Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ІДЕНТИФІКАЦІЇ ПОРІД КОТІВ ЗА ЇХ ЗОБРАЖЕННЯМ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-18-1
Мануйлова Н.Я.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Путятін Є.П.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Мануйловій Наталії Янівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Розробка та дослідження методу ідентифікації порід котів за їх зображенням*

затверджена наказом по університету від *«21» жовтня 2019 року № 1479Ст.*

2. Термін подання студентом роботи до екзаменаційної комісії *23 листопада 2019 р.*

3. Вихідні дані до роботи: *Програмна реалізація методу класифікації породи котячих за їх зображенням; теоретичні відомості про методи класифікації зображень*

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд методів класифікації зображень.
2. Огляд методів побудови нейронних мереж.
3. Модель класифікації породи кота.
4. Прогнозування породи кота за його зображенням.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *актуальність обраної теми, постановка задачі, статистична класифікація, машинне навчання з учителем, штучні нейронні мережі, згорткові нейронні мережі, програмна реалізація, схема згорткової нейронної мережі, демонстрація результатів роботи, висновки*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	21.10.2019	**Виконано**
2	Аналіз завдання, підбір літератури	22.10.19-24.10.19	**Виконано**
3	Аналіз літератури з досліджуваної проблеми	24.10.19-25.10.19	**Виконано**
4	Аналіз технічних засобів	25.10.19-26.10.19	**Виконано**
5	Розробка методу	26.10.19-12.11.19	**Виконано**
6	Програмна реалізація	12.11.19-18.11.19	**Виконано**
7	Оформлення пояснювальної записки	18.11.19-21.11.19	**Виконано**
8	Перевірка на плагіат	21.11.19	**Виконано**
9	Рецензування	21.11.19	**Виконано**
10	Підготовка презентації та доповіді	21-22.11.19	**Виконано**
11	Занесення роботи в електронний архів	23.11.19	**Виконано**
12	Попередній захист атестаційної роботи	23.11.19	**Виконано**

Дата видачі завдання «21» жовтня 2019 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: 67 с., 38 рис., 40 джерел.

КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, МАШИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ЛІНІЙНА КЛАСИФІКАЦІЯ, КРОС-ЕНТРОПІЙНА ФУНКЦІЯ ВТРАТ, МЕТОД СТОХАСТИЧНОГО ГРАДІЄНТА.

Метою даної атестаційної роботи є розробка та написання методу класифікації породи котів за їх зображенням.

Об’єктом дослідження є нейронні мережі та засоби побудови їх архітектури.

У процесі виконання роботи було розроблено архітектуру нейронної мережі на основі згорткової для її тренування, створення моделі та використання для класифікації породи котів. Було досліджено основні засоби розробки нейронної мережі, її навчання, підрахунку функції втрат, оптимізації моделі, зберігання, її валідації та застосування. Натреновану модель було протестовано на тестовому наборі даних; найкраща модель з точки зору валідації має точність 35 відсотків.

Результатом роботи є програмний застосунок для роботи з моделлю, а саме, її тренування, перевірку та застосування.

IMAGE CLASSIFICATION, MACHINE LEARNING, CONVOLUTIONAL NEURAL NETWORKS, LINEAR CLASSIFIER, CROSS-ENTROPIC LOSS FUNCTION, STOCHASTIC GRADIENT METHOD.

The purpose of this work is to develop and write a method for classifying cat breeds by their image.

The object of research is neural networks and the means of developing their architecture.

In the course of the work, a neural network architecture was developed based on convolutional training, model creation and use for cat breed classification. The basic means of neural network development, its training, loss function calculation, model optimization, storage, its validation and application were investigated. The trained model was tested on a test dataset; the best model in terms of validation has an accuracy of 35 percent.

The result of the work is an application to work with the model, namely, its training, validation and application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ANN – Artificial Neural Networks (штучні нейронні мережі)

SVM – Support-Vector Machines (метод опорних векторів)

k-NN – k-Nearest Neighbors Algorithm (алгоритм k-найближчих сусідів)

CNN, ConvNet – Convolutional Neural Network (згорткові нейронні мережі)

RGB – Red, Green, Blue (червоний, зелений, синій)

JPEG – Joint Photographic Experts Group (формат зображення, названий на честь компанії-розробника)

FC – Fully Connected (повністю з'єднанний)

HSV – Hue, Saturation, Value (тон, насиченість, значення)

CMYK – Cyan, Magenta, Yellow, Key/Black (блакитний, пурпуровий, жовтий, ключовий/чорний)

ReLU – Rectified Linear Unit (випрямлена лінійна одиниця)

SGD – Stochastic Gradient Descent (метод стохастичного градієнта)

IDE – Integrated Development Environment (інтегроване середовище розробки)

PNG – Portable Network Graphics (растровий формат зберігання графічної інформації)

CPU – Central Processor Unit (центральний процесор)

GPU – Graphics Processor Unit (графічний процесор)

ВСТУП

Задача сегментації зображень, класифікації знайдених на них об’єктів досить актуальна на сьогоднішній день. Подібні дослідження, певно, будуть корисні у будь-якій сфері життя людини.

Хоч вперше про розпізнавання образів заговорили ще на початку XX століття, а з 70-х років ця галузь науки переживає бурхливий розвиток, основною причиною якого є поява більш досконалих екземплярів обчислювальної техніки, задача повноцінного сприйняття машиною навколишнього світу через комп’ютерний зір нарівні з людиною до сих пір не розв’язана. Для штучного інтелекту на сьогоднішній день навіть виявити, яка тварина знаходиться на зображенні – процес важкий, потребуючий зусиль зі сторони інженерів, які його розроблюють (і навіть тоді, як правило, говорять не про гарантований результат, а скоріше про якийсь вагомий процент коректних відповідей, або хоча б таких, що мають сенс), тоді як для 3-річної дитини питання вивчення того, як виглядає навіть незнайома раніше річ, не передбачає ніяких складностей. Із урахуванням важкості такої проблеми, основні дослідження на даний момент зосереджені на розв’язанні конкретних задач, таких як розпізнавання зображень, з чим і пов’язана дана робота.

Задача класифікації полягає у розділенні певної множини об’єктів на класи за допомогою деяких методів. Існує вибірка, що містить ті об’єкти, класова належність яких відома. Інші об’єкти не мають інформації щодо того, до яких класів вони належать. Мета – розробити алгоритм, що класифікуватиме довільний об’єкт з вихідної множини.

Так, у нашому випадку порода кота – це клас, а зображення – об’єкт, що потрібно класифікувати належним чином.

Навчання з викладачем – задача для машинного навчання, що передбачає так званого вчителя, який «тренує» систему за допомогою набору даних, де пара «вхідні дані-вихідні дані» заздалегідь визначена. Алгоритм навчання з викладачем аналізує отримані правильні варіанти класифікації та

виробляє певну функцію, яку можна буде застосовувати для подальшого тренування за допомогою пар «ключ-значення». Оптимальний сценарій дозволить цьому алгоритму правильно визначати клас екземпляру, який до того не був використаний для корекції роботи програми. Кажучи інакше, вимога така, що алгоритм, який ми навчаємо, повинен перейти від прикладу з відомою відповіддю до задач, рішення яких він буде здатен знаходити самостійно так, що його власна відгадка буде мати сенс для людини.

Штучні нейронні мережі (artificial neural networks, ANN) – системи поєднання у комп'ютерних системах, прототипом яких можна вважати біологічні нейронні мережі, що є частиною мозку тварин. Такі системи «навчаються» виконувати завдання, розглядаючи приклади, як правило, не запрограмовані на конкретні задачі. Наприклад, у комп'ютерному зорі вони можуть вчитися розпізнавати об'єкти на зображенні та визначати, чи є там кіт або собака. Вони роблять це без початкових знань про те, що у таких тварин є шерсть, хвости, вуса, специфічні для котів або собак обличчя. Замість цього, вони автоматично визначають характеристики з прикладів, які такі мережі обробляють.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

1.1 Загальне поняття нейронних мереж

Штучні нейронні мережі (artificial neural networks, ANN) – системи поєднання у комп'ютерних системах, прототипом яких можна вважати біологічні нейронні мережі, що є частиною мозку тварин. Такі системи «навчаються» виконувати завдання, розглядаючи приклади, як правило, не запрограмовані на конкретні задачі. Вони автоматично визначають характеристики з прикладів, які такі мережі обробляють [1].

Перед тим, як розглядати мережу у цілому, доцільно взяти до уваги один нейрон, базову одиницю вищезгаданої структури. Нейрон приймає на вхід якісь дані, опрацьовує їх за допомогою математичних перетворень та створює певне вихідне значення [23]. (рис. 1.1)

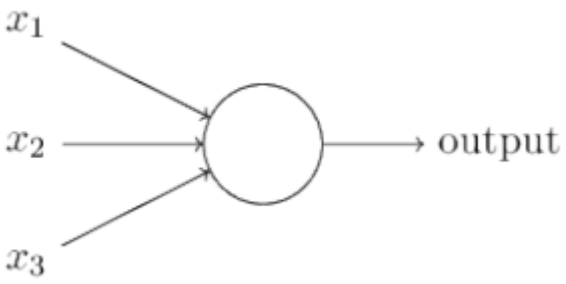

Рисунок 1.1 – Схема одного з найпростіших типів нейрону – персептрон

У наведеній схемі нейрону існує три вхідних параметра – x_1, x_2 та x_3. Взагалі їх може бути скільки завгодно. Вчений Франк Роузенблатт, який, власне, винайшов концепцію персептрону, пропонував декілька простих правил для того, щоб порахувати вихідне значення. Він представив так звані ваги w_1, w_2, …, що є дійсними числами, які математично відображають важливість кожної з вхідних характеристик [38]. «Вихід» нейрону, 0 або 1, визначається на основі того, чи є сума

$$sum = \sum_{j=1}^{n} w_j x_j ,$$

де n – кількість вхідних параметрів, більшою або меншою, чим певний поріг. Так як і ваги, поріг є дійсним числом – параметром нейрону [7]. Інакше кажучи, вихід нейрону можна порахувати наступним чином:

$$output = \begin{cases} 0 \text{ if } \sum_{j=1}^{n} w_j x_j \leq threshold, \\ 1 \text{ if } \sum_{j=1}^{n} w_j x_j > threshold. \end{cases}$$

Це – базова математична модель. Вираз, який дозволяє визначити значення вихідного параметру, ще називають функцією активації. Окрім жорсткої порогової функції, що була продемонстрована вище, існують наступні активатори:

– лінійний поріг, гістерезис – нескладна кусочно-лінійна функція; має дві лінійних ділянки, де функція активації тотожно дорівнює мінімально та максимально допустимим значенням; є також ділянка, де вона строго монотонно зростає;

– сигмоідальна функція, сигмоід – монотонно зростаюча здатна до диференціювання S-подібна нелінійна функція з насиченням; дозволяє посилювати слабкі сигнали та не перенасичуватися від сильних; прикладом можуть служити логістична функція або гіперболічний тангенс;

– гаусова крива – застосовується у випадках, коли реакція нейрону повинна бути максимальною для деякого певного набору вхідних значень [26].

Графіки цих функцій наведені на рисунку 1.2.

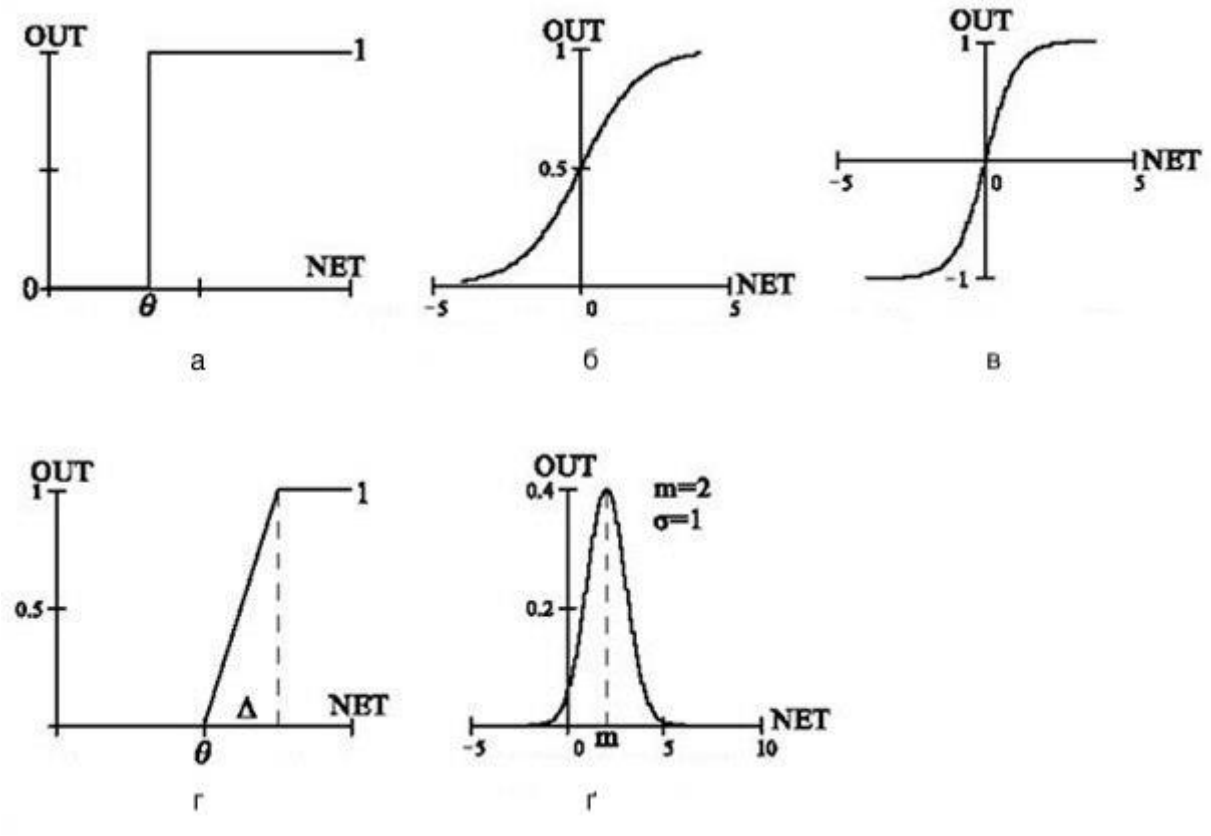

Рисунок 1.2 – Типи функцій активації нейрону: а – жорстка порогова функція, б – сигмоід (логістична функція), в – гіперболічний тангенс, г – лінійний поріг, гістерезис, ґ – гаусова крива

Таким чином, нейронна мережа – не більше, ніж поєднані між собою нейрони. Нейрони представляють собою примітивний прилад для прийняття рішення на основі певного алгоритму, тоді як мережа у цілому імітує модель людського сприйняття кроків рішення поставленої задачі [2].

На рисунку 1.3 зображена модель нейронної мережі. Виділяють шар (layer) вхідних даних, прихований шар, який по факту є будь-яким шаром між першим та останнім шарами (до речі, їх може бути більш одного), та вихідний шар з одним нейроном. Усі нейрони, усі шари пов’язані між собою, вхідні дані одного нейрону є вихідними іншого.

Саме це дає нам можливість називати цю структуру мережею.

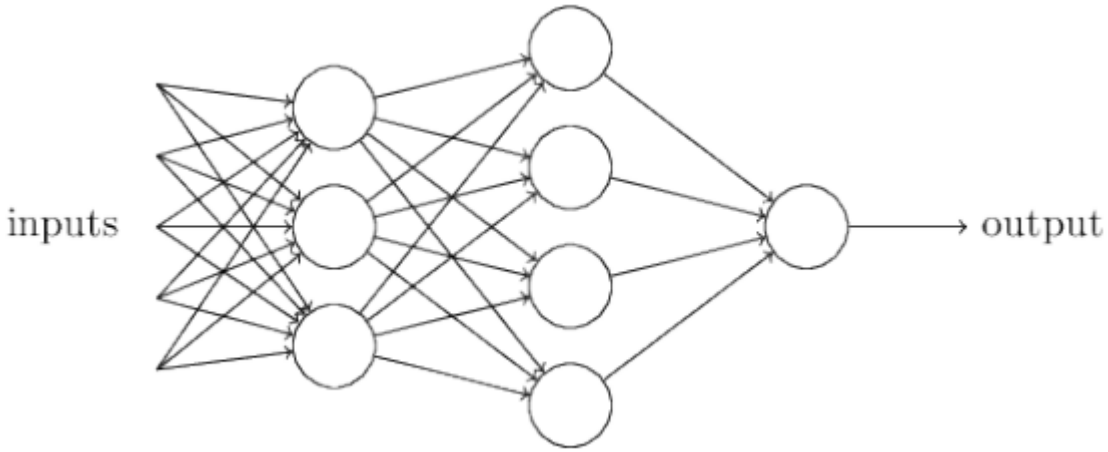

Рисунок 1.3 – Модель нейронної мережі

1.2 Статистична класифікація

1.2.1 Загальне поняття класифікації

Розглянемо таку проблему: припустимо, ви лікар. До вас щойно потрапив пацієнт із серцевим нападом. Як ви думаєте, чи стане він жертвою ще одного такого через, скажемо, шість місяців? Або, припустимо, ви – банк, і є особа, яка хоче взяти кредит для малого бізнесу. Погасить вона заборгованість або ні? А як щодо цього: яким чином сервер електронної пошти може визначити, які листи – спам, а які – реальні?

Інтуїтивно можна прийти до висновку, що всі перераховані вище ситуації реально вирішити, з’ясувавши відповідні емпіричні дані та визначивши, які фактори важливі в кожній. Наприклад, у випадку серцевого нападу, можливо, захочеться переглянути записи про госпіталізацію пацієнтів, у яких протягом шести місяців він стався двічі, і порівняти, чи нагадує ваш пацієнт таких за віком, демографією, звичками харчування, фізичною активністю та іншими клінічними вимірюваннями; це, можливо, дозволило би надати рекомендації щодо корекції способу життя задля запобігання погіршень у майбутньому. Щодо кредитів, можна оцінити майбутніх позичальників за їхнім кредитним рейтингом, доходом, кількістю наявних у

них кредитних карток та прийняти рішення на основі попередніх клієнтів із подібними профілями.

Вищенаведені ситуації є прикладами задач класифікації. Класифікація – статистичний метод, що використовується для побудови предикативних моделей для розділення та класифікації нових даних. Розглянемо приклад фільтрації спаму більш докладно. Є певний набір об'єктів, які ми бажаємо відокремити один від одного, – це небажана пошта, повідомлення від рекламодавців або хакерів, які заповнюють вашу поштову скриньку і завдають головний біль, і, звісно, справжні електронні листи. Далі, припустимо, що у нас вже існує колекція електронних листів, скажімо, їх N. Деякі з них непотрібні, а деякі справжні, і, використовуючи таку вибірку, ми хочемо визначити засоби фільтрації спаму. Цю колекцію електронних листів ми називаємо навчальним набором (training set), який, як правило, є набором все класифікованих раніше даних, які ми використовуємо для побудови власної класифікаційної моделі [10].

Визначимо тренувальні дані як певну множину $X = \{x_1, x_2, \ldots, x_N\}$. Тепер, за допомогою них ми повинні визначити, які змінні, що називають властивостями, ми хочемо виміряти, щоб вирішити, чи буде майбутній лист спамом або ні. Властивості можуть бути дискретними або неперервними. Так, у нашому випадку приклад дискретної властивості – маркер, що містить інформацію про те, чи містить адреса відправника частку «.edu» або ні, а приклад неперервної – процент листів, що мають певний набір слів або букв (таких, як «безкоштовно», «ваша», «!» і таке інше). Якщо деяка властивість m виміряна, кожен лист містить вектор $m \times 1$ x_i, де $i \in \{1, \ldots, N\}$ даних, і складається простір R^m, так званий простір властивостей. Тренувальні дані – матриця розмірністю $N \times m$ елементів, де входження x_{ij} представляє j-ту властивість i-го листа.

В кінці кінців, використовуючи тренувальну вибірку X, буде створена функція рішення D(x) – функція, що буде здатна взяти новий об'єкт та

створити передбачення стосовно його належності до визначеного класу (її також називають класифікатором).

Слід зазначити, що у термінології машинного навчання задача класифікації вирішується тренуванням з учителем. Відповідне тренування без учителя відноситься до кластеризації [35].

І класифікація, і кластеризація є прикладами більш загальної проблеми розпізнавання шаблонів, яка полягає у присвоєнні якогось вихідного значення заданому вхідному значенню [36]. Окрім цих, існують ще такі задачі:

– регресія, яка зіставляє розраховані заздалегідь результати кожному набору вхідних параметрів;

– маркування послідовності, що присвоює клас кожному члену послідовності значень (наприклад, тегування частин мови, призначення частини мови до кожного слова у вхідному реченні);

– розділ, який присвоює дереву розділу вхідне речення, описуючи синтаксичну структуру речення.

Поширеним підкласом класифікації є імовірнісна класифікація. Такі алгоритми використовують статистичні висновки, щоб знайти найкращий клас для даного екземпляра. На відміну від інших алгоритмів, які просто виводять клас як «найкращий», ймовірнісні алгоритми розраховують також ступінь того, наскільки екземпляр належить до кожного з можливих класів. Найкращий клас зазвичай вибирається як клас з найбільшою ймовірністю [4]. Такий алгоритм має численні переваги перед неімовірнісними класифікаторами:

– він може вивести значення довіри, пов’язане з його вибором (загалом, класифікатор, який може це зробити, відомий як такий, що зважений на довіру);

– він може зважати і на те, коли впевненість у виборі будь-якого конкретного результату буде занадто низькою, і не обрати ніякого класу взагалі;

– через наявні ймовірності такі класифікатори можуть бути застосовані у задачах машинного навчання більш ефективно; як наслідок, вони частково або повністю уникають проблем розповсюдження помилки [22].

1.2.2 Алгоритми класифікації

Так як не існує універсальної форми класифікації, що була би придатною для будь-яких даних, було розроблено дуже великий набір алгоритмів. Найчастіше використовують наступні алгоритми:

1. Лінійні класифікатори базуються на значенні лінійної комбінації характеристик:

– лінійний дискримінант Фішера знаходить лінійну комбінацію властивостей, що характеризують або розділяють два або більше класи, випадки; результат комбінації застосовують для лінійної класифікації або, частіше, для зменшення розмірності перед наступною класифікацією;

– логістична регресія використовується для моделювання ймовірності існування певного класу; для досліджуваного об’єкта розраховується ймовірність належати до кожного з класів від діапазону від 0 до 1;

– наївні класифікатори Байєса є сімейством простих імовірнісних класифікаторів, заснованих на застосуванні теореми Байєса з сильними (наївними) припущеннями незалежності між ознаками;

– персептрон – алгоритм для навчання з учителем для бінарних класифікаторів.

2. Метод опорних векторів (support-vector machine, SVM) будує модель, що присвоює нові приклади до тієї чи іншої з категорії, роблячи її неімовірнісним бінарним лінійним класифікатором; по суті, це алгоритми контрольованого навчання, що аналізують дані, які використовуються для класифікації та регресійного аналізу. Підвидом є метод опорних векторів з найменшими квадратами; відрізняється під класичної моделі тим, що можна

знайти рішення, розв’язавши набір лінійних рівнянь замість необхідності вирішення задачі опуклого квадратичного програмування для класичних SVM.

3. Квадратичний класифікатор використовується для розділення вимірювань двох або більше класів об’єктів квадратичною поверхнею; це більш загальна версія лінійного класифікатора.

4. Оцінка ядра – це форма оцінки щільності ядра, в якій розмір ядер, використовуваних в оцінці, змінюється залежно від місця розташування зразків або місця тестової точки; це особливо ефективна методика у випадку, коли пробний простір є багатовимірним. Окремо слід вирізнити алгоритм k-найближчих сусідів (k-nearest neighbors algorithm, k-NN), непараметричний метод, що використовують для класифікації та регресії.

5. Бустинг – мета-алгоритм, направлений на зменшення відхилення у навчанні з учителем; також сімейство алгоритмів машинного навчання, що перетворюють декілька «слабких» учнів на одного «сильного». «Слабкий» учень – класифікатор, що мало корелює зі справжньою класифікацією (мало відрізняється від випадкового вгадування). «Сильний» – навпаки, має добру кореляцію з контрольними результатами.

6. Дерева рішень є прогностичними моделями, що, виходячи зі спостережень про об’єкт (представлений у гілках) до висновків стосовно цільового значення (листя дерева). Вирізняють так звані випадкові ліси або ліси з випадковим рішенням, коли будують безліч дерев рішень під час навчання.

7. Нейронні мережі (artificial neural networks, ANN) – комп’ютерні системи, що імітують роботу біологічних нейронних мереж, що є частиною мозку тварин. Базується на колекції поєднаних між собою вузлів, що іменують нейронами [19].

1.3 Актуальність задачі

Насправді, рішення такої конкретної прикладної задачі могло б знадобитися власникам котів, або тим, хто цікавиться в цілях навчання, або ж спеціалізованим притулкам для створення додаткової реклами безпритульної тварини (куди приємніше усвідомлювати, що потенційний вихованець на 70 відсотків – персидський кіт і на 30 – мейн-кун, ніж звичайний надпис «безпородний»).

Але, навіть без цього задача сегментації зображень у цілому, класифікації знайдених на них об’єктів досить актуальна на сьогоднішній день – важко уявити собі галузь, де подібні дослідження не були б корисними.

Більш того, хоч вперше про розпізнавання образів заговорили ще на початку XX століття, а з 70-х років ця галузь науки переживає бурхливий розвиток, основною причиною якого є поява більш досконалих екземплярів обчислювальної техніки, задача повноцінного сприйняття машиною навколишнього світу через комп’ютерний зір нарівні з людиною до сих пір не розв’язана [40].

Для штучного інтелекту на сьогоднішній день навіть виявити, яка тварина знаходиться на зображенні – процес важкий, потребуючий зусиль зі сторони інженерів, які його розроблюють (і навіть тоді, як правило, говорять не про гарантований результат, а скоріше про якийсь вагомий процент коректних відповідей, або хоча б таких, що мають сенс), тоді як для 3-річної дитини питання вивчення того, як виглядає навіть незнайома раніше річ, не передбачає ніяких складностей.

Із урахуванням важкості такої проблеми, основні дослідження на даний момент зосереджені на розв’язанні конкретних задач, таких як розпізнавання зображень, з чим і пов’язана дана робота.

1.4 Сфери застосування

Взагалі, нейронні мережі здатні вирішувати наступний спектр задач:

– наближення функцій, або регресійний аналіз, включно з передбаченням часових рядів, наближенням пристосованості та моделюванням;

– класифікація, включно з розпізнаванням образів та послідовностей, виявленням нововведень та послідовним ухвалюванням рішень;

– обробка даних, включно з фільтруванням, кластеризацією, сліпим відокремлюванням сигналу та стисненням [33].

– робототехніка, включно зі скеровуванням маніпуляторів та протезів;

– автоматичне керування, включно з числовим програмним керуванням [17].

Завдяки своїй особливості реалізовувати та моделювати нелінійні процеси штучні нейронні мережі знайшли застосування у дуже великій кількості сфер людського життя, такими як:

– керування транспортними засобами, передбачення траєкторії, автоматизація виробничих процесів, природокористування;

– квантова хімія;

– гідрологія, моделювання океану, прибережна інженерія, геоморфологія;

– теорія ігор, прийняття рішень;

– радарні системи, розпізнавання обличчя, класифікація сигналів, ідентифікація об’єктів;

– розпізнавання жестів, мовлення, рукописного тексту [27];

– медична діагностика (наприклад, діагностика раку, де ці технології використовувались задля відрізнення ліній ракових клітин, схильних до метастазування);

– фінанси;

– добування даних та їх обробка;

– машинний переклад;

– соціально-мережеве фільтрування.

Вищезазначені сфери – не кінцевий список, де можливо якось застосувати мережі [5].

1.5 Постановка задачі дослідження

Практична задача – розробка методу класифікації породи котячих за їх зображенням.

Вхідні дані – зображення у форматі JPEG, що містить кота, без зайвих об’єктів, бажано з однорідним фоном та твариною у повний ріст. Вихідні дані – порода, ймовірність того, що такий висновок є істиною.

Для досягнення цієї мети поставлені наступні завдання:

– збір теоретичних відомостей щодо питань, які досліджуються в роботі;

– реалізація програмного застосунку, який класифікуватиме зображення та видаватиме результат, що відповідає вимогам до вихідних даних;

– оцінка якості роботи програмного застосунку;

– оцінка продуктивності роботи застосунку в плані часу, що потребується для виконання задачі, та потужностей, що необхідні для комфортного користування застосунком;

– підведення підсумків дослідження.

2 ОПИС МЕТОДУ ІДЕНТИФІКАЦІЇ ПОРОДИ КОТА

2.1 Машинне навчання з учителем

Поняття машинного навчання дуже тісно пов’язане із задачею класифікації. Фактично, воно є частиною будь-якого класифікатора.

Так, машинне навчання із тренером – процес тренування функції з метою змусити її видавати очікувані результати, базуючись на попередньо вивчених прикладах коректних пар «вхідні параметри – вихідне значення». Функція створюється за допомогою помічених тренувальних даних, що складаються з набору типових об’єктів з уже відомим результатом. Кожен зразок є парою, що складається з вхідного об’єкту (звичайно вектору) та бажаної відповіді (іноді її ще називають наглядовим сигналом). Алгоритм контрольованого машинного навчання аналізує отримані через тренера дані та виробляє функцію, що може бути використана для перевірки нових прикладів. В кінці кінців, добре навчений алгоритм буде здатен правильно визначати назву класу для об’єктів, яких він ще ні разу не приймав до обробки. Вимога така, щоб результат навчання узагальнив отриману під час тренування інформацію та міг сприймати неочікувані ситуації так, щоб це мало сенс з точки зору людини [3].

2.1.1 Загальний алгоритм машинного навчання з учителем

Уявимо, що у нас є певна задача, яку ми плануємо вирішити за рахунок натренованої системи. Щоб реалізувати її, зробимо наступне:

1. Визначимося з початковими даними. Перед тим, як щось робити, дуже важливо зазначити, у якому саме вигляді будуть подані об’єкти. Наприклад, у випадку з розпізнаванням тої ж породи кота це повинне бути його зображення, можливо, з певним фоном або позою тварини тощо.

2. Сформуємо тренувальний набір. Він повинен представляти собою імітацію використання алгоритму у реальних умовах. Згідно з вимогами, що були висловлені на попередньому етапі, збирається набір вхідних об’єктів, створюються відповідні їм правильні відповіді.

3. Обумовимо формат вхідних параметрів функції, що навчається. Точність відповіді сильно залежить від того, як представлений вхідний об’єкт. Зазвичай він перетворюється у вектор властивостей, який містить низку ознак, які є описовими для об’єкта. Кількість особливостей не повинна бути занадто великою через так зване «прокляття розмірності», але має містити достатньо інформації, щоб якомога точніше передбачити вихідне значення.

4. Виберемо структуру засвоєної функції та відповідний алгоритм навчання. Наприклад, інженер може обрати метод опорних векторів або дерева рішень.

5. Завершимо розробку функції. Запускаємо алгоритм навчання на зібраному раніше наборі тренувальних даних. Деякі з них вимагають від користувача деякі параметри управління. Їх можна відрегулювати завдяки оптимізації продуктивності на підмножині (набір валідації) навчальної збірки, або за допомогою перехресної перевірки (крос-валідація).

6. Після навчання та коригування параметрів, коли все зроблено, залишається лише оцінити точність передбачень виробленої функції. Вимірювання підсумкової продуктивності слід робити не на тренувальному наборі, а на окремо зібраних тестових даних [6].

2.1.2 Особливості вибору алгоритму навчання з тренером

Існує багатий вибір алгоритмів навчання з учителем, кожен зі своїми сильними та слабкими сторонами. Нажаль, на сьогоднішній день не існує універсального рішення, який працював би гарно у всіх задачах навчання з тренером [8].

На перешкоді стають розповсюджені проблеми контрольованого навчання.

Першою з таких є проблема компромісу зсуву та дисперсії. В ідеалі потрібно обирати модель, яка і закономірності в своїх тренувальних даних схоплює добре, і в той же час гарно узагальнює їх для ще не бачених даних. На жаль, на практиці неможливо приділяти увагу цим двом аспектам одночасно. Методи навчання з високою дисперсією можуть бути здатними добре представляти свої тренувальні набори, але перебувають під загрозою перенавчання зашумлених або нехарактерних тренувальних даних. На противагу їм, алгоритми з великим зсувом, зазвичай видають простіші моделі, не схильні до перенавчання, але можуть погано підлаштуватися до своїх тренувальних даних, виявляючись нездатними схопити важливі закономірності.

Ми вимушені шукати компроміс між зсувом та дисперсією. Ключовим аспектом багатьох контрольованих методів навчання є те, що вони здатні регулювати цей компроміс між зміщенням та дисперсією (автоматично або шляхом надання відповідних параметрів, які користувач може коригувати) [16].

Другою складністю є кількість даних для тренування, доступних для «справжньої» функції. Якщо вона проста, то «негнучкий» алгоритм навчання з великим зсувом і низькою дисперсією зможе навчатися на невеликій вибірці даних. Але якщо справжня функція є дуже складною (наприклад, тому, що вона включає складні взаємодії між багатьма різними вхідними властивостями і поводиться по-різному в різних частинах вхідного простору), то ця функція буде здатна тренуватися лише на дуже великій кількості навчальних даних, і у випадку використання «гнучких» алгоритмів навчання з низьким ухилом і великою дисперсією.

Третє питання – розмірність вхідного простору. Якщо вектори вхідних функцій мають дуже високу розмірність, задача навчання може бути важкою, навіть якщо функція залежить лише від невеликої кількості цих властивостей.

Це тому, що безліч «зайвих» вимірів може заплутати алгоритм навчання і викликати велику дисперсію. Отже, велика розмірність вводу зазвичай вимагає налаштування класифікатора на низьку дисперсію та велике зміщення. На практиці, якщо інженер може вручну видалити зайві властивості з вхідних даних, це, ймовірно, підвищить точність засвоєної функції. Крім того, існує багато алгоритмів вибору функцій, які прагнуть визначити функції, що підходять, та відкинути невідповідні. Це приклад більш загальної стратегії зменшення розмірності, яка прагне зіставити вхідні дані в меншу розмірність до запуску керованого алгоритму навчання.

Четверта проблема полягає у ступені шуму в бажаних вихідних значеннях (контрольні цільові змінні). Якщо бажані вихідні дані часто неправильні (через помилки людини або сенсора), алгоритм навчання не повинен намагатися знайти функцію, яка точно відповідає навчальним прикладам. Намагання надто ретельно підходити до даних призводить до перенавчання. Ви можете дійти до цього навіть тоді, коли немає помилок вимірювання (стохастичний шум), якщо функція, яку ви намагаєтеся навчити, занадто складна для вашої моделі навчання. У такій ситуації частина цільової функції, яку неможливо моделювати, «пошкоджує» ваші навчальні дані. Це явище отримало назву детермінованого шуму. Якщо присутній будь-який тип шуму, краще перейти на більш високий зсув, понизивши дисперсію.

На практиці існує декілька підходів до зменшення шуму у вихідних значеннях, таких як раннє припинення для запобігання перенапруження, а також виявлення та вилучення шумних прикладів навчання перед навчанням алгоритму керованого навчання.

Існує кілька алгоритмів, які визначають шумні дані і видаляють приклади, що визивають підозру у шумі, і це зменшує загальну похибку зі статистичною значимістю.

Інші фактори, які слід враховувати при виборі та застосуванні алгоритму навчання:

– гетерогенність даних – якщо вектори властивостей включають до себе надто багато різних типів (дискретні, дискретні впорядковані, неперервні значення), деякі алгоритми застосовуються легше, ніж інші; багато алгоритмів (SVM, k-NN, лінійна та логістична регресії тощо) вимагають, щоб вхідні дані були обов'язково числовими та мали однаковий діапазон (наприклад, інтервал [-1;1]); методи, що застосовують функцію дистанції (k-NN, SVM з ядрами Гауса), особливо чутливі до цього; проте, дерева рішень, наприклад, легко пораються з обробкою неоднорідних даних;

– надмірність даних – якщо вхідні функції містять зайву інформацію (наприклад, дві властивості з високим рівнем кореляції), деякі алгоритми навчання (наприклад, лінійна та логістична регресії, методи, засновані на відстані) не будуть добре працювати через числові нестабільності; накладаючи певну форму регуляризації, ці проблеми часто можна вирішити;

– наявність зв'язків та нелінійності – якщо кожна з особливостей вносить незалежний внесок у визначення результату, то алгоритми, що засновані на лінійних функціях (лінійна та логістична регресії, SVM, наївний Байєс) та функції відстані (k-NN, SVM з ядром Гауса), як правило, працюють без проблем; однак, якщо серед властивостей існують складні зв'язки, то дерева рішень і нейронні мережі працюють краще, оскільки вони спеціально винайдені для такого роду задач; лінійні методи допустимо використовувати, але інженеру доведеться власноруч встановлювати усі взаємодії [18].

Під час розробки нової програми інженер може порівняти декілька алгоритмів навчання та експериментально визначити, який з них найкраще вирішує проблему. Підвищення продуктивності алгоритму навчання може зайняти багато часу. Враховуючи обмежені ресурси, часто краще витратити більше часу на збір додаткових навчальних даних та більш інформативні функції, ніж витратити зайвий час на налаштування алгоритмів навчання [20].

2.2 Згорткові нейронні мережі

У нейронних мережах, згорткові нейронні мережі (convolutional neural network, CNN, іноді ConvNet) є одним з головних засобів для реалізації розпізнавання зображень, їх класифікації. Виявлення об’єктів, ідентифікація обличчя є прикладами задач, для вирішення яких такі мережі застосовуються душе широко. Потрібно зазначити, що швидкість попередньої обробки значно нижча, ніж у інших алгоритмів класифікації [28].

Як і інші нейронні мережі, архітектура згорткової мережі є аналогічною схемі зв’язку нейронів у людському мозку. Під час її розробки натхненням вченим служила зорова кора. У живих істот окремі нейрони реагують на подразники лише в обмеженій області зорового поля, відомої як рецептивне поле. Колекція таких полів перекривають один одного, щоб охопити усе поле зору.

Класифікатори зображень, засновані на CNN, опрацьовують їх та класифікують на певні категорії (наприклад, «кіт», «собака», «тигр», «лев»). Комп’ютер сприймає вхідне зображення як масив пікселів, розмір якого залежить від роздільної здатності зображення. Базуючись на цій характеристиці, маємо h x w x d (h – height, висота; w – width, ширина; d – dimension, роздільна здатність). Приклади можна спостерігати на рисунку 2.1 –RGB-зображення має представлення у вигляді масиву 6 x 6 x 3 (3 – розмірність), а матриця з представленням 4 x 4 x 1 – масив у градаціях сірого. Існує ряд таких кольорових просторів – градації сірого, RGB, HSV, CMYK тощо.

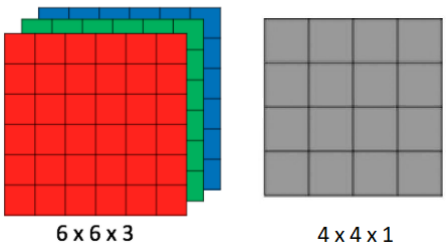

Рисунок 2.1 – Масиви матриць RGB та у градаціях сірого

Ви можете собі уявити, як важко буде обробити, скажімо, 8K-зображення (7680 × 4320). Роль ConvNet полягає в тому, щоб зменшити зображення у форму, яку легше обробляти, не втрачаючи тих особливостей, що є критичними для формування достатньо великої точності праці нейромережі. Це важливо у випадках, коли потрібно створити архітектуру, яка не тільки добре навчається, але також має здатність масштабуватися до масивних наборів даних.

Під час глибокого навчання моделей CNN для кожне вхідне зображення пройде через серію згорткових шарів (convolutional layers) з фільтрами (ядрами), шар поєднання (pooling layer), шар повного з'єднання (FC, fully connected layer) та функцію Softmax, що класифікуватиме об'єкт з імовірнісними значеннями у діапазоні [0; 1]. На рисунку 2.2 представлений повний процес роботи CNN для обробки вхідного зображення та класифікації об'єктів на основі властивостей [25].

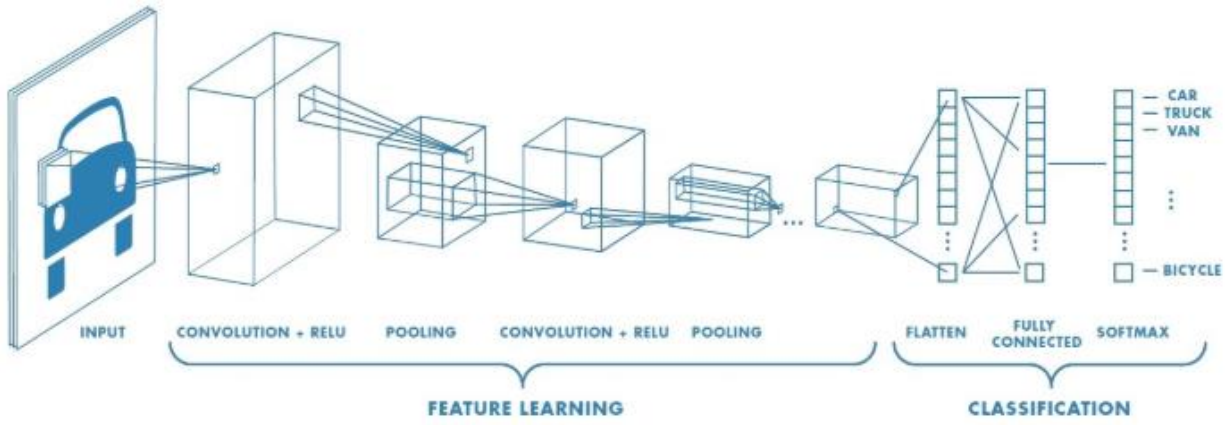

Рисунок 2.2 – Нейронна мережа з декількома згортковими шарами

Якщо говорити загалом, алгоритм згорткової нейронної мережі наступний:

– надати до згорткового шару вхідне зображення;

– обрати параметри, застосувати фільтри зі зсувами, додати відступи, якщо це необхідно;

– виконати операцію згортки на зображенні;
– застосувати функцію активації ReLU на вихідну матрицю;
– виконати об’єднання для того, щоб зменшити розмірність;
– додавати шари згортки, доки результат не задовільнить;
– вирівняти вихід та передати його до шару повного поєднання;
– вивести клас, використовуючи функцію активації [29].

2.2.1 Згортковий шар (convolutional layer)

Згортка – це перший шар, який вибирає властивості із вхідного зображення. Він зберігає зв’язок між пікселями, вивчаючи особливості зображення за допомогою невеликих квадратів вхідних даних. Це математична операція, яка має два параметри – матриця зображення і, власне, ядро.

Уявимо, що є деяке зображення 5 x 5 зі значеннями пікселів 0 та 1, та матриця-фільтр 3 x 3, як показано на рисунку 2.3.

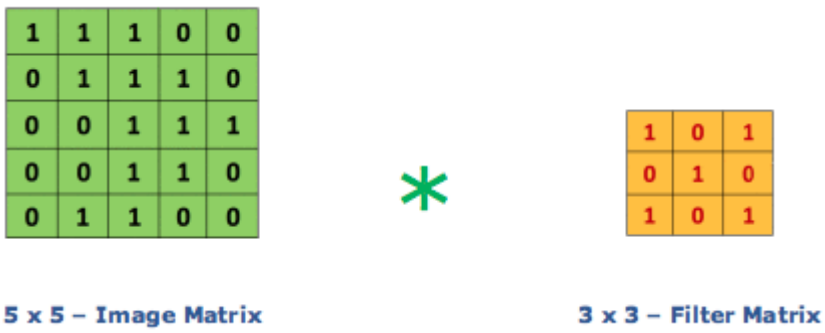

Рисунок 2.3 – Множення матриці зображення на ядро (матрицю-фільтр)

Виконується операція множення матриці зображення на ядро. Отримуємо матрицю 3 x 3, так звану таблицю властивостей (рис. 2.4).

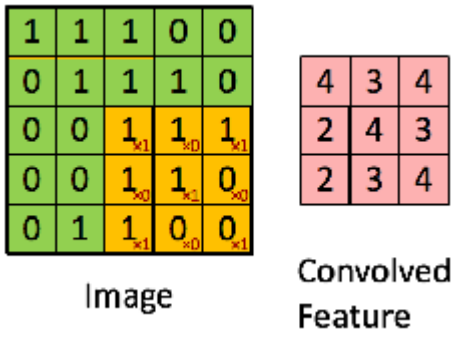

Рисунок 2.4 – Вихідна матриця властивостей

Згортка зображення з різними фільтрами може дати різний результат, такий, як визначення країв, розмиття або, навпаки, різкість [15].

Також для операції згортки важливий такий параметр, як величина кроку. Якщо вона дорівнює одиниці, значить, вікно буде рухатися на один піксель за ітерацію, якщо двійці – на два, і так далі. На рисунку 2.5 можна спостерігати, як рухатиметься ядро у разі, якщо величина кроку буде два.

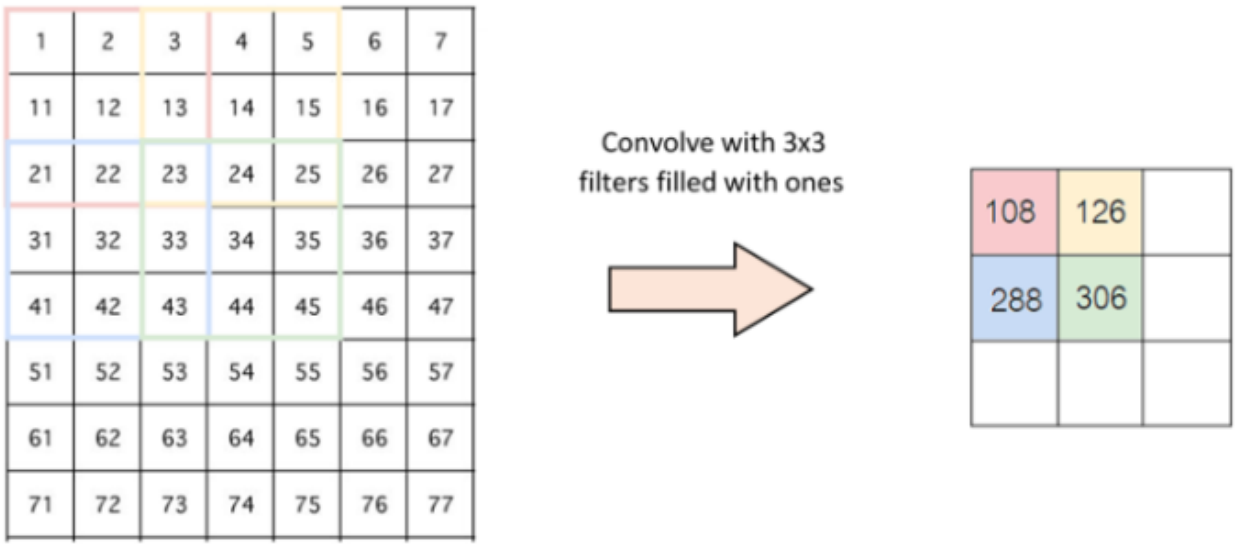

Рисунок 2.5 – Здвиг на два пікселі

Іноді фільтр не підходить ідеально для вхідного зображення. У такому разі ми можемо або заповнити нулями простір навколо зображення (zero-padding) або, навпаки, обрізати його там, де фільтр не може поміститися,

залишивши важливу частину зображення цілою (value padding). Підхід обирають в залежності від особливостей реалізації класифікатора.

2.2.2 Шар поєднання (pooling layer). Max pooling

Частина з шаром поєднання існує задля зменшення кількості параметрів у випадках, коли зображення надто великі. Просторове об’єднання також називають підвідбором (subsampling) або зменшенням кількості зразків (downsampling). Воно зменшує розмірність кожної таблиці, але залишає важливу інформацію, та може бути декількох типів:

- максимальне поєднання (max pooling);
- середнє поєднання (average pooling);
- сумарне поєднання (sum pooling) [32].

Частково це робиться для того, щоб допомогти максимально наблизитися до шуканих значень через більш абстраговану форму видіхних даних.

Це зменшує обчислювальні витрати за рахунок зменшення кількості параметрів для вивчення та забезпечує основну інваріантність перекладу для внутрішнього представлення.

Максимальне поєднання здійснюється за допомогою застосування максимального фільтра до поточного положення вікна вихідної матриці, що, зазвичай, не перекривається з наступними [21].

Маємо матрицю 4 х 4, що представляє початковий вхід. Скажімо, що є фільтр 2 х 2, який буде опрацьовувати вхідні дані, величина кроку 2, перекриття не передбачено.

Для кожної з областей, представлених фільтром, ми візьмемо максимальне значення та створимо нову вихідну матрицю, де кожним елементом є максимум з областей вхідної (рис. 2.6, 2.7).

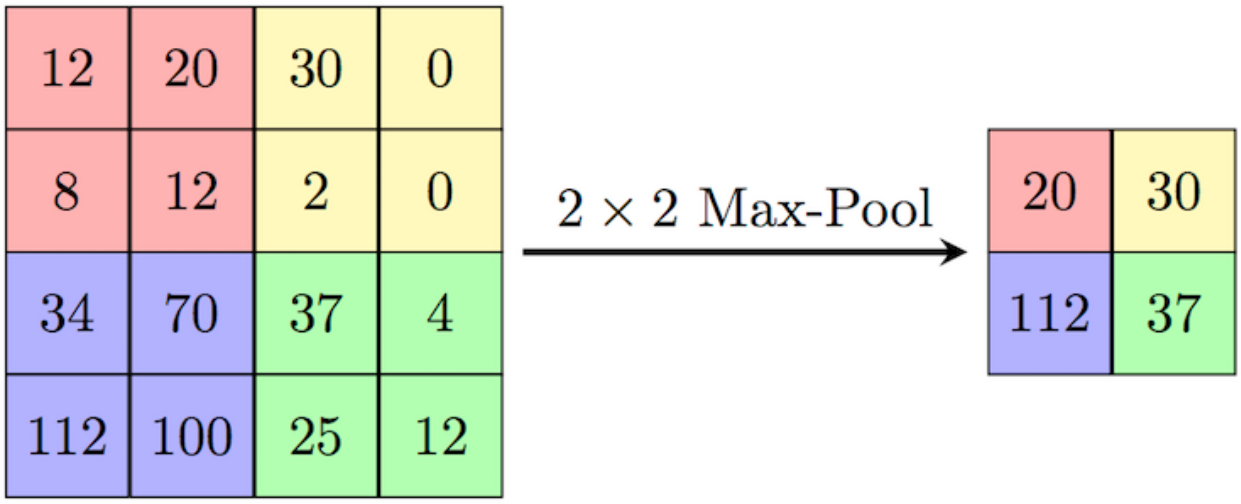

Рисунок 2.6 – Приклад роботи алгоритму max pooling

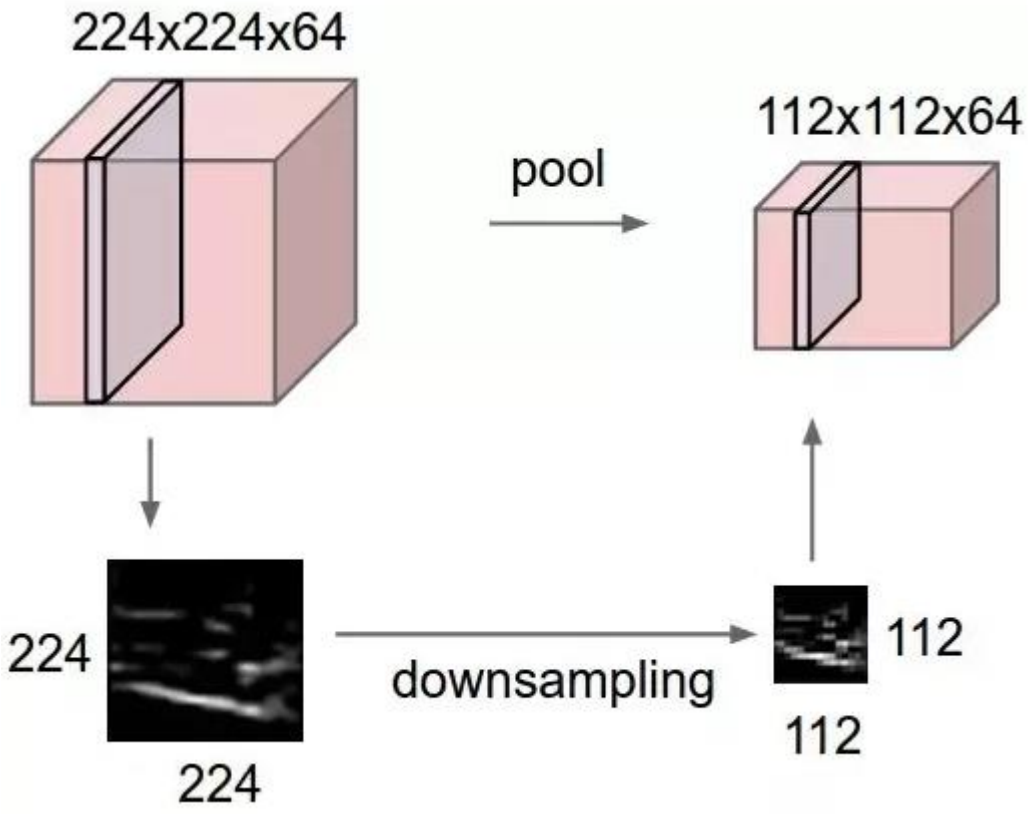

Рисунок 2.7 – Робота алгоритму max pooling на реальному зображенні

2.2.3 Випрямляч. Rectified linear unit (ReLU)

Випрямляч є активаційною функцією, що повертає позитивну частину аргументу:

$$f(x)= x^{+}=\max(0,x) ,$$

де x – вхідні дані нейрону.

Мета ReLU – ввести елемент нелінійності у згорткову нейронну мережу, та звести дані до не негативних (рис. 2.8).

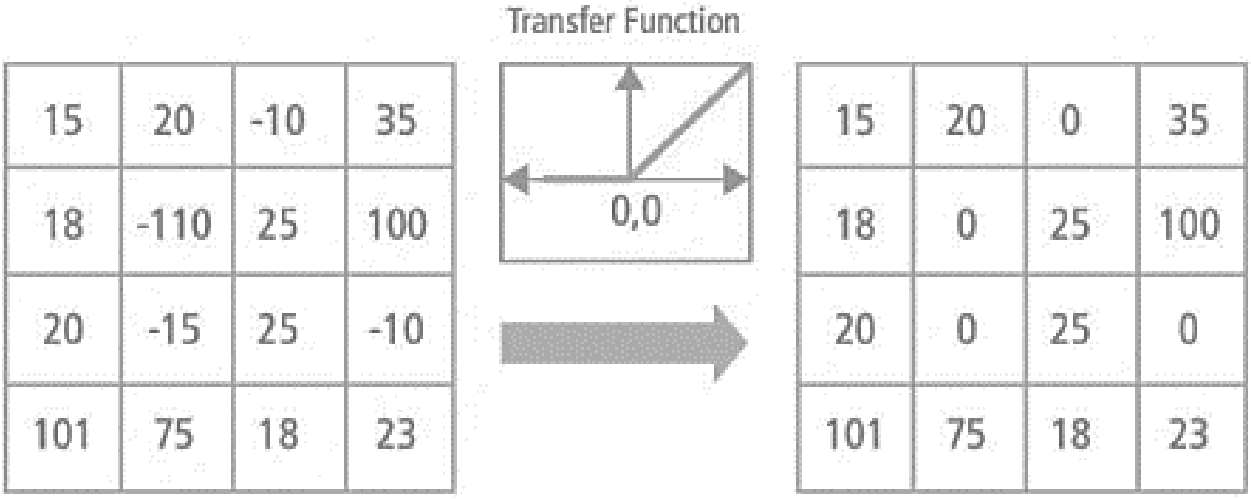

Рисунок 2.8 – Процес роботи ReLU

Як вже було зазначено у попередніх розділах, існує велика кількість функцій, у тому числі нелінійних, таких, як, сигмоід або гіперболічний тангенс, однак, науковці зазначають більш високу ефективність саме ReLU.

Існує також версія ReLU зі згладжуванням, так звана softplus:

$$f(x)=\log(1+\exp x).$$

Випрямляч на даний момент є найбільш популярною функцією активації для глибоких нейронних мереж. Його часто використовують для вирішення задач комп’ютерного зору та розпізнавання усного мовлення.

Графіки обидвох функцій активації можна побачити на рисунку 2.9.

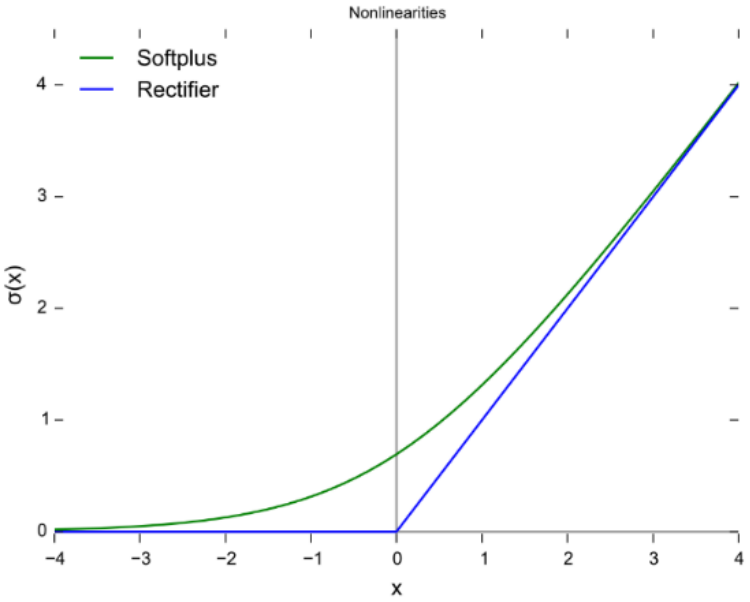

Рисунок 2.9 – Графік ReLU (синя лінія) та softplus (зелена лінія)

2.2.4 Повне поєднання (fully connected). Лінійна класифікація

У шару, що називають повним поєднанням, або fully connected, формують вектор з матриці та застосовують як вхідні дані до цього шару, подібно як до ще однієї нейронної мережі. За допомогою цього шару всі властивості поєднують разом для того, щоб сформувати модель. В кінці кінців, необхідно застосувати функцію активації задля того, щоб отримати клас, до якого належить зображення.

Розберемо детальніше лінійну класифікацію як один з варіантів застосування у цьому шарі.

Лінійна класифікація приймає рішення щодо занесення якогось об’єкта до класу на основі лінійної комбінації характеристик. Ці характеристики також відомі як значення властивостей і, як правило, представлені у векторі, який називається вектором властивостей. Такі класифікатори добре працюють для практичних проблем, таких як класифікація документів, і в цілому для проблем з багатьма змінними (властивостями), досягаючи рівнів точності,

порівнянних з нелінійними класифікаторами, і при цьому забираючи менше часу на тренування та використання.

Цей тип класифікатора працює краще, коли класи можна розділити лінією (рис. 2.10).

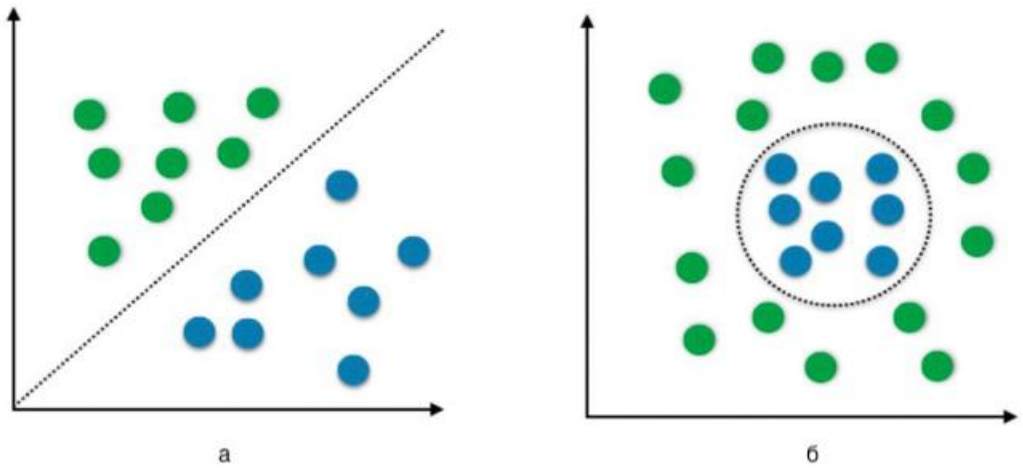

Рисунок 2.10 – Приклади лінійного (а) та нелінійного (б) розподілення об’єктів

Його часто використовують у випадках, коли швидкість роботи алгоритму є важливою, оскільки часто він працює у рази менше за часом у порівнянні з альтернативами (особливо коли вибірка добре розріджена). Також він добре проявляє себе, коли розмірність матриці ваг досить велика, і класифікатор доволі легко регулювати [31].

На рисунку 2.11 зображено приклад спроби розподілити заповнені та пусті усередині точки за допомогою лінійного класифікатора. Як видно, блакитна лінія (H1) та червона (H2) коректним чином розподілили об’єкти на два класи, тоді як зелений класифікатор H3 працює неправильно. Проте, треба зазначити той факт, що класифікатор H2 слід визнати кращим за блакитний H1, бо він рівновіддалений від обох скупчень об’єктів, і напевно матиме більш високий процент правильних попадань до класу.

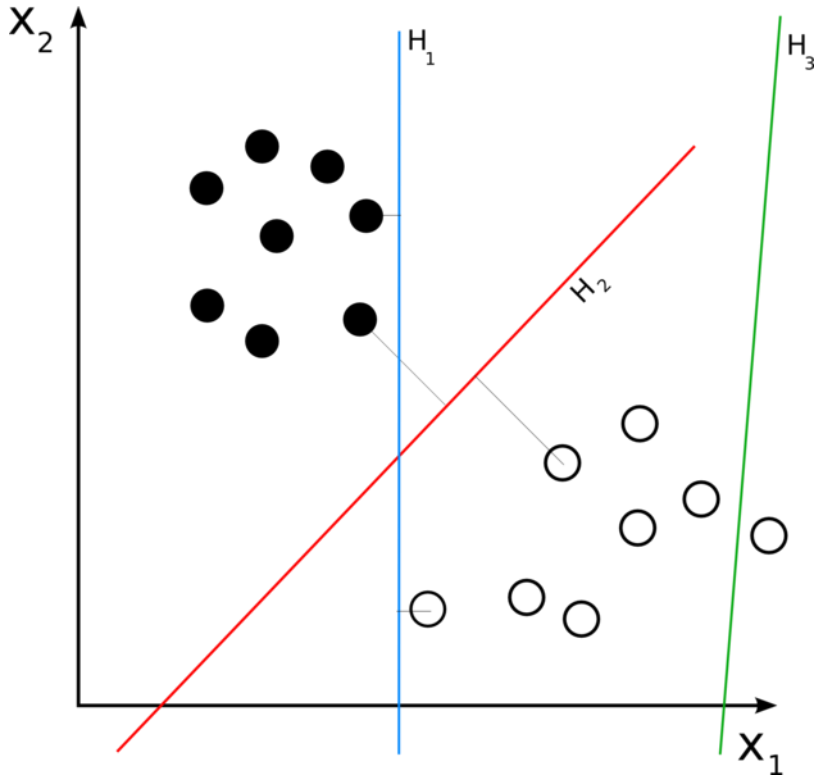

Рисунок 2.11 – Приклад спроб розділити простір на класи за допомогою лінійного класифікатора

Функція графіку матиме наступний вигляд:

$$y = f(\vec{x}, \vec{W}, \vec{b}) = \sum_{j} W_j x_j + b,$$

де x – вектор вхідних параметрів, W – матриця ваг, b – вектор відхилення, f – функція, що розрахує вихідне значення (це може бути порогова функція, що поверне значення у вигляді булевого параметру, або щось більш складне, здатне розрахувати ступінь належності до класу, ймовірність цього) [30].

Матриця ваг має по одному рядку для кожного класу та по одному стовпчику для кожної властивості. На рисунку 2.12 кожна лінія представлена рядком у матриці:

При зміні ваги будь-якого з параметрів буде змінюватися кут нахилу лінії, а корекція зміщення рухатиме її вгору-вниз.

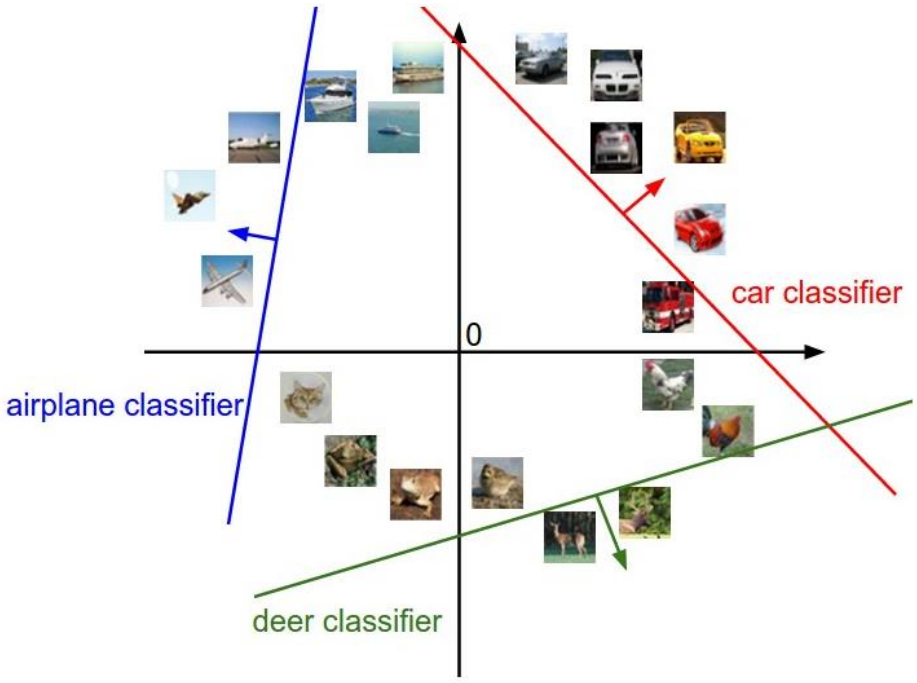

Рисунок 2.12 – Зразок розподілення об’єктів на класи (літак, машина, олень) за допомогою лінійних класифікаторів

Розглянемо параметричний підхід. Ідея полягає у тому, що у гіпотези або моделі є певні параметри, що допоможуть розмістити вхідні дані до специфічного класу. Параметрична модель має два важливих компонента:

– функція підрахунку – функція f(x, W, b), що розмістить неопрацьований вхідний вектор у розрахований;

– функція втрат – визначає, наскільки добре поточний набір ваг трансформує вхідні параметри у вихідне значення; розраховується під час тренувань.

При такому підході, на фазі тренування ми постійно коригуємо модель задля кращого результату. На цьому етапі, що займається проблемою оптимізації, єдине, що ми можемо змінювати – ваги W та здвиг b.

Розглянемо приклад (рис. 2.13). Нам потрібно розпізнати об’єкт на зображенні, та розрахувати його належність до кожного з класів. У цьому випадку, нам потрібно перетворити матрицю чисел (пікселі зображення) у одновимірний вектор. Наше зображення стає масивом 4 × 1, з допомогою якого нам потрібно класифікувати об’єкт.

У сірому квадраті матриця ваг перемножується на вектор вхідних параметрів, після чого до результату множення додають значення відхилення. Отримавши вихідне значення, ми коригуємо ваги за допомогою функції втрат, що порівнює фактичну, вірну відповідь з передбаченням алгоритму.

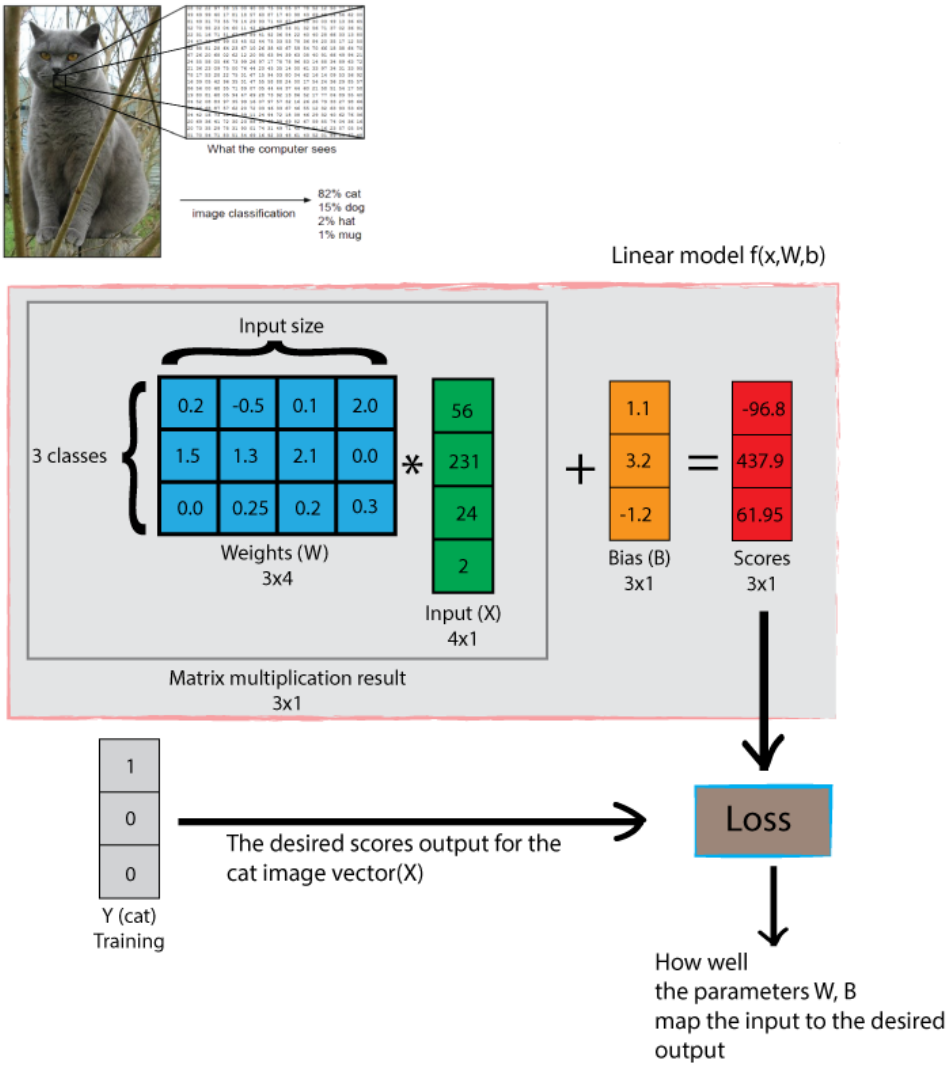

Рисунок 2.13 – Схема роботи лінійного класифікатора на прикладі ідентифікації об’єкта на зображенні

Тезисно резюмуємо процес роботи класифікатора:

1. Вхідне зображення x «стягується» до одновимірного вектору. Таким чином, просторова інформація втрачається.

2. Матриця ваг матиме по одній колонці для кожного вхідного параметру.

3. Матриця ваг матиме по одному рядку для кожного вихідного елементу (класу).

4. Відхилення матиме один вектор, кількість елементів якого дорівнюватиме кількості вихідних значень.

5. Втрати розраховуватимуться на основі порівняння отриманих значень з бажаним результатом для поточного зображення x.

Розрахуємо один рядок матриці ваг, щоб знайти відповідність параметрів для класу «кіт». Значення обчислюється за допомогою добутку вхідного вектору X та рядка, що відповідає цьому класу. Так, отримуємо результат розрахунку величини, що дозволить визначити ступінь належності зображення до класу котів:

$$score_{cat}=[0.2\times56 - 0.5\times231 + 0.1\times24 + 2\times2] + 1.1= -96.8.$$

2.3 Функції втрат. Крос-ентропійна функція втрати

Функція втрат грає дуже важливу роль у навчанні нейронної мережі. Вона використовується для розрахунку помилки між реальними і згенерованими відповідями. Глобальна мета – мінімізувати цю помилку. Таким чином, функція втрат ефективно наближає навчання нейронної мережі до виконання цієї задачі.

Фактично, вона вимірює, наскільки гарно нейронна мережа справляється з навчальною вибіркою та як точно визначає, до якого класу належить об’єкт. Вона також може залежати від таких змінних, як ваги і зміщення.

Функція втрат одномірна і не є вектором, оскільки вона оцінює, наскільки добре нейронна мережа працює в цілому.

Деякі відомі функції втрат:

– квадратична (середньоквадратичне відхилення);

– крос-ентропія (іноді перехресна ентропія);
– експоненціальна (AdaBoost);
– відстань Кульбака-Лейблера або приріст інформації.

Розглянемо детальніше крос-ентропійну функцію втрат.

Перехресна ентропія вимірює ефективність класифікаційної моделі, вихід якої є величиною ймовірності між 0 і 1. Тож прогнозування ймовірності, наприклад, 0.012, коли справжня ймовірність дорівнює 1, з точки зору коректності є далекою від істини, і призведе до високого рівня втрат. У ідеальної моделі функція втрат дорівнюватиме 0.

На графіку (рис. 2.14) показано діапазон можливих значень втрат при справжньому спостереженні 1. У міру наближення прогнозованої ймовірності до 1 функція втрати поступово зменшується. Коли ж передбачена машиною ймовірність падає, функція втрат швидко росте. З цього можна зробити висновок, що впевнені та неправильні прогнози будуть виділятися дуже сильно, і такі значення можуть сигналізувати про погано побудовану модель.

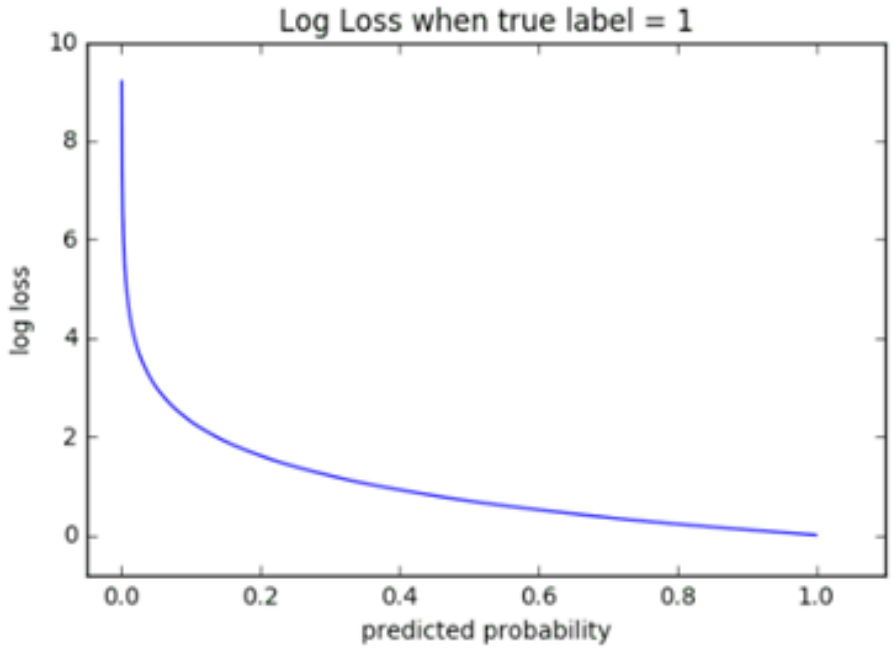

Рисунок 2.14 – Графік залежності рівня втрат від ступеня близькості до правильної відповіді

У двійковій класифікації, де кількість класів M дорівнює 2, перехресна ентропія може бути обчислена як:

$$\text{crossentropy} = -(y \log(p) + (1-y) \log(1-p)) .$$

Якщо M > 2, рахують кожну втрату окремо для кожного класу відносно об’єкту, а результат сумують:

$$\text{crossentropy} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) .$$

У вищенаведених формулах M – кількість класів, y – бінарний індикатор (0 або 1), що демонструє, чи є клас c правильною відповіддю для об’єкта o, p – передбачена ймовірність того, що об’єкт o належить до класу c.

2.4 Метод стохастичного градієнта (stochastic gradient descent)

Градієнт означає нахил або нахил поверхні. Тож спуск градієнта буквально означає спуск по схилу, щоб досягти найнижчої точки на цій поверхні.

Візьмемо у якості прикладу двовимірний графік, такий як парабола на рисунку 2.15.

У наведеному графіку найнижча точка параболи виникає при x = 1. Завдання алгоритму спуску градієнта – знайти значення x, таке, що y є мінімальним. y називається цільовою функцією, над якою працює алгоритм спуску градієнта, щоб спуститися до нижньої точки.

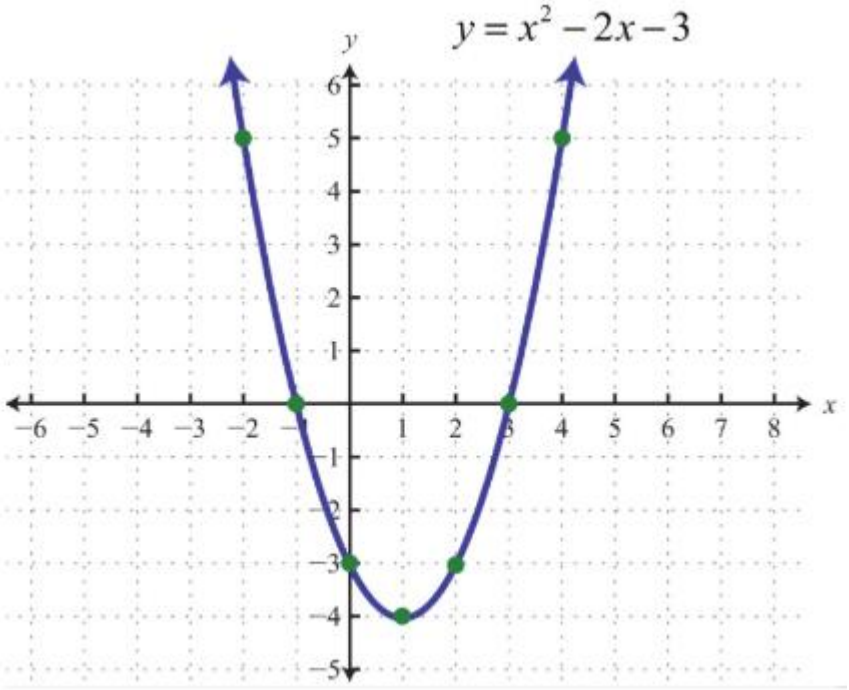

Рисунок 2.15 – Параболічна функція у двовимірному просторі

Спуск градієнта – це ітеративний алгоритм, який починається з випадкової точки функції і рухається вниз по її схилу кроками, поки не досягне нижньої точки цієї функції.

Цей алгоритм корисний у тих випадках, коли оптимальних точок неможливо знайти, прирівнюючи нахил функції до 0. У випадку лінійної регресії можна подумки зіставити суму квадратичних залишків як функцію у та вектору ваг як x в параболі вище [34].

У цьому суть алгоритму. Загальна ідея полягає в тому, щоб почати з випадкової точки (у прикладі з параболою починаємо з випадкового x) і знаходити спосіб оновити цю точку з кожною ітерацією, щоб спуститися по схилу.

Кроки алгоритму наступні:

1. Знайти нахил цільової функції стосовно кожного параметра/ознаки. Іншими словами, обчислити градієнт функції.

2. Обрати випадкові початкові значення для параметрів (в прикладі з параболою слід диференціювати у відносно x; якщо було би більше

особливостей, таких як x_1, x_2 тощо, можна було би взяти часткову похідну від у стосовно кожної з ознак).

3. Оновити функцію градієнта, додавши значення параметрів.

4. Обчислити розміри кроків для кожної функції наступним чином: розмір кроку = градієнт * швидкість навчання.

5. Обчислити нові параметри так: нові параметри = старі параметри – розмір кроку.

6. Повторювати кроки 3-5, поки градієнт не буде достатньо близький до 0.

Згадана вище «швидкість навчання» є гнучким параметром, який сильно впливає на конвергенцію алгоритму. Більш високі темпи навчання змушують алгоритм робити величезні кроки вниз по схилу, і він може стрибати через мінімальну точку, тим самим не дістаючи до неї. Отже, завжди добре дотримуватися низького рівня навчання, наприклад, 0.01. Математично може бути показано, що алгоритм спуску градієнта робить більші кроки вниз по схилу, якщо вихідна точка знаходиться вище, і робить маленькі кроки, коли він досягає ближче до пункту призначення, щоб бути обережним та не пропустити його, і в той же час бути досить швидким.

Існує кілька недоліків алгоритму спуску градієнта. Потрібно більш детально розглянути кількість обчислень, які треба робити для кожної ітерації алгоритму.

Скажімо, у нас є 10 000 точок даних та 10 функцій. Сума залишків у квадраті складається з такої кількості варіантів, скільки є точок даних, тому в нашому випадку 10 000. Нам потрібно обчислити похідну цієї функції відносно кожної з особливостей, тому фактично ми будемо робити $10000 \cdot 10 = 100\,000$ обчислень за ітерацію. У випадку з 1000 ітераціями, насправді необхідно зробити $100\,000 \cdot 1000 = 100\,000\,000$ обчислень для завершення алгоритму. Як наслідок, на великій кількості даних алгоритм спуску градієнта буде повільним.

Щоб уникнути цього недоліку, застосовують метод стохастичного градієнта. Під час вибору точок даних на кожному кроці для обчислення похідних обирають не усі точки, а лише одну на кожній ітерації. Це значно скорочує обчислення.

Також прийнято вибирати невелику кількість точок даних замість лише однієї точки на кожному кроці, і це називається «міні-пакетним» градієнтним спуском. Міні-пакет намагається встановити баланс між точністю градієнтного спуску та швидкістю SGD.

У нейронних мережах метод застосовують для оптимізації обчислень та визначення тих властивостей, що не грають ролі у обчисленні результативного класу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ РОЗПІЗНАВАННЯ ПОРОДИ КОТА

3.1 Обґрунтування вибору середовища програмної реалізації

У рамках атестаційної роботи було розроблено програмний застосунок для тренування, перевірки та застосування моделі нейронної мережі, архітектура якої передбачена для класифікації зображень (у першу чергу для розпізнавання порід котів). Програма не має обмежень стосовно операційної системи. Вона була написана на мові програмування Python із використанням середовища розробки PyCharm. Таке IDE було обрано у першу чергу тому, що він створений для роботи с обраною мовою програмування. У PyCharm легко редагувати код завдяки автодоповненню, перевірці коду «на льоту», підсвічуванню помилок і швидких виправлень, автоматичного рефакторингу і зручної навігації. Також є інтерактивна консоль, зручна у випадку, коли потрібно щось швидко перевірити, можливість підключати бібліотеки Anaconda, а також працювати з такими для наукових обчислень і аналізу даних (наприклад, використані у роботі PyTorch, OpenCV та ін.). PyCharm надає вбудований налагоджувач і інструмент запуску тестів, профілювальник Python, вбудований термінал, інструменти для роботи з базами даних і інтеграцію з популярними системами контролю версій. Також можна переглядати документацію до методів, не переходячи до самого файлу, де вони описані.

Щоб залежності були ізольовані, можна використовувати окреме оточення Conda для кожного проекту. PyCharm дозволяє легко створювати і вибирати правильне оточення.

За роки розробки платформи IntelliJ було створено більше 50 плагінів для PyCharm, які забезпечують підтримку додаткових систем контролю версій, інтеграцію з інструментами і фреймворками, а також розширюють можливості редактора, наприклад за рахунок емуляції Vim.

3.2 Мова програмування для розробки застосунку

Python – високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і простоти читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує структурне, об’єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основні архітектурні риси:

- динамічна типізація;
- автоматичне керування пам’яттю;
- повна інтроспекція;
- механізм обробки виключень;
- підтримка багатопоточних обчислень;
- високорівневі структури даних.

Підтримується розбиття програм на модулі, які, в свою чергу, можуть об’єднуватися в пакети.

Python активно розвивається. Нові версії з додаванням/зміною властивостей виходять приблизно раз в два з половиною роки.

Серед основних її переваг можна назвати такі:

- код, що легко читати та писати;
- програму легко переносити на інші машини (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв’язання простих задач);
- зручний для розв’язання математичних проблем;
- відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження [24].

Окрім самої мови, було використано наступні бібліотеки:

– io – бібліотека для роботи з вхідними/вихідними потоками;

– os – бібліотека, що містить функції для взаємодії з операційною системою;

– glob – бібліотека для пошуку файлів за заданим шаблоном;

– PyTorch – бібліотека, що містить засоби для розробки алгоритмів глибокого навчання, тренування та перевірки, застосування навчених моделей;

– Python Imaging Library, PIL – бібліотека, що містить засоби роботи з зображеннями;

– OpenCV – бібліотека алгоритмів комп’ютерного зору, обробки зображень та чисельних алгоритмів загального призначення;

– PySimpleGUI – бібліотека для створення графічного інтерфейсу програми.

3.3 Програмна реалізація

Даний застосунок містить декілька скриптів, кожен з яких відповідає за певну частину роботи програми. Розглянемо кожен з них окремо.

Файл consts.py містить основні константи, необхідні для роботи інших скриптів. Вони мають наступне призначення:

– SCALE – містить функцію transforms.Resize для приведення зображень до єдиного розміру 256 x 256;

– TO_TENSOR – містить функцію transforms.ToTensor для трансформації зображення у масив типу Tensor;

– NORMALIZE – містить функцію transforms.Normalize для нормалізації зображень (значення середніх для всіх каналів 0.5, стандартне відхилення – 0.5);

– TRANSFORMATIONS – містить набір вищеописаних трансформацій для попередньої обробки зображень з набору даних;

– CLASSES_PRINT_AMOUNT – кількість класів, що буде виводитися у застосунку, на даний момент 5;

– PATH_TO_TRAIN_DATASET – путь до набору даних для тренування;

– PATH_TO_TEST_DATASET – путь до набору даних для валідації моделі;

– MODEL_PATH – путь до файлу, що містить ваги натренованої моделі;

– BATCH_SIZE – кількість зображень, що будуть одночасно подаватися до класифікатора, встановлений на 1;

– NUM_WORKERS – кількість потоків, що будуть водночас опрацьовувати набір даних, встановлений на 0 через те, що можливість паралелізації процесу відсутня у Windows;

– EPOCH_NUM – кількість разів, коли весь тренувальний датасет пройде через алгоритм навчання, встановлений на 4;

– CLASSES – лист з назвами досліджуваних класів, у даному випадку – порід котів;

– CLASSES_NUM – кількість класів (кількість елементів у списку CLASSES).

Існує також утилітарний файл png_to_jpg.py, що був створений для приведення усіх зображень до формату JPEG (рис. 3.1). У ньому витягуються усі файли під певним каталогом, шукаються зображення з форматом PNG, конвертуються у JPEG, зберігаються, а старі видаляються.

```
from glob import glob
import cv2
import os

pngs = glob('test-dataset/*/*.png')

for j in pngs:
    img = cv2.imread(j)
    cv2.imwrite(j[:-3] + 'jpg', img)
    os.remove(j[:-3] + 'png')
```

Рисунок 3.1 – Файл png_to_jpg.py

Файл net.py містить саму архітектуру нейронної мережі. У ньому присутній клас Net, що представляє собою реалізацію згорткової нейронної мережі. Є константа COEFFICIENT, що дорівнює 16; вона впливає на кількість вихідних властивостей у другого згорткового шару, вхідних властивостей у першого лінійного класифікатора, та на представлення матриці властивостей після проходження через операції згортки [14].

Архітектура нейронної мережі побудована з двох згорткових шарів (у першому кількість вхідних параметрів дорівнює три, бо у зображення 3 канали, кількість вихідних – 6, розмір ядра – 5; у другого кількість вхідних каналів – 6, кількість вихідних дорівнює константі COEFFICIENT, розмір ядра – 5). Шар з’єднання є максимальним з’єднанням (max pooling) з розміром ядра 2 та величиною кроку 2. Присутні три шари повного поєднання (fully connected), заснованих на лінійній класифікації (перший шар: кількість вхідних властивостей – COEFFICIENT × 61 × 61, кількість вихідних

властивостей – 120; другий шар: кількість вхідних властивостей – 120, кількість вихідних властивостей – 84; третій шар: кількість вхідних властивостей – 84, кількість вихідних властивостей дорівнює кількості класів) [11].

У класу Net також є функція forward, що приймає у себе вхідні параметри (пакет зображень, кількість залежить від BATCH_SIZE). У ній застосовуються згорткові шари, які потім нормалізують за допомогою ReLU та стиснюють, використовуючи вище описане максимальне поєднання. Після цього вихідні дані перебудовують у матрицю, висота якої буде дорівнювати BATCH_SIZE. У кінці застосовуються три шари повного поєднання, перші два з яких «обгорнуті» випрямлячем ReLU. Функція повертає результат обчислень, матрицю з класами та мірою належності зображення до них [12].

Код класу наведений у рисунку 3.2.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=COEFFICIENT, kernel_size=5)
        self.fc1 = nn.Linear(in_features=COEFFICIENT * 61 * 61, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=CLASSES_NUM)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(BATCH_SIZE, COEFFICIENT * 61 * 61)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Рисунок 3.2 – Клас Net, його реалізація

Файл cat_breed_trainer.py містить код для тренування нейронної мережі та створення моделі. Передбачено тренування як на потужностях CPU, так і на GPU. За рахунок чого проводити тренування, обирається автоматично на

початку, перевага надається графічному процесору, якщо той наявний у комп’ютера.

Після вибору пристрою до програми завантажується пакет тренувальних даних, який на жорсткому диску представлений у вигляді папки з папками зображень, що відсортовані та наіменовані згідно з назвами класів. До всіх об’єктів класифікації застосовують трансформації задля приведення їх до певної однорідності [39].

Після первинної обробки дані завантажуються у спеціальний завантажувач train_loader, де кількість зображень у одному пакеті дорівнює BATCH_SIZE, об’єкти випадковим чином перемішані, а кількість потоків, у яких оброблятимуться дані, дорівнює NUM_WORKERS.

Далі ініціалізують клас з нейронною мережею, завантажуємо модель, якщо така існує (можливо у випадках, коли її тренували раніше, зберегли, а потім вирішили ще попрацювати над нею). Також створено змінні criterion та optimizer, що відповідатимуть за підрахунок функції втрати (обрано крос-ентропійну) та процес оптимізації ваг нейронної мережі за допомогою методу стохастичного градієнта відповідно.

Процес завантаження даних до нейронної мережі та тренувального завантажувача наведено на рисунку 3.3.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

train_set = ImageFolder(PATH_TO_TRAIN_DATASET, transform=TRANSFORMATIONS)

train_loader = torch.utils.data.DataLoader(train_set, batch_size=BATCH_SIZE,
                                           shuffle=True, num_workers=NUM_WORKERS)

net = Net()
net.to(device)
if os.path.isfile(MODEL_PATH):
    net.load_state_dict(torch.load(MODEL_PATH))

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Рисунок 3.3 – Підготовка даних та нейронної мережі до навчання

Після виконання вищезазначеного починається тренування мережі. В залежності від того, скільки разів було вирішено «прогнати» модель через тренувальні дані (визначається константою EPOCH_NUM), формується відповідний цикл. У ньому ініціалізують початкове значення втрати 0 та створюють новий цикл, який ітерується вже по тренувальним даним.

У ньому спочатку оптимізатор видаляє градієнти всіх оптимізованих матриць. Потім, власне, вхідні дані поміщують у згорткову нейронну мережу, реалізовану у класі Net, підраховують функцію втрати для кожного об’єкта у пакеті для підрахованої імовірності кожного класу, та робиться один крок оптимізації ваг. Підрахована втрата сумується до загальної.

Після є невелика логіка виведення у консоль статистики щодо рівня функції втрати кожні 1 000 опрацьованих зображень. Якщо ми знаходимося на певній ітерації, де потрібно надрукувати поточне положення справ, значення загальної суми втрат ділиться на 1 000 та виводиться у консоль, а після обнуляється. Вивід такої інформації зроблений для того, щоб слідкувати за прогресом навчання моделі та запобігти ситуації, коли результат не тільки не покращується, а навпаки. Після тренування модель зберігається у файловій системі. Процес навчання нейронної мережі наведений на рисунку 3.4.

```
for epoch in range(EPOCH_NUM):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 1000 == 999:  # print every 1000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1000))
            running_loss = 0.0

print('Finished Training')

torch.save(net.state_dict(), MODEL_PATH)
```

Рисунок 3.4 – Процес тренування нейронної мережі

Файл cat_breed_model_validate.py призначений для перевірки натренованої моделі. Початок роботи схожий зі схемою тренування, проте дані не перемішуються, а модель завантажується у будь-якому випадку – без неї застосунок не працюватиме (рис. 3.5).

```
test_set = ImageFolder(PATH_TO_TEST_DATASET, transform=TRANSFORMATIONS)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=BATCH_SIZE,
                                          shuffle=False, num_workers=NUM_WORKERS)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net()
net.to(device)
net.load_state_dict(torch.load(MODEL_PATH))
```

Рисунок 3.5 – Завантаження тестових даних та моделі

Починається підрахунок загальної точності роботи алгоритму. Функція градієнту вимикається задля підвищення швидкості визначення результатів (на етапі тестування оптимізація не проводиться). Потім у коді ітеруються по пакету даних, дістають передбачені класи з моделі, підраховують загальну кількість зображень та кількість об’єктів, клас для яких було визначено коректно. Після застосунок виводить у консоль процент правильних передбачень системи (рис. 3.6) [9].

```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on test images: %d %%' % (100 * correct / total))
```

Рисунок 3.6 – Алгоритм обчислення точності передбачень моделі

Після підраховується точність попадань відносно кожного класу. Створюються списки із загальною кількістю об’єктів, що належать до категорії, та з кількістю коректних відповідей. Так же само, без оптимізації, проходять по тестовому пакету даних ще раз. До кожного зображення застосовують модель, беруть клас з найбільшою ймовірністю належності, за версією програми. Є ще один цикл, що проходить по самому пакету зображень, розмір якого дорівнює BATCH_SIZE, підраховує кількість коректних відповідей, записує до списку [37].

В кінці кінців, списки використовуються для того, щоб вивести до консолі процент попадань для кожного з класів (рис. 3.7).

```
class_correct = list(0. for i in range(CLASSES_NUM))
class_total = list(0. for i in range(CLASSES_NUM))
with torch.no_grad():
    for data in test_loader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(BATCH_SIZE):
            label = labels[i]
            if BATCH_SIZE == 1:
                class_correct[label] += c.item()
            else:
                class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(CLASSES_NUM):
    print('Accuracy of %5s : %2d %%' % (
        CLASSES[i], 100 * class_correct[i] / class_total[i]))
```

Рисунок 3.7 – Підрахунок ефективності розпізнавання кожного окремо взятого класу

Останній файл, cat_breed_classifier.py, призначений для кінцевого користувача. Він просто застосовує попередньо треновану модель задля класифікації породи кота, має графічний інтерфейс. Незважаючи на простоту функціоналу, є найбільш масивним.

Містить утилітарний метод get_img_data, призначений для конвертації зображення у тому форматі, що підходить для виводу зображення у графічне вікно. Також масштабує його до розміру 800 x 600, для зручності відображення (рис. 3.8).

```
def get_img_data(f, maxsize=(800, 600), first=False):
    image = Image.open(f)
    image.thumbnail(maxsize)
    if first:  # tkinter is inactive the first time
        bio = io.BytesIO()
        image.save(bio, format="PNG")
        del image
        return bio.getvalue()
    return ImageTk.PhotoImage(image)
```

Рисунок 3.8 – Метод для конвертації зображення у формат tkinter-сумісного зображення

Перший крок той же самий – завантажується модель, обирається засіб обробки моделі, CPU або GPU (рис. 3.9).

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = Net()
model.load_state_dict(torch.load(MODEL_PATH))
model.to(device)
```

Рисунок 3.9 – Завантаження тестових даних та моделі

Потім формується об’єкт, що містить параметри для стартового вікна. Він пропонує обрати файл для розпізнавання, містить кнопки «Browse» (обрати файл), «Ok» (вивести результат класифікації) та «Cancel» (закрити програму). Кольорова гама виконана у золото-червоних тонах. Друге вікно, де буде відображено результат роботи, неактивне та не показується, доки кнопка «Ok» не буде натиснута (рис. 3.10).

```
sg.change_look_and_feel('Reds')
layout1 = [
    [sg.Text('Select image to process'), sg.InputText(), sg.FileBrowse()],
    [sg.Button('Ok'), sg.Button('Cancel')]]

window1 = sg.Window('Cat breed classifier', layout1)

window2_active = False
```

Рисунок 3.10 – Програмування зовнішності першого вікна

Поки перше вікно не закрили, програма очікуватиме зображення на вхід. Після натискання кнопки «Ok» у модель завантажуються отримані дані, попередньо опрацьовані за допомогою трансформаторів, що були описані раніше. Для того, щоб відобразити ймовірність належності до певного класу, застосовують функцію softmax. Формується двовимірна матриця результатів класифікації (рис. 3.11).

```
while True:
    event1, values1 = window1.read()
    if event1 in (None, 'Cancel'):
        break

    IMAGE = values1[0]
    tensor_img = TRANSFORMATIONS(Image.open(IMAGE))

    tensor_gpu = tensor_img.to(device)

    outputs = model(tensor_gpu[None, ...])

    outputs = F.softmax(outputs, dim=1)

    probabilities, predicted = torch.topk(outputs, CLASSES_PRINT_AMOUNT)

    data = []
    header_list = ['Number', 'Class name', 'Probability, %']
    for i in range(CLASSES_PRINT_AMOUNT):
        item = [i + 1, CLASSES[predicted[0][i].item()], '%.2f' % round(probabilities[0][i].item() * 100, 2)]
        data.append(item)
```

Рисунок 3.11 – Робота моделі, форматування результатів у зручному вигляді

Виводиться зображення, що класифікували, та таблиця з першими п’ятьома класами, їх ймовірностями на друге вікно (рис. 3.12).

```
if not window2_active and event1 == 'Ok':
    window2_active = True
    output_layout = [
        [sg.Image(data=get_img_data(IMAGE, first=True))],
        [sg.Table(values=data, headings=header_list,
                  hide_vertical_scroll=True,
                  auto_size_columns=True,
                  num_rows=CLASSES_PRINT_AMOUNT,
                  background_color='#872d2d',
                  alternating_row_color='#820909',
                  select_mode=sg.TABLE_SELECT_MODE_NONE,
                  text_color='#d48e48')]
    ]
    window2 = sg.Window('Cat breed classifier', output_layout)

if window2_active:
    event2, values2 = window2.read()
    if event2 is None or event2 in (None, 'Cancel'):
        window2_active = False
        window2.close()
```

Рисунок 3.12 – Кодування зовнішнього вигляду другого вікна

3.4 Оцінка результатів роботи програми

Результатом роботи є програмний застосунок, що ідентифікує породу кота за його зображенням. Тренувальний пакет даних містив близько 4 200 зображень з котами, розподілення зображень по класах більш-менш однорідне. Тестовий пакет містив 360 файлів, по 30 на кожен клас. Загалом модель тренувалася розпізнавати дванадцять порід котів – абіссинська кішка, бенгальська кішка, бірманська кішка, бомбейська кішка, британська короткошерста кішка, єгипетська мау, мейн-кун, персидська кішка, регдол, російська блакитна кішка, сіамська кішка та сфінкс.

Більшість параметрів нейронної мережі було обрано емпірично, шляхом багаторазового тренування моделей та спостерігання за змінами у точності передбачення тестових зразків. Найліпшим видався описаний у пункті 3.3 варіант, з загальною точністю попадань 35 відсотків. Інші об’єктивні результати роботи моделі на тестових даних можна побачити на рисунку 3.13.

```
Accuracy of the network on the 360 test images: 35 %
Accuracy of Abyssinian : 43 %
Accuracy of Bengal : 70 %
Accuracy of Birman : 30 %
Accuracy of Bombay : 36 %
Accuracy of British shorthair : 30 %
Accuracy of Egyptian mau : 30 %
Accuracy of Maine coon : 13 %
Accuracy of Persian : 30 %
Accuracy of Ragdoll : 50 %
Accuracy of Russian blue : 20 %
Accuracy of Siamese : 20 %
Accuracy of Sphynx :  10 %
```

Рисунок 3.13 – Розрахунок точності попадань натренованої мережі

На перший погляд, такий результат невтішний, але, зважаючи на те, що класів дванадцять, і виводяться перші п’ять з найвищою порівняно з іншими ймовірністю, насправді точність не є такою поганою. Фактично, у нас великі шанси зустріти у вибірці з п’яти класів правильну породу, а при підрахунку вищезазначеної метрики ураховувалися тільки перше, найвище за ймовірністю попадання.

Для того, щоб покращити результат, перш за все треба збільшити кількість тренувальних даних. Очевидно, що 4 000 зображень недостатньо для такої кількості класів і такої складності вхідних параметрів [13].

І тренування, і валідація моделі проходять досить швидко. У режимі роботи GPU з 6 гігабайтами оперативної пам’яті (відеокарта Nvidia GeForce 2060 RTX) перший етап при даних параметрах проходив за десять хвилин, валідація – приблизно за двадцять секунд. У разі роботи у режимі CPU процес іде дещо повільніше (тестування проходили на Intel Core i5-9600K та Intel Core i7-7600U), проте, можливість вибору дає свободу та знімає обмеження на випадок, якщо у комп’ютері немає відеокарти. У режимі роботи через графічний інтерфейс результат надається за лічені секунди у незалежності від режиму роботи.

Дизайн лаконічний та не містить нічого зайвого – на рисунку 3.14, наприклад, зображене стартове вікно, що містить лише поле зі шляхом до зображення, кнопку «Browse», та «Ok», «Cancel».

Рисунок 3.14 – Стартове вікно застосунку

Незважаючи на доволі низький відсоток, наданий метрикою, ключова потреба до нейронної мережі все ж таки не досягнути ста відсотків точності, а робити такі висновки, щоб вони мали сенс для людини. Очікувано, що вона часто плутає російську голубу з британцем або сіамську кішку з регдолом або бірманською – для нашого ока вони досить схожі. Проте, на рисунку 3.15 мережа достатньо природньо для людини зазначає, що зображений кіт майже однаково схожий на бірманську кішку та сіамську, все ж таки роблячи невелику перевагу не у той бік. У більшості випадків мережа дає якщо не коректну, то хоча б близьку до такої відповідь.

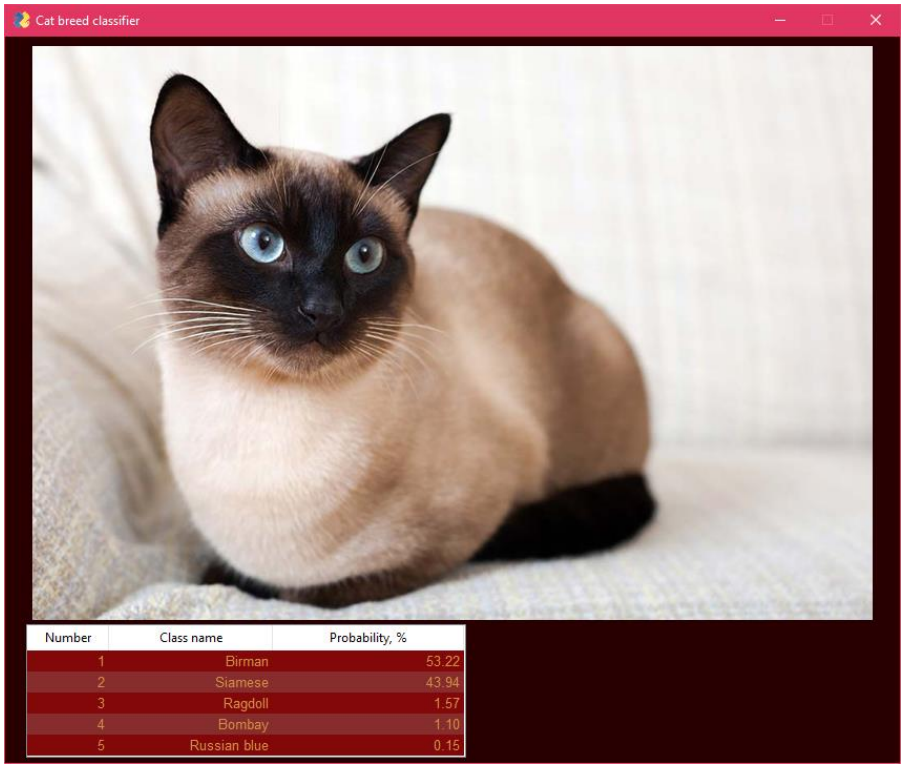

Рисунок 3.15 – Приклад не зовсім коректного розпізнавання мережею породи

Часто у таких ситуаціях у таблиці схожості можна спостерігати обидві породи, що мають схожу зовнішність. Так, на рисунку 3.16 російська голуба розпізнана вірно, проте присутній доволі високий процент за британського кота. Такий же випадок на рисунку 3.17. Перші два класи є схожими зовнішньо.

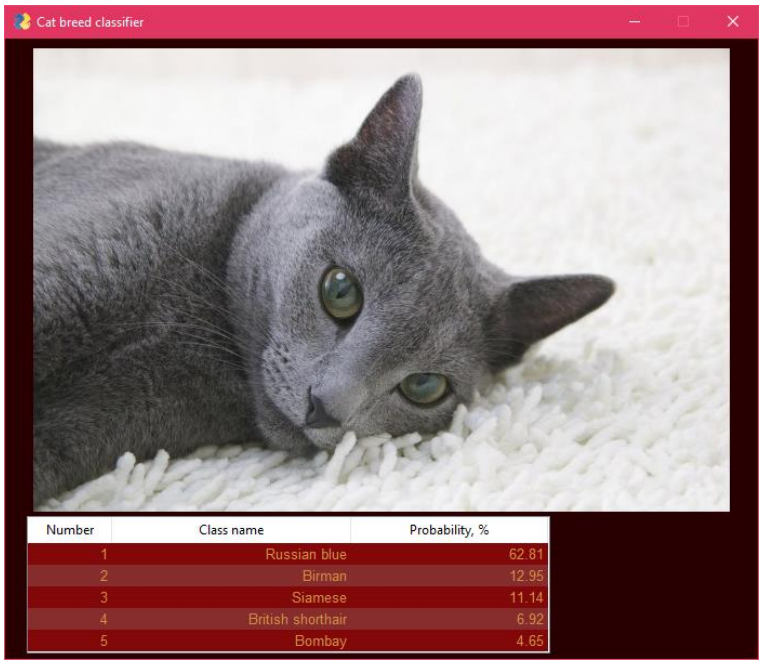

Рисунок 3.16 – Приклад коректного розпізнавання російської блакитної

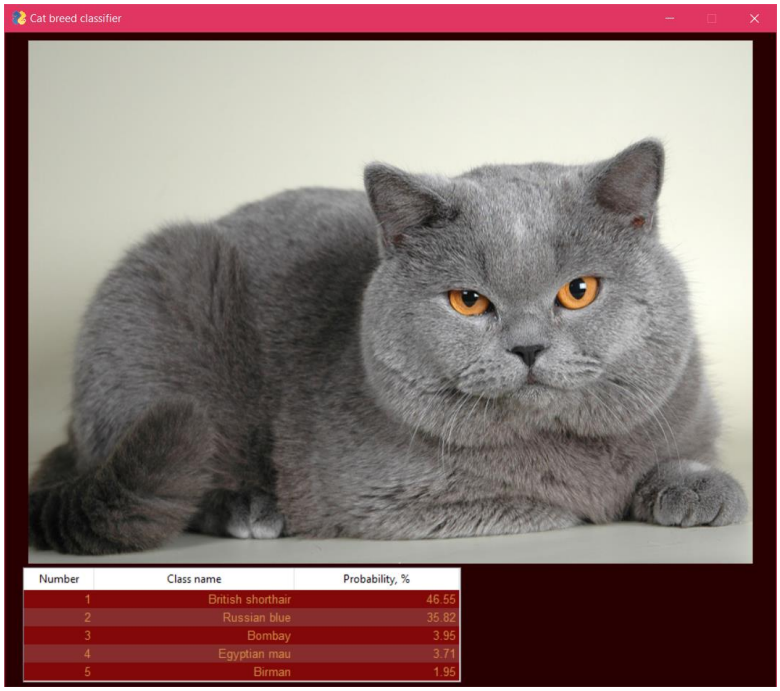

Рисунок 3.17 – Приклад розпізнавання британської кішки

Бувають і доволі впевнені, чіткі відповіді, як на рисунку 3.18.

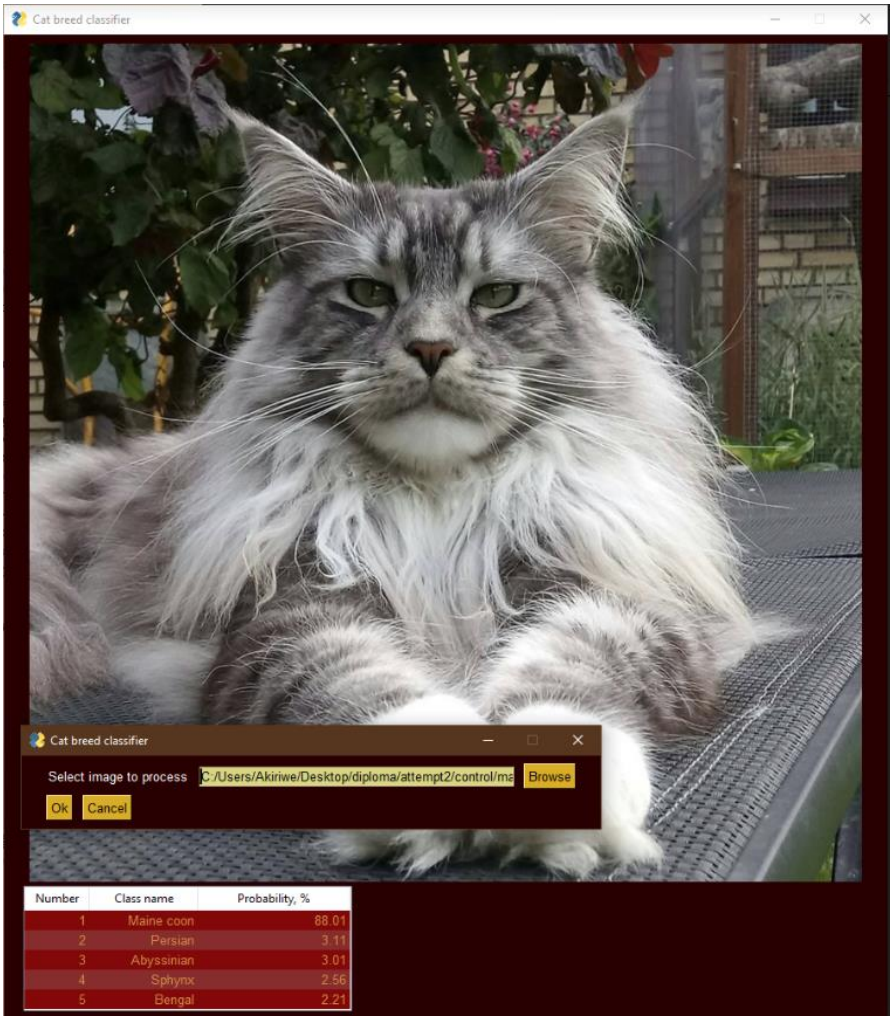

Рисунок 3.18 – Приклад впевненої та правильної відповіді нейронної мережі

Іноді алгоритму навіть не заважають зайві предмети на зображенні, проте, це скоріше вдача, ніж правило.

Так, на рисунку 3.19 йому не завадила наявність банану, і британського короткошерстого визначено правильно, хоч фрукт, напевно, все ж таки трохи «ввів в оману» – у результативній таблиці бачимо невідомо звідкіля доволі значні майже вісім процентів сфінкса.

На рисунку 3.20 людина у кадрі дуже сильно вплинула на результат – текстура шкіри дуже схожа на того ж сфінкса.

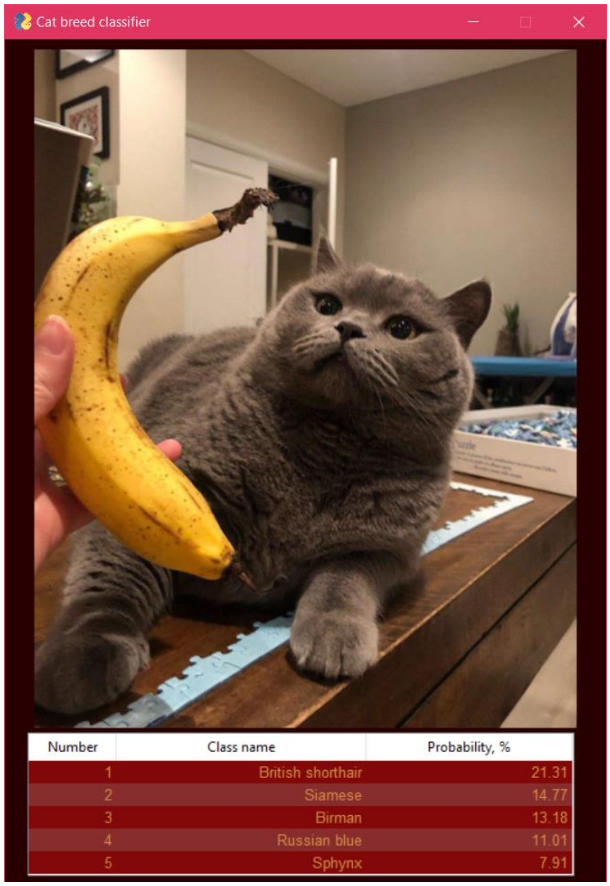

Рисунок 3.19 – Приклад класифікації зображення із зайвим об’єктом

Рисунок 3.20 – Помилкова класифікація через текстуру людської шкіри

Як висновок, можна сказати, що нейронна мережа значно краще порається з задачею при таких умовах:

- контрастний однорідний фон;
- кішка представлена у повний ріст, мордою до камери;
- кішка повинна займати більшу частину зображення;
- окрім неї, на зображенні немає ніяких рівних за розміром або навіть більших об’єктів.

ВИСНОВКИ

У процесі виконання роботи було розроблено архітектуру нейронної мережі на основі згорткової для її тренування, створення моделі та використання для класифікації породи котів. Було досліджено основні засоби розробки нейронної мережі, її навчання, підрахунку функції втрат, оптимізації моделі, зберігання, її валідації та застосування.

Результатом роботи є програмний застосунок для роботи з моделлю, а саме, її тренування, перевірку та застосування.

Тренувальний пакет даних містив близько 4 200 зображень з котами, розподілення зображень по 12 класах більш-менш однорідне. Тестовий пакет містив 360 файлів, по 30 на кожен клас. Більшість параметрів нейронної мережі було обрано емпірично, шляхом багаторазового тренування моделей та спостерігання за змінами у точності передбачення тестових зразків.

Натреновану модель було перевірено на тестовому наборі даних; найкраща модель з точки зору валідації має точність 35 відсотків.

На перший погляд, такий результат невтішний, але, зважаючи на те, що класів дванадцять, і виводяться перші п’ять класів з найвищою порівняно з іншими ймовірністю, насправді точність не є такою поганою.

Ключова потреба до нейронної мережі все ж таки не досягнути ста відсотків точності, а робити такі висновки, щоб вони мали сенс для людини. Очікувано, що вона часто плутає російську голубу з британцем або сіамську кішку з регдолом або бірманською – для нашого ока вони досить схожі.

Система краще порається з зображеннями, що мають наступні властивості:

– контрастний однорідний фон;

– кіт представлений на зображенні у повний ріст, мордою до камери;

– кіт повинен займати більшу частину зображення;

– окрім нього, на зображенні немає ніяких рівних за розміром або навіть більших об’єктів.

Для того, щоб покращити результат, перш за все треба збільшити кількість тренувальних даних, що дозволить врегулювати ваги залежно від особливостей зображень, котів, що на них представлені.

Підсумовуючи, можна сказати, що у згорткових нейронних мереж є значні переваги. Це – один з найкращих алгоритмів для розпізнавання та класифікації зображень. Така мережа має меншу кількість ваг, що можна налагоджувати, на відміну від інших алгоритмів, що змушує її ж узагальнювати інформацію про зображення, а не рахувати кожен піксель. Навчання зручно проводити на графічному процесорі, є засоби для паралелізації і, як наслідок, пришвидшення процесу. CNN відносно непогано розпізнає зображення. що повертали або здвигали. Етап корекції помилок також пришвидшує навчання та робить його більш якісним.

Проте, під час використання стало зрозуміло, що у мережі надто багато параметрів. Нажаль, з’ясувати їх призначення точно важко, тому що нема розуміння, як саме вплинуть ці налаштування на роботу алгоритму. Так, до них можна віднести: кількість шарів, розмірність ядра згортки для кожного з шарів, кількість ядер для кожного з шарів, крок зсуву ядра при обробці шару, необхідність шарів з’єднання, ступінь зменшення ними розмірності, вибір функції по зменшенню розмірності, функція активації нейронів, наявність і параметри іншої нейронної мережі після роботи згорткової. Ті, хто застосовує ConvNet, змушені підбирати усе це емпірично. Хоча існують деякі вдалі інструкції щодо встановлення параметрів, за яких мережа працює гарно, але ми не можемо заперечувати, що не існує повноцінних рекомендацій щодо створення архітектури згорткових нейронних мереж під власні задачі.

Роботу рекомендовано для ознайомлення з практичним застосуванням вищеописаних методів задля класифікації зображень. Сфера застосування – визначення порід, фелінологія.

Результати атестаційної роботи були апробовані на 23-ому міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті». в 2019 р.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., & Montreal, U. (2008). Advances in neural information processing systems 19. Chinese Medical Ethics, 23, 80-83.

2. Bodyanskiy, Y. V., Tyshchenko, O. K., & Mashtalir, S. V. (2019, June). Fuzzy Clustering High-Dimensional Data Using Information Weighting. In International Conference on Artificial Intelligence and Soft Computing (pp. 385-395). Springer, Cham.

3. Bodyanskiy, Y., Grimm, P., Mashtalir, S., & Vinarski, V. (2010, July). Fast training of neural networks for image compression. In Industrial Conference on Data Mining (pp. 165-173). Springer, Berlin, Heidelberg.

4. Bodyanskiy, Y., Shafronenko, A., & Mashtalir, S. (2019, May). Online Robust Fuzzy Clustering of Data with Omissions Using Similarity Measure of Special Type. In International Scientific Conference “Intellectual Systems of Decision Making and Problem of Computational Intelligence” (pp. 637-646). Springer, Cham.

5. Bogucharskiy, S., & Mashtalir, V. (2015, September). Image segmentation via X-means under overlapping classes. In 2015 Xth International Scientific and Technical Conference" Computer Sciences and Information Technologies"(CSIT) (pp. 45-47). IEEE.

6. Dönmez, P. (2013). Introduction to Machine Learning, by Ethem Alpaydın. Cambridge, MA: The MIT Press2010. ISBN: 978-0-262-01243-0. $54/£ 39.95+ 584 pages. Natural Language Engineering, 19(2), 285-288.

7. Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.

8. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

9. Grimm, P., Kinoshenko, Y., Mashtalir, S., & Shlyahov, V. Metrical properties of nested partitions for image retrieval. In Information technology and

electrical engineering-devices and systems, materials and technologies for the future (Vol. 54).

10. Kim, M.S. (2010). Statistical Classification. URL: http://pages.pomona.edu/~jsh04747/Student%20Theses/MinsooKim10.pdf.

11. Kinoshenko, D., Mashtalir, S., & Shlyakhov, V. TEMPORAL VIDEO SEGMENTATION VIA SPATIAL IMAGE SEGMENTATION.

12. Kinoshenko, D., Mashtalir, S., Shcherbinin, K., & Yegorova, E. (2007). IMAGE PARTITION TRANSFORMS FOR FAITHFUL SEGMENTATION SEARCH. In International Conference «Information Research & Applications»-i. Tech (p. 1).

13. Lyashenko, V., Kobylin, O., & Baranchykov, Y. (2018, October). Ideology of Image Processing in Infocommunication Systems. In 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T) (pp. 47-50). IEEE.

14. Mashtalir, S., & Mashtalir, V. (2016, August). Sequential temporal video segmentation via spatial image partitions. In 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP) (pp. 239-242). IEEE.

15. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2017). Video shot boundary detection via sequential clustering. International Journal of Information Theories and Applications, 24, 50-59.

16. Mashtalir, V. P., Shlyakhov, V. V., & Yakovlev, S. V. (2014). Group structures on quotient sets in classification problems. Cybernetics and Systems Analysis, 50(4), 507-518.

17. Mashtalir, V., & Bogucharskiy, S. (2015). Covering Image Segmentation via Matrix X-means and J-means Clustering. Sensors & Transducers, 195(12), 56-61.

18. Mills, P. (2011). Efficient statistical classification of satellite measurements. International Journal of Remote Sensing, 32(21), 6109-6132.

19. Mitchell, T. M. (2010). Generative and discriminative classifiers: Naive bayes and logistic regression. Machine learning, 1-17.

20. Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning. MIT press.

21. Nagi, J., Ducatelle, F., Di Caro, G. A., Cireşan, D., Meier, U., Giusti, A., ... & Gambardella, L. M. (2011, November). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA) (pp. 342-347). IEEE.

22. Nakamoto, P. (2017). Neural Networks and Deep Learning: Deep Learning explained to your granny A visual introduction for beginners who want to make their own Deep Learning Neural Network. CreateSpace Independent Publishing Platform.

23. Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA, USA:: Determination press.

24. Ozer, R. (2017). Advanced Python Programming: The Insider Guide to Advanced Python Programming Systems.

25. Prabhu, R. Understanding of Convolutional Neural Network (CNN)— Deep Learning.

26. Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.

27. Ripley, B. D. (2007). Pattern recognition and neural networks. Cambridge university press.

28. Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited.

29. Saha, S. (2018). A comprehensive guide to convolutional neural networks–the ELI5 way.

30. Santos, L.A. Artificial Intelligence, GitBook. URL: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/.

31. Yuan, G. X., Ho, C. H., & Lin, C. J. (2012). Recent advances of large-scale linear classification. Proceedings of the IEEE, 100(9), 2584-2603.

32. Айвазян, С. А., Бухштабер, В. М., Енюков, И. С., & Мешалкин, Л. Д. (1989). Прикладная статистика: Классификация и снижение размерности.

33. Богучарский, С. И., & Машталир, С. В. (2015). Модифицированный метод кластеризации Х-средних в задачах сегментация изображений. Електротехнічні та комп'ютерні системи, (20), 106-110.

34. Богучарский, С. И., & Машталир, С. В. (2014). Кластеризация коллекций изображений в больших базах данных на основе рекурентной оптимизации.

35. Богучарский, С. И., Каграманян, А. Г., & Машталир, С. В. (2014). Иерархическая агломеративная кластеризация изображений в больших базах данных. Системи обробки інформації, (8), 93-97.

36. Богучарский, С. И., Каграманян, А. Г., & Машталир, С. В. (2015). Модификация метода J-средних в задачах фрагментной сегментации изображений.

37. Егорова, Е. А., Киношенко, Д. К., Машталир, С. В., & Шляхов, Д. В. (2006). Метрическое сравнение результатов сегментации изображений. Радиоэлектроника и информатика, (2).

38. Машталир, С. В., & Сакало, Е. С. (2008). Адаптивное нейросетевое сжатие сигналов большой размерности на основе взвешенного информационного критерия.

39. Путятин, Е. П. (2018). Нормализация и распознавание изображений. URL: http://sumschool. sumdu. edu. ua/is-02/rus/lectures/pytyatin/pytyatin. htm.

40. Четирбок, П. В. (2015). Розпізнавання об'єктів на основі векторної міри близькості образів у просторі похибок (Doctoral dissertation, Національний технічний університет України «Київський політехнічний інститут).