кут між гілками задається при описі кожного нейрона в геномі. Зв'язок між нейронами встановлюється, якщо одна з гілок графа має підхід до іншого нейрона менше, ніж встановлене заздалегідь порогове значення.

Спочатку навчання записаних таким чином мереж окремо не проводилось, вони еволюціонували разом з ваговими коефіцієнтами, які також записувалися в геном. Але подібне представлення може застосовуватися і виключно для побудови структури мережі, а ваги можуть розраховуватися і за стандартними алгоритмами.

Можливо кодування нейронів прихованого шару. Нейрони, що включаються до генотипу, отримують мінімально можливий індекс.

Оператор мутації для нейронної мережі вибирає випадковим чином один з нейронів і довільно змінює його. Для нейрона можуть бути виконані наступні дії:

– додавання прихованого нейрона;

– видалення випадково обраного прихованого нейрона разом з усіма вхідними та вихідними зв'язками;

– додавання зв'язку;

– видалення випадково обраного зв'язку;

– зміна ваги випадково обраного зв'язку на випадкову величину з діапазону [-0,5; 0,5].

При застосування генетичного алгоритму для зменшення кількості нейронів прихованого шару виникає необхідність використання генетичних операторів. Оператор схрещування - кросинговер, є одним з основних операторів в генетичних алгоритмах і

використовується для отримання хромосом нащадків шляхом рекомбінації батьківських хромосом. Разом з оператором мутації, який вносить довільні зміни в хромосоми популяції, оператор кросинговеру визначає положення популяції нащадків в просторі пошуку. Таким чином схрещування є своєрідним аналогом градієнтного спуску і багато в чому від нього залежить швидкість знаходження оптимального рішення [11].

Найчастіше схрещування проводиться над двома особинами з числа кращих - тих, чиї значення функцій пристосованості найближче розташовані до оптимуму. Результатом є також зазвичай дві особини з компонентами, узятими від батьків. Мета цього оператора - поширення хороших генів популяції і стягування щільності популяції до тих областей, які мають порівняно хорошу пристосованість [11]. І хоча немає гарантії того, що отримані рішення будуть краще, ніж батьківські, все ж в цілому даний процес призводить до знаходження нових рішень і поліпшення пристосованості популяції.

В процесі оптимізації топології мережі шляхом використання генетичного алгоритму проводиться операція схрещування батьківських хромосом для отримання двох нащадків на основі одноточкового кросинговеру. У одноточковому варіанті кросинговеру на першому етапі схрещування вибираються пари хромосом з батьківської популяції (батьківського пулу). Потім вибирається деяка точка розриву і в результаті обміну генетичною інформацією виходять два нових рішення, званих нащадками.

У двоточковому варіанті кросинговеру, як і в попередньому варіанті, на першому етапі схрещування вибираються пари хромосом з батьківської популяції. Далі вибираються дві точки розриву, а потім виходять два рішення-нащадка.

Отримані в результаті кросинговеру рішення включаються в популяцію, а потім беруть участь у відборі нарівні з іншими рішеннями.

В отриманій системі для пошуку рішення використовується популяція нейронних мереж, тобто кожна особина є штучною нейронною мережею. В якості початкового наближення (представників початкової популяції) використовуються саме нейронні мережі з одношарової топологією і випадково підібраними вагами. Що, однак, не виключає того, що в результаті еволюційного процесу можуть бути отримані багатошарові нейронні мережі.

У процесі оцінювання перевіряється працездатність закодованої нейронної мережі, яка визначає пристосованість даної особини. Після оцінювання всі особини спочатку сортуються в порядку зменшення пристосованості, а потім з отриманою популяцією взаємодіють генетичні оператори. Даний процес триває до тих пір, поки не виконується критерій зупинки.

У одноточковому варіанті кросинговеру на першому етапі схрещування вибираються пари хромосом з батьківської популяції (батьківського пулу). Це тимчасова популяція, що складається з хромосом, відібраних в результаті селекції і призначених для подальших

перетворень операторами схрещування і мутації з метою формування нової популяції нащадків. На даному етапі хромосоми з батьківської популяції об'єднуються в пари. Це проводиться випадковим способом відповідно до ймовірності схрещування. Далі для кожної пари відібраних таким чином батьків розігрується позиція гена у хромосомі, що визначає так звану точку схрещування. Якщо хромосома кожного з батьків складається з L генів, то очевидно, що точка схрещування lk є натуральне число, менше L. Тому фіксація точки схрещування зводиться до випадкового вибору числа з інтервалу [1, L-1]. Обидві батьківські структури розриваються на два сегменти по цій точці. Потім, відповідні сегменти різних батьків склеюються і виходять два генотипи нащадків. Дія одноточкового кросинговеру описані дії наведено на рисунку1.7.

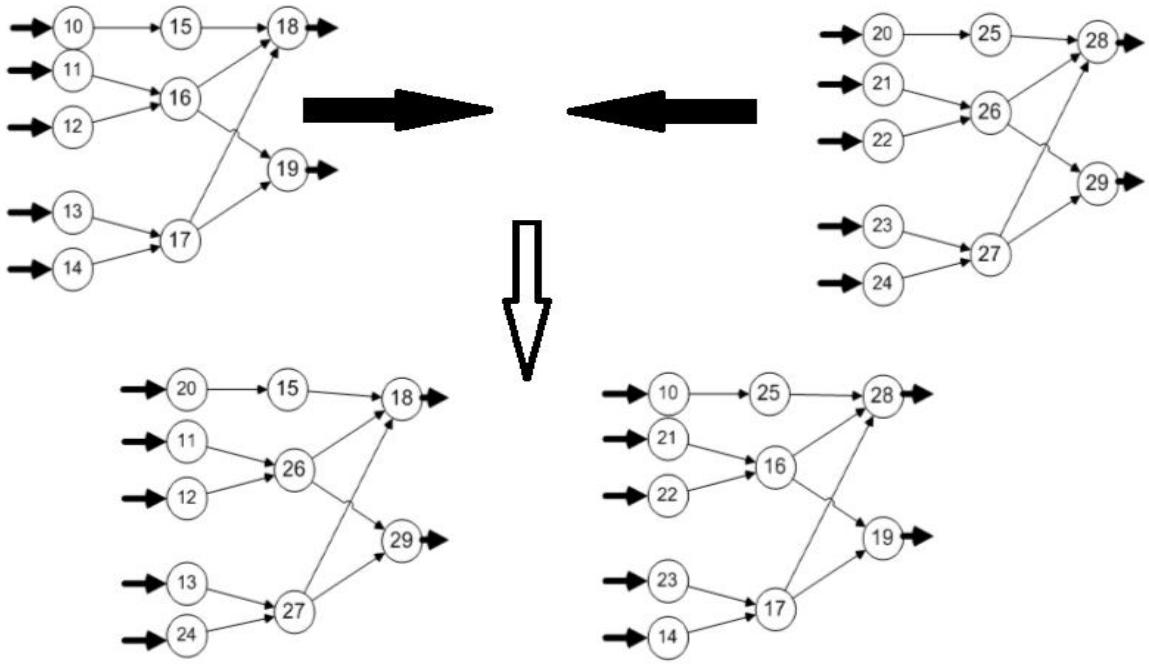


Рисунок 1.7 – Приклад одноточкового кросинговеру особин нейронних мереж

В результаті схрещування пари батьківських хромосом виходить пара нащадків:

1-ий нащадок, хромосома якого на позиціях від 1 до *lk* складається з генів першого з батьків, а на позиціях від *lk* + 1 до *L* - з генів другого з батьків;

2-ий нащадок, хромосома якого на позиціях від 1 до *lk* складається з генів другого з батьків, а на позиціях від *lk* + 1 до *L* - з генів першого з батьків.

Даний варіант оператора схрещування є найбільш поширеним варіантом, оскільки він є найбільш простим і при цьому досить ефективним варіантом отримання нових варіантів вирішення завдання.

У двоточковому варіанті кросинговеру, як і в попередньому варіанті, на першому етапі схрещування вибираються пари хромосом з батьківської популяції. Потім вибираються дві

точки розриву, і батьківські хромосоми обмінюються ділянкою генетичного коду, який знаходиться між двома цими точками. Тому фіксація точок схрещування зводиться до випадкового вибору числа з інтервалу $[1, L-1]$. Обидві батьківські структури розриваються на три сегменти за цими точкам. Відповідні сегменти різних батьків склеюються і виходить два генотипи нащадків. Після схрещування пари батьківських хромосом виходить також пара нащадків:

1-ий нащадок, хромосома якого на позиціях від 1 до $lk1$ складається з генів першого з батьків, на позиціях від $lk1$ +1 до $lk2$ - з генів другого з батьків, а потім знову $lk2$ + 1 до L - з генів першого;

2-ий нащадок, хромосома якого на позиціях від 1 до $lk1$ складається з генів другого з батьків, на позиціях від $lk1$ +1 до $lk2$ - з генів першого з батьків, а потім знову $lk2$ + 1 до L - з генів другого. Приклад роботи цього варіанту кросинговеру наведено на рисунку 1.8.

Оператор мутації для нейронної мережі вибирає випадковим чином один з нейронів та видаляє його.

Іншим генетичним оператором є відбір вирішенні, для додавання в нову популяцію. Існують різні стратегії:

- турнірний відбір;
- елітний відбір;
- рулеточний відбір.

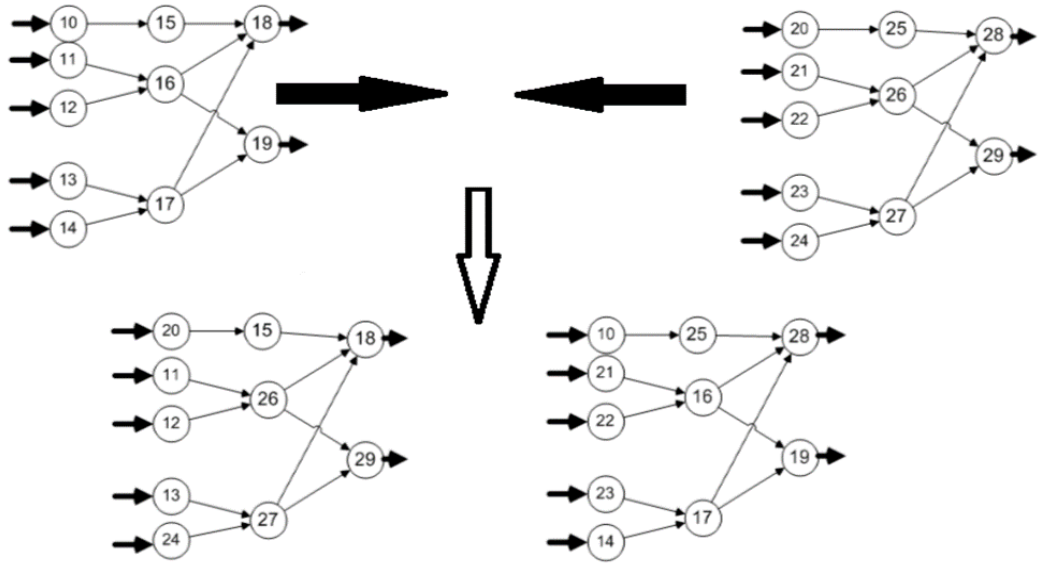


Рисунок 1.8 – Приклад двоточкового кросинговеру особин нейронних мереж

При турнірній селекції все особини популяції розбиваються на підгрупи з подальшим вибором в кожній з них особини з найкращого пристосованістю. Підгрупи можуть мати

довільний розмір, але в популяція поділяється на підгрупи по дві особини в кожній (рис. 1.8). Турнірний метод придатний для вирішення завдань оптимізації функції, до яких і зводиться задача оптимізації топології нейронних мереж. Турнірний метод діє ефективніше, ніж метод рулетки.

Елітна стратегія полягає в захисті найкращих хромосом на наступних ітераціях. При даному підході найбільш пристосовані особини завжди переходять в наступне покоління. Відбирається рівно половина від популяції, оскільки після виконання оператора схрещування розмір популяції збільшується вдвічі.

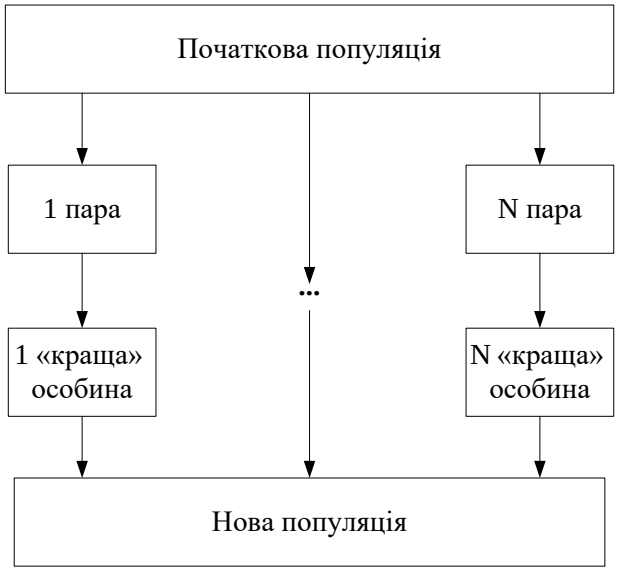


Рисунок 1.9 – Турнірна селекція

Рисунок 1.10 – Елітна селекція

У рулеточному варіанті відбору ймовірність і-й особини взяти участь в схрещуванні пропорційна значенню її пристосованості. Після цього відбирається n особин, які включаються в нову популяцію. Даний метод дозволяє уникнути передчасної збіжності, хоча і знижує середню пристосованість одержуваної популяції.

1.5 Постановка задачі дослідження

Таким чином, метою роботи є розробка і реалізація на ПЕОМ штучної нейронної мережі для розпізнавання місця знаходження автомобільних номерних знаків. Основною задачею роботи є навчання ШНМ. З поняттям навчання тісно пов'язане поняття узагальнюючої здібності. Узагальнююча здатність - це властивість моделі відображати вихідні дані в необхідні результати ($X \rightarrow Y$) на всіх множинах вихідних даних (у всіх сценаріях, а не тільки на тренувальних прикладах). Величину узагальнення оцінюють через зворотну величину - відхилення або помилку. Помилка – це чисельно виражена різниця між відповіддю моделі і необхідним реальним значенням. У більш загальному сенсі узагальнююча здатність - здатність моделі знайти певний закон, який буде описувати невідомий прихований взаємозв'язок вхідних і вихідних даних. Таким чином, спираючись на поняття узагальнюючої здатності, можна виділити основні проблемні питання машинного навчання:

– кількість даних для знаходження в них корисних знань;
– спроможність моделі навчитися на наявних даних;
– поширення отриманого закон на всю можливу множину X;
– оцінка якості роботи навченої моделі.

З математичної точки зору задача наближення знаходиться в області оптимізації або мінімізації помилки.

Таким чином, загальний алгоритм вирішення завдань в сфері машинного навчання складається з наступних кроків:

– математична постановка задачі;
– обробка даних і виділення ключових ознак;
– побудова моделі (або моделей);
– навчання моделі (моделей) і оцінка якості;
– експлуатація моделі при досягненні необхідної якості, або повернення до одного з попередніх кроків (переналаштування моделі, видобуток нових даних і тому подобне).

2 ОПИС МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Опис архітектури згорткової нейронної мережі

Одна з базових здібностей людей і тварин полягає в розпізнаванні візуальних образів під будь-яким кутом і в будь-якому положенні. Звичайно, за умови того, що витриманий певний поріг зашумленості, освітленості і т.д. Системи з розпізнавання тексту, на жаль, не володіють такими можливостями. Або вони навчені розпізнавати певний тип шрифту, або вони дуже чутливі до зрушень та нахилів, або до масштабу, або взагалі до всього. І незважаючи на те, що нейронні мережі в якості прототипу використовували принципи роботи мозку, довгий час не було вирішення цієї проблеми. Даний недолік називається проблемою чутливості до просторових спотворень. А домогтися потрібно так званої інваріативності до таких спотворень трьох основних типів: зсувам, поворотам, масштабуванню.

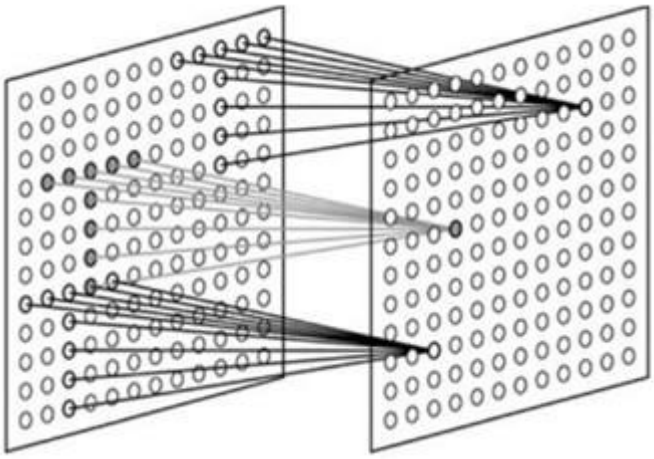


Рисунок 2.1 – Біологічний прототип згорткової нейронної мережі - рецептивні поля простих клітин, налаштованих на пошук обраного патерну в різних позиціях

Нейрони тут також упаковані не тільки в шари, але і двовимірні карти. Інформація циркулює зліва направо по нейронам такого ж типу (як і в MLP). Але головна особливість полягає в тому, що мережа не повнозв’язна, тобто кожен нейрон має свою невелику область видимості. Таке локальне сприйняття і узагальнення від шару до шару і дає вирішення проблеми чутливості до просторових спотворень, про які говорилося вище. Іншими словами, Згорткова Нейронна Мережа (ЗНМ) здатна обробляти просторову топологію.

Саме ця архітектура ШНМ лягла в основу так званих Глибоких мереж, що породило поняття глибокого навчання (Deep Learning) [5,8]. Безумовно, даний метод не позбавлений недоліків, але у багатьох задачах саме візуального розпізнавання ЗНМ є провідним рішенням.

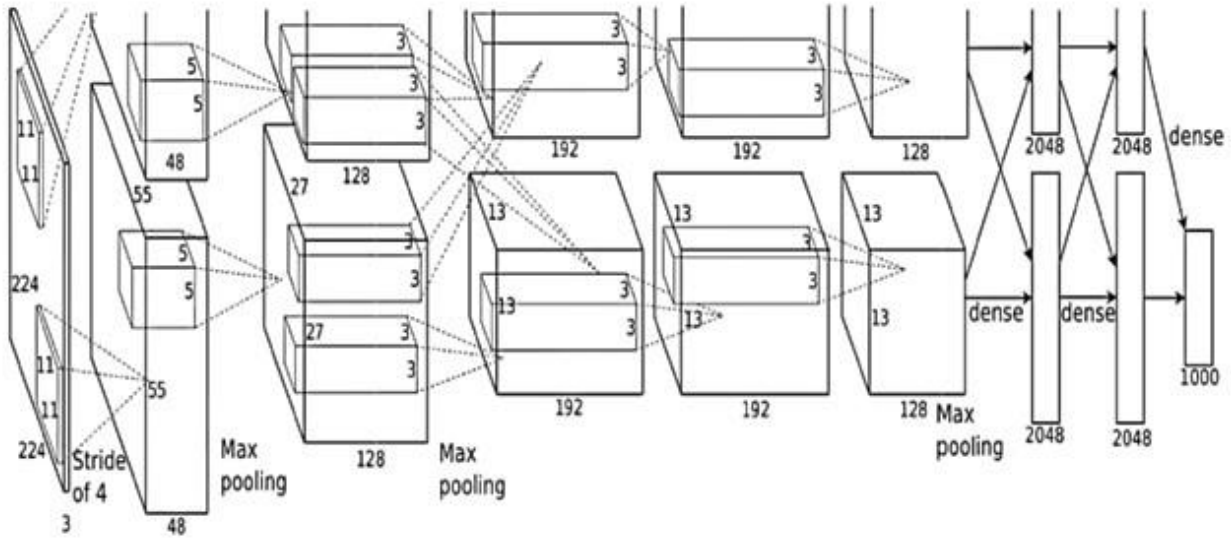


Рисунок 2.2 – Схематичне представлення глибокої мережі

Але крім цього важливу роль відіграє різноманітне комбінування певних блоків карт. У глибоких мережах дуже багато різних модулів і відгалужень (може бути і таке, що мережа має кілька входів на різних рівнях).

Однак нарощування шарів призводить до цілого ряду труднощів, як з об'ємом обчислень, так і з навчанням мережі, з чого складаються особливості, притаманні саме глибокому навчанню.

2.2 Розробка алгоритму навчання згорткової нейронної мережі

Є два класи навчальних методів: детерміністський і стохастичний. Детерміністський метод навчання крок за кроком здійснює процедуру корекції ваги мережі, засновану на використанні їх поточних значень, а також величин входів, фактичних виходів і бажаних виходів. Стохастичні методи навчання виконують псевдовипадкові зміни величин ваг, зберігаючи ті зміни, які ведуть до покращення результату.

Навчання нейронної мережі полягає в рішенні задачі багатомірної оптимізації, яка уявляє собою здачу зменшення суми квадратів похибок між експериментальним та модельним значенням:

$$\sum (Y_e - Y_m)^2 \to \min . \tag{2.1}$$

Для навчання мережі може бути використана гіпотеза Д. О. Хебба про те, що мозок збільшує підсилювальні властивості синапсів в випадках, коли нейрони перед і за синапсамі одночасно активізовані.

У більш суворому формулюванні гіпотеза Д. О. Хебба звучить так: властивість сінапса (посилення або уповільнення) змінюється пропорційно добутку перед і після синаптичної активності. Іншими словами: нейронні сполуки посилюються, якщо вони використовуються частіше.

Алгоритм реалізації цього правила для нейронної мережі може бути представлений в математичній формі: якщо w_{ij} – вага в графічній мережі від входу x_j до виходу y_j, то вага w_{ij} в момент надходження вхідного сигналу змінюється на величину $\Delta w_{ij} = \sigma x_i y_j$.

В алгоритмах навчання правило Д.О. Хебба використовується у трансформаційнгму вигляді. Замість виходу y_j приймають, причому величина дорівнює різниці між бажаним виходом і фактичним виходом, від початку моменту навчання. Тоді отримують дельта-правило:

$$\Delta w_{ij} = \sigma x_i \Delta y_j . \tag{2.2}$$

При необхідності практичного застосування нейронної мережі потрібно виконати наступні заходи:

1) поставити вхідні вектори x_1, x_2,, ..., x_n і відповідні їм вихідні вектори y_1, y_2,..., y_n.

2) вибрати для мережі необхідну топологію.

3) визначити такі значення ваг, при яких мережа буде реалізовувати задані функції. Для визначення ваги використовується процедура навчання мережі.

Після навчання ваг мережа буде підготовлена до роботи, її подальше використання може бути багаторазовим.

Алгоритм навчання мережи включає наступні етапи:

1) Для вагових коефіцієнтів w_{kj} обираються початкові значення.

2) Задається начальний вхідний вектор $\overline{X}$.

3) Змінюються ваги згідно з співвідношенням:

$$w_{ij}^{\text{н}} = w_{ij}^{\text{п}} + \Delta w_{ij}, \tag{2.3}$$

де w_{ij}^{n}, w_{ij}^{u} — минулий і наступний вагові коефіцієнти,

$$\Delta w_{ij} = (\alpha x_j \varepsilon_i), \tag{2.4}$$

$$\varepsilon_i = y_i - f\left(\sum_k w_{ik} x_k\right) = y_i - f(net_i), \tag{2.5}$$

де ε_i – різниця між цільовим і фактичним виходом в точці i; x_i, y_i – вхідний і вихідний сигнали мережі;.

4) Повернення до етапу 2.

Робота алгоритму завершується в момент, коли мережа для всіх вхідних векторів стане видавати правильний цільовий вектор.

При виконанні роботи для навчання мережі використано алгоритм зворотного поширення помилки, для реалізації якого так само необхідно підготувати навчальні дані, які складаються з векторів вхідних сигналів і бажаного вихідного результату.

Навчання являє собою послідовність ітерацій (повторень). У кожній ітерації вагові коефіцієнти нейронів змінюються з використанням нових даних з тренувальних прикладів.

Змінення вагових коефіцієнтів і складають суть алгоритму. Кожен крок навчання починається з надходження вхідних сигналів з тренувальних прикладів. Після цього необхідно визначити значення вихідних сигналів для всіх нейронів в кожному шарі мережі. Далі відбувається поширення сигналу через прихований шар.

На наступному кроці алгоритму, вихідний сигнал мережі порівнюється з бажаним вихідним сигналом, який зберігається в тренувальних даних. Різниця між цими двома сигналами називається помилкою вихідного шару мережі.

Ідея алгоритму полягає в поширенні сигналу помилки (обчисленого за крок навчання) назад на всі нейрони, чиї вихідні сигнали були вхідними для останнього нейрона.

Вагові коефіцієнти, використовувані для зворотного поширення помилки, дорівнюють тим же коефіцієнтам, що використовувалися під час обчислення вихідного сигналу, але змінюється напрямок потоку даних (сигнали передаються від виходу до входу).

Цей процес повторюється для всіх шарів мережі. Якщо помилка прийшла від декількох нейронів - вона підсумовується.

Коли обчислено величину помилки сигналу для кожного нейрона - можна скорегувати вагові коефіцієнти кожного вузла нейрона з використання похідної.

Обчислення похідної необхідно, тому що для коригування вагових коефіцієнтів при навчанні ШНМ за допомогою алгоритму зворотного поширення, використовується метод градієнтного спуску.

Коли оброблені таким чином вхідні сигнали досягають нейрона, вони збуджують його лише в тому випадку, якщо сума всіх величин потенціалів перевершить деяке порогове значення. У моделі нейрона формується сума всіх входів, множення на значення синапсів, і порівнюється з граничним значенням. Якщо сума перевищує, вихідний сигнал нейрона встановлюється на 1, в іншому випадку на 0. Якщо $\sum w_j x_j > \theta$ то нейрон збуджений, тобто $Y = 1$.

Сумуючий блок що відповідає тілу біологічного елемента, складає зважені входи алгебраїчно, створюючи вихід.

Розрізняють два основних типи мереж: мережі зі зворотним зв'язком, в яких вихідний сигнал подається назад на вхід мережі, і мережі, в яких для кожного вхідного вектору розраховується вихідний вектор, зчитування з вихідних нейронів, то прямонаправлені мережі.

Алгоритм навчання «з учителем» включає чотири етапи:

Крок 1. Для ваг w_{kj} вибираються випадкові числа.

Крок 2. Здається випадковий вхідний вектор. Позначивши його як, приймаються значення 1 або 0.

Крок 3. Змінюють ваги по співвідношенню:

$$w_{ij}^{n} = w_{ij}^{n} + \Delta w_{ij}\ , \tag{2.6}$$

де w_{ij}^{n}, w_{ij}^{n} колишній і новий ваги,

$$\Delta w_{ij} = (\alpha x_j \varepsilon_i)\ , \tag{2.7}$$

$$\varepsilon_i = y_i - f\left(\sum\nolimits_k w_{ik} x_k\right) = y_i - f(net_i)\ , \tag{2.8}$$

де ε_i - різниця між цільовим і фактичним виходом в точці; x_j, y_i - вхідний і вихідний сигнали мережі; $\alpha > 0$ - мале число.

Крок 4. Повернення до кроку 2.

Робота алгоритму завершується в момент, коли мережа для всіх вхідних векторів стане видавати правильні цільові вектори.

В процесі навчання вагові коефіцієнти w1, ..., wn і порогова величина встановлюються такими, щоб класифікація відповідала поставленому завданню.

Для випадку більш високої розмірності отримаємо розділяючу гіперплощину. При навчанні задаються вектори обох класів. Якщо мережа невірно класифікує той чи інший вектор, розділова лінія буде зміщена так, що при наступній спробі помилка не виникне.

Будь-яка задача, для якої при навчанні існує рішення, буде в кінцевому рахунку вирішена за кілька кроків. Для багатовимірної функції з n вхідними бітами і одним вихідним бітом:

– якщо задана функція з n вхідними бітами і одним вихідним бітом, вона буде лінійно розділятися тоді, якщо її гіперплощина в n-вимірному просторі допускає поділ вхідних точок на дві групи, з котрих одна відображається на 1, а інша на 0 або -1;

– якщо задана функція з n вхідними бітами і k вихідними бітами, вона лінійно розділяється тоді, якщо кожна координата її вихідного вектора має лінійну розділяємість;

– тільки лінійно розділювальна функція піддається поданням одно-шарової мережею.

В реальності вихідні дані про об'єкти у вигляді даних навчальної вибірки можуть бути неповними, неточними і навіть нечисловими. У кожному разі використовуються різні методи машинного навчання, тому вони досить різноманітні та численні.

Машинне навчання працює з об'єктами - елементарними одиницями даних, що виникають в конкретних завданнях, які характеризуються змінними, які спостерігаються, - $\bar{x}$ і прихованими змінними - $\bar{t}$, які приймають значення з деяких заздалегідь відомих множин. Головним завданням машинного навчання є автоматичне визначення взаємозалежностей між змінними, що спостерігаються, та прихованими змінними об'єкта, з тим, щоб для довільного об'єкта за його відомими компонентам можна було оцінити можливі значення прихованих компонент. Як правило, можливі взаємозалежності попередньо задаються за допомогою параметричних вирішальних правил, визначених значенням параметрів – ваг $\bar{w}$. Конкретні значення вагових коефіцієнтів визначаються в результаті навчання з використанням навчальної вибірки, яка представляє собою множину об'єктів з відомими прихованими змінними та змінними, які спостерігаються, що відповідає навчанню «з учителем» або тільки змінними, які спостерігаються, у разі навчання «без учителя» [1]. При цьому задача визначення вагових коефіцієнтів вирішального правила $\bar{w}$ за навчальною вибіркою називається задачею навчання

або налаштування (training), а задача визначення допустимих значень прихованої змінної $\bar{t}$ по заданим спостережуваним компонентам $\bar{x}$ об'єкта і заданим вагам вирішального правила $\bar{w}$ – задачею виведення (inference). Зазвичай передбачається, що кожен об'єкт описується одним і тим же набором змінних, а номенклатура спостережуваних і прихованих змінних для всіх об'єктів однакова. Прикладом такої стандартної задачі є задача класифікації, в якій прихована змінна для кожного об'єкта одна і приймає значення з кінцевої дискретної множини, а кожна спостережувана змінна може приймати дійсні, або дискретні значення. Якщо прихована змінна об'єкта є не дискретною, а безперервною, задача називається задачею відновлення регресії, що є ще однією зі стандартних і добре вивчених задач машинного навчання.

Загальна задача процесу машинного навчання проілюстрована за наведеною на рисунку 2.3 схемою

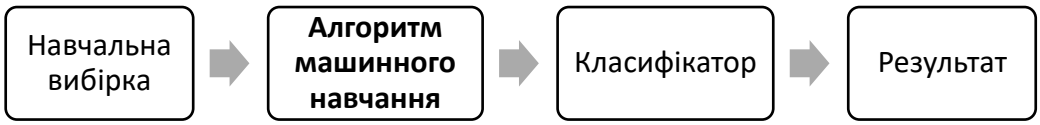


Рисунок 2.3 - Загальна задача машинного навчання

У різний час робилися неодноразові спроби ввести деяку універсальну мову опису різних постановок і методів розв'язання задач машинного навчання. Починаючи з 90-х р.р. минулого століття широкого поширення отримав байєсівський формалізм [5, 8]. При його використанні передбачається, що залежність між спостережуваними змінними об'єкта, вагами вирішального правила і прихованими змінними об'єкта моделюється з допомогою спільного розподілу на ці групи змінних $p(\bar{x}, \bar{t}, \bar{w})$. Якщо має місце тільки задача визначення прихованих змінних за спостережуваними, розглядаються дискримінативні моделі (discriminative models) $p(\bar{t}, \bar{w} \mid \bar{x})$. Значення спостережуваних змінних $\bar{x}$ в цьому випадку моделюються через припущення, що вони є відомими на всіх етапах виконання завдання, і спільний розподіл стає простішим. В стандартних постановках задачі машинного навчання передбачалося, що приховані змінні кожного об'єкта залежать тільки від спостережуваних змінних цього об'єкта, причому вид залежності визначається параметрами $\bar{w}$. Це відповідає представленню:

$$\rho(T, \overline{w} \mid X) = \prod_{i=i}^{n} \rho(\overline{t_l} \mid \overline{x_l}, \overline{w}) \rho(\overline{w}). \tag{2.4}$$

При використанні такого формалізму задача налаштування параметрів $\bar{w}$ вирішується, наприклад, знаходженням найбільш вірогідного значення:

$$\arg\max \frac{\rho(T^{tr}, \bar{w} \mid x^{tr})}{\rho(T^{tr} \mid x^{tr})} = \arg\max \rho(T^{tr}, \bar{w} \mid X^{tr}). \tag{2.5}$$

В той час задача виведення вирішується наступним шляхом:

$$\arg\max \frac{\rho(T^{tr}, \bar{w} \mid x^{tr})}{\rho(T^{tr} \mid x^{tr})} = \arg\max \rho(\bar{t} \mid \bar{x}, \bar{W}_{MP}). \tag{2.6}$$

Таким чином, для формулювання і рішення задачі машинного навчання достатньо знати дві функції: p($\bar{t}$| $\bar{x}$, $\bar{w}$) і p ($\bar{w}$). Перша функція, звана функцією правдоподібності, характеризує ступінь «істинності» отриманого прогнозу прихованої змінної, другий компонент, є апріорним розподілом, при зміні якого існує можливість впливу на результат процедури налаштування, тобто на $\bar{w}$. При цьому спосіб адекватного визначення апріорного розподілу неочевидний. Апріорний розподіл є ефективним способом контролю складності вирішального правила і дозволяє здійснювати регуляризацію процедури налаштування. Замість знаходження ваг, що забезпечують найменшу помилку прогнозу на навчальній вибірці та загрожує ефектом перенавчання, доцільно дещо зменшити точність, але забезпечити здатність отримати ту ж помилку прогнозу на інших об'єктах генеральної сукупності. В будь-якій моделі машинного навчання можна виділити найпростіше вирішальне правило (наприклад, що відповідає нульовим значенням ваг), в якому міститься мода унімодального апріорного розподілу. Чим більше відстань поточних значень ваг від моди, тим менше значення p($\bar{w}$). Ширина апріорного розподілу задається параметром регуляризації, який може бути досить ефективно знайдений процедурою змінного контролю або байєсівською процедурою вибору моделі [6].

Перевагою байєсівського формалізму виявилася можливість враховувати численні апріорні знання про можливі залежності між спостережуваними та прихованими змінними об'єктів, які є в багатьох прикладних задачах. Наприклад, відомо, що надійність позичальника (прогнозована змінна) повинна позитивно корелювати з його доходом і освітою (спостережувані змінні). Такі попередні значення у алгоритмах загального призначення, виражені у вигляді апріорного розподілу на $\bar{w}$, дозволили домогтися значного збільшення точності і знизити ефект перенавчання, завдяки адаптації їх під специфіку конкретного завдання.

Зведення практично будь-якої задачі машинного навчання до подібного формалізму відкриває уніфікований спосіб аналізу різних моделей машинного навчання, наприклад, з метою дослідження їх узагальнюючої здатності або вироблення ефективних наближених методів настройки і виведення загального призначення.

У рамках байєсівського формалізму розроблений ряд спільних методів для оцінки апостеріорних розподілів, байєсівського виведення, автоматичного вибору моделі та ін. Не менш важливим успіхом байєсівського формалізму стала можливість успішного узагальнення результатів і методів класичного машинного навчання на абсолютно нові задачі [6].

Методи глибинного навчання (deep learning) використовують механізми нейронних мереж [8]. Недоліками традиційних нейронних мереж стали:

– критична залежність якості настройки ваг мережі від вибору початкового наближення;

– велика схильність до перенавчання укупі зі слабкими можливостями контролю узагальнюючої здатності мережі;

– велика кількість локальних мінімумів функціонала якості, більшість з яких є несприйнятими.

З іншого боку, незаперечною сильною стороною нейронних мереж стало відкриття методу зворотного поширення помилки, який дозволив відстежувати вплив внутрішніх шарів мережі на якість прогнозу прихованих змінних об'єктів навчальної вибірки.

Останнім часом став активно розвиватися напрямок, який одержав назву глибинного навчання [5]. В його основі лежать нейронні мережі, які зазнали значних змін, а саме либинне навчання будує не дискримінативність, а породжує моделі, в яких моделюється загальний розподіл $p(\bar{x}, \bar{t}, \bar{w})$, на відміну від дискримінативних моделей, що дозволяє, наприклад, генерувати нові об'єкти.

У найбільш поширеній постановці всі змінні об'єктів передбачаються бінарними. Це полегшує моделювання залежностей між змінними об'єкта.

Кожен шар мережі спочатку навчається незалежно, проходячи процедуру преднавчання. Це дозволяє знайти хороше початкове наближення для подальшого запуску алгоритму зворотного поширення помилки. Кожен шар, в залежності від обраної моделі, являє собою обмежену машину Больцмана або згорткову нейронну мережу.

Для навчання використовуються сотні тисяч і мільйони об'єктів. Такі гігантські вибірки дозволяють налаштовувати мережі з десятками тисяч параметрів, без ризику перенавчання. Навчені таким чином мережі, не просто дозволяють моделювати складні об'єкти (наприклад, тексти або зображення), але і генерують в процесі навчання інформативні ознакові описи, які

можуть бути використані іншими, більш простими алгоритмами машинного навчання як спостережувані змінні об'єкта.

Методологія глибинного навчання дозволила домогтися нових результатів при навчанні на великих і надвеликих обсягах даних. В даний час вона є одним з найбільш перспективних шляхів розвитку машинного навчання.

Традиційно, методи непараметричної статистики визначалися як розділ статистики, в якій число параметрів, що описують дані (наприклад, параметри щільності розподілу об'єктів) не фіксоване, а зростає зі збільшенням числа об'єктів [6]. Широкого застосування набула задача визначення числа кластерів в зростаючій вибірці об'єктів. Дана задача тим більш актуальна, що загальноприйнятих методів визначення на сьогоднішній день не існує. Чим більше об'єктів надходить в розпорядження, тим з більшим дозволом ми можемо знаходити в них структуру, виділяючи кластери схожих між собою об'єктів. У разі досить неоднорідної вибірки число кластерів має поступово збільшуватися в міру надходження нових об'єктів. Тому постає задача визначення швидкості зростання числа кластерів з ростом даних вибірки. Формально дана залежність може бути описана формулою Байєса, яка об'єднує апріорні уявлення з поточними спостереженнями:

$$Posterior = \frac{Likelihood * \Pr ior}{Evidence}. \tag{2.7}$$

У непараметричному випадку розподіл задається за допомогою випадкових процесів. В даному випадку, це процес Діріхле [6]. З його допомогою, вдається не тільки розрахувати для будь-якого розбиття довільного числа об'єктів на кластери його апріорну ймовірність, але і врахувати характеристики об'єктів (їх спостережувані змінні), щоб перейти до апостеріорного розподілу на можливе розбиття.

Як це часто буває при застосуванні байєсівських методів, апостеріорний розподіл має гострий пік, який відповідає сталому розбиттю вибірки об'єктів на деяке число кластерів. Фактично, процес Діріхле дозволяє задавати розподіл над можливими дискретними розподілами. При виведенні використовуються наближені методи Монте-Карло з марківськими ланцюгами і методи варіаційного виведення. Описана схема допускає численні узагальнення на випадок ієрархій кластерів та множинних вибірок.

Ще однією областю машинного навчання, яка активно розвивається, є навчання з підкріпленням, призначене для навчання агентів (автономних модулів, самостійно приймаючих рішення в реальному часі на підставі наявних даних) в умовах невизначеності, що породжується як неповнотою інформації про навколишнє оточення, так і можливими діями

інших агентів. Залежно від поточного стану середовища і дій агентів розраховується функція вигоди, яку отримає агент в наступний момент часу. В ролі спостережуваних змінних об'єкта виступає інформація, що надається агентом, а прихованими змінними є довгострокові оцінки отриманої вигоди. Важливою перевагою алгоритмів навчання з підкріпленням є можливість навчання агента «з нуля» за рахунок балансованого поєднання режимів «дослідження-використання» і вивчання стратегій, які дозволяють жертвувати малим зараз заради отримання більшої вигоди в подальшому [6].

Побудова математичної моделі будь-якої оптимізаційної складається з наступних етапів:

– визначення цілі;

– визначення сукупності шуканих величин;

– запис цільової функції як функції шуканих величин, яка визначає ціль вирішуваної задачі;

– запис системи обмежень як функцій тих самих шуканих величин та представлення у вигляді нерівностей згідно змісту задачі.

Вирішення задачі навчання представлено наступними алгебраїчними об’єктами:

– величиною Z*, яка є кількісним значенням цілі;

– сукупністю $\{ x^*_1, x^*_2, \dots x^*_n \}$, що є оптимальними значеннями шуканих величин, які складають варіант x*.

Для вирішення задач математичного програмування, окрім градієнтних методів [7,8] використовуються методи можливих напрямків [11]. Ідея методів можливих напрямків, близька до ідеї градієнтних методів для задач з обмеженнями, полягає в наступному: на кожній ітерації визначається допустимий напрям на множині X, уздовж якого функція f (x) зростає (такий напрямок називається можливим напрямом зростання функції f (x)) , і по ньому здійснюється крок. Фактично в методі проекції градієнта і в методі умовного градієнта знаходимо такі напрямки. У методах можливих напрямків вихідним пунктом є опис всіх допустимих напрямів і вибір з них такого, уздовж якого функція f (x) зростає і бажано якнайшвидше.

Група методів множників Лагранжа заснована на зведенні рішення задачі математичного програмування до знаходження сідлової точки функції Лагранжа [7, 15]. Методи множників Лагранжа є різні процедури пошуку сідлової точки. В якості такої процедури можна взяти, наприклад, ітеративний процес як в методі проекції градієнта, застосувавши його для підйому по змінної x і спуску по змінної y.

Наявність обмежень істотно ускладнює завдання пошуку екстремуму функції. Майже всі викладені методи вирішення екстремальних завдань включали спеціальні складні процедури, які враховують наявність обмежень. Від цього недоліку позбавлені методи штрафних функцій [8], які дозволяють в досить загальному випадку зводити екстремальні завдання зі складними

обмеженнями до завдань без обмежень або з простими обмеженнями, а в поєднанні з методами пошуку безумовного екстремуму служать універсальним засобом вирішення завдань математичного програмування.

В процесі навчання нейронної мережі виникає задача визначення найкращих значень її параметрів та структури. Така задача є оптимізаційною. В сучасних умовах оптимізаційні задачі і задачі прийняття рішень моделюються і вирішуються в самих різних областях техніки [1]. До навичок математичного обґрунтування прийняття рішень відносяться навички математичного моделювання оптимізаційних задач, вибору адекватного математичного забезпечення (методу, алгоритму, програмної системи) з необхідним обґрунтуванням, аналізу отриманих результатів та їх інтерпретації в термінах предметної області.

Завдання оптимізації в цілому зводиться до задачі пошуку екстремуму (мінімуму або максимуму) цільової функції з заданими обмеженнями. Її математична постановка виглядає наступним чином: необхідно визначити значення вектору змінних x = (x1, x2,..., Xm), які задовольняють обмеженням виду:

$$gi(x1, x2, \ldots, xm) \leq bi \tag{2.8}$$

для всіх i = 1,. . . , K і при яких досягається максимум або мінімум цільової функціїf (x1, x2,..., xm):

$$f(x1, x2,..., xm) \rightarrow (\max, \min). \tag{2.9}$$

Допустимим рішенням задачі називається таке рішення, яке задовольняє її обмеженням (1). Сукупність допустимих рішень задачі називають областю допустимих рішень (ОДР). Остаточним рішенням задачі є пара (x*, f*(x*)), що складається з оптимального рішення і оптимального значення цільової функції.

Основні твердження теорії оптимізації стосуються, в першу чергу, виду допустимої множини рішень X, а також існування і властивостей оптимальних рішень задачі. Можливі випадки допустимої множини рішень задачі полягають у наступному [2, 4].

– допустима множина рішень є порожньою.У даному випадку відповідає взаємна суперечливість обмежень, що входять в задачу;

– допустима множина - опуклий обмежений багатогранник;

– допустима множина - опукла необмежена багатогранна множина.

Два останніх випадки досить легко уявити в дво- або тривимірному вимірюванні. У просторі більшої розмірності поняття багатогранника (багатогранної множини) вводиться

абстрактно як перетин гіперплоскостей і гіперполуплоскостей, що визначаються відповідними лінійними рівняннями і нерівностями, що входять до складу обмежень задачі. Характерною властивістю багатогранника є наявність в ньому особливих точок - вершин.

Можливі випадки оптимальних рішень задачі оптимізації:

1) Задача не має оптимальних рішень. Даний випадок може виникнути: або тоді, коли допустима множина рішень є порожньою, або коли допустима множина являє собою необмежену багатогранну множину, і цільова функція на ньому необмежено зростає (якщо L -> max) або необмежено убуває (при L -> min).

2) Задача має єдине рішення (єдиний оптимальний план). Це рішення обов'язково збігається з однією з вершин допустимої множини.

3) Задача має множину оптимальних рішень, задану деяким лінійним об’єктом - ребром, межею, гіпергранью і т.д. Серед точок цього лінійного об’єкту є і вершини допустимої множини.

Якщо перевірка на оптимальність наявного варіанту - мета, то її досягнення пов'язане з визначенням того, що відсутній небазисний вектор, який може поліпшити наявний варіант. Має існувати правило такого вектора, тобто всі небазисні вектори оцінюються певним чином, і по даній оцінці визначається потенціал кожного вектора.

Методи оптимізації дають велику різноманітність алгоритмів рішення даного завдання. В цілому алгоритми пошуку реалізують методи спуску до екстремуму, при яких значення цільової функції послідовно покращується до досягнення екстремуму. Залежно від можливості знаходження алгоритмом локального або глобального екстремуму, вони діляться на алгоритми локального і глобального пошуку.

Алгоритми пошуку локального екстремуму призначені для визначення одного з локальних екстремумів на множині допустимих рішень, в якій цільова функція приймає максимальне або мінімальне значення. При їх побудові можуть використовуватися як детермінований спуск в область екстремуму, так і випадковий пошук. Серед детермінованих методів розрізняють методи нульового порядку і градієнтні (1-го і 2-го порядку). Перші засновані на обчисленнях тільки значень оптимізованої функції. Другі використовують часткові похідні відповідного порядку. Для пошуку екстремуму у випадках, коли вид оптимізованої функції відомий не повністю, або її структура занадто складна, застосовуються методи стохастичного програмування або нейронних мереж. Ефективність процедури пошуку оптимуму - можливість відшукання рішення і збіжність до вирішення по швидкості залежать від виду функції і застосовуваного для неї методу. Виникає необхідність визначення стратегії кожного методу більш детально, досліджуючи для визначеності мінімізацію цільової функції.

З прямих методів нульового порядку найбільш відомі методи:

– координатного спуску - почергова оптимізація параметрів уздовж осей одним з відомих одновимірних методів;

– спірального координатного спуску;

– обертових координат (метод Розенброка);

– симплексного пошуку;

– Хука-Дживса з пошуком за зразком і інше.

Метод координатного спуску полягає в тому, що в якості напрямків траєкторії спуску від попередньої точки пошуку X(k-1) до подальшої X(k) приймаються по черзі напрямки координатних осей Xi (i = 1, 2,..., N). Після спуску на один крок по координаті X1 відбувається перехід до спуску на один крок по координаті X2, а потім рух уздовж координати X3 і т.д., поки не буде знайдена наступна точка пошуку X(k) з координатами $X_1^{(k)}$, $X_2^{(k)}$,. . . , $X_n^{(k)}$. Рух по траєкторії спуску від попередньої точки X(k-1) до подальшої X(k) триває до тих пір, поки не буде досягнуто точку мінімуму X * цільової функції, що визначаються точністю обчислень. Для пошуку координат точки X(k) на кожному кроці ітерації можна використовувати будь-який з методів одновимірної мінімізації: метод золотого перетину, метод поділу відрізка навпіл, метод інтерполяції-екстраполяції та ін.

Метод спірального координатного спуску відрізняється від розглянутого вище лише тим, що крок h змінюється кожен раз при переході від пошуку мінімуму по одній змінній до пошуку мінімуму за іншою змінною. У тривимірному просторі це нагадує спуск в западину по спіралі. Зазвичай цей метод дає деяке скорочення часу пошуку. Методи координатного спуску недостатньо ефективні для поверхонь з "оврагами", так як в цьому випадку отримання рішення з необхідною точністю не гарантоване. Це викликано тим, що в разі "оврагу", повернутого щодо осей, спроба просування в будь-якому напрямку може викликати "погіршення" цільової функції. У той же час просування уздовж "оврагу" може давати "покращення" цільової функції.

Метод Розенброка спрямований на усунення одного з недоліків метода покоординатного спуску - високу чутливість до вибору системи координат. В процесі пошуку методом Розенброка проводиться поворот координатних осей так, щоб одна з осей була направлена вздовж напрямку "оврагу". На кожній ітерації процедура здійснює ітеративний пошук вздовж n лінійно незалежних і ортогональних напрямків. Коли отримана нова точка в кінці ітерації, будується нова множина ортогональних векторів. Побудова напрямків пошуку полягає в наступному: нехай d1,. . . , dn - лінійно незалежні вектори, за нормою рівні одиниці. Приймемо, що ці вектори взаємно ортогональні (di * dj = 0) для i=j. Починаючи з поточної точки x(k), цільова функція послідовно мінімізується уздовж кожного з напрямків, в результаті чого виходить точка x(k + 1). Зокрема, x(k + 1)-x (k) = Σ (λj * dj), де λj - довжина кроку у напрямку dj.

Новий набір напрямків q1,. . . , qn будується за допомогою процедури Грамма-Шмідта. Нові напрямки, побудовані описаним чином, є лінійно незалежними і ортогональними.

Таким чином, метод Розенброка дозволяє уникнути проблем, пов'язаних з отриманням рішення заданої точності для поверхонь с "оврагами". Однак такий підхід порівняно збільшує час пошуку, що є відносним недоліком описаного методу.

Регулярний симплекс в N-мірному просторі - це багатогранник, утворений N+1 рівновіддаленими точками - вершинами симплекса. Його важливою властивістю є можливість побудови нового симплекса на будь-якій грані вихідного шляхом перенесення обраної вершини на деяку відстань уздовж прямої, що з'єднує цю вершину з центром ваги інших вершин симплекса.

Робота алгоритму починається з побудови регулярного симплекса в просторі незалежних змінних задачі і оцінювання значення цільової функції в його вершинах. Потім точка x(j) з найбільшим значенням функції відбивається через центр ваги інших точок:

$$x_c = \frac{1}{N}\sum_{i=0}^{N} X_i. \tag{2.9}$$

Нова точка використовується як вершина нового симплекса. Ітерації тривають до тих пір, поки або не буде досягнута точка мінімуму, або не почнеться циклічний рух по двох або більше симплексах. При цьому слід користуватися трьома правилами:

– якщо точка з максимальним значенням функції отримана на попередній ітерації, то замість неї береться точка з наступним за величиною значенням функції;

– якщо деяка вершина симплекса не виключається більш ніж на N ітераціях, то зменшити розміри симплекса за допомогою деякого коефіцієнта і побудувати новий симплекс, використовуючи в якості базової точку з найменшим значенням функції;

– пошук закінчується, коли розміри симплекса і різниці значень функції в вершинах стануть досить малі.

Всі точки прямої, що проходить через x(j) і x_c визначаються формулою:

$$x = x^{(j)} + \lambda(x_c - x^{(j)}). \tag{2.10}$$

Переваги методу:

– простота реалізації;

– мала кількість заздалегідь встановлених параметрів;

– алгоритм ефективний при помилках у визначенні значень цільової функції.

Недоліки методу:

– виникають труднощі пов'язані з масштабуванням задачі (в реальних задачах різні змінні часто не співставні між собою за значеннями);

– алгоритм працює повільно (не використовується інформація попередніх ітерацій);

– не існує простого способу зміни розмірів симплекса без перерахунку всіх значень цільової функції.

Алгоритм пошуку по симплексу можна вдосконалити шляхом введення множини векторів, які задають напрямки пошуку. Ці вектори повинні бути лінійно-незалежні і утворювати базис в просторі незалежних змінних. Таким умовам задовольняє система координатних напрямків. Метод Хука-Дживса - це комбінація пошуку за напрямками і пошуку за зразком.

Досліджуваний пошук визначається наступним чином. Задається величина кроку, яка може бути різною для різних координатних напрямків і змінюватися в процесі пошуку. Якщо значення цільової функції в пробній точці не перевищує значення у вихідній, то крок пошуку розглядається як успішний. В іншому випадку, необхідно повернутися в попередню точку і зробити крок в протилежному напрямку. Після перебору всіх N координат досліджуваний пошук закінчується. Отримана точка називається базовою.

Пошук за зразком полягає в реалізації єдиного кроку з отриманої базової точки вздовж прямої, що з'єднує її з попередньою базовою точкою.

Нова точка будується за формулою:

$$x_p^{(k+1)} = x^{(k)} + \lambda(x^k - x^{(k-1)}). \tag{2.11}$$

де x(k) - поточна базова точка; x(k-1) - попередня базова точка; x(k+1) - точка, побудована при русі за зразком; λ - параметр алгоритму.

Якщо рух за зразком не приводить до зменшення цільової функції, то точка x (k+1) фіксується в якості тимчасової базової точки і знову проводиться досліджуваний пошук з цієї точки. Якщо в результаті виходить точка зі значенням функції меншим, ніж в x(k), то вона розглядається як нова базова точка x(k + 1). Якщо досліджуваний пошук невдалий, то відбувається повернення в x(k) і проводиться пошук в протилежному напрямку. Якщо він також не приводить до успіху, то величина кроку зменшується і відновлюється досліджує пошук. Пошук завершується, коли величина кроку стає досить малою.

Перевагами даного методу є проста стратегія пошуку і невеликий обсяг необхідної пам'яті. Однак алгоритм заснований на циклічному русі по координатам, що може привести до виродження алгоритму в нескінченну послідовність досліджуваний пошуків без пошуку за зразком.

Ітераційні процеси оптимізації, напрямок пошуку яких на кожному кроці співпадає з антиградієнтом функції, називаються градієнтними методами [2].

Алгоритм найшвидшого спуску реалізує ітераційну процедуру руху до мінімуму з довільно вибраної початкової точки в напрямку найбільш сильного зменшення функції, визначеному в околиці поточного значення аргументу, мінімізованої функції. Такий напрям протилежний напрямку, що задається вектором градієнта ∇f (x) функції, що мінімізується f(x). Загальна формула для знаходження значення аргументу:

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^k. \tag{2.12}$$

Алгоритми найшвидшого спуску розрізняються за способом визначення кроку λ(k). Якщо крок λ(k) не залежить від k (є постійним), то в околиці екстремуму спостерігатимуться незгасаючі коливання, амплітуда яких залежить від величини λ і від форми функції, що мінімізується. Використання постійного кроку:

– дозволяє побудувати найбільш простий варіант алгоритму;

– при великих значеннях λ забезпечує швидкий рух до екстремуму, але призводить до помітних змін на околицях;

– при малих значеннях λ призводить до низької швидкості збіжності до екстремуму;

– інформація про прийнятну величину кроку λ може бути отримана тільки в ході налагодження алгоритму, тому що ніяка інформація про властивості функції, що мінімізується, не використовується.

Покращенням методу з постійним кроком, що дозволяє усунути коливання в околиці екстремуму без істотного ускладнення алгоритму, є використання кроку λ, величина якого зменшується по ходу ітераційного процесу і залежить від номера ітерації k. Поряд з перевагами такий підхід має і недолік, а саме незв'язаність значень λ(k) з формою функції. Якщо далеко від екстремуму функція f(x) має малий градієнт, швидкість збіжності може виявитися неприпустимо повільної. Ця проблема вирішується шляхом модифікації алгоритму, яка може бути здійснена різними способами.

Існує алгоритм найшвидшого спуску з кроком, довжина якого залежить від властивостей функції, що мінімізується (використання похідної по напрямку). Нехай знайдена точка x(k) і в

цій точці визначено напрямок руху до мінімуму, що задається вектором одиничної довжини s(k). Тоді положення наступної точки x(k+1) і значення функції f (x(k+1)) в цій точці залежать тільки від єдиної скалярної змінної λ(k). Тоді виникає ідея не змінювати напрямки до тих пір, поки функція f(x) в напрямку s(k) убуває. Точка x(k+1) відповідає мінімуму f(x) у напрямку s(k). Тоді в точці x(k + 1) має бути визначено новий напрямок руху до мінімуму s(k + 1), який буде ортогональним попередньому s(k). Значення визначається з умови мінімуму квадратичної апроксимації f(x(k+1)) по λ (k).

Ітераційний процес закінчується при виконанні однієї з наступних умов:

– якщо в завданні необхідно значення функції в точці оптимуму (а не значення аргументу x), то слід припинити обчислення, якщо, починаючи з k* тої ітерації, для якої абсолютне значення нормованої різниці між значеннями функції в сусідніх точках не перевищує наперед заданого малого числа ε> 0. Збіжність такого типу називається збіжністю по функціоналу;

– якщо згідно постановки завдання необхідно визначити саме значення аргументу (а не функції), то слід припинити обчислення, якщо, починаючи з k** ітерації, для якої абсолютне значення нормованої різниці між значеннями аргументу в "сусідніх" точках не перевищує наперед заданого малого числа δ>0. Збіжність такого типу називається збіжністю за параметрами.

При використанні обох правил зупинки необхідні додаткові заходи для запобігання занадто тривалих обчислень (наприклад, переривання обчислень при k> k***, де k*** - задане число.

Існує алгоритм найшвидшого спуску з кроком, довжина якого залежить від властивостей функції, що мінімізується (використання других похідних). Метод Ньютона заснований на квадратичній апроксимації функції, що мінімізується в околиці точки x(k). Мінімум квадратичної функції легко знайти, прирівнюючи її градієнт нулю. Можна відразу ж обчислити положення екстремуму і вибрати його в якості наступного наближення до точки мінімуму. Обчислюючи точку нового наближення за формулою: $x^{k+1} = x^k + \Delta x^k$ і розкладуючи f(x(k+1)) в ряд Тейлора, отримаємо формулу квадратичної апроксимації функції в наступній точці.

Переваги методу Ньютона:

– якщо мінімізована функція є квадратичною, то метод дозволить знайти мінімум за один крок;

– якщо мінімізована функція належить до класу поверхонь обертання (тобто володіє симетрією), то метод також забезпечує збіжність за один крок;

– якщо функція несиметрична, то метод не забезпечує збіжність за кінцеве число кроків. Але для багатьох функцій досягається набагато більш висока швидкість збіжності, ніж при використанні інших модифікацій методу найшвидшого спуску.

Недоліки методу Ньютона пов'язані з необхідністю обчислень і обернення матриць других похідних. При цьому не стільки витрачається машинний час, скільки можуть з'явитися значні обчислювальні похибки, якщо матриця виявиться погано обумовленою.

2.3 Аналіз перспектив розвитку згорткових нейронних мереж при вирішенні задач знаходження місця розташування об’єктів

Серед задач, вирішуваних на основі згорткових нейронних мереж, мають місце такі області:

– розпізнавання образів;
– розпізнавання мови;
– розпізнавання рукописного введення;
– розпізнавання жестів;
– діагностика (медична, технічна);
– біоінформатика;
– хемоінформатика;
– прогнозування часових рядів;
– виявлення спаму;
– біржовий технічний аналіз;
– ранжирування результатів пошуку;
– створення рекомендаційних систем.

Одним з найбільш вагомих результатів використання методів машинного навчання для опису завдань обробки даних є апарат імовірнісних графічних моделей [5]. Графічні моделі дозволили радикально переглянути області застосування методів машинного навчання і аналізу даних за рахунок відмови від вимоги незалежності прихованих змінних для різних об'єктів. Дискримінативна модель вибірки об'єктів задається спільним розподілом, який, на відміну від класичного випадку, не факторизується за окремими об'єктами:

$$p(\bar{t}, \overline{w} \mid \bar{x}) = p(\bar{t} \mid \bar{x}, \overline{w}) * p(\overline{w}). \tag{2.13}$$

Прикладами областей, в яких використовується даний підхід, заснований на відмові від припущення про незалежність прихованих змінних, є:

1) Соціальні мережі. Користувачі соціальних мереж характеризуються, як спостережувані змінні (наприклад, анкетна інформація, яку користувач повідомив про себе в мережі), так і прихованими змінними (наприклад, його реальні інтереси, схильність до позитивної реакції на адресну рекламу тощо). Хоча є можливість формального аналізу кожного користувача незалежно, видається досить очевидним, що інформація про значення прихованих змінних його друзів, може значно розширити уявлення про даного користувача.

2) Комп'ютерний зір. У задачі семантичної сегментації зображень, що є першим етапом будь системи комп'ютерного зору, потрібно зіставити кожному пікселю деяку мітку класу, яка відповідає предметові, в зображення якого входить даний піксель. Очевидно, що крім інформації про даний піксель (колір, значення дескрипторів, інтенсивність та ін.) або інші пікселі, важливу роль відіграють мітки сусідніх пікселів, тому що неявно передбачається, що сусідні пікселі частіше всього мають однакові мітки.

3) Імітаційне моделювання. При моделюванні середовищ взаємодіючих агентів (наприклад, транспортних потоків в містах) стан кожного агента залежить від станів інших агентів, що знаходяться в межах зони взаємодії. Стан кожного агента можна розглядати як приховану змінну об’єкта, що залежить від прихованих змінних інших об'єктів. Дослідження таких взаємодій грає важливу роль, тому що дозволяє встановити умови стрибкоподібних переходів від локальних взаємодій до глобальних (фазові переходи), наприклад, коли через різке короткочасне гальмування однієї машини в потоці виникає багатокілометрова пробка.

4) Коллаборативна фільтрація (collaborative filtering). З розвитком інтернеткомерції все більшу актуальність отримують рекомендаційні сервіси. В ситуації, коли відвідувач фізично не може переглянути весь асортимент інтернет-магазину, що включає в себе десятки тисяч найменувань, виникає задача формування обмеженого списку товарів, які його потенційно можуть зацікавити. Ясно, що крім спостережуваних змінних об'єкта, яким виступає клієнт, що характеризують його соціально-демографічний профіль і історію покупок, необхідно аналізувати покупки інших клієнтів і близькість їх переваг до переваг розглянутого клієнта.

Характерне число об'єктів у вибірці, з яким доводиться стикатися в сучасних задачах, становить величину порядку десятків тисяч - мільйонів. Основна складність, яка виникає при спробі побудувати ймовірнісну модель, яка містить взаємозалежності між прихованими змінними об'єктів, полягає в неможливості задати такий розподіл в загальному вигляді. Нехай є тисяча об'єктів, у кожного з яких є одна прихована змінна, що приймає два значення. Для того, щоб задати $p(\bar{t} \mid \bar{x}, \bar{w})$ знадобилося б задати $2^{1000} \approx 10^{300}$ значень ймовірностей. Така кількість на

багато порядків перевершує обсяги доступної пам'яті будь-якого сховища даних. При використанні графічних моделей передбачається, що спільний розподіл може бути представлено у вигляді добутку факторів, кожен з яких залежить від невеликої підмножини об'єктів, причому підмножини перетинаються. Завдяки цьому вдається змоделювати ситуації, коли прихована компонента довільного об'єкта залежить від прихованої компоненти кожного з решти об'єктів вибірки. З іншого боку, за рахунок факторизації, можна зменшити вимоги до пам'яті аж до лінійних за кількістю об'єктів, що дозволяє зберігати спільні розподіли на сотні тисяч об'єктів.

Ключовим поняттям, необхідним для розуміння логіки роботи апарату графічних моделей, є поняття умовної незалежності випадкових величин. Випадкові величини *a* і *b* називаються незалежними за умови *c*, якщо виконується рівняння:

$$p(a, b \mid c) = p(a \mid c) * p(b \mid c). \quad (2.14)$$

Найпростішим прикладом умовно незалежних величин є: зріст людини (величина *a*), довжина його волосся (величина *b*) і його стать (величина *c*). Добре відомо, що зріст зворотно корелює з довжиною волосся, однак, після додавання в ймовірнісну модель фактора статі людини, зріст і довжина волосся стають незалежними величинами.

Існує два основних правила роботи з випадковими величинами. Розглянемо спільну щільність *n* випадкових величин $p(a_1,..., a_n)$. Правило добутку говорить про те, що будь-яку багатовимірну щільність можна представити у вигляді добутку одновимірних умовних щільностей:

$$p(a_1,..., a_n) = p(a_n \mid a_1,..., a_{n-1}) \times p(a_{n-1} \mid a_1,..., a_{n-2}). . . p(a_2 \mid a_1)\, p(a_1). \quad (2.15)$$

Аналогічні рівняння можна виписати для довільного перевпорядкованих змінних. Правило підсумовування дозволяє отримувати безумовні розподіли меншої розмірності шляхом виключення частини змінних:

$$p(a_1,\cdots,a_k)=\int p(a_1,\cdots,a_n)da_{k+1}\cdots da_n = \int p(a_1,\cdots,a_n). \quad (2.16)$$

Всі операції, які здійснюються з ймовірнісними моделями при використанні байєсівського формалізму, спираються на застосування цих двох правил. Байєсівські мережі дозволяють моделювати причинно-наслідкові зв'язки між величинами. Для цього на множині

змінних Y = ($\bar{x}$, $\bar{t}$, $\bar{w}$) ймовірнісної моделі задається орієнтований граф, в якому ребра відображають відносини причинності. За змістом побудови в такому графі заборонені орієнтовані цикли. Граф причинності задає систему факторизації спільного розподілу

$$p(Y) = \prod_{i=1}^{n} p(y_i \mid pa_i), \tag{2.17}$$

де pa_i - множина батьків i-ї вершини.

Слід зазначити, що розмір кожного фактора (а саме розмірність чинників служить мірою складності розподілу як на етапі його завдання, так і на етапі роботи з ним) визначається числом батьків вершини. Така система факторизації значно спрощує розрахунки довільних умовних і маргінальних розподілів, а саме до цього, зводяться завдання настройки і виведення в байєсівських моделях. Часто виникає необхідність моделювати системи випадкових величин між якими є залежності, але некоректно говорити про причинно-наслідкові зв'язки. Прикладом таких величин можуть бути мітки сусідніх пікселів в завданні сегментації зображень або профілі друзів у соціальній мережі. Для моделювання таких залежностей на множині величин задається неорієнтований граф, який визначає факторизацію спільного розподілу таким чином:

$$p(Y) =, \frac{1}{z}\prod_{c\in C}\psi_c(Y_c) = \frac{\prod_{c\in C}\psi_c(Y_c)}{\sum_Y \prod_{c\in C}\psi_c(Y_c)}. \tag{2.18}$$

де $\psi_c()$ - невід'ємні функції.

Слід зазначити, що на відміну від байєсівських мереж, фактори не мають ймовірнісного сенсу, тому необхідна додаткове нормування добутку чинників. Якщо величини *y'* та *y''* ніколи не входять в один фактор (тобто не з'єднані ребром у графі), то вони є незалежними за умови, що всі інші величини відомі. Таким чином, ребра графа визначають відносини умовної незалежності. Однією з переваг систем факторизації, які задаються графічними моделями, нарівні з зручністю представлення багатовимірних розподілів, є можливість паралельної і розподіленої обробки інформації при підрахунку умовних розподілів, наприклад, за допомогою інтерфейсу передачі повідомлень (message-passing interface). Апарат графічних моделей використовується для точного або наближеного вирішення наступних основних завдань:

- навчання з учителем $arg\ max_{\bar{w}} p$ ($\bar{\mathrm{w}} \mid \overline{\mathrm{X^{tr}}}, \overline{\mathrm{T^{tr}}}$);
- навчання без учителя формула 2.19;
- підрахунок нормованої константи Z;

– підрахунок найбільш ймовірної конфігурації прихованих змінних формула 2.20;

– підрахунок маргінального розподілу фіксованої змінної $p(t_i \mid X, \bar{w})$.

$$\arg\max_{\bar{w}} p(\bar{w} \mid \overline{X^{tr}}) = \arg\max_{\bar{w}} \sum_T p(\bar{w}T \mid \overline{X^{tr}}). \tag{2.19}$$

$$\arg\max_T p(T \mid X, \bar{w}). \tag{2.20}$$

Слід зазначити, що всі ці завдання зводяться до підрахунку тих чи інших умовних розподілів на невідомі змінні за умови спостережуваних змінних і, можливо, маргіналізації за нерелевантними змінними. Можна помітити, що ті ж завдання виникають в класичному машинному навчанні. Перенесення класичних результатів на більш складні графічні моделі є одним з найважливіших напрямків робіт в сучасному машинному навчанні.

В ході огляду сучасних тенденцій використання згорткових нейронних мереж виділено такі перспективні напрямки фундаментальних і прикладних досліджень в даній області:

– теоретичні дослідження в області моделей штучного інтелекту в поєднанні з аналізом автоматично побудованих моделей;

– практичні дослідження мультизадачних, генеративних (породжуючих) моделей;

– більш широке поширення автоматизованих засобів машинного навчання;

– розвиток і уніфікація інструментальних засобів, в тому числі хмарних засобів і сервісів інтелектуального аналізу даних;

– розробка нових інтелектуальних продуктів користувальницького рівня, заснованих на досягненнях методології машинного навчання.

2.4 Оцінка переваг та недоліків використання згорткової нейронної мережі для знаходження місця розташування автомобільних номерів

Недоліками згорткових нейронних мереж стали [5-8]:

– критична залежність якості настройки ваг мережі від вибору початкового наближення;

– велика схильність до перенавчання укупі зі слабкими можливостями контролю узагальнюючої здатності мережі;

– велика кількість локальних мінімумів функціонала якості, більшість з яких є несприйнятими.

З іншого боку, незаперечною сильною стороною нейронних мереж стало відкриття методу зворотного поширення помилки, який дозволив відстежувати вплив внутрішніх шарів мережі на якість прогнозу прихованих змінних об'єктів навчальної вибірки.

Загальними перевагами згорткових нейронних мереж є:

– нелінійність моделі, що дає можливість апроксимувати нелінійні функції;

– локальність сприйняття, що полягає в тому, що кожен нейрон отримує не весь вхідний вектор. Це дозволяє сегментувати дані і працювати з більш складними ситуаціями;

– каскад шарів. У поєднанні з пунктом 2 це дає здатність сприймати більш абстрактні ознаки;

– можливість якісно працювати з комбінаціями ознак;

– нейронна мережа дає кілька механізмів для контролю перенавчання;

– нейронна мережа здатна донавчатися при несуперечливості нових образів.

Останнім часом став активно розвиватися напрямок глибинного навчання [5]. В його основі лежать згорткові нейронні мережі, які зазнали значних змін. Глибинне навчання будує не дискримінативність, а породжує моделі, в яких моделюється загальний розподіл $p(\bar{x}, \bar{t}, \bar{w})$, на відміну від дискримінативних моделей, що дозволяє, наприклад, генерувати нові об'єкти.

У найбільш поширеній постановці всі змінні об'єктів передбачаються бінарними. Це полегшує моделювання залежностей між змінними об'єкта.

Кожен шар мережі спочатку навчається незалежно, проходячи процедуру преднавчання. Це дозволяє знайти хороше початкове наближення для подальшого запуску алгоритму зворотного поширення помилки. Кожен шар, в залежності від обраної моделі, являє собою обмежену машину Больцмана або згорткову нейронну мережу.

Для навчання використовуються сотні тисяч і мільйони об'єктів. Такі гігантські вибірки дозволяють налаштовувати мережі з десятками тисяч параметрів, без ризику перенавчання. Навчені таким чином мережі, не просто дозволяють моделювати складні об'єкти (наприклад, тексти або зображення), але і генерують в процесі навчання інформативні ознакові описи, які можуть бути використані іншими, більш простими алгоритмами машинного навчання як спостережувані змінні об'єкта.

Методологія глибинного навчання дозволила домогтися нових результатів при навчанні на великих і надвеликих обсягах даних. В даний час вона є одним з найбільш перспективних шляхів розвитку машинного навчання.

Таким чином, можна зробити висновок про те, що в даний час спостерігається прогрес у розвитку методик використання нейронних мереж шляхом побудови ефективних навчальних

моделей аналізу даних, які можна застосувати до багатьох практичних завдань інтелектуального аналізу даних. Постійний розвиток методів машинного навчання викликаний зростанням можливостей сучасних обчислювальних систем, ще більш стрімким зростанням обсягів даних, доступних для аналізу, а також постійним розширенням області застосування методів машинного навчання на все більш широкий клас задач обробки даних.

Однією з важливих характеристик алгоритмів машинного навчання є узагальнююча здатність. Однак з нею пов'язані ще два поняття: недонавчання і перенавчання.

Недонавчання виникає при навчанні по прецедентах і характеризується тим, що алгоритм не дає задовільно малої середньої помилки на навчальній множині. Як правило, це явище виникає внаслідок використання недостатньо складних моделей.

Протилежне цьому явищу - перенавчання. Його суть полягає в тому, що ймовірність помилки натренованого алгоритму на об'єктах тренувальної вибірки виявляється істотно менше, ніж на об'єктах тестової. Найчастіше перенавчання виникає через використання занадто складних моделей.

Найчастіше при побудові алгоритмів навчання використовується метод мінімізації емпіричного ризику (середньої помилки алгоритму на навчальній вибірці). Його суть полягає в тому, щоб для поточної моделі підібрати алгоритм, що мінімізує значення середньої помилки на даній навчальній вибірці.

З перенавчанням методу мінімізації емпіричного ризику пов'язані три твердження, що пояснюють його причину:

1) Мінімізація емпіричного ризику не гарантує малу ймовірність помилки на тестових даних. Можна легко побудувати алгоритм, який мінімізує емпіричний ризик до нуля, однак не буде здатний до навчання. Суть полягає в тому, що цей алгоритм запам’ятовує навчальну вибірку, потім порівнює зразок з пропонованим. При збігу об'єкта та зразка навчальної вибірки алгоритм видає вірну відповідь, інакше - довільну. Тобто емпіричний ризик дорівнює нулю, однак узагальнюючої здатності у алгоритму немає.

2) Перенавчання з'являється внаслідок мінімізації емпіричного ризику. Нехай задано кінцеву множину з D алгоритмів, які допускають помилки незалежно і з однаковою ймовірністю. Число помилок будь-якого з цих алгоритмів на заданій навчальній вибірці підпорядковується одному того ж біноміальному розподілу. Мінімум емпіричного ризику - це випадкова величина, що дорівнює мінімуму з D незалежних однаково розподілених біноміальних випадкових величин, очікуване значення якої зменшується з ростом D. Відповідно, з ростом D збільшується перенавчання - різниця ймовірності помилки і частоти помилок на навчанні. Однак в реальній ситуації алгоритми мають різні ймовірності помилок, не є незалежними, а множина алгоритмів, з якого вибирається найкращий, може бути

нескінченною. З цих причин виведення кількісних оцінок перенавчання є складним завданням, яким займається теорія обчислювального навчання. До сих пір залишається відкритою проблема сильної завищеності верхніх оцінок імовірності перенавчання.

3) Перенавчання з'являється в зв'язку з надмірною складністю моделі. Завжди можна знайти оптимальне значення складності моделі, при перенавчання буде мінімальним.

Одним із способів вимірювання ймовірності перенавчання є емпірично й метод Монте-Карло, або метод ковзаючого контролю. У літературі також зустрічаються назви крос-перевірка, або крос-валідація. Суть його в наступному: проводиться певна кількість розбиття вихідної вибірки на навчальну і контрольну. Для кожного розбиття відбувається навчання алгоритму на навчальній підвибірки і оцінка середньої помилки на контрольній. Потім обчислюється оцінка змінного середнього, як середня по всім розбиттям величина обчисленої помилки. Для незалежної вибірки цей показник дає незміщену оцінку ймовірності помилки. Метод змінного контролю є стандартним способом порівняння і оцінювання алгоритмів класифікації, регресії, прогнозування.

При використанні крос-валідації можна зробити наступні висновки:

– якщо помилка велика на більшості ділянок, то швидше за все проблема в моделі;

– якщо дані навчальної вибірки характеризуються сильно зміщеними математичним очікуванням і дисперсією, то рівень узагальнення буде низьким, що може бути пов'язано з перенавчанням;

– якщо знайдені сильні відхилення на певних підвибірках, то, ймовірно, проблема в цих ділянках даних або модель недообучается;

– при малому обсязі навчальної вибірки кросвалідація може стати способом боротьби з перенавчанням.

Якщо ж говорити про прямі способи боротьби з перенавчанням, то можна виділити наступні:

– спрощення моделі;

– підготовка більшого числа навчальних даних (можливо, за допомогою генерації);

– регуляризація.

Зупинимося докладніше на останньому. Регуляризація є додавання деякої додаткової інформації умові мінімізації помилки. Виконується це, щоб вирішити некоректно поставлене завдання або запобігти перенавчання. Найчастіше додана інформація набуває вигляду штрафу за складність моделі. Наприклад, введені обмеження по нормі векторного простору або гладкості результуючої функції. Або ж, з байєсівської точки зору, додані апріорні розподілу на параметри моделі.

Іншими словами, перенавчання в більшості випадків виявляється в тому, що в які утворюються многочленах занадто великі коефіцієнти. Відповідно і боротися з цим можна досить природним способом: потрібно просто додати в цільову функцію штраф, який би карав модель за надто великі коефіцієнти.

З приводу застосування тієї чи іншої регуляризації: регуляризацію можна застосовувати з будь-яким методом МО-класифікації, який заснований на математичному рівнянні. Приклади включають лінійну, логістичну регресію, нейронні мережі. Оскільки це зменшує величину вагових значень в моделі, регуляризацію іноді називають скороченням ваг. Основна перевага застосування регуляризації в тому, що воно часто призводить до створення більш точної моделі. Головний недолік полягає у введенні додаткового параметра, значення якого потрібно визначити, - вагового значення регуляризації. У разі логістичної регресії це не дуже серйозно, так як в цьому алгоритмі зазвичай використовується лише параметр швидкості навчання, але при використанні більш складного методу класифікації, зокрема нейронних мереж, додавання ще одного так званого гіперпараметра може зажадати маси додаткової роботи для підбору комбінації значень двох параметрів.

L1- і L2-регуляризація - процеси схожі. Дослідниками сформульовані теоретичні правила щодо того, яка регуляризація краще для певних завдань, але на практиці доведеться поекспериментувати, щоб визначити, який тип регуляризації краще в кожному випадку і чи варто взагалі використовувати будь-яку регуляризацію.

Застосування L1-регуляризації іноді може давати корисний побічний ефект, що викликає прагнення одного або більше вагових значень до 0.0, а це означає, що відповідний ознака не надає значного впливу на результуючий, тобто включення його в модель не потрібно. Це одна з форм того, що називають «селекцією ознак». На відміну від L1, L2-регуляризація обмежує вагові значення моделі, але зазвичай не призводить до повного обнулення цих значень. Тому може здатися, що L1-регуляризація краще L2-регуляризації. Однак недолік застосування L1-регуляризації в тому, що цей метод непросто використовувати з деякими алгоритмами машинного навчання. Наприклад, тими, в яких використовуються чисельні методи для обчислення так званого градієнта. L2-регуляризацію можна використовувати з будь-яким типом алгоритму навчання.

Таким чином, можна зробити висновок, що L1-регуляризація іноді дає корисний побічний ефект видалення непотрібних ознак, привласнюючи пов'язаним з ними ваг значення 0.0, але L1-регуляризація стабільно працює не з усіма формами навчання. L2-регуляризація працює з усіма формами навчання, але не забезпечує неявній селекції функцій. На практиці ж слід використовувати метод проб і помилок, щоб визначити, яка форма регуляризації (якщо

вона взагалі потрібна) краще для конкретного завдання. Розглянуті методи боротьби з перенавчанням використовуються для регресії або нейронних мереж.

Гарантувати збереження і доступність навчальних даних можна комбінуванням різних стратегій [12]:

1) Стандартизація – одна з основних передумов доступності електронної інформації протягом тривалого часу; стандартизації підлягають файлові формати, метадані інформаційного об'єкта, типи і властивості носіїв інформації тощо. Можливість тривалого зберігання електронного документа повинна враховуватися вже на стадії створення файлу, зокрема, рекомендується використовувати так звані стабільні формати, які характеризуються значним поширенням, відкритою специфікацією (нормуванням) так як їх створювали для тривалого зберігання даних (наприклад, XML, TIFF, PDF, JPEG , PNG).

2) Міграція – це збереження доступності інформації протягом необмеженого часу шляхом регулярного перенесення даних з застарілого в нове системне оточення шляхом використання технічних компонентів, призначених для зберігання даних і управління ними. Оригінальність і справжність інформації, авторські права і заборона копіювання можуть бути порушені в процесі міграції; так само сумнівна можливість повноцінного пошуку в перенесених даних, якщо під час міграції дані, контекстна інформація або метадані змінилися. Тому ще на стадії організації документу потрібно передбачити можливість своєчасної міграції, що буде відбуватися без порушення прав, без змін і втрат інформації; для цього, наприклад, може знадобитися створення оригінального програмного забезпечення для міграції форматів даних.

3) Емуляція передбачає створення нових комп'ютерних систем і програмного забезпечення, здатних симулювати власні старі версії і таким чином працювати з даними, що генеруються в старих комп'ютерах і операційних системах.

4) Інкапсуляція – процес, який вважається особливо важливим як підготовчий до емуляції; сутність процесу полягає в зберіганні разом з інформаційними об'єктами програмного забезпечення, за допомогою якого вони можуть бути відтворені, а також відповідних метаданих. Недоліками стратегії є занадто великий розмір об'єкта («капсули»), а також відсутність гарантії, що розроблене ПЗ може бути використано в майбутньому.

5) Конвертація в процесі використання – стратегія, яка передбачає постійну наявність в системі програм - конверторів та оглядачів, здатних при відкритті файлу в застарілому форматі перевести його в більш актуальний. Відмінність від емуляції полягає в тому, що в умовах конвертації інформаційний об'єкт пристосовується до наявного системного оточення, а при емуляції навпаки – нове оточення симулює те, в якому був створений об'єкт. Недоліками такої конвертації є проблематичність збереження структури документів.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Використані технології

3.1.1 Python

У чому полягає робота фахівця по машинному навчанню? Він повинен збирати, систематизувати і аналізувати дані, а потім на основі отриманої інформації створювати алгоритми для штучного інтелекту.

Python найкраще підходить для виконання таких завдань, тому що він досить зрозумілий в порівнянні з іншими мовами. Більш того, у нього відмінна продуктивність при обробці даних.

Одна з основних причин, чому Python використовується для машинного навчання полягає в тому, що у нього є безліч фреймворків, які спрощують процес написання коду і скорочують час на розробку. У наукових розрахунках використовується Numpy, в просунутих обчисленнях - SciPy, в добуванні і аналізі даних - SciKit-Learn. Ці бібліотеки працюють в таких фреймворків, як TensorFlow, котрий використовується у цій роботі, CNTK і Apache Spark. Існує фреймворк для Python, розроблений спеціально для машинного навчання - це PyTorch.

Python - сама високорівнева і зрозуміла мова, з якою зручно працювати. Завдяки її лаконічності і зручності читання віна добре підходить для навчання ПЗ.

Простий синтаксис мови Python допомагає розробнику тестувати складні алгоритми з мінімальною витратою часу на їх реалізацію[27].

3.1.2 Keras

Keras - це API, призначений для людей, а не для машин. Він дотримується кращих практик зменшення когнітивного навантаження: він пропонує послідовні та прості API, він мінімізує кількість дій користувачів, необхідних для випадків загального використання, та забезпечує чіткий та дієвий зворотний зв'язок при помилці користувача.

Це робить Keras простим у навчанні та легким у використанні. Як користувач Keras, ви більш продуктивні, що дозволяє вам швидше спробувати більше ідей, ніж ваша конкуренція - що, в свою чергу, допомагає вигравати змагання з машинного навчання.

Така зручність використання не є ціною зниженої гнучкості: оскільки Керас інтегрується з мовами глибокого вивчення нижчого рівня (зокрема TensorFlow), це дозволяє реалізувати все,

що ви могли побудувати на базовій мові. Зокрема, як tf.keras, API Keras легко інтегрується з вашими робочими процесами TensorFlow[28].

3.1.3 Tensorflow

TensorFlow - це потужна бібліотека машинного навчання, орієнтована на потік даних, створена командою Brain Google і створена з відкритим кодом у 2015 році. Вона створена для зручного використання та широкого застосування як для проблем, орієнтованих на числові, так і для нейромережевих мереж, а також інших областей.

По суті, TensorFlow - це інструментарій низького рівня для занять складною математикою, і він орієнтований на дослідників, які знають, що вони роблять для створення експериментальних навчальних архітектур, щоб грати з ними та перетворювати їх на працюєче ПЗ.

Це можна розглядати як систему програмування, в якій ви представляєте обчислення як графіки. Вузли на графіку представляють математичні операції, а ребра представляють багатовимірні масиви даних (тензори), що передаються між ними[29].

3.2 Створення та підготовка набору даних

Для початку навчання необхідно було зібрати тренувальний набір зображень. На просторах інтернету був зібраний два набор зображень котрій складаються з фотографій автомобілів з встановлених на ними українськими номерами останнього формату, від 2015 року. Розмір набору склав більш ніж 600 зображень. Приклад зображень наведено у ДОДАТКУ А.

Рисунок 3.1 – Приклад фото для навчання з разметкою

Після цього необхідно всі фотографії обробити і розмітити. В ДОДАТКУ Б наведено фрагмент коду модуля data.py, котрий формує базу зображень для подальшої обробки системою. В ході навчання було помічено, що оригінальні зображення, з яких складається набір даних, при навчанні вимагають багато відеопам'яті і через кілька годин навчання програма могла викинути помилку про нестачу пам'яті. Для вирішення можна було змінити конфігурацію, що могло зменшити якість і швидкість навчання, було прийнято рішення зменшити розмір зображень навчальної вибірки. Для цього була написана програма, яка зменшує розмір і змінює конфігураційний файл з розміткою під нові розміри зображень. Всі зображення були зменшені так, щоб висота була 500 пікселів. Такі дії допомогли зменшити час навчання і уникнути проблем з нестачею пам'яті при тих же конфігураціях.

3.3 Навчання нейронної мережі

Для того щоб нейронна мережа була спроможна знайти автомобільний номер на зображенні спочатку необхідно навчити нейронну мережу. Для навчання мережі необхідно

підготувати великий набір різних зображень на яких будуть присутні автомобільні номери які необхідно буде знайти. Після збору необхідних зображень необхідно зробити файл який буде містити координати номерів на зображеннях. Зазначені дії реалізовано у модулі models.py, лістинг якого наведено в ДОДАТКУ В.

Для навчання була використана бібліотека TensorFlow (ДОДАТОК Г). Навчання за допомогою цієї бібліотеці потребує потужної відео карти що дуже підвищує швидкість навчання. Для того щоб нейронна мережа почала працювати була підібрана мінімальна кількість даних для навчання - для цієї роботи, шляхом збільшення навчальних даних було виявлено, що це 200 зображень. Але після навчання на такої кількості зображень було виявлено, що якість роботи програми не є задовільною, тому послідовно кількість навчальних зображень будо збільшено до більш ніж 600 зображень. Але при збільшенні кількості зображень дуже виросла кількість необхідного часу для здійснення навчання. Спочатку навчання здійснювалося за допомогою відеокарти вбудованої у процесор, але вона має дуже малу продуктивність та малу кількість відеопам’яті та навчання проходило дуже повільно та з низькими параметрами налаштовування якості. Отже була куплена відеокарта MSI GTX 980ti з 6GB відеопам’яті, з якою навчання проходило набагато швидше, але все одне для 600 фотографій навчання займало майже добу.

3.4 Тестування системи

Для тестування програми були використані звичайні зображення та зображення до яких було додано нормального(Гаусовського) шуму. Нормальний шум задається функцією Гауса. Для того щоб додати шум також була створена програма (ДОДАТОК Д) котра додавала нормальний шум до тестових зображень та зберігає їх в окрему папку. Ця програма використовує бібліотеку OpenCV в котрій є вже реалізована функція котра створює зображення з нормальним шумом, а потім цій шум просто додається до оригінального зображення.

Параметри котрі були враховані для порівняння роботи це кількість вірних знаходжень, кількість вірних знаходжень але не повністю вірним розташуванням, кількість не знайдених номерів, кількість помилкових знаходжень.

Рисунок 3.2 – приклад вихідного зображення програми

а) б)

в) г)

Рисунок 3.3 – Типи тестових зображень: а – нормальне зображення, б – нормальній шум сигма = 20, в – нормальній шум сигма = 50, г – нормальній шум сигма = 100

Таблиця 3.1 – Порівняння працездатності при додаванні шуму

Тип	Вірних	Майже вірних	Не знайдених	Помилкових
Оригінал	182	20	6	2
	86,7%	9,5%	2,8%	0,95%
З шумом 20	178	20	12	0
	84,8%	9,5%	5,7%	0%
З шумом 50	177	8	25	0
	84,2%	3,8%	11,9%	0%
З шумом 100	122	3	85	0
	58%	0,01%	0,4%	0%

4 ОХОРОНА ПРАЦІ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

4.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання ДСанПіН 3.3.2.007-98 [31], які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне елект-рообладнання, зокрема персональні комп’ютери (ПК) та периферійні пристрої.

Основними робочими характеристиками персонального комп'ютера є наступні:

– – робоча напруга U = +220В ± 5%;

– – робочий струм I = 2А;

– – споживана потужність P = 350 Вт.

Робочі місця мають відповідати вимогам Державних санітарних правил і норм роботи з ві-зуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [31].

За умов роботи з ПК виникають наступні небезпечні та шкідливі чинники: несприятливі мі-крокліматичні умови, освітлення, електромагнітні випромінювання, забруднення повітря шкідливими речовинами (джерелом, яких можуть бути: принтер, сканер та інші джерела виділення багатьох хімічних речовин - напр., озону, оксидів азоту та аерозолів високодисперсних частинок тонера), шум, вібрація, електричний струм, електростатичне поле, напруженість трудового процесу та інше.

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.1).

Таблиця 4.1 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	Джерела факторів (види робіт)	Кількісна оцінка	Нормативні документи
1	2	3	4
фізичні			
- підвищена температура поверхонь обладнання	експлуатація ЕОМ, принтерів, сканерів чи/або серверного обладнання для роботи	2	[32]
- підвищений рівень шуму на робочому місці	-//-	2	[33]
- підвищена або знижена рухливість повітря	-//-	1	[32]
- підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	4	[35]
- підвищена напруженість електричного поля	-//-	2	[35]
- недостатність природного світла	порушення умов праці (вимог до приміщень)	2	[36]
- недостатнє освітлення робочої зони	порушення гігієнічних параметрів виробничого середовища	3	[36]

Продовження таблиці 4.1

1	2	3	4
психофізіологічні:			
- нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових)	- пошук інформації для постановки теми; - пошук та аналіз аналогів і літератури; - пошук наявних технологій, моделювання та аналіз алгоритмів; - виконання роботи за темою диплома, тестування; - оформлення роботи	4	[31] [40]
- фізичні (статичне – сидіння)	порушення умов праці (організації місця праці- сидіння користувача,) та організації робочого часу - безпервна робота)	2	[31] [40]

4.2 Гігієнічні вимоги до параметрів виробничого середовища

4.2.1 Мікроклімат

Оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають ДСН 3.3.6.042-99 [32] і наведені в табл. 4.2:

Таблиця 4.2 – Норми мікроклімату робочої зони об’єкту

Період року	Категорія робіт	Температура C^0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	легка-1 а	22 - 24	40 – 60	0,1
Тепла	легка-1 а	23 - 25	40 – 60	0,1

Дане приміщення обладнане системами опалення, кондиціонування повітря або припливно-витяжною вентиляцією. У приміщенні на робочому місці забезпечуються

оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до ДСН 3.3.6.042-99 [2]. Для забезпечення оптимальних параметрів мікроклімату в приміщенні проводяться перерви в роботі співробітників, з метою його провітрювання. Існують спеціальні системи кондиціонування, які забезпечують підтримання в приміщенні балансу оптимальних параметрів мікроклімату. Контроль параметрів мікроклімату в холодний і теплий період року здійснюється не менше 3-х разів на зміну (на початку, середині, в кінці).

4.2.2 Освітлення

У приміщенні, де розташовані ЕОМ передбачається природне бічне освітлення, рівень якого відповідає ДБН В. 2.5-28:2018 [36]. Джерелом природного освітлення є сонячне світло. Регулярно повинен проводитися контроль освітленості, який підтверджує, що рівень освітленості задовольняє ДБН і для даного приміщення в світлий час доби достатньо природного освітлення.

Розрахунок освітлення.

Для будівель виробництв світловий коефіцієнт приймається в межах 1/6 - 1/10:

$$\sqrt{a^2 + b^2} \cdot S_b = (1/8 \div 1/10) \cdot S_n \quad (4.1)$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2.

$$S_n = a \cdot b = 5 \cdot 5 = 25 \text{ м}^2$$

$$S_{вік} = 1/8 \cdot 25 = 3{,}125 \text{ м}^2$$

Приймаємо 2 вікна площею $S = 1{,}6 \text{ м}^2$ кожне.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 5 м, світильниками ЛПО2П, оснащеними лампами типа ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників *n* виробляється по формулі (4.2):

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M} \tag{4.2}$$

де *E* – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2; S = 25 м^2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п. – 0,575

M – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$\mathrm{n} = \frac{300 \cdot 25 \cdot 1{,}1 \cdot 1{,}5}{5400 \cdot 0{,}575 \cdot 2} \approx 2.$$

Приймаємо освітлювальну установку, яка складається з 2-х світильників, які складаються з двох люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

4.2.3 Шум та вібрація, електромагнітне випромінювання

Рівень шуму, що супроводжує роботу користувачів персональних комп’ютерів (зумовлений як роботою системних блоків, клавіатури, так і друкуванням на принтерах, а також зовнішніми чинниками), коливається у межах 50–65 дБА [33]. Шум такої інтенсивності на тлі високого ступеня напруженості праці негативно впливає на функціональний стан користувачів. Тому на практиці рекомендують знижувати фактичний рівень шуму у приміщеннях, де створюють комп’ютерні програми, виконують теоретичні та творчі роботи, проводять навчання до 40 дБА, а в приміщеннях, де виконують роботу, що потребує зосередженості, — до 55 дБА.

У залах опрацювання інформації та комп'ютерного набору рівні шуму не повинні перевищувати 65 дБА.

Шум часто є причиною зниження рівня працездатності, підвищення рівня загальної та професійної захворюваності, частоти виробничих травм. Шум є загальнобіологічним подразником, який негативно впливає на всі органи і системи організму. У разі тривалого систематичного впливу шуму може виникнути патологія з переважним ураженням слуху, центральної нервової і серцево- судинної систем.

Для зниження шуму на шляху його поширення передбачається розміщення в приміщенні штучних поглиначів. Для зниження рівня шуму стелю або стіни вище 1.5 - 1.7 метра від підлоги повинні облицьовуватися звукопоглинальним матеріалом з максимальним коефіцієнтом звукопоглинання в області частот 63-8000 Гц. Додатковим звукопоглинанням в КВТ можуть бути фіранки, підвішені в складку на відстані 15-20 см. Від огорожі, виконані з щільної, важкої тканини. У приміщенні з ЕОМ коректований рівень звукової потужності не перевищує 45 дБА. Оскільки рівень шуму не перевищує гранично допустимих величин, які встановлені санітарними нормами, заходи для зниження шуму не проводяться.

Віброізоляцію можливо здійснювати за допомогою спеціальної прокладки під системний блок, який послаблює передачу вібрацій робочого столу. Вібрація на робочому місці в приміщенні, що розглядається, відповідає нормам [33]. Допустимий рівень вібрацій на робочому місці: для 1 ступеня шкідливості до 3 дБ; для 2-3 - 1-6 дБ; для 3 - більше 6 дБ.

Для захисту від електромагнітного випромінювання передбачаються наступні заходи:

1) застосування нових плазмових моніторів, LG W2271TC,

2) віддалення робочого місця не менше, ніж на 0,4-0,5 м, оскільки напруженість електричного поля зменшується при віддаленні від джерела поля,

3) встановлення раціональних режимів роботи персоналу (обмеження часу перебування),

4) раціональне розміщення в робочому приміщенні устаткування, що випромінює електромагнітну енергію.

4.2.4 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції (вентиляційні шахти) і установки в віконному отворі автономного кондиціонера БК-2000. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП (30 $м^3$ на годину на одного працюючого).

Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.3 Заходи з організації виробничого середовища та попередження виникнення надзвичайних situацій

Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Загальний опір захисного заземлення визначається за формулою:

$$R_{ззп} = \frac{R_з \cdot R_п}{R_п \cdot n \cdot \eta_з + R_з \cdot \eta_п}, \quad (4.3)$$

де Rз - опір заземлення, якими когут бать труби, опори, кути і т.п., Ом;

Rш - опір опори, яке з'єднує заземлювачі, Ом;

п - кількість заземлювачів;

ηз - коефіцієнт екранування заземлювача; приймається в межах 0,2 ÷ 0,9; ηз = 0,7

ηш - коефіцієнт екранування сполучної стійки; приймається в межах 0,1 ÷ 0,7; ηш = 0,5;

Опір заземлення визначається за формулою:

$$R_з = \frac{\rho}{2\pi \cdot l} \cdot \left(\ln \frac{2 \cdot l}{d} + \frac{1}{2} \ln \frac{4 \cdot t + l}{4 \cdot t - l} \right), \quad (4.4)$$

де ρ - питомий опір ґрунту, залежить від типу грунту, Ом·м;

для піску - 400 ÷ 700 Ом·м; приймаємо ρ = 400 Ом·м;

l - довжина заземлювача, м; для труб - 2-3 м; l = 3 м;

d - діаметр заземлювача, м; для труб - 0,03-0,05 м; d = 0,05 м;

t - відстань від середини забитого в ґрунт заземлювача до рівня землі, м;

t = 2м.

$$R_з = \frac{400}{2 \cdot 3{,}14 \cdot 3} \left(\ln \frac{2 \cdot 3}{0{,}05} + \frac{1}{2} \ln \frac{4 \cdot 2 + 3}{4 \cdot 2 - 3} \right) = 110 \text{, Ом}$$

Опір смуги, що з'єднує заземлювачі, визначається за формулою:

$$R_{ш} = \frac{\rho}{2\pi \cdot L} \cdot \ln \frac{2 \cdot L^2}{b \cdot t^1}, \quad (4.5)$$

де L - довжина смуги, що з'єднує заземлювачі (м) і приблизно дорівнює периметру будівлі: $П_{буд.} = 42 \cdot 2 + 38 \cdot 2 = 160$ м; L = 160 м;

b - ширина смуги, м; b = 0,03 м;

t_1 - глибина заземлення від рівня землі, м; $t_1 = 0,5$ м.

$$R_n = \frac{400}{2 \cdot 3,14 \cdot 160} \cdot \ln \frac{2 \cdot 160^2}{0,03 \cdot 0,5} = 5,99 \text{ , Ом}$$

Кількість заземлювачів захисного заземлення визначається за формулою:

$$n = \frac{2 \cdot R_з}{4 \cdot \eta_з}, \quad (4.6)$$

де 4 - допустимий загальний опір, Ом;

2 - коефіцієнт сезонності.

Визначаємо загальний опір захисного заземлення:

$$R_{ззп} = \frac{110 \cdot 5,99}{5,99 \cdot 79 \cdot 0,7 + 110 \cdot 0,5} = 1,7 \text{ Ом}$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{ззп} < 4$ Ом.

3) При виникненню пожеж при роботі на ПЕОМ від таких можливими джерел запалювання як:

– іскри і дуги коротких замикань;

– перегрів провідників, резисторів та інших радіодеталей ПЕОМ, від тривалої перевантаження та наявність перехідного опору;

– іскри при розмиканні і розмиканні ланцюгів;

– розряди статичної електрики;

– необережному поводженню з вогнем, а також вибухи газо-повітряних і паро-повітряних сумішей.

Важливу увагу слід звернути на пожежну безпеку підприємства в цілому і окремих його приміщень. В приміщеннях не повинно накопичуватися сміття, непотрібний папір, мотлох та ін. речі, які не використовуються у виробничому процесі. Наявний вільний аварійний вихід за межі приміщення в разі пожежі, бути передбачені вогнегасники. Вони повинні бути в робочому стані і перевірятися згідно з нормами. У приміщеннях повинна бути пожежна сигналізація, вогнегасник. У разі виникнення пожежі необхідно повідомити в найближчу пожежну частину, убезпечити інших працівників і по можливості прийняти кроки по запобіганню можливих наслідків та усуненню пожежі.

4.3 Екологія

Діяльність за темою магістерської роботи в процесі її виконання впливає на навколишнє природнє середовище і регламентується нормами діючого законодавства: Законом України «Про охорону навколишнього природного середовища» [42], Законом України «Про забезпечення санітарного та епідемічного благополуччя населення» [43], Законом України «Про відходи» [44].

В процесі діяльності користувача виникають процеси поводження з відходами ІТ галузі. Нижче надано перелік відходів, що утворюються в процесі роботи:

– відпрацьовані люмінесцентні лампи – I клас небезпеки;

– змінні носії інформації – IV клас небезпеки;

– макулатура – IV клас небезпеки;

– побутові відходи – IV клас небезпеки.

ВИСНОВКИ

В результаті виконання роботи виконано дослідження та програмна розробка ШНМ для знаходження місця розташування автомобільних номерів на зображенні. Вирішено наступні завдання:

– створення програмного модуля завантаження та обробки зображення з відокремленням робочою області зображення;

– вибір типу ШНМ;

– структурна ідентифікація топології ШНМ;

– навчання ШНМ;

– оцінка якості функціонування системи розпізнавання місця розташування автомобільних номерів з проведенням експериментів для тестової вибірки різнорідних зображень.

Таким чином, нейронна мережа являє собою сукупність нейронних елементів та зв'язків між ними. Основний елемент нейронної мережі - це формальний нейрон, що здійснює операцію нелінійного перетворення суми здобутків вхідних сигналів на вагові коефіцієнти.

Персептрон є однією з найпопулярніших реалізацій нейронних мереж. Причиною його популярності є відносна простота реалізації на тлі універсальності і широкого кола завдань, які можуть вирішувати персептрони.

Для вирішення більш складних завдань використовують багатошарові нейронні мережі [1-2], котрі й були використані у цій роботі. Але для навчання багатошарових нейронних мереж потрібні більш складні алгоритми навчання нейронних мереж, що було однією з проблем при навчанні ШНМ в рамках цієї роботи. ШНМ навчається за допомогою деякого процесу, що модифікує її ваги. Якщо навчання успішне, то передача мережі множини вхідних сигналів призводить до появи бажаної множини вихідних сигналів.

Проведений аналіз нейронних мереж і алгоритмів їх навчання дозволив виявити ряд недоліків і проблем, що виникають:

– невизначеність у виборі числа шарів і кількості нейронних елементів в шарі;

– повільна збіжність градієнтного методу з постійним кроком навчання;

– складність вибору відповідної швидкості навчання. Так як маленька швидкість навчання призводить до скочування нейронної мережі в локальний мінімум, а велика швидкість навчання може привести до пропуску глобального мінімуму і зробити процес навчання розбіжним;

– неможливість визначення точок локального і глобального мінімуму, так як градієнтний метод їх не розрізняє;

– вплив випадкової ініціалізації вагових коефіцієнтів нейронної мережі на пошук мінімуму функції середньоквадратичної помилки.

При розробці був використаний тип ШНМ YOLO3 котрий добре показав себе у задачі детектування автомобільних номерів. Для збільшення точності необхідно здобути багато нових зображень для тренування з різною якістю, нахилом та також різні тестові зображення та деякі з них повинні мати об’єкти схожі на номери, для того щоб зменшити помилкові знаходження.

Як подальший розвиток проекту може бути реалізован модуль заповнення регіону з номером, що допоможе спростити та здобути велику якість нормалізації регіону з номером для подальшого розпізнавання. Також важливим розвитком буде додавання модулю розпізнавання самого номеру з зображення та відображення його у текстовому виді. Така система може бути використана у великій кількості сфер автоматизації, таких як ідентифікація правопорушників, та інший сферах, котрі з’являються кожний рік.

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, чинників впливу на навколишнє природнє середовище з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в дипломній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника. Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Було наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1) Воронцов К. В. (2004). Машинное обучение: курс лекцій. Дата звернення 01.11.2020 http://www.machinelearning.ru/wiki/

2) Christopher M. (весна 2006). Bishop Pattern Recognition and Machine Learning.

3) Хайкин С. (2006). Нейронные сети: полный курс, 2-е изд.

4) Зайченко Ю.П. (2008). Нечёткие модели и методы в интеллектуальных системах. Учебное пособие для студентов высших учебных заведений.

5) Штовба С.Д. (2007). Проектирование нечетких систем средствами MATLAB.

6) Круглов В.В., В.В. Борисов (2001). Принятие решений на основе нечетких моделей : примеры использования.

7) Банди Б. (1988). Методы оптимизации. Вводний курс.

8) Андриевская Н. В., Бочкарев Бочкарев С. В. (2008). Моделирование систем: учебное пособие.

9) Малыхина М.П., Шичкин М.П. (2013). Аспекты практического применения цветового различия для распознавания и выделения границ зображений. Политематический сетевой электронный научный журнал Кубанского государственного аграрного университета. № 89(09), 676-688.

10) Грузман И.С.б Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А. (2002). Цифровая обработка изображений в информационных системах: Учебное пособие.

11) Городецкий А.Е., И.Л.Тарасова. (2005). Управление и нейронные сети.

12) Захарова Е.М., Минашина И.К. (2014). Обзор методов многомерной оптимизации. Математические модели. Вычислительные методы. Т. 14. № -3. 256-274.

13) Граничин О.Н., Поляк Б.Т. (2003). Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах.

14) Бенджио И., Гудфеллоу Я., Курвилль А. (2018). Глубокое обучение.

15) Воронцов К. Математические методы обучения по прецедентам. http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_%28курс_лекций%2C_К.В.Воронцов%29

16) Deng L., Yu, D. (2014). Deep Learning: Methods and Applications // Foundations and Trends in Signal Processing. Vol. 7, No. 3–4. 197-387.

17) Николенко С., Кадурин А., Архангельская Е. (2018). Глубокое обучение. Погружение в мир нейронных сетей.

18) Лаврищева Е.М. (2006). Методы программирования. Теория, инженерия, практика.

19) Лавріщева К.М. (2003)/ Основні напрямки досліджень в програмній інженерії і шляхи їхнього розвитку. Проблеми програмування. № 3–4. 44–58.

20) Ладогубець В.В., Ладогубець Т.С., Ладогубець О.В. (2006). Алгоритми параметричної оптимізації складних систем.

21) Соколов В.Ю. (2010). Інформаційні системи і технології.

22) Тоценко В.Г. (2002). Методы и системы поддержки принятия решений.

23) Фаддеев, Д. К., Фаддеева В.Н. (2002). Вычислительные методы линейной алгебры.

24) Гольштейн Е.Г., Третьяков И.В. Модифицированные функции Лагранжа: Теория и методы оптимизации. М.: Наука, 1989. 400 с.

25) Нурминский Е.А. (1991). Численные методы выпуклой оптимизации.

26) Евтушенко Ю.Г., Мазурик В.П. (1989). Программное обеспечение систем оптимизации.

27) Беклемешева А. Why Use Python for AI and Machine Learning? Дата звернення 12.11.2020 https://steelkiwi.com/blog/python-for-ai-and-machine-learning/

28) Why use Keras? Дата звернення 12.11.2020 https://keras.io/why-use-keras/

29) Патідар Ш. (31.05.2018). What Is TensorFlow and What Is New in It? Дата звернення 12.11.2020 https://dzone.com/articles/what-is-tensorflow-and-what-is-new-in-it.

30) Сторінка проекту tensorflow . Дата звернення 12.11.2020 https://www.tensorflow.org/?hl=en

31) Державні санітарни норми і правила. ДСанПіН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» Режим доступу: WWW. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98

32) Державні санітарни норми України. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» Режим доступу: WWW. URL: https://zakon.rada.gov.ua/rada/show/va042282-99

33) Державні санітарни норми України. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» Режим доступу: WWW. URL: https://zakon.rada.gov.ua/rada/show/va037282-99

34) Державні санітарни норми України. ДСН 3.3.6.039-99 «Санітарні норми виробничої загальної та локальної вібрації» Режим доступу: WWW. URL: https://zakon.rada.gov.ua/rada/show/va039282-99

35) Державний стандарт України. ГОСТ 13109-97 «Электрическая энергия. Совместимость технических средств электромагнитных. Нормы качества электроэнергоснабжения общего назначения» Режим доступу: WWW. URL: http://odz.gov.ua/lean_pro/standardization/files/elektromagnitnaja_sovmestimost_2014_03_11_1.pdf

36) Державні будівельні норми України. ДБН В.2.5-28:2018 «Природнє і штучне освітлення» Режим доступу: WWW. URL: https://okna.ua/img_all/oknaua/dbn-V-2-5-28-2018-ed.pdf

37) Нормативно-правові акти з охорони праці. НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів» Режим доступу: WWW. URL: https://zakon.rada.gov.ua/laws/show/z0093-98

38) Державні будівельні норми України. ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування» Режим доступу: WWW. URL: https://dnaop.com/html/32609/doc-%D0%94%D0%91%D0%9D_%D0%92.2.5-67_2013

39) Державний стандарт України. ГОСТ 12.1.044-89 «ССБТ. Пожаровзрывоопасность веществ и материалов. Номенклатура показателей и методы их определения» Режим доступу: WWW. URL: http://online.budstandart.com/ru/catalog/doc-page?id_doc=51048

40) НПАОП 0.00-7.15-18 Вимоги щодо безпеки та захисту здоров`я працівників під час роботи з екранними пристроями Міністерство доходів і зборів України Наказ від 05.09.2013 р. № 443 «Про затвердження Примірної інструкції з охорони праці під час експлуатації електронно-обчислювальних машин» Режим доступу: WWW. URL: http://sop.zp.ua/norm_npaop_0_00-7_15-18_01_ua.php

41) Нормативно-правові акти з охорони праці. НПАОП 0.00-4.15-98 «Про розробку інструкцій з охорони праці» Режим доступу: WWW. URL: http://sop.zp.ua/norm_npaop_0_00-4_15-98_01_ru.php

42) Закон України «Про охорону навколишнього природного середовища». Вводиться в дію Постановою ВР № 1268-XII від 26.06.91, ВВР, 1991, № 41, ст.547. Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/1264-12

43) Закон України «Про забезпечення санітарного та епідемічного благополуччя населення». Вводиться в дію Постановою ВР № 4005-XII від 24.02.94, ВВР, 1994, № 27, ст.219. - Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/4004-12

44) Закон України «Про відходи». Відомості Верховної Ради України (ВВР), 1998, № 36-37, ст.242. Режим доступу: www. URL: https://zakon.rada.gov.ua/laws/show/187/98-вр

ДОДАТОК А.

Тестові зображення

ДОДАТОК Б.

Лістинг модулю формування набору даних

```
"""
data.py
"""
from collections import Counter
import glob
import os

import cv2
import numpy as np
from sklearn.cluster import KMeans
import scipy.stats as st
import tensorflow as tf

from colors import colors_dict


def get_dominant_color(image, k=6, image_processing_size=(200, 200), thresh=0.0):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    # resize image if new dims provided
    if image_processing_size is not None and \
            image.shape[0] > image_processing_size[0] or image.shape[1] > image_processing_size[1]:
        image = cv2.resize(image, image_processing_size,
                           interpolation=cv2.INTER_AREA)

    # reshape the image to be a list of pixels
    image = image.reshape((image.shape[0] * image.shape[1], 3))

    # cluster and assign labels to the pixels
    clt = KMeans(n_clusters=k)
    labels = clt.fit_predict(image)

    # count labels to find most popular
    label_counts = Counter(labels).most_common(k)

    output, mapped = [], []
    for count in label_counts:
        if count[1] > thresh*image.shape[0]:
            color = clt.cluster_centers_[count[0]]
            output.append(color)

    return np.array(output)

def convert_to_lab(image):
    """Convert RGB image to LAB."""
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    return image.reshape((image.shape[0] * image.shape[1], 3))

def map_to_colors(lab_in, lemotif_colors, sim_thresh=1500, freq_thresh=0.2, one_hot=True):
    """Extract dominant colors from image based on set of specified colors."""
    all_colors = []
    for color, info in lemotif_colors.items():
        all_colors.append(lemotif_colors[color]['lab'])
    targets = np.array(all_colors)

    diff = np.sum(np.square(lab_in[:, None, ...] - targets), axis=(2))
```

```
        best_match = np.argsort(-diff, axis=1)[:,diff.shape[1]-1::]

        if one_hot:
            matches = np.zeros(diff.shape[1], dtype='float32')
            for target in range(diff.shape[1]):
                pixel_idx = np.where(best_match == target)[0]
                if len(pixel_idx) > 0:
                    pixel_diffs = diff[pixel_idx, np.ones((len(pixel_idx)), dtype='int32')*target]
                    n_below_thresh = np.where(pixel_diffs < sim_thresh)[0].shape[0]
                    if n_below_thresh > freq_thresh*diff.shape[0]:
                        matches[target] = 1.0
            return matches
        else:
            matches = []
            for match in range(best_match.shape[0]):
                if diff[match, best_match[match][0]] < sim_thresh:
                    matches.append((list(lemotif_colors)[best_match[match][0]],
                                    lemotif_colors[list(lemotif_colors)[best_match[match][0]]]['color']))
            return set(matches)

    def generate_crops(source_dir, out_dir, crop_size, up_factor):
        """Generate random crops for training."""
        img_files = []
        for e in ['*.jpg', '*.jpeg', '*.JPG', '*.JPEG', '*.png']:
            img_files += glob.glob(os.path.join(source_dir, e))
        for img_file in img_files:
            img = cv2.imread(img_file)
            if img.shape[0] <= crop_size or img.shape[1] <= crop_size:
                max_factor = up_factor*crop_size/min(img.shape[0], img.shape[1])
                img = cv2.resize(img, (int(max_factor*img.shape[1]), int(max_factor*img.shape[0])))
            for c in range(int(img.shape[0]/crop_size)):
                rand_row, rand_col = np.random.randint(0, img.shape[0]-256), np.random.randint(0,
img.shape[1]-256)
                crop = img[rand_row:rand_row+256, rand_col:rand_col+256]
                lab = convert_to_lab(crop)
                mapped = map_to_colors(lab, colors_dict, sim_thresh=1250, freq_thresh=0.3)

                if np.sum(mapped) > 0:
                    extension = img_file.split('.')[-1]
                    name = img_file.replace('.' + extension, '')\
                        .replace(source_dir, out_dir)
                    new_file = name + '_' + str(c) + '.jpg'

                    cv2.imwrite(new_file, crop)
                    np.savetxt(new_file.replace('jpg', 'txt'), mapped)
                    np.save(new_file.replace('.jpg', ''), mapped)

    class DataGenerator():

        def __init__(self, **kwargs):
            self.source_dir = kwargs['source_dir']
            self.shape = kwargs['shape']
            self.batch_size = kwargs['batch_size']
            self.tags = kwargs['tag_dict']

        def extract_colors(self, img):
            dominant = get_dominant_color(img, thresh=0.2)
            mapped = map_to_colors(dominant, self.tags)
            return mapped

        def parse_cond(self, file, label_file):
            img = tf.image.decode_jpeg(tf.io.read_file(file), channels=3)
```

```
            img = (tf.image.resize(img, self.shape, align_corners=True, preserve_aspect_ratio=False) -
127.5)/127.5
            label = tf.py_func(np.load, [label_file], tf.float32)

            return img, label

        def parse_blur(self, file):
            img = tf.image.decode_jpeg(tf.io.read_file(file), channels=3)
            img = (tf.image.resize(img, self.shape, align_corners=True, preserve_aspect_ratio=False) -
127.5)/127.5

            return img

        def gen_dataset(self, parser):
            img_files = []
            for e in ['*.jpg', '*.jpeg', '*.JPG', '*.JPEG']:
                img_files += glob.glob(os.path.join(self.source_dir, e))

            if parser == 'parse_cond':
                labels = [img.replace(img.split('.')[-1], 'npy') for img in img_files]
                dataset_source = (img_files, labels)
            else:
                dataset_source = img_files

            dataset = tf.data.Dataset.from_tensor_slices(dataset_source).repeat().\
                shuffle(len(img_files)).map(getattr(self, parser)).batch(self.batch_size)

            return dataset
```

ДОДАТОК В.

Лістинг модулю підготовки моделі

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.layers as kl

from utils import instance_norm


# Fading function
def blend_resolutions(upper, lower, alpha):
    upper = tf.multiply(upper, alpha[..., tf.newaxis, tf.newaxis])
    lower = tf.multiply(lower, tf.subtract(1.0, alpha)[..., tf.newaxis, tf.newaxis])
    return kl.Add()([upper, lower])


def cond_stylegan(out_size, fade_outputs, z_dim, num_classes, conv_init, batch_size,
                  map_layers=4, start_size=8, start_filters=512, n_channels=3):
    conv_loop = int(np.log2(out_size / start_size)) + 2

    z = kl.Input(shape=(z_dim,))
    fade = kl.Input(shape=(1,))
    conditioning = kl.Input(shape=(num_classes,))
    constant = kl.Input(shape=(1,))

    # Mapping network
    # w = tf.concat((z, conditioning), 1)  # Concatenate noise input with condition labels
    w = kl.Concatenate(axis=-1)([z, conditioning])

    for layer in range(map_layers):
        w = kl.Dense(units=start_filters, name='dense_map' + '_' + str(layer))(w)
        w = kl.LeakyReLU(alpha=.2)(w)

    # Synthesis network
    lower_res = None
    for resolution in range(conv_loop):
        filters = max(start_filters // 2 ** (resolution), 4)
        if resolution == 0:  # 4x4
            with tf.variable_scope('Constant'):
                x = kl.Dense(units=4 * 4 * filters)(constant)
                x = kl.Reshape([4, 4, filters])(x)
                # x = kl.Lambda(lambda x: add_noise(x, self.batch_size))(x)
                x = kl.LeakyReLU(alpha=.2)(x)

                # Adaptive instance normalization and style moderation.
                x = kl.Lambda(lambda x: instance_norm(x))(x)
                style_mul = kl.Dense(units=filters, name='dense_' + str(resolution) + '_0_0')(w)
                style_add = kl.Dense(units=filters, bias_initializer='Ones',
                                     name='dense_' + str(resolution) + '_0_1')(w)
                x = kl.Multiply()([x, style_mul])
                x = kl.Add()([x, style_add])

            # Single convolution.
            with tf.variable_scope('Conv_' + str(resolution) + '_0'):
                x = kl.Conv2D(filters=filters, kernel_size=3, padding='same',
                              name='conv_' + str(resolution) + '_0')(x)
                # x = kl.Lambda(lambda x: add_noise(x, self.batch_size))(x)
```

```
                x = kl.LeakyReLU(alpha=.2)(x)

                # Adaptive instance normalization operation and style moderation.
                x = kl.Lambda(lambda x: instance_norm(x))(x)
                style_mul = kl.Dense(units=filters, name='dense_' + str(resolution) + '_1_0')(w)
                style_add = kl.Dense(units=filters, bias_initializer='Ones',
                                     name='dense_' + str(resolution) + '_1_1')(w)
                x = kl.Multiply()([x, style_mul])
                x = kl.Add()([x, style_add])
            else:
                x = kl.UpSampling2D(interpolation='bilinear')(x)
                for conv in range(2):
                    with tf.variable_scope('Conv_' + str(resolution) + '_' + str(conv)):
                        x = kl.Conv2D(filters=filters, kernel_size=3, padding='same',
                                      name='conv_' + str(resolution) + '_' + str(conv))(x)
                        # x = kl.Lambda(lambda x: add_noise(x, self.batch_size))(x)
                        x = kl.LeakyReLU(alpha=.2)(x)

                        # Adaptive instance normalization operation and style moderation.
                        x = kl.Lambda(lambda x: instance_norm(x))(x)
                        style_mul = kl.Dense(units=filters, name='dense_' + str(resolution) + '_' + str(conv) +
'_0')(w)
                        style_add = kl.Dense(units=filters, bias_initializer='Ones',
                                             name='dense_' + str(resolution) + '_' + str(conv) + '_1')(w)
                        x = kl.Multiply()([x, style_mul])
                        x = kl.Add()([x, style_add])
                if resolution == conv_loop - 2 and conv_loop > 1:
                    lower_res = x

            # Conversion to 3-channel color
            convert_to_image = kl.Conv2D(filters=n_channels, kernel_size=1, strides=1, padding='same',
                                         kernel_initializer=conv_init, use_bias=True, activation='tanh',
                                         name='conv_to_img_' + str(x.get_shape().as_list()[-1]))
            pre_img_filters = x.get_shape().as_list()[-1]
            x = convert_to_image(x)

            if fade_outputs:
                if lower_res.get_shape().as_list()[-1] == pre_img_filters:
                    convert_to_image_lower = convert_to_image
                else:
                    convert_to_image_lower = kl.Conv2D(filters=3, kernel_size=1, strides=1, padding='same',
                                                       kernel_initializer=conv_init, use_bias=True, activation='tanh',
                                                       name='conv_to_img_' + str(lower_res.get_shape().as_list()[-1]))
                # Fade output of previous resolution stage into final resolution stage
                lower_upsampled = kl.UpSampling2D(interpolation='bilinear')(lower_res)
                lower_upsampled = convert_to_image_lower(lower_upsampled)
                x = kl.Lambda(lambda args: blend_resolutions(args[0], args[1], args[2]))([x, lower_upsampled,
fade])

            return tf.keras.models.Model(inputs=[z, conditioning, fade, constant], outputs=x, name='generator')


        def cond_progan(out_size, fade_outputs, z_dim, num_classes, conv_init, batch_size, cond_start=False,
                        start_size=8, start_filters=512, n_channels=3):
            conv_loop = int(np.log2(out_size / start_size)) + 2

            z = kl.Input(shape=(z_dim,))
            fade = kl.Input(shape=(1,))
            conditioning = kl.Input(shape=(num_classes,))
            if cond_start:
                # z_cond = tf.concat((z, conditioning), 1)
                z_cond = kl.Concatenate(axis=-1)([z, conditioning])
            else:
```

```
            z_cond = z

        # Synthesis network
        lower_res = None
        for resolution in range(conv_loop):
            filters = max(start_filters // 2 ** (resolution), 4)
            if resolution == 0:  # 4x4
                with tf.variable_scope('Dense_' + str(resolution) + '_0'):
                    x = kl.Dense(units=4 * 4 * filters, name='dense_1')(z_cond)
                    x = kl.Reshape([4, 4, filters])(x)
                    x = kl.BatchNormalization(name='bn_'+str(resolution) + '_0_' +str(filters))(x)
                    x = kl.LeakyReLU(alpha=.2)(x)

                # Single convolution.
                with tf.variable_scope('Conv_' + str(resolution) + '_1'):
                    x = kl.Conv2D(filters=filters, kernel_size=3, padding='same', kernel_initializer=conv_init,
                                  name='conv_' + str(resolution) + '_0')(x)
                    x = kl.BatchNormalization(name='bn_'+str(resolution) + '_1_' +str(filters))(x)
                    x = kl.LeakyReLU(alpha=.2)(x)
            else:
                x = kl.UpSampling2D(interpolation='bilinear')(x)
                for conv in range(2):
                    with tf.variable_scope('Conv_' + str(resolution) + '_' + str(conv)):
                        x = kl.Conv2D(filters=filters, kernel_size=3, padding='same', kernel_initializer=conv_init,
                                      name='conv_' + str(resolution) + '_' + str(conv))(x)
                        x = kl.BatchNormalization(name='bn_'+str(resolution) + '_' + str(conv) + '_' +str(filters))(x)
                        x = kl.LeakyReLU(alpha=.2)(x)
            if resolution == conv_loop - 2 and conv_loop > 1:
                lower_res = x

        if not cond_start:
            cond_tiled = kl.Lambda(tf.tile, arguments={'multiples': [1, out_size ** 2]})(conditioning)
            cond_layers = kl.Reshape((out_size, out_size, num_classes))(cond_tiled)
            x = kl.Concatenate(axis=-1)([x, cond_layers])
            x = kl.Conv2D(filters=filters, kernel_size=1, strides=1, padding='same', kernel_initializer=conv_init,
                          use_bias=True, name='conv_pre_img_' + str(out_size) + '_' + str(x.get_shape().as_list()[-
1]))(x)

        # Conversion to 3-channel color
        convert_to_image = kl.Conv2D(filters=n_channels, kernel_size=1, strides=1, padding='same',
                                     kernel_initializer=conv_init, use_bias=True, activation='tanh',
                                     name='conv_to_img_' + str(x.get_shape().as_list()[-1]))
        pre_img_filters = x.get_shape().as_list()[-1]
        x = convert_to_image(x)

        if fade_outputs:
            if lower_res.get_shape().as_list()[-1] == pre_img_filters:
                convert_to_image_lower = convert_to_image
            else:
                convert_to_image_lower = kl.Conv2D(filters=3, kernel_size=1, strides=1, padding='same',
                                                   kernel_initializer=conv_init, use_bias=True, activation='tanh',
                                                   name='conv_to_img_' + str(lower_res.get_shape().as_list()[-1]))
            # Fade output of previous resolution stage into final resolution stage
            if not cond_start:
                lower_cond_tiled = kl.Lambda(tf.tile, arguments={'multiples': [1, int((out_size/2) **
2)]})(conditioning)
                lower_cond_layers = kl.Reshape((int(out_size/2), int(out_size/2),
num_classes))(lower_cond_tiled)
                lower_res = kl.Concatenate(axis=-1)([lower_res, lower_cond_layers])
                ls = lower_res.get_shape().as_list()
                lower_res = kl.Conv2D(filters=filters*2, kernel_size=1, strides=1, padding='same',
                                      kernel_initializer=conv_init, use_bias=True,
                                      name='conv_pre_img_' + str(ls[1]) + '_' + str(ls[-1]))(lower_res)
```

```
            lower_upsampled = kl.UpSampling2D(interpolation='bilinear')(lower_res)
            lower_upsampled = convert_to_image_lower(lower_upsampled)
            x = kl.Lambda(lambda args: blend_resolutions(args[0], args[1], args[2]))([x, lower_upsampled,
fade])

        return tf.keras.models.Model(inputs=[z, conditioning, fade], outputs=x, name='generator')


    def encoder_basic(in_size, z_dim, conv_init, max_filters=512):

        conv_loop = int(np.log2(in_size)) - 2

        img = kl.Input(shape=(in_size, in_size, 3,))

        start_filters = int(max(max_filters / 2 ** conv_loop, 4))
        x = img
        for resolution in range(conv_loop):
            filters = start_filters * 2 ** (resolution + 1)
            x = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same',
                          kernel_initializer=conv_init,
                          name='conv_' + str(in_size / 2 ** (resolution)) + '_0_' + '_' + str(filters))(x)
            x = kl.BatchNormalization()(x)
            x = kl.LeakyReLU(.2)(x)

            x = kl.Conv2D(filters=filters, kernel_size=3, strides=2, padding='same',
                          kernel_initializer=conv_init,
                          name='conv_' + str(in_size / 2 ** (resolution)) + '_1_' + '_' + str(filters))(x)
            x = kl.BatchNormalization()(x)
            x = kl.LeakyReLU(.2)(x)

        return tf.keras.models.Model(inputs=[img], outputs=[z], name='encoder')


    def encoder_residual(in_size, z_dim, conv_init, max_filters=512):
        img = kl.Input(shape=(in_size, in_size, 3,))

        conv_loop = int(np.log2(in_size)) - 1
        start_filters = int(max(max_filters / 2 ** (conv_loop - 1), 4))
        x = kl.Conv2D(filters=start_filters, kernel_size=1, strides=1, padding='same',
kernel_initializer=conv_init,
                      kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                      name='conv_from_img_' + str(start_filters))(img)

        for resolution in range(conv_loop):
            filters = start_filters * 2 ** (resolution)
            strides = 2 if resolution > 0 else 1
            x1 = kl.Conv2D(filters=filters, kernel_size=3, strides=strides, padding='same',
kernel_initializer=conv_init,
                           kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                           name='conv_' + str(in_size / 2 ** (resolution)) + '_0_' + '_' + str(filters))(x)
            x1 = kl.BatchNormalization()(x1)
            x1 = kl.LeakyReLU(.2)(x1)

            x2 = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same',
kernel_initializer=conv_init,
                           kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                           name='conv_' + str(in_size / 2 ** (resolution)) + '_1_' + '_' + str(filters))(x1)
            x2 = kl.BatchNormalization()(x2)

            if resolution > 0:
                x = kl.Conv2D(filters=filters, kernel_size=1, strides=2, padding='same',
kernel_initializer=conv_init,
                              kernel_regularizer=tf.keras.regularizers.l2(1e-4),
```

```
                    name='conv_resid_down_' + str(x.get_shape().as_list()[-1]))(x)
        x = kl.Add()([x, x2])
        x = kl.LeakyReLU(.2)(x)

    return tf.keras.models.Model(inputs=[img], outputs=[z], name='encoder')


def decoder_basic(out_size, z_dim, conv_init, max_filters=512):

    conv_loop = int(np.log2(out_size / 8)) + 2

    z = kl.Input(shape=(z_dim,))

    for resolution in range(conv_loop):
        filters = max(max_filters // 2 ** (resolution), 4)
        if resolution == 0:  # 4x4
            with tf.variable_scope('Dense_' + str(resolution) + '_0'):
                x = kl.Dense(units=4 * 4 * filters, name='dense_' + str(filters))(z)
                x = kl.Reshape([4, 4, filters])(x)
                x = kl.BatchNormalization()(x)
                x = kl.LeakyReLU(alpha=.2)(x)

            # Single convolution.
            with tf.variable_scope('Conv_' + str(resolution) + '_1'):
                x = kl.Conv2D(filters=filters, kernel_size=3, padding='same', kernel_initializer=conv_init,
                              name='conv_' + str(resolution) + '_0')(x)
                x = kl.BatchNormalization()(x)
                x = kl.LeakyReLU(alpha=.2)(x)
        else:
            x = kl.UpSampling2D(interpolation='bilinear')(x)
            for conv in range(2):
                with tf.variable_scope('Conv_' + str(resolution) + '_' + str(conv)):
                    x = kl.Conv2D(filters=filters, kernel_size=3, padding='same', kernel_initializer=conv_init,
                                  name='conv_' + str(resolution) + '_' + str(conv))(x)
                    x = kl.BatchNormalization()(x)
                    x = kl.LeakyReLU(alpha=.2)(x)

    # Conversion to 3-channel color
    x = kl.Conv2D(filters=3, kernel_size=1, strides=1, padding='same',
                  kernel_initializer=conv_init, use_bias=True, activation='tanh',
                  name='conv_to_img_' + str(x.get_shape().as_list()[-1]))(x)

    return tf.keras.models.Model(inputs=[z], outputs=[x], name='decoder')


def discriminator_basic(out_size, conv_init, max_filters=512):

    conv_loop = int(np.log2(out_size)) - 2

    img = kl.Input(shape=(out_size, out_size, 3,))

    start_filters = int(max(max_filters / 2 ** (conv_loop - 1), 4))
    x = img
    for resolution in range(conv_loop):
        filters = start_filters * 2 ** resolution
        x = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same', kernel_initializer=conv_init,
                      name='conv_' + str(out_size / 2 ** (resolution)) + '_0_' + '_' + str(filters))(x)
        x = kl.BatchNormalization()(x)
        x = kl.LeakyReLU(.2)(x)

        x = kl.Conv2D(filters=filters, kernel_size=3, strides=2, padding='same', kernel_initializer=conv_init,
                      name='conv_' + str(out_size / 2 ** (resolution)) + '_1_' + '_' + str(filters))(x)
        x = kl.BatchNormalization()(x)
```

```
            x = kl.LeakyReLU(.2)(x)

        # Convert to single value
        x = kl.Flatten()(x)
        disc_out = kl.Dense(1, name='dense_to_score_' + str(x.get_shape().as_list()[-1]))(x)

        return tf.keras.models.Model(inputs=[img], outputs=[disc_out], name='discriminator')


    def discriminator_residual(out_size, conv_init, max_filters=512):
        img = kl.Input(shape=(out_size, out_size, 3,))

        conv_loop = int(np.log2(out_size)) - 1
        start_filters = int(max(max_filters / 2 ** (conv_loop - 1), 4))
        x = kl.Conv2D(filters=start_filters, kernel_size=1, strides=1, padding='same',
kernel_initializer=conv_init,
                      kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                      name='conv_from_img_' + str(start_filters))(img)

        for resolution in range(conv_loop):
            filters = start_filters * 2 ** (resolution)
            strides = 2 if resolution > 0 else 1
            x1 = kl.Conv2D(filters=filters, kernel_size=3, strides=strides, padding='same',
kernel_initializer=conv_init,
                           kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                           name='conv_' + str(out_size / 2 ** (resolution)) + '_0_' + '_' + str(filters))(x)
            x1 = kl.BatchNormalization()(x1)
            x1 = kl.LeakyReLU(.2)(x1)

            x2 = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same',
kernel_initializer=conv_init,
                           kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                           name='conv_' + str(out_size / 2 ** (resolution)) + '_1_' + '_' + str(filters))(x1)
            x2 = kl.BatchNormalization()(x2)

            if resolution > 0:
                x = kl.Conv2D(filters=filters, kernel_size=1, strides=2, padding='same',
kernel_initializer=conv_init,
                              kernel_regularizer=tf.keras.regularizers.l2(1e-4),
                              name='conv_resid_down_' + str(x.get_shape().as_list()[-1]))(x)
            x = kl.Add()([x, x2])
            x = kl.LeakyReLU(.2)(x)

        # Convert to single value
        x = kl.Flatten()(x)
        disc_out = kl.Dense(1, name='dense_to_score_' + str(x.get_shape().as_list()[-1]))(x)

        return tf.keras.models.Model(inputs=[img], outputs=[disc_out], name='discriminator')

    def discriminator_progressive(out_size, fade_inputs, batch_size, n_channels, num_classes, conv_init,
max_filters=512):

        # Fading function
        def blend_resolutions(upper, lower, alpha):
            upper = tf.multiply(upper, alpha[..., tf.newaxis, tf.newaxis])
            lower = tf.multiply(lower, tf.subtract(1.0, alpha)[..., tf.newaxis, tf.newaxis])
            return kl.Add()([upper, lower])

        conv_loop = int(np.log2(out_size)) - 1

        img = kl.Input(shape=(out_size, out_size, n_channels,))
        conditioning = kl.Input(shape=(num_classes,))
        fade = kl.Input(shape=(1,))
```

```
        # Convert from RGB
        start_filters = int(max(max_filters / 2 ** (conv_loop), 4))
        x = kl.Conv2D(filters=int(start_filters), kernel_size=1, strides=1, padding='same',
                  kernel_initializer=conv_init, use_bias=True,
                  name='conv_from_img_'+str(out_size))(img)
        x = kl.BatchNormalization(name='bn_from_img_' + str(out_size) + '_' + str(start_filters))(x)
        x = kl.LeakyReLU(.2)(x)

        if fade_inputs:
            # Calculate discriminator score using alpha-blended combination of new discriminator layer outputs
            # versus downsampled version of input images
            downsampled = kl.AveragePooling2D(pool_size=(2, 2), padding='same')(img)
            downsampled = kl.Conv2D(filters=int(start_filters*2), kernel_size=1, strides=1, padding='same',
                            kernel_initializer=conv_init, use_bias=True,
                            name='conv_from_img_'+str(int(out_size/2)))(downsampled)
            downsampled = kl.BatchNormalization(name='bn_from_img_' + str(int(out_size/2)) + '_' +
str(int(start_filters*2))
                                        )(downsampled)
            downsampled = kl.LeakyReLU(.2)(downsampled)

        for resolution in range(conv_loop):
            filters = start_filters * 2 ** (resolution + 1)

            # if resolution == conv_loop - 1:
                # x = minibatch_stddev_layer(x, batch_size)
                # x = kl.Lambda(minibatch_stddev_layer, arguments={'batch_size': batch_size})(x)

            x = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same',
                      kernel_initializer=conv_init,
                      name='conv_' + str(out_size / 2 ** (resolution)) + '_0_' + '_' + str(filters))(x)
            x = kl.BatchNormalization(name='bn_' + str(resolution) + '_0_' + str(filters))(x)
            x = kl.LeakyReLU(.2)(x)

            if resolution < conv_loop - 1:
                x = kl.Conv2D(filters=filters, kernel_size=3, strides=1, padding='same',
                          kernel_initializer=conv_init,
                          name='conv_' + str(out_size / 2 ** (resolution)) + '_1_' + '_' + str(filters))(x)
                x = kl.BatchNormalization(name='bn_' + str(resolution) + '_1_' + str(filters))(x)
                x = kl.LeakyReLU(.2)(x)
                x = kl.AveragePooling2D(pool_size=(2, 2), padding='same')(x)

            if resolution == 0 and fade_inputs:
                x = kl.Lambda(lambda args: blend_resolutions(args[0], args[1], args[2]))([x, downsampled, fade])

        # Convert to single value
        x = kl.Flatten()(x)
        x = kl.Dense(max_filters, name='dense_pre_'+str(x.get_shape().as_list()[-1]))(x)
        x = kl.LeakyReLU(.2)(x)

        # Label prediction
        labels_out = kl.Dense(num_classes, name='dense_to_labels_'+str(x.get_shape().as_list()[-1]))(x)

        # Discriminator score
        x = kl.Concatenate(axis=-1)([x, conditioning])

        disc_out = kl.Dense(1, name='dense_to_score_'+str(x.get_shape().as_list()[-1]))(x)

        return tf.keras.models.Model(inputs=[img, conditioning, fade], outputs=[disc_out, labels_out],
name='discriminator')
```

ДОДАТОК Г.

Лістинг модулів навчання системи

```
"""
train_ae.py
Train AE-style model.
"""
import os
import tensorflow as tf

from autoencoder import AutoEncoderGAN
from data import DataGenerator
from colors import colors_dict


PARAMS = {
    'dataset': 'wikiart',
    'source_dir': '/home/alice/data/wikiart-abstract-crop/',
    'tag_dict': colors_dict,
    'in_size': 16,
    'out_size': 256,
    'disc_size': 256,
    'enc_model': 'encoder_residual',
    'dec_model': 'decoder_basic',
    'disc_model': 'discriminator_residual',
    'shape': (256, 256),
    'batch_size': 8,
    'z_dim': 512,
    'max_filters': 512,
    'iterations': 1000000,  # Number of steps
    'lr': 0.00001,  # Learning rate
    'b1': 0.5,  # Adam beta1
    'b2': 0.99,   # Adam beta2
    'loss_weights': {'l2_gen': 1000, 'l2_loss_size': 256, 'discriminator': 0.01},
    'blur_kernel': 0,
    'save_int': 20000,  # Number of steps before generating output
    'save_dir':   '/home/alice/lemotif/outputs/wikiart_ae_nodisc_resid_16-256_blur0/',      # Output data
directory
    'verbose': True
}

if __name__ == '__main__':
    if not os.path.isdir(PARAMS['save_dir']): os.mkdir(PARAMS['save_dir'])
    f = open(os.path.join(PARAMS['save_dir'], "args.txt"), "w")
    f.write(str(PARAMS))
    f.close()

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.per_process_gpu_memory_fraction = 0.6
    tf.compat.v1.enable_eager_execution(config=config)
    with tf.device("/cpu:0"):
        dataset = DataGenerator(**PARAMS).gen_dataset(parser='parse_blur')
    model = AutoEncoderGAN(**PARAMS)
    model.train(train_data=dataset, **PARAMS)

"""
train_cgan.py
```

```
Train conditional GAN style model.
"""
import os
import tensorflow as tf

from condgan import ConditionalGAN
from data import DataGenerator
from colors import colors_dict


PARAMS = {
    'dataset': 'wikiart',
    'generator_model': 'cond_stylegan',
    'discriminator_model': 'discriminator_progressive',
    # 'source_dir': '/home/alice/lemotif/data/wikiart-abstract-crop-2/',
    'source_dir': '/home/alice/data/wikiart-abstract-crop-2/',
    'tag_dict': colors_dict,
    'n_classes': 18,
    'max_filters': 512,
    'shape': (256, 256),
    'batch_size': 8,
    'z_dim': 256,
    'img_size': 256,
    'start_size': 64,
    'n_channels': 3,
    'iterations': {32: 250000, 64: 500000, 128: 500000, 256: 1000000, 512: 1000000},  # Number of steps
    'lr': {32: 0.00001, 64: 0.00001, 128: 0.00001, 256: 0.000001, 512: 0.00001},  # Learning rate
    'b1': 0.5,  # Adam beta1
    'b2': 0.99,   # Adam beta2
    'loss_weights': {'class_label': 1000, 'gen_label': 1000},
    'save_int': 10000,  # Number of epochs before generating output
    'save_dir': '/home/alice/lemotif/outputs/wikiart_residuals_acgan_sg_2/',  # Output data directory
    'verbose': True
}

if __name__ == '__main__':
    if not os.path.isdir(PARAMS['save_dir']): os.mkdir(PARAMS['save_dir'])
    f = open(os.path.join(PARAMS['save_dir'], "args.txt"), "w")
    f.write(str(PARAMS))
    f.close()

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.per_process_gpu_memory_fraction = 0.8
    tf.compat.v1.enable_eager_execution(config=config)
    with tf.device("/cpu:0"):
        dataset = DataGenerator(**PARAMS).gen_dataset(parser='parse_cond')
    model = ConditionalGAN(**PARAMS)
    model.train(train_data=dataset, **PARAMS)
```

ДОДАТОК Д.

Лістинг модулю тестування системи

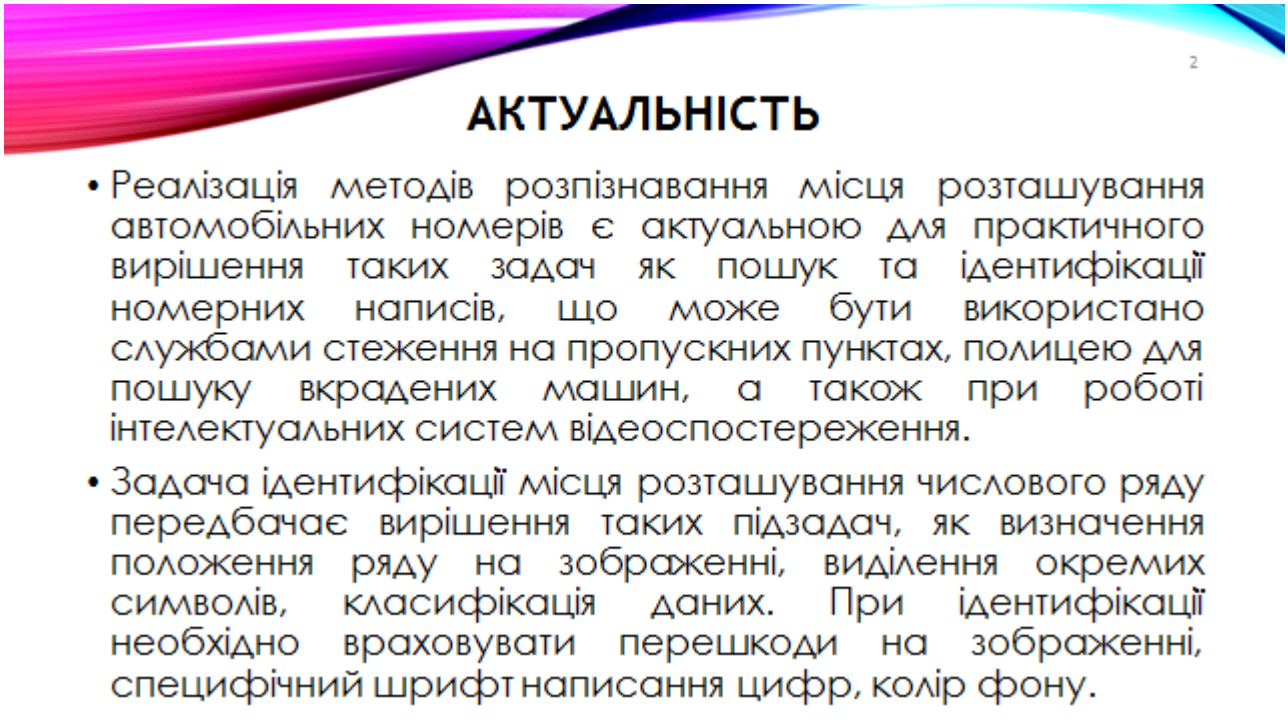
```
"""
utils.py
IO utilities and custom operations.
"""

import numpy as np
import scipy.stats as st
import tensorflow as tf
import tensorflow.keras.layers as kl
from tensorflow.keras import models, initializers, regularizers, constraints


def gauss_kernel(blur_kernel, nsig=3):
    interval = (2*nsig+1.)/(blur_kernel)
    x = np.linspace(-nsig-interval/2., nsig+interval/2., blur_kernel+1)
    kern1d = np.diff(st.norm.cdf(x))
    kernel_raw = np.sqrt(np.outer(kern1d, kern1d))
    kernel = kernel_raw/kernel_raw.sum()

    kernel = tf.expand_dims(tf.stack([kernel, kernel, kernel], axis=2), axis=3)
    return tf.cast(kernel, tf.float32)


def minibatch_stddev_layer(x, batch_size):
    _, h, w, _ = x.get_shape().as_list()
    new_feat_shape = [batch_size, h, w, 0]
    mean, var = tf.nn.moments(x, axes=[0], keep_dims=True)
    stddev = tf.sqrt(tf.reduce_mean(var, keepdims=True))
    new_feat = tf.tile(stddev, multiples=new_feat_shape)
    return tf.concat([x, new_feat], axis=3)


def add_noise(x, batch_size):
    """
    Adds noise to outputs of convolutional layer.
    Args:
        x (tensor): Feature map.
    Returns:
        Feature map with stochastic variation introduced.
    """
    # Add randomly generated single-channel noise images of the same shape as current layer.
    shape = x.get_shape().as_list()
    noise = tf.random.normal([batch_size, shape[1], shape[2], 1], dtype=x.dtype)
    # Apply per-channel scaling factor to noise input through element-wise multiplication.
    noise_scaled = kl.Conv2D(filters=shape[-1], kernel_size=1, padding='same',
                             kernel_initializer=initializers.Zeros())(noise)
    x = kl.Add()([x, noise_scaled])

    return x


def instance_norm(x, epsilon=1e-8):
    """
    Instance normalization of x.
    Args:
        x (tensor): Layer to normalize.
        epsilon (float): Parameter adjusting the standard deviation of x.
```

```
    Returns:
      Normalized layer.
    """
    orig_dtype = x.dtype
    x = tf.cast(x, tf.float32)
    x -= tf.reduce_mean(x, axis=[1, 2], keepdims=True)
    epsilon = tf.constant(epsilon, dtype=x.dtype, name='epsilon')
    x *= tf.rsqrt(tf.reduce_mean(tf.square(x), axis=[1, 2], keepdims=True) + epsilon)
    x = tf.cast(x, orig_dtype)

    return x


class ELRDense(kl.Layer):
    """
    Custom layer for fully connected layer using learning rate equalization.
    """

    def __init__(self, units, gain=np.sqrt(2), lrmul=1.0, bias=True, name=None, **kwargs):
        self.filters = units if isinstance(units, int) else units.value
        self.gain = gain
        self.lrmul = lrmul
        self.use_bias = bias
        self.he_std = None
        self.bias = None
        super(ELRDense, self).__init__(name=name)

    def build(self, input_shape):
        fan_in = np.prod(input_shape.as_list()[1:])
        self.he_std = self.gain / np.sqrt(fan_in)
        self.kernel = self.add_weight(name='kernel',
                                      shape=(fan_in, self.filters),
                                      initializer=initializers.RandomNormal(0, 1.0 / self.lrmul),
                                      trainable=True)
        if self.use_bias:
            self.bias = self.add_weight(name='bias',
                                        shape=(self.filters),
                                        initializer=initializers.Zeros(),
                                        trainable=True)
        super(ELRDense, self).build(input_shape)

    def call(self, x):
        if len(x.shape) > 2:
            x = tf.reshape(x, [-1, np.prod([d.value for d in x.shape[1:]])])
        out = tf.matmul(x, self.kernel * self.he_std * self.lrmul)
        if self.use_bias:
            if len(x.get_shape().as_list()) > 2:
                self.bias = tf.reshape(self.bias, [1, 1, 1, -1])
            out = out + self.bias * self.lrmul
        return out

    def get_config(self):
        config = {'filters': self.filters,
                  'gain': self.gain,
                  'lrmul': self.lrmul,
                  'bias': self.use_bias}
        base_config = super(ELRDense, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.filters)
```

```
class ELRConv2D(kl.Layer):
    """
    Custom layer for 2D convolution layer using learning rate equalization.
    """

    def __init__(self, kernel_size, filters, padding, gain=np.sqrt(2), lrmul=1.0, bias=True, name=None,
**kwargs):
        self.kernel_size = kernel_size
        self.filters = filters if isinstance(filters, int) else filters.value
        self.padding = padding
        self.gain = gain
        self.lrmul = lrmul
        self.use_bias = bias
        self.he_std = None
        self.bias = None
        super(ELRConv2D, self).__init__(name=name)

    def build(self, input_shape):
        fan_in = self.kernel_size * self.kernel_size * input_shape.as_list()[-1]
        self.he_std = self.gain / np.sqrt(fan_in)
        self.kernel = self.add_weight(name='kernel',
                                      shape=(
                                          self.kernel_size, self.kernel_size, input_shape.as_list()[-1], self.filters),
                                      initializer=initializers.RandomNormal(0, 1.0 / self.lrmul),
                                      trainable=True)
        if self.use_bias:
            self.bias = self.add_weight(name='bias',
                                        shape=(self.filters),
                                        initializer=initializers.Zeros(),
                                        trainable=True)
        super(ELRConv2D, self).build(input_shape)

    def call(self, x):
        out = tf.nn.conv2d(x, self.kernel * self.he_std * self.lrmul, strides=[1, 1, 1, 1],
                           padding=self.padding.upper())
        if self.use_bias:
            if len(x.get_shape().as_list()) > 2:
                self.bias = tf.reshape(self.bias, [1, 1, 1, -1])
            out = out + self.bias * self.lrmul
        return out

    def get_config(self):
        config = {'kernel_size': self.kernel_size,
                  'filters': self.filters,
                  'padding': self.padding,
                  'gain': self.gain,
                  'lrmul': self.lrmul,
                  'bias': self.use_bias}
        base_config = super(ELRConv2D, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return input_shape
```

ДОДАТОК Е.

Електронні плакати



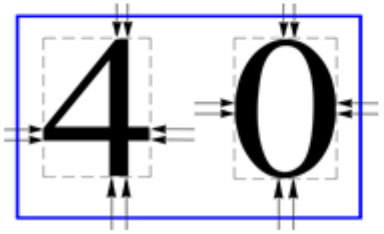
ПОСТАНОВКА ЗАДАЧІ

Метою роботи є дослідження та програмна розробка ШНМ для знаходження місця розташування автомобільних номерів на зображенні. **Об'єктом дослідження** є процес знаходження місця розташування автомобільних номерів. **Предметом дослідження** є ШНМ для знаходження місця розташування автомобільних номерів на зображенні.

ВИДІЛЕННЯ ЦИФРИ НА ЗОБРАЖЕННІ

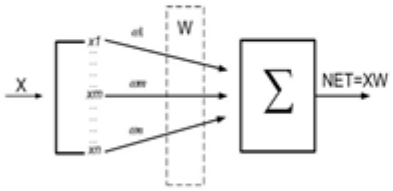
$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$Y = 255(0.21r + 0.72g + 0.07b)^{1/2.2},$$



- завдання значення Yзд в діапазоні [0, 255], що відповідає градації сірого;
- попіксельне зчитування зображення;
- фіксація координати «крайніх» (ліворуч і праворуч, вгорі і внизу) пікселів, значення яких Y>Yзд (фон темніший за цифри);
- обрізка зображення по отриманим «крайніх» точок – вилучення області зображення з цифрою.

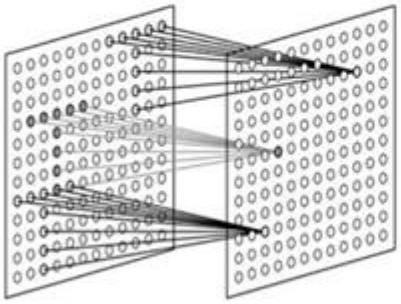
ВИЗНАЧЕННЯ РОЗТАШУВАННЯ ОБ'ЄКТІВ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

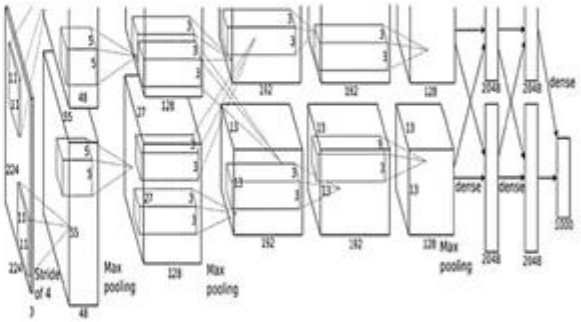


- невизначеність у виборі числа шарів і кількості нейронних елементів в шарі;
- повільна збіжність градієнтного методу з постійним кроком навчання;
- складність вибору відповідної швидкості навчання. Так як маленька швидкість навчання призводить до скочування нейронної мережі в локальний мінімум, а велика швидкість навчання може привести до пропуску глобального мінімуму і зробити процес навчання розбіжним;
- неможливість визначення точок локального і глобального мінімуму, так як градієнтний метод їх не розрізняє;
- вплив випадкової ініціалізації вагових коефіцієнтів нейронної мережі на пошук мінімуму функції середньоквадратичної помилки.

ВИЗНАЧЕННЯ РОЗТАШУВАННЯ ОБ'ЄКТІВ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ

АРХІТЕКТУРА ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ





АЛГОРИТМ НАВЧАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

$$\sum (Y_e - Y_m)^2 \to \min . \quad (1)$$

$$\Delta w_{ij} = \sigma x_i y_j \quad (2)$$

1) Для вагових коефіцієнтів w_{kj} обираються початкові значення.
2) Задається начальний вхідний вектор X
3) Змінюються ваги згідно з співвідношенням:

$$w_{ij}^{н} = w_{ij}^{п} + \Delta w_{ij}$$

де $w_{ij}^{п}, w_{ij}^{н}$ — минулий і наступний вагові коефіцієнти,

$$\Delta w_{ij} = (\alpha x_j \varepsilon_i)$$

$$\varepsilon_i = y_i - f\left(\sum\nolimits_k w_{ik} x_k\right) = y_i - f(net_i)$$

де ε_i – різниця між цільовим і фактичним виходом в точці i; x_i, y_i – вхідний і вихідний сигнали мережі

4) Повернення до етапу 2.

ОЦІНКА ПЕРЕВАГ ТА НЕДОЛІКІВ ВИКОРИСТАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ЗНАХОДЖЕННЯ МІСЦЯ РОЗТАШУВАННЯ АВТОМОБІЛЬНИХ НОМЕРІВ

Недоліками згорткових нейронних мереж стали:

- критична залежність якості настройки ваг мережі від вибору початкового наближення;
- велика схильність до перенавчання укупі зі слабкими можливостями контролю узагальнюючої здатності мережі;
- велика кількість локальних мінімумів функціонала якості, більшість з яких є несприйнятими.

Загальними перевагами згорткових нейронних мереж є:

- нелінійність моделі, що дає можливість апроксимувати нелінійні функції;
- локальність сприйняття, що полягає в тому, що кожен нейрон отримує не весь вхідний вектор. Це дозволяє сегментувати дані і працювати з більш складними ситуаціями;
- каскад шарів. У поєднанні з пунктом 2 це дає здатність сприймати більш абстрактні ознаки;
- можливість якісно працювати з комбінаціями ознак;
- нейронна мережа дає кілька механізмів для контролю перенавчання;
- нейронна мережа здатна донавчатися при несуперечливості нових образів.

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

СТВОРЕННЯ ТА ПІДГОТОВКА НАБОРУ ДАНИХ

НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.layers as kl

from utils import instance_norm

# Fading function
def blend_resolutions(upper, lower, alpha):
    '''
    return kl.Add()([upper, lower])

def cond_stylegan(out_size, fade_outputs, z_dim, num_classes, conv_init, batch_size,
                  map_layers=4, start_size=8, start_filters=512, n_channels=3):
    '''
    # Mapping network
    '''
    # Synthesis network
    '''
    # Conversion to 3-channel color
    '''
    return tf.keras.models.Model(inputs=[z, conditioning, fade, constant], outputs=x, name='generator')

def cond_progan(out_size, fade_outputs, z_dim, num_classes, conv_init, batch_size, cond_start=False,
                start_size=8, start_filters=512, n_channels=3):
    '''
    # Synthesis network
    '''
    # Conversion to 3-channel color
    '''
    return tf.keras.models.Model(inputs=[z, conditioning, fade], outputs=x, name='generator')

def encoder_basic(in_size, z_dim, conv_init, max_filters=512):
    '''
    return tf.keras.models.Model(inputs=[img], outputs=[z], name='encoder')

def encoder_residual(in_size, z_dim, conv_init, max_filters=512):
    '''
    return tf.keras.models.Model(inputs=[img], outputs=[z], name='encoder')

def decoder_basic(out_size, z_dim, conv_init, max_filters=512):
    '''
    return tf.keras.models.Model(inputs=[z], outputs=[x], name='decoder')

def discriminator_basic(out_size, conv_init, max_filters=512):
    '''
    # Convert to single value
    '''
    return tf.keras.models.Model(inputs=[img], outputs=[disc_out], name='discriminator')

def discriminator_residual(out_size, conv_init, max_filters=512):
    '''
    # Convert to single value
    '''
    return tf.keras.models.Model(inputs=[img], outputs=[disc_out], name='discriminator')

def discriminator_progressive(out_size, fade_inputs, batch_size, n_channels, num_classes, conv_init,
                              max_filters=512):
    # Fading function
    '''
    # Convert from RGB
    '''
    # Convert to single value
    '''
    # Label prediction
    '''
    # Discriminator score
    '''
    return tf.keras.models.Model(inputs=[img, conditioning, fade], outputs=[disc_out, labels_out],
name='discriminator')
```

ТЕСТУВАННЯ СИСТЕМИ

а) б) в) г)

Типи тестових зображень:
а – нормальне зображення,
б – нормальній шум сигма = 20,
в – нормальній шум сигма = 50,
г – нормальній шум сигма = 100

Порівняння працездатності при додаванні шуму

Тип	Вірних	Майже вірних	Не знайдених	Помилкових
Оригінал	182	20	6	2
	86,7%	9,5%	2,8%	0,95%
З шумом 20	178	20	12	0
	84,8%	9,5%	5,7%	0%
З шумом 50	177	8	25	0
	84,2%	3,8%	11,9%	0%
З шумом 100	122	3	85	0

ВИСНОВКИ

В результаті виконання роботи виконано дослідження та програмна розробка ШНМ для знаходження місця розташування автомобільних номерів на зображенні. Вирішено наступні завдання:

- створення програмного модуля завантаження та обробки зображення з відокремленням робочою області зображення;
- вибір типу ШНМ;
- структурна ідентифікація топології ШНМ;
- навчання ШНМ;
- оцінка якості функціонування системи розпізнавання місця розташування автомобільних номерів з проведенням експериментів для тестової вибірки різнорідних зображень.

Про ці дослідження було зроблено доповідь на міжнародній конференції «Майбутній науковець 2020»