ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Технологія оптимізації нейромережних класифікаторів на основі AutoKeras»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Власенко Єгор Сергійович
Керівник: Слюсар Вадим Іванович
Рецензент: Муравльов Володимир Вячеславович

Полтава – 2024 року

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується масштабним використанням штучного інтелекту в значній кількості галузей людської діяльності: від медицини та фінансів до промисловості та освіти. Зростаючі потреби в практичних кейсах реалізації інтелектуального функціоналу для класифікації об'єктів за допомогою нейронної мережі, вимагає з одного боку скорочення часу на їх розробку, а з іншого – підвищення якості класифікації об'єктів. Для автоматизації та спрощення процесу створення моделей машинного навчання можуть використовуватись засоби Automated Machine Learning (AutoML). Однак, питання розробки рекомендацій щодо використання AutoKeras для оптимізації нейромережних класифікаторів потребують додаткових досліджень.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної ініціативної тематики «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» (ДРН 0123U105060, 2023-2028 рр.), що реалізується на кафедрі інформаційних систем та технологій, тематиці досліджень навчально-дослідної лабораторії інтелектуальних систем, комп'ютерних мереж та інтернет речей кафедри інформаційних систем та технологій Полтавського державного аграрного університету.

Метою кваліфікаційної роботи є поліпшення якості класифікації об'єктів за допомогою нейронної мережі на основі використання AutoKeras.

Завданнями кваліфікаційної роботи є:

– визначення особливостей застосування AutoKeras для оптимізації нейромережного класифікатора зображень;

– порівняльній аналіз моделей нейромережного класифікатору зображень;

– формування рекомендацій щодо застосування AutoCeras для синтезу класифікаторів.

Об'єктом дослідження є процес навчання та функціонування нейронних мереж.

Предметом дослідження є нейронні мережі різної архітектури, що застосовуються для класифікації зображень.

Методами дослідження є аналітичний, інформаційно-пошуковий, методи синтезу та навчання нейронних мереж класифікації зображень, робота з фреймворком AutoKeras.

Інформаційна база кваліфікаційної роботи сформована з ресурсів, що містять інформацію про маши, інструментарій для виконання глибокого машинного навчання.

Елементи наукової новизни роботи полягають у створенні архітектур нейронних мереж для класифікації зображень.

Практична значущість полягає в розробці рекомендацій щодо застосування AutoCeras для синтезу класифікаторів, які можуть бути використані для подальших досліджень за даною тематикою та при проектуванні хмарних сервісів.

Апробація результатів відбувалася за матеріалами VI Міжнародної студентської наукової конференції «Тренди та перспективи розвитку мультидисциплінарних досліджень» (жовтень 2024 р., м. Кременчук), науково-практичної конференції за підсумками проходження виробничої практики здобувачів вищої освіти спеціальності 126 Інформаційні системи та технології (16 жовтня 2024 р., м. Полтава).

Структура кваліфікаційної роботи логічно пов'язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 65 сторінок формату А4. Вона містить 37 рисунків.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ НЕЙРОМЕРЕЖНИХ КЛАСИФІКАТОРІВ ОБ'ЄКТІВ

1.1 Загальні відомості про нейромережних класифікатори об'єктів

Певним компромісом між параметричним і метричними методами є використання для рішенні завдань класифікації нейронних мереж. Нейронні мережі належать до непараметричних моделей, які не потребують припущень щодо ймовірнісного розподілу даних і не спираються на відстані як ключовий показник. Завдяки цьому вони є універсальними інструментами для класифікації, дозволяючи знаходити рішення навіть у тих випадках, де інші методи, такі як параметричні або метричні, виявляються неефективними.

Класифікація займає важливе місце серед завдань інтелектуального аналізу даних. Для її реалізації використовуються спеціальні математичні моделі, відомі як класифікатори. Популярність цього підходу пояснюється відносною простотою його алгоритмів і методів, а також високою результативністю в порівнянні з іншими інструментами аналізу [1].

На сьогоднішній день існує широкий спектр класифікаторів, що ґрунтуються як на статистичних методах (зокрема, логістичній регресії чи аналізі дискримінанта), так і на алгоритмах машинного навчання (нейронні мережі, дерева рішень, метод найближчих сусідів, машини опорних векторів тощо).

Використання різноманітних підходів до класифікації зумовлене специфікою задач, які необхідно вирішувати. Наприклад, ці завдання можуть відрізнятися кількістю класів (бінарна чи багатокласова класифікація), обсягом і якістю даних або їхньою розмірністю. Така варіативність вимагає уважного вибору класифікатора, що найбільш повно відповідає характеристикам конкретної проблеми, адже це безпосередньо впливає на точність отриманого результату. Різні види класифікаторів мають свої

переваги і недоліки. Так, класифікатори, в яких використовуються методи статистики мають хорошу математичну обгрунтованість, але при цьому складні у використанні і вимагають знання імовірнісного розподілу початкових даних і оцінки його параметрів (тому їх називають параметричними), а також мають фіксовану структуру моделі. Окрім цього, статистичні методи оцінюють тільки вірогідність належность об'єкту класу, але не пояснюють чому.

Класифікатори, грунтовані на машинному навчанні не вимагають оцінки параметрів розподілу початкових даних, а міра схожості в них формалізується за допомогою функції відстані. Такі класифікатори називаються метричними. Як правило, вони простіше в реалізації і використанні, чим параметричні, а їх результати зручніше для інтерпретації і розуміння. Але при цьому метричні класифікатори є евристичними моделями – забезпечують рішення тільки в обмеженому числі практично значимих випадків, можуть дати неточне або не єдине рішення.

Слід зазначити, що задача класифікації для НМ, взагалі кажучи, не являється основною (як, наприклад, для дерев рішень або алгоритму k найближчих сусідів). Спочатку, основним завданням для НМ є чисельний результат, коли на вході і виході моделі числові значення, що іноді не зовсім коректно називають регресією [2].

Нейронні мережі можуть бути адаптовані для роботи з категоріальними даними шляхом перетворення категорій у числові значення. Це дозволяє таким моделям приймати категорії як вхідні дані і генерувати відповідні категоріальні виходи.

Серед основних переваг нейронних мереж як класифікаторів можна відзначити їх здатність до самонавчання, що мінімізує потребу в втручанні користувача. Вони також виступають універсальними апроксиматорами, здатними точно моделювати будь-яку безперервну функцію. Крім того, нелінійність нейронних мереж робить їх ефективними для вирішення завдань класифікації навіть у випадках, коли класи не піддаються лінійному

розділенню. Завдяки гнучкості архітектури, нейронні мережі можуть бути пристосовані до вирішення найрізноманітніших завдань. Окрім цього, вони демонструють високу стійкість до шумів у вхідних даних, що підвищує точність результатів. (рис. 1.1).

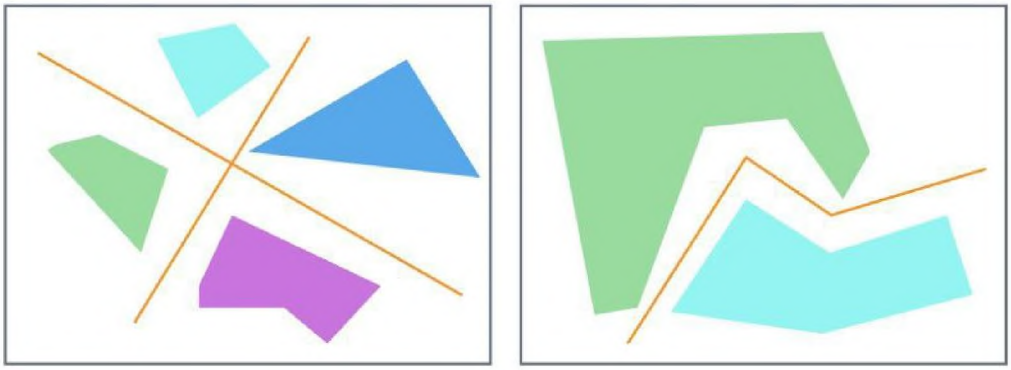

Рисунок 1.1 – Лінійна роздільність класів

Певним компромісом між параметричним і метричними методами є використання для рішенні завдань класифікації нейронних мереж (НМ). Дійсно, НМ є належать до класу непараметричних моделей, які не передбачають припущень щодо ймовірнісного розподілу даних і не використовують відстань як основний критерій. Завдяки цьому вони виступають універсальними класифікаторами, здатними забезпечувати результати навіть у тих ситуаціях, де параметричні та метричні підходи не дають задовільного розв’язання задачі [3].

1.2 Засоби автоматизації проєктування нейромереж

Алгоритми машинного навчання являються невід'ємною частиною великого числа сучасних інформаційних систем. Вони стали ключовим інструментом, вживаним в найрізноманітніших областях, від охорони здоров'я і фінансів до маркетингу і робототехніки. Вони дозволяють вирішувати

завдання класифікації, регресії, прогнозування і багато що інше. Проте створення і оптимізація моделей машинного навчання вимагають глибоких знань, часу і ресурсів. Це особливо важливо для організацій, що не мають великого досвіду в області даних або достатньої кількості фахівців. У таких випадках стає незамінним автоматизоване машинне навчання.

Автоматизоване машинне навчання є підходом, спрямованим на спрощення процесів розробки моделей машинного навчання, автоматизуючи ключові етапи, які раніше вимагали участі експертів. AutoML включає методи і інструменти, які дозволяють користувачам мінімізувати ручні налаштування і скоротити складність розробки моделей. Головна мета AutoML – зробити машинне навчання доступним для широкої аудиторії, включаючи фахівців з різних областей, що не обов'язково мають глибокі знання в області програмування або статистики [4].

AutoML – це набір концепцій і методів, що використовуваних для автоматизації процесів формування моделей машинного навчання, проходять через наступні етапи:

- попередня обробка даних;
- виділення ознак;
- вибір функцій;
- вибір найбільш відповідного алгоритму;
- налаштування гіперпараметрів.

Пошук нейронної архітектури – це процес автоматизації проектування нейронних мереж. Для цих цілей використовуються алгоритми, що навчаються або еволюційні. У навчанні з підкріпленням моделі на низьку точність та високу. Використовуючи цю техніку, модель завжди прагнутиме отримати високу точність. Пошук нейронної архітектури показує найкращі результати для вирішення задач, що вимагають виявлення нових архітектур.

Попередня обробка даних (Data Preprocessing) є ключовим етапом у підготовці даних до аналізу чи моделювання, оскільки від її якості залежить точність і ефективність роботи алгоритмів.

Цей етап має критичне значення, оскільки дані, як правило, містять шум, пропуски або надмірну інформацію. Автоматизація обробки даних включає наступні дії:

– автоматичне виявлення і усунення пропущених або помилкових значень;

– заміна пропусків середнім, медіанним значенням або спеціальною ознакою (наприклад, "Немає даних");

– видалення дублікатів записів.

Нормалізація і стандартизація – це процес автоматизації приведення даних до єдиного масштабу, наприклад, нормалізація в діапазоні від 0 до 1, що підвищує стабільність роботи алгоритмів.

Обробка викидів передбачає автоматичне виявлення аномальних даних, які можуть спотворити результати навчання, зокрема за допомогою методів, аналізують розподіл даних і виділяють значення, що сильно відрізняються від середніх показників.

Кодування категоріальних даних:

– створення бінарних ознак для категоріальних змінних;

– заміна категорій числовими значеннями.

Розділення даних відбувається таким чином, вибірка розбивається на повчальну, валідаційну і тестову частині (наприклад, 70/15/15).

Виділення ознак (Feature Engineering) створення нових, інформативних характеристик із наявних даних для підвищення ефективності алгоритмів машинного навчання.

Автоматичне створення нових ознак з початкових даних є одним із найскладніших і креативних завдань у сфері машинного навчання. Цей процес включає генерацію нових ознак шляхом перетворення існуючих даних для створення більш інформативних характеристик. Наприклад, для тимчасових даних можна створювати ознаки, які представляють дні тижня, місяці або певні часові інтервали. Для географічних даних часто використовують розрахунок відстані між точками або кластеризацію на основі регіонів.

Також важливим аспектом є збагачення даних, яке передбачає використання зовнішніх джерел, таких як погодні умови або економічні показники, для покращення якості аналізу. Додатково застосовується вибір функцій (Feature Selection), який спрямований на зменшення розмірності даних, прискорення обчислень і покращення точності передбачень. На цьому етапі автоматизовані інструменти відбирають найбільш релевантні ознаки, що дозволяє підвищити продуктивність моделі [5].

Основними методами вибору ознак є аналіз їх важливості, видалення нерелевантних ознак і використання методів регуляризації. Аналіз важливості ознак може здійснюватися за допомогою вбудованих методів, таких як визначення важливості ознак у випадковому лісі (Random Forest) або застосування підходу SHAP (SHapley Additive exPlanations), який оцінює вклад кожної ознаки в результат моделі.

Видалення нерелевантних ознак передбачає автоматичне виключення характеристик, які мають низьку кореляцію з цільовою змінною, що допомагає зменшити розмірність даних і спростити модель.

Методи регуляризації, такі як L1-регуляризація, використовуються для відсіювання зайвих ознак шляхом накладання штрафу на їх коефіцієнти. Це дозволяє уникнути переобучення і зосередитися на найбільш важливих даних для прогнозування.Вибір найбільш відповідного алгоритму.

Цей етап автоматизує завдання вибору алгоритму, відповідного для конкретного завдання (класифікація, регресія, кластеризація і так далі). Процес включає генерацію і тестування безлічі моделей.

AutoML проводить порівняльний аналіз різних алгоритмів, таких як:

– лінійні моделі (логістична регресія, лінійна регресія);

– нелінійні моделі (дерева рішень, випадковий ліс, градієнтний бустинг);

– глибоке навчання (нейронні мережі для складних завдань).

Автоматичний вибір метрик для оцінки якості моделі, таких як:

– точність (Accuracy) для класифікації;

– середньоквадратична помилка (MSE) для задач регресій;

– ROC-AUC, F1-score, Precision, Recall для завдань з незбалансованими класами.

Для невеликих наборів даних перевага віддається простим алгоритмам, тоді як для складних завдань використовуються потужні методи, наприклад, ансамблеві моделі.

Налаштування гіперпараметрів (Hyperparameter Tuning), цей етап визначає оптимальні значення параметрів алгоритму для досягнення найкращого результату.

Сітковий пошук (Grid Search) дійснює перебір всіх можливих комбінацій заданих параметрів моделі, щоб знайти найкращі значення для досягнення оптимальних результатів. Повний перебір усіх можливих комбінацій параметрів в заданому діапазоні.

Випадковий пошук (Random Search) дозволяє швидше знаходити оптимальні гіперпараметри порівняно з сітковим пошуком.

Баєсова оптимізація – це метод, який використовує моделі для більш інтелектуального пошуку оптимальних рішень. Вона дозволяє ефективно знаходити найкращі параметри для задач, де обчислювальні ресурси обмежені або тестування різних варіантів є дуже витратним.

Метод Hyperband:

– адаптивний розподіл обчислювальних ресурсів для прискорення налаштування;

– переваги детальної автоматизації;

– зменшує вірогідність помилок, пов'язаних з ручним налаштуванням;

– економить час фахівців;

– забезпечує кращу модель, що інтерпретується.

Можливість побудови і застосування моделей машинного навчання без навичок програмування, предметної експертизи і математичного аналізу.

Можливість автоматизованої обробки первинних даних, самостійного визначення вирішуваної задачі, вибору оптимальної нейромережевої

архітектури, оптимізації гіперпараметрів алгоритму навчання та вирішення інших завдань, включаючи візуалізацію, дозволяє знижувати зміщення та дисперсію моделей. Це також сприяє зменшенню часу, необхідного для розробки та тестування моделей, що робить процес побудови та налаштування більш ефективним і менш ресурсоємним.

Недоліки AutoML:

– новизна концепції несе в собі відповідні ризики і вимагає обережності при застосуванні останніх версій бібліотек;

– рішення AutoML являються дуже ресурсомісткими і вимагають залучення хмарних обчислень, так як час їх роботи на локальних комп'ютерах досить великий.

Розглянемо деякі існуючі рішення, сумісні з мовою Python, що дозволяють здійснювати вибір моделі і пошук гіперпараметрів, але кожен інструмент має свої особливості.

H2O – це платформа машинного навчання з відкритим початковим кодом. Вона доступна на Python. Цей пакет забезпечує підтримку методів машинного навчання з використанням швидких алгоритмів, і з 2018 року він отримав підтримку глибокого навчання. На тлі популяризації Apache Spark, H2O обзавівся інтерфейсом Sparkling Water для об'єднання можливостей Apache Spark і H2O. H20.ai є абсолютним лідером за швидкістю виконання, але продукт дає відносно середню точність.

H2O.ai активно використовується в банківській сфері, страхуванні, охороні здоров’я, ритейлі, телекомунікаціях та інших галузях для вирішення завдань прогнозування, оптимізації та виявлення шахрайства.

Інструмент оптимізації на основі дерева (TPOT) бібліотека для автоматичного машинного навчання, яка використовує еволюційні алгоритми для автоматичного пошуку оптимальних моделей [6].

Метою TPOT є автоматизація побудови конвеєрів ML шляхом об'єднання гнучкого представлення дерева конвеєрів виразів з алгоритмами пошуку, такими як генетичне програмування. TPOT використовує основану на

Python бібліотеку scikit-learn в якості меню ML. Пакет не уміє взаємодіяти з звичайною мовою і категоріями ознак.

Основна ідея TPOT – автоматизувати процес вибору, налаштування і комбінування моделей і методів попередньої обробки даних, щоб прискорити розробку рішень.

TPOT підходить для завдань швидкого дослідженнях, особливо якщо ви хочете заощадити час на підборі алгоритмів і їх налаштування.

TPOT має кілька обмежень, які варто враховувати. По-перше, процес оптимізації може бути ресурсомістким, особливо для великих наборів даних, що може ускладнити його використання на обмежених обчислювальних потужностях. По-друге, інструмент обмежений лише тими моделями та трансформерами, які надає бібліотека scikit-learn, що може звузити вибір доступних методів. Ще одним недоліком є відсутність підтримки категорійних змінних – перед використанням TPOT їх необхідно попередньо закодувати, наприклад, за допомогою one-hot encoding.

Незважаючи на ці обмеження, TPOT чудово підходить для швидкого створення базових моделей і автоматичного підбору гіперпараметрів, що позбавляє необхідності налаштовувати їх вручну. Він також стане корисним у дослідженнях великих наборів даних, де потрібно швидко знаходити ефективні рішення. Крім того, цей інструмент є оптимальним вибором у ситуаціях, коли обмежений час на розробку вимагає автоматизації процесу побудови та налаштування моделей.

Cloud AutoML – це набір продуктів для машинного навчання, який дозволяє розробникам з обмеженим досвідом в області ML навчати високоякісні моделі, що відповідають бізнес потребам, використовують передові технології навчання Google і технологію пошуку нейронної архітектури. Як стверджують розробники, готове для промислової експлуатації рішення можна отримати впродовж робочого дня. Має простий призначений для користувача інтерфейс і може використовуватися

безкоштовно впродовж року. Для комерційних рішень продукт являється платним.

Cloud AutoML від Google Cloud є потужним інструментом для автоматизації процесів машинного навчання, який робить передові технології доступними навіть для користувачів з мінімальним досвідом в цій області. Цей набір продуктів дозволяє навчати високоякісні моделі, що відповідають специфічним бізнес-потребам, використовуючи інфраструктуру і алгоритми Google. Завдяки технології пошуку нейронної архітектури Cloud AutoML автоматично знаходить оптимальну структуру моделі, що спрощує процес і скорочує час розробки.

Серед ключових переваг Cloud AutoML – інтуїтивно зрозумілий призначений для користувача інтерфейс, який робить процес налаштування і навчання моделей зручним навіть для непрофесіоналів. Сервіс орієнтований на швидке впровадження рішень, дозволяючи отримати готову модель впродовж декількох годин або робочого дня. При цьому користувачі мають можливість безкоштовно тестувати продукт впродовж року, що робить його привабливим для досліджень і початкових етапів розробки. Проте для комерційного використання продукт надається на платній основі, з можливістю масштабування залежно від вимог бізнесу.

Cloud AutoML інтегрується з іншими сервісами Google Cloud, що спрощує обробку даних, розгортання моделей і їх інтеграцію в існуючі бізнес-процеси. Це рішення ідеально підходить для завдань, що вимагають мінімального часу на налаштування і високу точність, наприклад, класифікації зображень, аналізу тексту або обробки природної мови.

Auto-Sklearn – це автоматизований пакет машинного навчання, грунтований на scikit-learn. Уміє генерувати ознаки на основі сирих даних. Переваги фреймворка у бейсовій мові оптимізації, метанавчанні і ансамблевій побудові. Ефективно працює тільки з невеликими датасетами. По відгуках користувачів, Auto-Sklearn дає велику точність, але є найповільнішим серед конкурентів. Інструмент краще всього показує себе при роботі з невеликими

наборами даних, де він може продемонструвати високу точність завдяки своїй інтелектуальній оптимізації. Проте користувачі відмічають, що Auto-Sklearn значно повільніше порівняно з конкурентами, такими як TPOT або H2O AutoML, із-за складних алгоритмів оптимізації і великого пошуку по простору моделей.

Незважаючи на повільність, Auto-Sklearn особливо корисний в тих випадках, коли важлива точність моделей і є можливість чекати завершення процесів оптимізації. Він підтримує роботу з різними алгоритмами машинного навчання, доступними в scikit-learn, що робить його гнучким інструментом для завдань класифікації і регресії.

1.3 Аналіз особливостей фреймворку AutoKeras

AutoKeras – це потужна програмна бібліотека з відкритим початковим кодом, створена для автоматизації завдань машинного навчання і глибокого навчання. Основна мета AutoKeras – зробити технології глибокого навчання доступними для фахівців з мінімальним досвідом роботи з великими даними і машинним навчанням, усуваючи необхідність в детальному налаштуванні архітектури нейронних мереж і підборі гіперпараметрів. Проект розроблений лабораторією DATA Lab Техаського університету A&M спільно з учасниками співтовариства, що гарантує активний розвиток і підтримку.

AutoKeras унікальний завдяки своїй здатності автоматизувати складні процеси глибокого навчання, які традиційно вимагають глибоких знань в області машинного навчання і значного досвіду роботи з нейронними мережами. Ця бібліотека робить доступним потужний інструмент для тих, хто хоче використати переваги глибокого навчання, але не має можливості вручну настроювати моделі або розробляти архітектуру з нуля.

Одним з ключових чинників, які виділяють AutoKeras, є його використання технології пошуку нейронної архітектури (NAS, Neural

Architecture Search). Цей підхід дозволяє автоматично знаходити оптимальну архітектуру нейронних мереж для заданого завдання. У традиційному процесі розробки машинного навчання розробник мережі вручну підбирає шари, їх кількість, послідовність, параметри активації, розмір фільтрів і багато інших елементів. Це вимагає значних тимчасових і інтелектуальних витрат. AutoKeras бере на себе це завдання, виконуючи автоматизований пошук з використанням методів оптимізації і експериментуючи з тисячами можливих конфігурацій, щоб знайти найбільш ефективне вирішення.

Крім того, AutoKeras фокусується на оптимізації гіперпараметрів моделі. Цей процес включає вибір оптимальних значень для параметрів, які безпосередньо впливають на процес навчання і продуктивність моделі, таких як швидкість навчання, кількість епох, розмір міні-батча, тип оптимізатора і інші. За рахунок автоматизації підбору гіперпараметрів бібліотека дозволяє користувачам добитися високої якості моделі без необхідності в довгих експериментах.

Ще однією унікальною особливістю AutoKeras є його орієнтованість на мультимодальні дані. У реальному світі дані часто є комбінацією різних типів, наприклад, зображення, текст і числові табличні дані. AutoKeras автоматично визначає, як обробляти і інтегрувати ці різні типи даних в одну модель, що робить його особливо потужним інструментом для комплексних завдань, які важко вирішити з використанням традиційних підходів [7].

Крім того, AutoKeras пропонує високу міру інтеграції з екосистемою TensorFlow. Це дозволяє користувачам експортувати навчені моделі і при необхідності допрацювати їх за допомогою стандартних інструментів TensorFlow або Keras. Наприклад, якщо знайдена AutoKeras архітектура вимагає незначної модифікації або тонкого налаштування, користувач може внести зміни вручну, не розпочинаючи процес навчання з нуля.

AutoKeras робить процес глибокого навчання інтуїтивно зрозумілим і доступним завдяки мінімалістичному інтерфейсу. Користувачам досить вказати завдання (наприклад, класифікація зображень, аналіз тексту або

прогнозування тимчасових рядів) і надати дані, після чого бібліотека сама виконує увесь цикл – від попередньої обробки даних до навчання і оцінки моделі. Такий підхід економить час і знижує вірогідність помилок, пов'язаних з людським чинником, роблячи AutoKeras універсальним інструментом для вирішення завдань машинного навчання.

У області обробки тексту AutoKeras надає інструменти для завдань класифікації, генерації тексту і аналізу чутливості, забезпечуючи гнучкість при роботі з текстовими даними. Також бібліотека застосовна для прогнозування тимчасових рядів, що робить її корисною при аналізі тимчасових даних для виявлення трендів і прогнозування майбутніх значень.

Унікальною особливістю AutoKeras є її здатність працювати з мультимодальними даними, тобто об'єднувати зображення, текст і табличні дані у рамках одного завдання. Це дозволяє користувачам вирішувати складні проблеми, що вимагають комплексного підходу до аналізу різнорідних джерел даних.

AutoKeras, незважаючи на свої вражаючі можливості, має декілька обмежень, які варто враховувати перед використанням. Однією з головних труднощів є його ресурсоємність. Процес автоматичного пошуку нейронної архітектури (NAS) і оптимізації гіперпараметрів вимагає значних обчислювальних потужностей. Це особливо помітно при роботі з великими наборами даних або при виконанні великої кількості експериментів, що може зробити процес тривалим і дорогим. Користувачі, у яких немає доступу до високопродуктивним GPU або хмарним обчислювальним платформам, можуть зіткнутися з труднощами у використанні усіх можливостей [8].

Автоматизація, яка є основною перевагою AutoKeras, може обернутися недоліком у разі простіших завдань. Процес пошуку архітектури і гіперпараметрів може бути невиправданий повільним в порівнянні з традиційним ручним налаштуванням, особливо якщо завдання не вимагає складної моделі. У таких випадках використання AutoKeras може привести до перевитрати часу і ресурсів.

Ще одне обмеження пов'язане з архітектурою і гіперпараметрами, які підтримує AutoKeras. Оскільки бібліотека побудована поверх TensorFlow, її функціонал обмежується тільки тією архітектурою, яка реалізована в екосистемі TensorFlow. Це може бути перешкодою для користувачів, яким потрібні специфічні моделі або підходи, що не входять в стандартний набір інструментів TensorFlow.

Крім того, бібліотека може зазнавати труднощі з обробкою сильно спеціалізованих даних або виконанням складних призначених для користувача вимог. Наприклад, якщо завдання вимагає нестандартних шарів нейронних мереж, складних методів регуляризації або специфічних метрик, AutoKeras не завжди здатний ефективно впоратися з такими сценаріями. Хоча бібліотека і дозволяє експортувати модель в TensorFlow для подальшого доопрацювання, цей процес може звести нанівець первинну мету автоматизації [9].

Ще однією потенційною проблемою є залежність від якості вхідних даних. AutoKeras може ефективно автоматизувати пошук архітектури і оптимізацію гіперпараметрів, але він не вирішує проблеми, пов'язані з низькою якістю даних, відсутністю їх достатнього об'єму або неправильною розміткою. У таких випадках навіть кращі алгоритми, створені бібліотекою, можуть демонструвати низьку продуктивність.

Для деяких завдань, де потрібно високу інтерпретованість моделі, AutoKeras може бути не кращим вибором. Автоматично створені моделі часто виявляються складними і такими, що мало інтерпретуються, що може бути недоліком в областях, де необхідно пояснювати логіку роботи моделі, наприклад, в медицині або фінансах.

Незважаючи на перераховані недоліки, AutoKeras залишається потужним інструментом, який знижує бар'єр входу в глибоке навчання, особливо для користувачів, які цінують зручність і простоту використання. Проте для завдань, що вимагають високої гнучкості, нестандартних підходів або глибокого контролю над процесом моделювання, можуть знадобитися додаткові налаштування або інші інструменти [10].

Висновок до розділу 1

Розгляд основ нейромережних класифікаторів, методів їх оптимізації, а також технологій автоматизованого проєктування нейромереж (AutoML) дозволяє отримати глибше розуміння сучасних підходів у машинному навчанні, зокрема в задачах класифікації. Ключову роль нейромережних класифікаторів у розв'язанні різноманітних задач, починаючи від медичної діагностики і закінчуючи фінансовими прогнозами. Нейромережі дозволяють знаходити складні залежності між даними та успішно застосовуються в тих сферах, де традиційні методи машинного навчання не здатні досягти таких результатів.

Проте, для досягнення високої точності та ефективності моделі, важливим є застосування методів оптимізації, зокрема таких, як градієнтний спуск, що дозволяє мінімізувати функцію втрат та оптимізувати параметри моделі. Важливим аспектом є також регуляризація, яка допомагає уникнути перенавчання моделі і зберегти її здатність. Правильна оптимізація моделей нейромереж є необхідною умовою для їхнього успішного застосування на реальних даних.

AutoML, значно спрощує процес створення, налаштування та оптимізації нейромереж. Вони дозволяють автоматично підбирати найбільш підходящі архітектури моделей для конкретних завдань, здійснювати пошук гіперпараметрів, а також проводити модернізацію на основі даних. В результаті, AutoML робить створення нейромереж доступним навіть для людей без глибоких знань в області машинного навчання, забезпечуючи високу ефективність та зручність у використанні.

Загалом, розвиток і впровадження технологій AutoML та бібліотек, таких як AutoKeras, забезпечують швидке й ефективне створення нейромереж, дозволяючи фахівцям з різних галузей зосередитися на вирішенні прикладних завдань, а не на технічних аспектах побудови моделей.

РОЗДІЛ 2
РОЗРОБКА ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ НЕЙРОМЕРЕЖНИХ КЛАСИФІКАТОРІВ НА ОСНОВІ AUTOKERAS

2.1 Вибір і налаштування гіперпараметрів нейромережного класифікатора зображень за допомогою AutoKeras

Гіперпараметричний пошук і оптимізація моделей є ключовими етапами в процесі машинного навчання.

При створенні моделей машинного навчання існує одна важлива складова, яка часто залишається за кадром, але має вирішальне значення для досягнення високої продуктивності і точності – це гіперпараметри.

Як архітектори будують основу для будівлі, так і вибір гіперпараметрів визначає фундамент для моделей машинного навчання. Гіперпараметри – це параметри, які налаштовуються до початку процесу навчання і визначають як саму структуру моделі, так і спосіб її навчання. Їх правильний вибір може значно вплинути на результати навчання, тоді як неправильно підібрані значення гіперпараметрів можуть привести до небажаних і недооцінених моделей.

Здатність моделі до узагальнення даних – ключовий аспект, що визначає її цінність в реальних завданнях. Гіперпараметри впливають на цю здатність, граючи роль регуляторів моделі. Правильно підібрані гіперпараметри допомагають збалансувати між підгонкою моделі під навчальні дані (перенавчання) і занадто узагальненим представленням, яке може упустити важливі закономірності (недонавчання).

Визначення гіперпараметрів і їх відмінність від параметрів моделі.

Гіперпараметри і параметри – це два ключові аспекти, які формують модель машинного навчання. Вони грають різні ролі в процесі навчання і впливають на поведінку і продуктивність моделі.

Гіперпараметри – це налаштування моделі, які визначають її загальну структуру і спосіб навчання. Ці параметри встановлюються до початку процесу навчання і не змінюються в процесі навчання моделі. Гіперпараметри роблять вплив на те, як модель навчатиметься, які ознаки враховуватимуться, і які обмеження будуть накладені на процес навчання. Прикладами гіперпараметрів може бути кількість шарів і нейронів в нейронній мережі, швидкість навчання, крок градієнтного спуску, коефіцієнт регуляризації [11].

Гіперпараметри виконують найважливішу роль в процесі машинного навчання, задаючи основу, на якій будується модель, її навчання і здатність до узагальнення. Ці параметри діляться на декілька категорій, кожна з яких впливає на різні аспекти моделі. Наприклад, архітектурні гіперпараметри визначають структуру моделі, кількість шарів, число нейронів в кожному шарі і організацію зв'язків між ними. Збільшення числа шарів може поліпшити здатність моделі до обробки складних даних, але вимагає обережного підходу, щоб уникнути перенавчання. Процес навчання залежить від гіперпараметрів, таких як швидкість навчання і розмір міні-батча. Швидкість навчання регулює, наскільки великими кроками модель змінює свої ваги в процесі оптимізації. Занадто висока швидкість може привести до нестабільності, тоді як низька уповільнить процес. Розмір міні-батча, розраховується, скільки прикладів використовується для одного кроку оновлення вагів, робить вплив на точність і обчислювальну ефективність. Наприклад, маленькі батчі забезпечують точніші градієнти, але вимагають більше часу на навчання.

Регуляризация також є важливим елементом управління моделлю. Вона допомагає запобігти перенавчанню, додаючи обмеження на величини вагів моделі або тимчасово відключаючи нейрони в процесі навчання (метод dropout).

Оптимізація гіперпараметрів, таких як вибір алгоритму для оновлення вагів (наприклад, Adam, SGD або RMSprop), дозволяє знайти баланс між точністю і швидкістю навчання. Різні оптимізатори мають свої переваги такі

як, Adam адаптує швидкість навчання для кожного параметра, роблячи процес стабільнішим на складних даних.

Обробка даних і правильне їх розділення на навчальну, валідаційную і тестову вибірки також підкорюється налаштуванню гіперпараметрів. Вибір методів попередньої обробки, таких як нормалізація або стандартизація, допомагає моделі краще розуміти структуру вхідних даних.

Підбір гіперпараметрів можна автоматизувати за допомогою інструментів Grid Search, Random Search, а також складніших методів, таких як баєсова оптимізація. Це робить процес налаштування ефективнішим, особливо для складних моделей, де вручну досліджувати усі комбінації параметрів практично неможливо.

Параметри моделі, з іншого боку, є внутрішніми вагами або коефіцієнтами, які модель навчає в процесі навчання на основі повчальних даних. Ці параметри змінюються в процесі навчання з метою мінімізації функції втрат і досягнення найкращої відповідності між прогнозами моделі і реальними значеннями цільової змінної. У разі нейронних мереж, параметри включають ваги між нейронами в різних шарах.

Відмінність між гіперпараметрами і параметрами моделі полягає в тому, що гіперпараметри задаються вручну до початку навчання і визначають характеристики усього процесу навчання, тоді як параметри моделі обчислюються в процесі навчання на основі даних і оптимізуються для досягнення найкращої продуктивності. Гіперпараметри можна порівняти з налаштуваннями інструменту, за допомогою якого створюється модель, а параметри моделі, – це результат роботи цього інструменту на конкретних даних.

Розуміння різниці між гіперпараметрами і параметрами моделі є ключовим для правильного налаштування моделей машинного навчання і досягнення оптимальних результатів [12].

Кожен алгоритм машинного навчання має унікальні характеристики, які вимагають точного налаштування гіперпараметрів для досягнення оптимальних результатів.

Гратчастий (Grid Search) передбачає систематичний перебір всіх можливих комбінацій заданих значень параметрів.

Один з найбільш поширених способів налаштування гіперпараметрів – це гратчастий підхід. Цей метод припускає завдання наборів значень для кожного гіперпараметра, які потім перебираються систематично для знаходження найкращої комбінації. Для кожної комбінації гіперпараметрів робиться навчання моделі і оцінка її продуктивності (рис.2.1-2.3) на валідаційних даних.

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 2.1 – Визначення параметрів і їх значень для перебору

```
rf_model = RandomForestClassifier()
grid_search = GridSearchCV(rf_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Рисунок 2.2 – Створення моделі і налаштування з використанням гратчастого пошуку

```
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

Рисунок 2.3 – Виведення найкращих гіперпараметрів і оцінки

Плюси гратчастого підходу в тому, що він гарантує повний перебір усіх заданих комбінацій гіперпараметрів, що може допомогти знайти найкращі значення. Проте це може бути дуже ресурсозатратним, особливо при великому числі гіперпараметрів і значень.

Випадковий пошук (Random Search) гіперпараметрів вибирає випадкові комбінації значень гіперпараметрів із заданого простору пошуку.

Інший метод – це випадковий пошук гіперпараметрів. Замість того щоб перебирати усі комбінації, випадковий пошук вибирає випадкові набори значень для кожного гіперпараметра. Цей метод може бути ефективнішим за часом (2.4-2.6), оскільки він зазвичай вимагає менше ітерацій, щоб знайти хороші значення.

```
param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 2.4 – Визначення діапазонів значень для випадкового пошуку

```
rf_model = RandomForestClassifier()
random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist
random_search.fit(X_train, y_train)
```

Рисунок 2.5 – Створення моделі і налаштування з використанням випадкового пошуку

```
print("Best Hyperparameters:", random_search.best_params_)
print("Best Cross-Validation Score:", random_search.best_score_)
```

Рисунок 2.6 – Виведення найкращих гіперпараметрів і оцінки

Випадковий пошук може бути ефективніший в пошуку оптимальних гіперпараметрів, особливо коли ресурси обмежені. Проте є вірогідність упустити деякі комбінації, які могли б бути кращими.

AutoKeras – це бібліотека з відкритим початковим кодом для Автоматизованого машинного навчання (Automated Machine Learning (AutoML). AutoKeras надає функції для автоматичного пошуку архітектури і гіперпараметрів моделей глибокого навчання.

Зараз вже звідусіль тільки і чутно, як використання технологій штучного інтелекту і машинного навчання застосовується для вирішення різноманітних завдань у бізнесі, медицині, творчості, робототехніці.

З іншого боку, вартість залучення програмістів продовжує рости, що дає серйозний поштовх розвитку систем AutoML, які стає усе більш популярними.

Компанія Google вже запустила свій сервіс для ШІ – публічно доступний хмарний сервіс для створення систем машинного навчання – Cloud AutoML.

Сервіс надає набір продуктів для машинного навчання і дозволяє користувачам без досвіду в області машинного навчання використати технології Google для пошуку нейронної архітектури, здатної відповідати їх бізнес потребам [13].

Працювати це повинно так, користувач завантажуєте туди свої дані (наприклад, картинки), а сервіс сам настроює модель (нейронну мережу) для розпізнавання образів на цих картинках.

Таким чином, AutoML надає доступні інструменти глибокого навчання, для користувачів, що має обмежені знання в області аналізу даних або машинного навчання.

AutoKeras, автоматично виконує пошук архітектури нейронної мережі Neural Architecture Search (NAS) – підбираючи її структуру і гіперпараметри для оптимального вирішення (рис.2.7-2.8) заданого завдання (класифікація, регресія).

```
1.  pip install autokeras
```

Рисунок 2.7 – Встановлення AutoKeras

```
1.  import autokeras as ak
2.
3.  clf = ak.ImageClassifier()
4.  clf.fit(x_train, y_train)
5.  results = clf.predict(x_test)
```

Рисунок 2.8 – Приклад застосування AutoKeras

Особливістю AutoKeras є його здатність адаптувати архітектуру моделі та налаштовувати гіперпараметри в залежності від специфіки задачі. Це значно скорочує час, необхідний для створення конкурентоспроможних моделей, та зменшує ризики, які можуть виникнути через людський фактор.

Таким чином, використання AutoKeras сприяє побудові ефективних моделей із високою точністю, що особливо важливо для складних задач сучасного машинного навчання [14].

2.2 Автоматизоване налаштування архітектури нейромережного класифікатору зображень

Автоматизація налаштування архітектури нейромережі, також відома як AutoML, є процесом, в якому використовуються алгоритми для автоматичного проектування, налаштування і оптимізації нейронних мереж. Це дозволяє значно скоротити час розробки, мінімізувати помилки, пов'язані з людським чинником, і підвищити якість моделей.

Ключовим елементом автоматизації є пошук архітектури нейромережі (Neural Architecture Search, NAS), де алгоритми визначають оптимальну структуру мережі, кількість шарів, їх типи, кількість нейронів і інші параметри. Для цього застосовуються методи, такі як еволюційні алгоритми, навчання з підкріпленням і баєсова оптимізація. Результатом є високоефективна архітектура, такі як EfficientNet.

Автоматизація включає і оптимізацію гіперпараметрів, наприклад, швидкості навчання, розміру батча і параметрів регуляризації. Використання таких інструментів, як Grid Search або Random Search, дозволяє знаходити найбільш ефективні налаштування. Важливу роль грає автоматичне проектування модулів мережі, наприклад згортальних блоків або механізмів уваги, а також налаштування етапів попередньої обробки даних, включаючи нормалізацію і аугментацію.

Переваги автоматизації очевидні, прискорення розробки, поліпшення продуктивності моделей і можливість застосування до широкого спектру завдань. Інструменти ніби Google AutoML або AutoKeras полегшують впровадження таких підходів навіть для користувачів з мінімальним досвідом в машинному навчанні. Автоматизація архітектури нейромережі стає важливим напрямом в розвитку штучного інтелекту, роблячи технології машинного навчання доступними і ефективними.

Активне впровадження ІТ в системи автоматизації будівель останнім часом спричинило нові можливості в частині використання сучасних засобів програмування і мов, зручній візуалізації для різних груп користувачів і відкрило шлях до швидкої обробки великих об'ємів даних від інженерного устаткування, що являється, по суті, елементом ШІ. Ці прогнозні методи управління інженерним устаткуванням будівель забезпечують перехід від інтелектуальної будівлі до навчаного або когнітивного.

Згаданий перехід відбувається за наявності ряду необхідних умов, до яких відносяться, принаймні, що відповідають будівельна інфраструктура, архітектура системи і додатка. Наявність цих умов дозволяє забезпечити перехід до системного навчання, що формує технології ШІ.

Архітектура системи – це платформа, що підтримує процеси навчання. Вона може бути як хмарною, так і серверною. Серверні мають більшу обчислювальну потужність, а хмарні володіють ширшим функціоналом Необхідно відмітити, що платформа ШІ будується на інфраструктурі системи

автоматичного управління будівлею, яка управляє інженерним устаткуванням будівлі і забезпечує контроль систем автоматизації приміщень (рис.2.9).

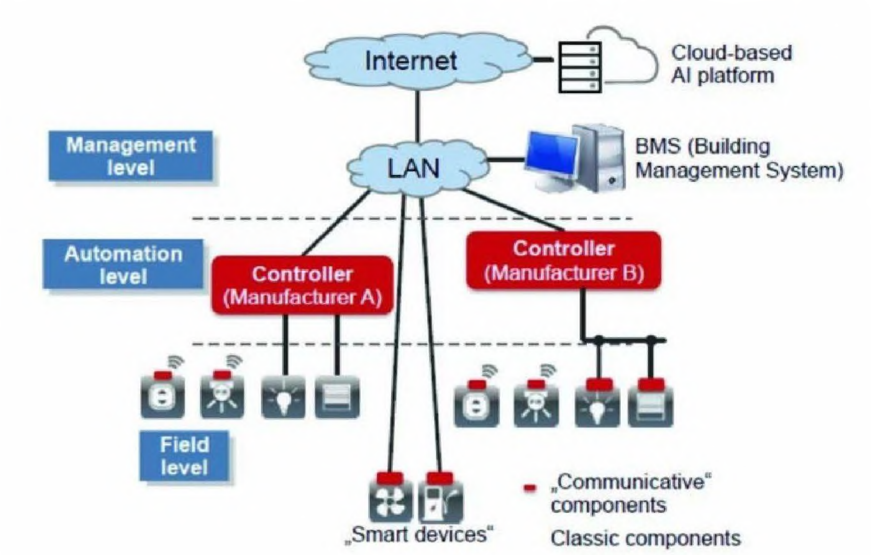

Рисунок 2.9 – Об'єднання автоматизації будівель з платформою (хмарною) штучного інтелекту

У цьому сенсі цікаво відмітити, що пропонована зарубіжними спеціалістами структура об'єднання автоматизації будівель з платформою хмарною штучного інтелекту багато в чому співпадає із структурою системи автоматизації будівлі, приведеної в стандарті ISO 16484-2 (рис.2.10). Обидві структури мають три рівні – польовий, апаратного управління і диспетчеризації (менеджменту), тільки при об'єднанні з хмарною платформою активніше використовуються хмарні технології, і на рівні менеджменту з'являється хмарна платформа ШІ, що забезпечує новий якісний рівень рішення, на якому можлива реалізація будівлі [15].

Це дозволяє інтегрувати новітні технології, такі як аналіз великих даних та прогнозування на основі штучного інтелекту, що значно покращує ефективність управління енергоспоживанням, безпекою та комфортом в будівлях. Завдяки використанню хмарних сервісів, система отримує можливість адаптуватися до змінних умов у реальному часі.

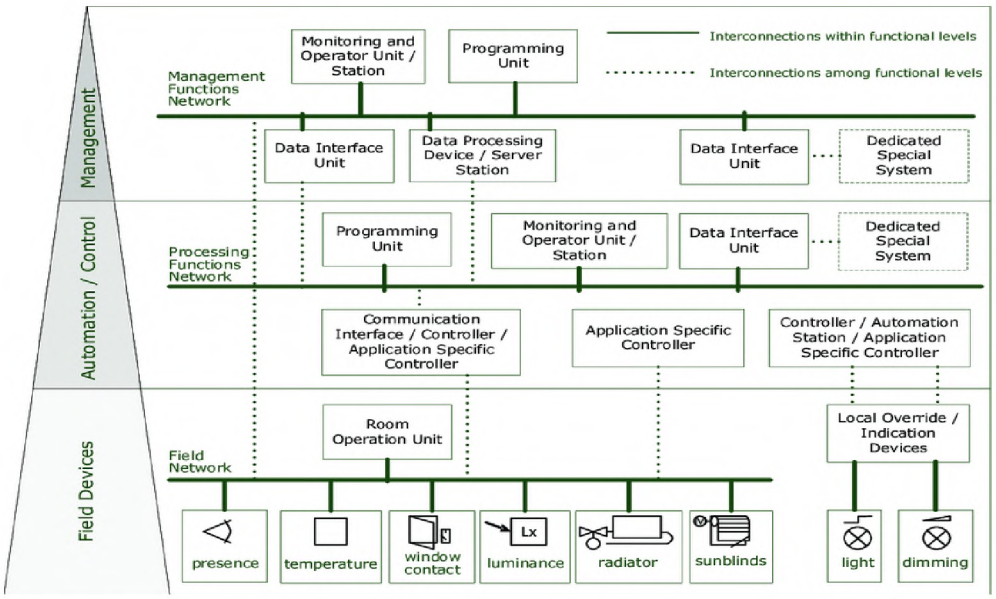

Рисунок 2.10 – Структура системи автоматизації будівлі за стандартом ISO 16484-2

Додатково, інтеграція машинного навчання в цю структуру відкриває можливості для більш складного аналізу даних та адаптивного управління будівлею. На польовому рівні алгоритми машинного навчання дозволяють аналізувати дані з датчиків у реальному часі, оптимізуючи роботу систем вентиляції, освітлення та енергопостачання. На рівні апаратного управління такі алгоритми сприяють створенню моделей прогнозування для запобігання аварійним ситуаціям та забезпечення довгострокової ефективності обладнання.

Хмарна платформа ШІ, з інтегрованими алгоритмами машинного навчання (рис.2.11), додає новий рівень функціональності, розпізнавання, передбачення енергоспоживання, оптимізація витрат ресурсів та персоналізоване налаштування умов для мешканців чи працівників. Завдяки цьому система автоматизації може не лише реагувати на зміни, але активно покращувати свою роботу, враховуючи накопичений досвід і дані.

Таким чином, завдяки інтеграції машинного навчання, система автоматизації будівлі стає більш інтелектуальною та самонавчальною, що дозволяє зменшити витрати на енергоспоживання, підвищити рівень комфорту для мешканців та знизити ризик технічних збоїв.

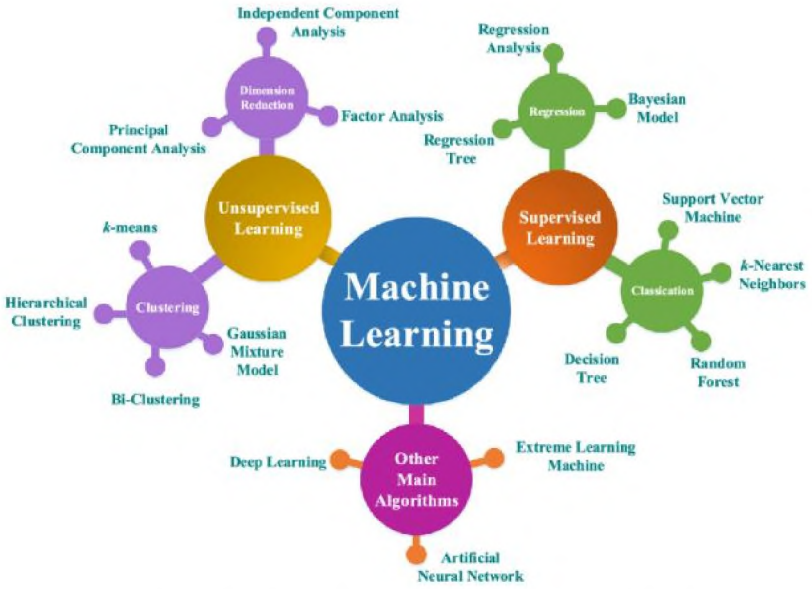

Рисунок 2.11 – Машинне навчання

Таким чином, поєднання автоматизації будівель, хмарного штучного інтелекту та машинного навчання дозволяє створювати більш ефективні, адаптивні й екологічні будівлі, які активно сприяють підвищенню комфорту користувачів та зменшенню експлуатаційних витрат. Крім цього, такі технології забезпечують можливість динамічного моніторингу та аналізу даних у режимі реального часу, що дозволяє швидше реагувати на зміни в умовах експлуатації. Наприклад, системи можуть виявляти аномалії у споживанні ресурсів або змінах мікроклімату та автоматично приймати рішення для їх усунення. Інтеграція технологій інтернету речей (IoT) дає змогу підключати до єдиної платформи різноманітні пристрої, забезпечуючи ще більш детальну аналітику і контроль [16].

2.3 Стратегії підвищення ефективності класифікації при використанні AutoKeras

Завдання класифікації фотографій в машинному навчанні і комп'ютерному зорі полягає в тому, щоб автоматично привласнювати кожній вхідній фотографії або зображенню одну або декілька категорій (класів) на

основі утримуваного зображення (рис.2.12). Це одне з ключових завдань в області комп'ютерного зору, яка має безліч практичних застосувань, таких як розпізнавання осіб, класифікація об'єктів, медична діагностика по зображеннях, фільтрація контенту.

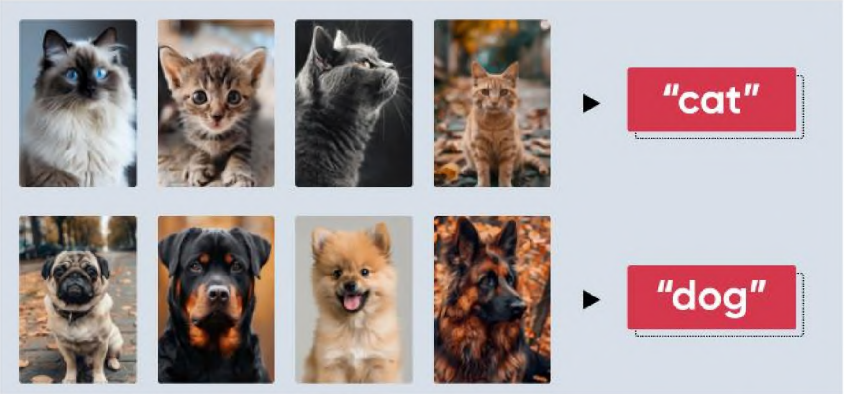

Рисунок 2.12 – Класифікація зображень на прикладі двох класів

Перший прототип (LeNet-1) з'явився в 1989 році, розроблений Яном Лекуном і його колегами для завдання розпізнавання рукописних цифр на наборі даних MNIST. До 1998 року з'явилася версія LeNet-5, що стала одним з перших успішних застосувань нейронних мереж для практичних завдань (згортальні і пулінгові шари можуть працювати разом для ознак і зменшення розмірності зображень, щоб ефективно розпізнавати об'єкти) і продемонструвала потужність глибокого навчання в комп'ютерному зорі. Так LeNet-5 є засадничою для подальших розробок в області комп'ютерного зору і глибинного навчання (рис.2.13).

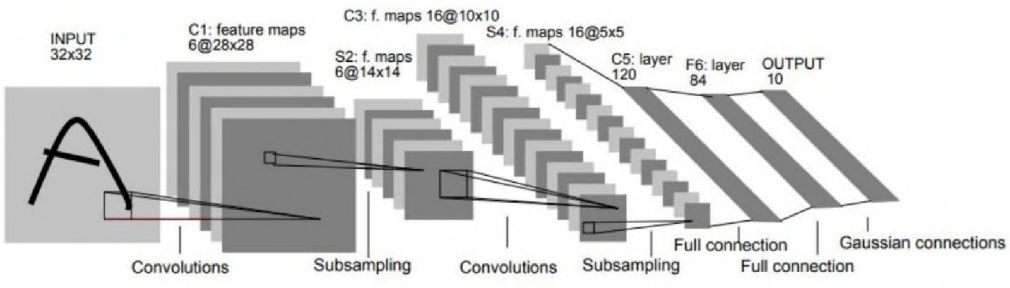

Рисунок 2.13 – Класифікація зображень версія LeNet-5

LeNet-5 стала однією з перших нейронних мереж, які застосовували згортальні шари для автоматичного виділення ознак із зображень, що дозволило відмовитися від традиційних методів ручної обробки ознак. Згортальні шари ефективно витягають ключові особливості зображення, а використання шарів пулінга після кожного згортального шару дозволяє зменшити просторовий дозвіл карти ознак. Це не лише знижує обчислювальні витрати і кількість параметрів моделі, але і підвищує стійкість мережі до зміщень і деформацій на зображення.

Однією з особливостей LeNet-5 являється використання функції активації tanh, яка у той час вважалася стандартним рішенням, хоча в сучасній архітектурі вона частенько замінена на ReLU. Важливу роль грають повнозв'язні шари, які йдуть за згортковими і пулінг-слоями. Вони сполучають усі нейрони попереднього шару з кожним нейроном наступного і відповідають за остаточну класифікацію зображення.

Спочатку LeNet-5 була розроблена для класифікації рукописних цифр, таких як цифри з бази даних MNIST. Проте архітектурні принципи цієї мережі виявилися універсальними і були успішно адаптовані для вирішення інших завдань комп'ютерного зору, включаючи розпізнавання об'єктів і осіб. LeNet-5 стала основою для подальшого розвитку згорткових нейронних мереж, відкривши новий етап в області глибокого навчання.

LeNet-5, незважаючи на свою інноваційність, має ряд обмежень, пов'язаних з обчислювальними потужностями того часу. Вона ефективно працює тільки з простими об'єктами, такими як зображення невеликого розміру і низького розширення, що робить її непридатною для обробки складних даних. Відсутність методів попередньої обробки зображень призводить до необхідності додаткової підготовки даних вручну, що знижує гнучкість мережі.

Також в LeNet-5 спостерігається проблема загасання градієнта, яка ускладнює навчання глибоких мереж, особливо при збільшенні числа шарів. Модель схильна до перенавчання, що обмежує її здатність до узагальнення на

нових даних, а для роботи мережі потрібно значний об'єм пам'яті, що ускладнює її використання на пристроях з обмеженими ресурсами. Ці недоліки демонструють, як далеко просунувся розвиток нейронних мереж з моменту появи LeNet-5.

AlexNet (SOTA 2012) виграла ImageNet Challenge 2012 (ILSVRC-2012). Для конкурсу була використана вибірка з 1,2 млн зображень з 1000 різними класами, серед яких були як узагальнені категорії (наприклад, кішка, собака), так і вужчі (конкретні породи кішок або собак). Завдяки перемозі AlexNet в ILSVRC-2012 почалася ера глибокого навчання. Ця архітектура була глибша і складніша, ніж попередня LeNet-5. AlexNet використовує комбінацію згортальних шарів і повнозв'язних шарів, 5 згортальних і 3 повнозв'язні шари, із загальною кількістю 60 млн параметрів, пулінг і ReLU − активацію замість tanh. Для налаштування архітектури розробники зіткнулися з множиною проблем, пов'язаних з підбором параметрів, таких як розміри пулінга, кернелів, страйдів і інших гіперпараметрів (рис.2.14) [17].

Інноваційним рішенням стало використання нормалізації зсуву яскравості (Local Response Normalization), яка сприяла кращій узагальнювальній здатності мережі. Крім того, AlexNet була однією з перших моделей, що активно використовувала графічні процесори (GPU), що значно прискорило навчання.

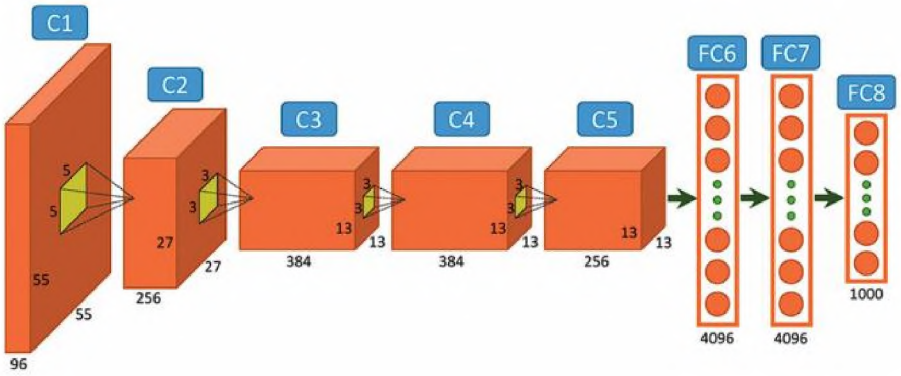

Рисунок 2.14 – Архітектура AlexNet (SOTA 2012)

Однією з ключових особливостей AlexNet є використання активації ReLU, яка допомагає вирішити проблему затухаючих градієнтів, оскільки на відміну від сигмоїди, ReLU при обчисленні похідної не створює малих значень. Для підвищення стійкості моделі застосовується Dropout з ймовірністю 0,5 на повнозв'язних шарах, що дозволяє системі адаптуватися до відключення випадкових нейронів. Також AlexNet активно використовує аугментацію даних, включаючи випадкові обрізки, повороти і дзеркальні відображення зображень, що збільшує обсяг навчальних даних і покращує здатність моделі до узагальнення.

Іншою інновацією стало використання локальної нормалізації зсуву яскравості (Local Response Normalization), яка збільшувала швидкість сходимості алгоритму. На той час це було ефективним рішенням для стабілізації навчання, хоча зараз частіше використовуються Batch Normalization або Layer Normalization, які забезпечують ще більш стабільне і швидке навчання. AlexNet також використовує max-pooling, що краще захоплює важливі особливості, такі як границі і текстури, і робить модель стійкішою до зсувів і деформацій об'єктів. Динамічне управління коефіцієнтом швидкості навчання дозволяє зменшувати його у 10 разів, якщо якість перестає покращуватися на валідаційній вибірці, що підвищує ефективність тренування.

Модель також підтримує навчання на кількох графічних процесорах (multi-GPU), що значно прискорило обробку великих обсягів даних. Регуляризація у різних формах допомагає зменшити ризик перенавчання і покращує здатність моделі до роботи з новими даними.

Однак AlexNet має свої недоліки. Вона містить велику кількість параметрів (понад 60 млн), що потребує значних обчислювальних ресурсів і великого обсягу пам’яті. Це створює високий ризик перенавчання і робить модель вимогливою до апаратного забезпечення. Проблема затухаючих градієнтів, хоча і зменшена завдяки ReLU, все ще може виникати. Обмежена глибина мережі знижує її здатність витягати складні ієрархічні ознаки

зображень, а фіксований розмір вхідного зображення вимагає додаткового попереднього масштабування, що іноді призводить до втрати інформації.

Архітектура, запропонована дослідницькою групою Visual Geometry Group Оксфордського університету. Брала участь в ILSVRC 2014 року і зайняла друге місце, поступившись лише GoogleNet. VGG NET складається із згортальних фільтрів 3×3, має декілька версій, які відрізняються кількістю шарів (рис.2.14). Найбільш відомі: VGG-16 (13 сверток і 3 повнозв'язні шари) і VGG-19 (16 сверток і 3 повнозв'язні шари) (рис.2.15).

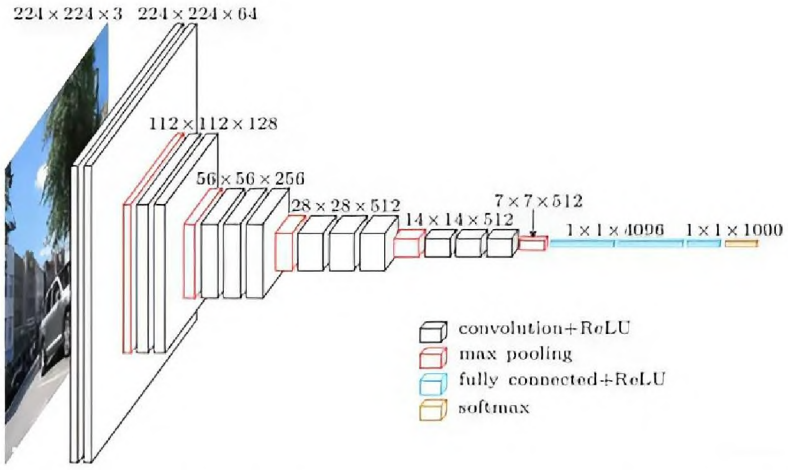

Рисунок 2.15 – Архітектура Visual Geometry Group 2014

VGG виділяється використанням маленьких згортальних фільтрів розміром 3×3 в усіх шарах, що дозволяє точніше захоплювати деталі зображення. Послідовне застосування таких невеликих фільтрів еквівалентне використанню більших фільтрів, наприклад 5×5 або 7×7, але з меншим числом параметрів. На відміну від AlexNet, яка використала великі фільтри, наприклад 11×11 в першому згортальному шарі, архітектура VGG оптимізує обчислення і покращує детектування ознак.

Для зменшення розмірності зображень VGG застосовує max-pooling з розміром вікна 2×2 і кроком 2. Цей метод дозволяє добитися інваріантності до зміщень і зменшити обчислювальну складність, причому pooling застосовується регулярніше, ніж в AlexNet. Повнозв'язні шари у кінці мережі

включають два шари з 4096 нейронами і вихідний шар, кількість нейронів в якому відповідає числу класів (наприклад, 1000 для ImageNet).

У VGG використовується функція активації ReLU, що дозволяє прискорити навчання і уникнути проблем, пов'язаних із затухаючими градієнтами. На відміну від AlexNet, тут не застосовується Local Response Normalization (LRN), що спрощує архітектуру і знижує обчислювальні витрати, не погіршуючи продуктивність. Проте, незважаючи на ці оптимізації, навчання моделі вимагало значних обчислювальних ресурсів, оскільки VGG була навчена на ImageNet, а кількість параметрів складала близько 138 мільйонів для VGG16 і 144 мільйони для VGG19, що майже удвічі перевищує параметри AlexNet.

Серед основних недоліків VGG – високі обчислювальні витрати і вимогливість до пам'яті із-за використання безлічі шарів з 3×3 фільтрами. Навчання на великих наборах даних займає багато часу, а результати стають повільними, що робить модель менш відповідною для завдань, що вимагають обробки в реальному часі. Також велика кількість параметрів збільшує ризик перенавчання, особливо на невеликих наборах даних, якщо не застосовується регуляризація.

Архітектура, розроблена Google. Використовує модуль Inception, основна ідея якого полягає в комбінуванні різних типів сверток з різними розмірами фільтрів і пулінгу для ефективнішого одержання ознак. Переможець ILSVRC 2014 (рис.2.16).

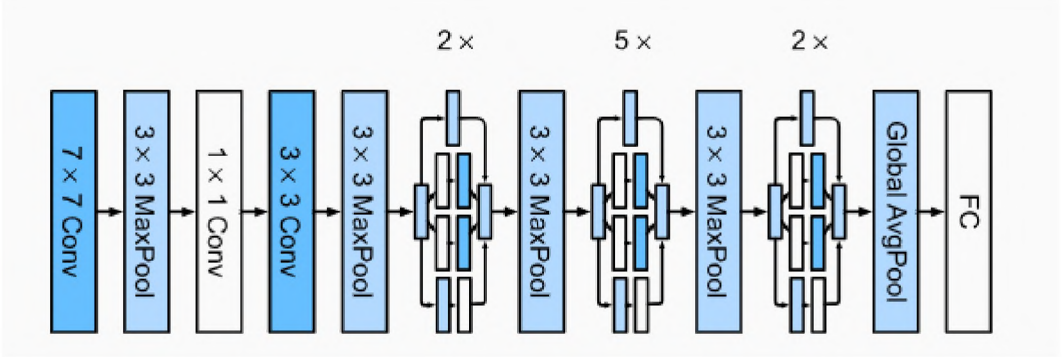

Рисунок 2.16 – Архітектура ILSVRC 2014

Однією з ключових особливостей архітектури Inception є використання унікальних Inception-блоків, які застосовують свертки з різними розмірами фільтрів (1×1, 3×3, 5×5) і max-pooling паралельно. Така структура дозволяє витягати ознаки на різних рівнях деталізації і ефективно комбінувати їх. Замість вибору одного фіксованого розміру фільтру, як це робилося у більше ранній архітектурі, результати від усіх розмірів фільтрів об'єднуються, що покращує якість витягання ознак і підвищує гнучкість моделі.

Додатковою перевагою стали допоміжні класифікатори, розташовані на проміжних рівнях мережі. Вони сприяють навчанню на ранніх етапах, допомагають запобігти загасанню градієнтів в глибоких мережах і покращують загальну продуктивність, додаючи елемент регуляризації. У фінальній частині мережі Inception використовує глобальний середній пулінг (global average pooling), який знижує кількість параметрів, роблячи модель стійкою до перенавчання, на відміну від традиційних повнозв'язних шарів.

Проте архітектура Inception має свої недоліки. Її модульна структура з різноманітністю фільтрів робить модель складною для реалізації, налаштування і відладки. Із-за безлічі паралельних операцій виникають складнощі з розпаралелюванням, що збільшує навантаження на пам'ять і процесор. Складна структура блоків утрудняє адаптацію архітектури під специфічні завдання, а навчання вимагає значних ресурсів і ретельного налаштування гіперпараметрів. Незважаючи на оптимізацію ресурсів в порівнянні з VGG, Inception залишається обчислювально витратною, особливо на етапі навчання.

ResNet (Residual Network) – це архітектура згортальних нейронних мереж, розроблена дослідницькою групою Microsoft під керівництвом К. Хе (Kaiming He), Х. Жаня (Xiangyu Zhang), С. Жэня (Shaoqing Ren) і Д. Суна (Jian Sun) в 2015 році. Використовує залишкові блоки (Residual Blocks), які дозволяють будувати надглибокі мережі (до 152 шарів) без проблеми затухаючих градієнтів. Завоювала перше місце на конкурсі ILSVRC 2015.

ResNet стала однією з найзначимішої архітектури в області глибокого навчання, оскільки вона вирішує проблему навчання дуже глибоких мереж. Стала основою для багатьох іншої архітектури, таких як ResNeXt, DenseNet і EfficientNet, які також використовують залишкові блоки і прямі переходи для досягнення кращої продуктивності (рис.2.17).

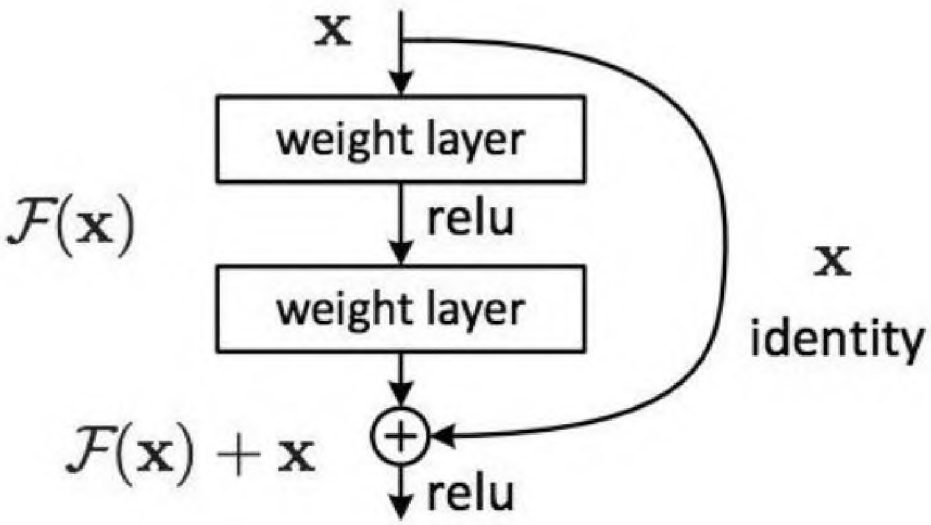

Рисунок 2.17 – ResNet (2015)

ResNet стала революційною архітектурою завдяки впровадженню резидуальних зв'язків, які дозволяють передавати інформацію через декілька шарів, обходячи проміжні. Основою архітектури є резидуальні блоки, які включають згортальні шари з активацією ReLU і прямим з'єднанням, що покращує збереження і передачу важливої інформації. Для оптимізації обчислень ResNet використовує згортальні шари 1×1 і bottleneck-блоки, які зменшують кількість параметрів без втрати продуктивності.

Проте глибокі мережі вимагають значних обчислювальних ресурсів і складного проектування. Збільшення глибини не завжди призводить до значного поліпшення точності, а підбір гіперпараметрів для навчання таких моделей стає складним завданням. Крім того, архітектура з пропускними з'єднаннями ускладнює інтерпретацію роботи мережі. Проте, ResNet задала

новий стандарт для побудови глибоких нейронних мереж, зробивши їх навчання стабільнішим і ефективнішим.

DenseNet (Densely Connected Convolutional Networks) – це архітектура згортальних нейронних мереж, запропонована Гао Хуангом (Gao Huang), Зи Лиу (Zhuang Liu), Лораном ван дер Маатеном (Laurens van der Maaten) і Кілером Вайнбергером (Kilian Weinberger) в 2017 році. DenseNet стала важливим кроком в розвитку глибоких нейронних мереж завдяки новому підходу до з'єднання шарів, дозволило поліпшити передачу інформації і градієнтів через мережу (рис.2.18).

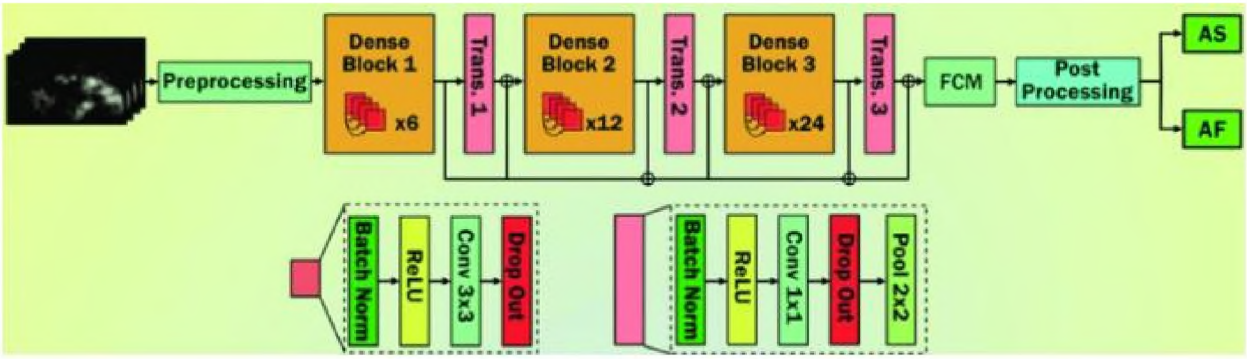

Рисунок 2.18 – DenseNet (2017)

DenseNet відрізняється унікальною архітектурою, де кожен шар сполучений з усіма попередніми, що дозволяє ефективно використати вже вичислені ознаки і покращувати нові. Таке рішення мінімізує кількість параметрів, незважаючи на велику кількість з'єднань, і полегшує передачу градієнтів, запобігаючи загасанню навіть в дуже глибоких мережах.

DenseNet використовує bottleneck- шари з 1×1 свертками для зниження обчислювальних витрат і компактності моделі, що робить її менш схильною до перенавчання. Щільні з'єднання забезпечують ефективніше навчання і кращу передачу інформації між шарами в порівнянні з ResNet, що особливо важливо для глибоких мереж [18].

Проте висока щільність з'єднань збільшує вимоги до пам'яті і обчислювальних ресурсів, уповільнюючи виконання моделі, особливо в

реальному часі. Незважаючи на ці обмеження, DenseNet залишається ефективною для витягання ознак і навчання глибоких мереж.

EfficientNet – це серія нейронних мереж, розроблена в 2019 році дослідницькою командою Google Brain під керівництвом Мингжера Тана і Квок Ле для підвищення ефективності згортальних нейронних мереж (CNN). Основна ідея EfficientNet полягає в тому, що масштабування мережі (збільшення числа шарів, ширини або дозволу) повинне відбуватися збалансованим чином для максимальної продуктивності і ефективності (рис.2.19).

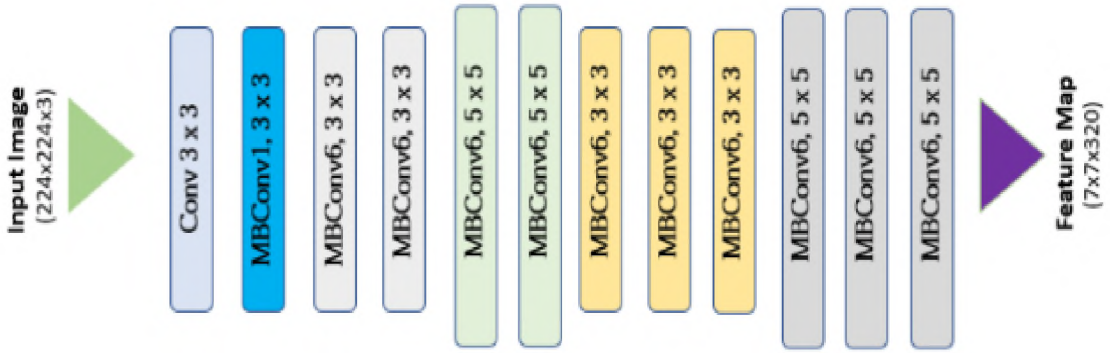

Рисунок 2.19 – EfficientNet (2019)

EfficientNet відрізняється інноваційним підходом до масштабування архітектури мережі в трьох вимірах: глибині, ширині і розширенні, завдяки методу Compound Scaling. Базова модель EfficientNet-B0 була розроблена з використанням автоматичного пошуку архітектури (NAS), а потім масштабована до більших моделей (B1-B7). Цей підхід забезпечує високу точність при значно менших обчислювальних витратах і об'ємі параметрів в порівнянні з ResNet або Inception. EfficientNet ефективно балансує продуктивність і витрати, пропонуючи компактні моделі з відмінним співвідношенням точності і ресурсів.

Основні недоліки пов'язані з високою обчислювальною вартістю автоматичного пошуку архітектури і складністю адаптації для завдань за

межами класифікації зображень. Для оптимального застосування в інших областях, таких як сегментація або детекція, часто потрібно додаткове налаштування і навчання [19].

Vision Transformer (ViT) – це архітектура, запропонована в 2020 році дослідниками з Google для обробки зображень з використанням трансформерів, які раніше застосовувалися, в основному, для завдань обробки послідовностей, таких як NLP (обробка природної мови). ViT став революційним підходом в комп'ютерному зорі, пропонуючи конкурентоздатну продуктивність в порівнянні з традиційними згортальними нейронними мережами (CNN) (рис.2.20).

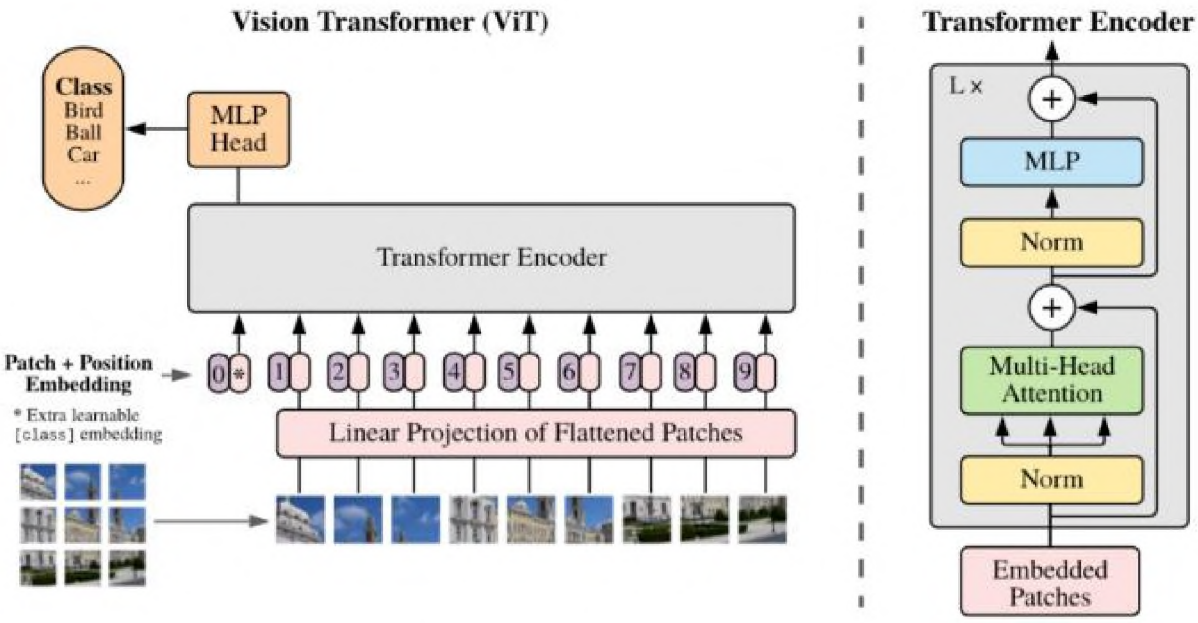

Рисунок 2.20 – ViT (2020)

ViT (Vision Transformer) використовує архітектуру трансформерів для обробки зображень, адаптуючи механізми self-attention, які зарекомендували себе в NLP. Замість сверток, як в CNN, ViT ділить зображення на маленькі патчі, які розглядаються як слова і передаються в трансформер.. Після обробки патчів трансформером використовується спеціальний токен [CLS] для класифікації. ViT ефективно моделює глобальні залежності між частинами

зображення і показує високі результати на великих наборах даних. На менших датасетах його продуктивність гірша, ніж у традиційних CNN [20].

Висновок до розділу 2

У розділі, присвяченому розробці методів оптимізації нейромережних класифікаторів із використанням AutoKeras, було розглянуто ключові аспекти автоматизації процесу побудови та налаштування моделей глибокого навчання. Основну увагу приділено трьом аспектам: вибору та налаштуванню гіперпараметрів, автоматичному проєктуванню архітектури нейронних мереж і впровадженню стратегій підвищення ефективності класифікації.

Дослідження підтвердило, що використання AutoKeras значно спрощує процес вибору оптимальних гіперпараметрів, оскільки інструмент застосовує сучасні методи пошуку, зокрема баєсову оптимізацію. Це дозволяє ефективно обирати найкращі параметри для моделі, зменшуючи потребу в ручному налаштуванні, що є складним і часовитратним завданням. AutoKeras забезпечує автоматизований підхід до оптимізації, адаптуючись до різноманітних наборів даних та специфіки задач.

Крім того, автоматизоване налаштування архітектури нейромереж, яке забезпечується AutoKeras, є важливим досягненням у галузі автоматизації машинного навчання. Завдяки використанню алгоритмів AutoML, система здатна будувати оптимальні архітектури навіть для складних задач класифікації, що дозволяє досягати високої точності моделі без потреби в глибоких знаннях у галузі машинного навчання.

Таким чином, завдяки інтеграції з іншими інструментами AutoML, таких як Keras Tuner або Hyperopt, AutoKeras надає можливість оптимізації моделей на більш високому рівні, забезпечуючи детальнішу настройку моделей на етапі пошуку найкращих гіперпараметрів.

РОЗДІЛ 3
РЕКОМЕНДАЦІЙ ЩОДО ЗАСТОСУВАННЯ AUTOKERAS ДЛЯ СИНТЕЗУ КЛАСИФІКАТОРІВ

3.1 Оцінка результатів навчання нейромережних класифікаторів зображень з використанням різних модифікацій архітектури

Розмір ядра згортки є важливим параметром, який визначає розмір фільтра, що рухається по вхідному зображенню в згортковому шарі нейронної мережі. Цей параметр впливає на кількість нейронів у шарі згортки, а також на розмір вихідного зображення. Використання більших ядер сприяє збереженню більшої кількості деталей в зображенні, проте водночас збільшує кількість параметрів, які потрібно оптимізувати під час навчання. Більші ядра також ефективно виявляють великі області зображення, забезпечуючи точніше розпізнавання особливостей.

Вибір розміру ядра залежить від специфіки задачі та розмірів вхідного зображення. При цьому важливо досягти балансу між точністю роботи моделі та кількістю її параметрів. Крім того, розмір ядра визначає здатність мережі виявляти як дрібні деталі, так і більш загальні особливості зображення. Згорткові нейронні мережі засновані на припущенні, що вхідні дані мають структуру зображення, що дозволяє їм формувати архітектуру, адаптовану до цього типу інформації.

На відміну від звичайних нейронних мереж, у згорткових нейронних мережах ядра згортки розташовуються в трьох вимірах: ширині, висоті та глибині. При цьому термін «глибина» стосується третього виміру ядра, а не загальної глибини нейронної мережі, яка визначається кількістю шарів. Це дозволяє згортковим шарам ефективно працювати з об'ємними структурами даних, такими як кольорові зображення [21].

CIFAR-10 (Canadian Institute for Advanced Research) є одним із найбільш популярних наборів даних у сфері комп'ютерного бачення. Він містить 60

тисяч зображень розміром 32 × 32 пікселі, розділених на 10 класів по 6 тисяч зображень у кожному. Класи включають різноманітні категорії об'єктів, такі як літаки, автомобілі, птахи, коти, собаки та інші. Цей набір широко застосовується для навчання та тестування алгоритмів класифікації, зокрема згорткових нейронних мереж, і є важливим інструментом у дослідженнях з розпізнавання зображень і машинного навчання.

Зображення з CIFAR-10 є тривимірними вхідними даними з формою 32 × 32 × 3, де третій вимір (глибина) відповідає кількості кольорових каналів (RGB). У згорткових мережах глибина фільтрів першого шару завжди відповідає кількості каналів вхідного зображення. Наприклад, якщо на вхід згорткового шару подається RGB-зображення з трьома каналами, а необхідно отримати 32 карти ознак, то шар має містити 32 фільтри, кожен з яких має глибину 3. Цей підхід забезпечує узгодженість між вхідними даними та параметрами фільтрів, дозволяючи згортковому шару ефективно витягувати важливі ознаки з зображення (рис. 3.1).

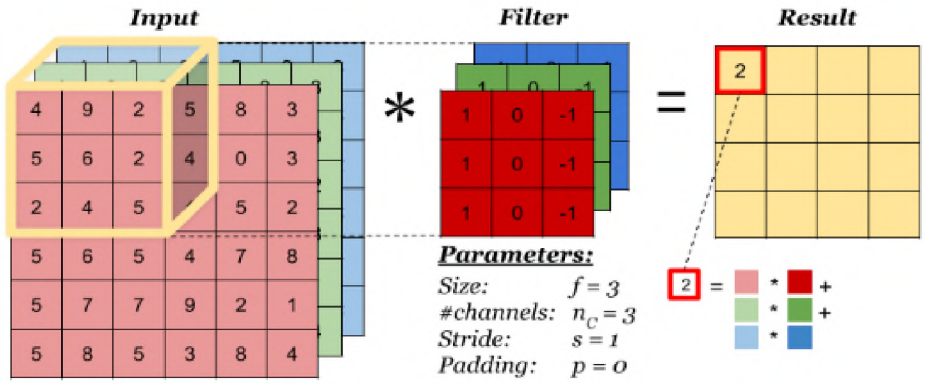

Рисунок 3.1 – Глибина згорткового шару

Обробка зображень є ключовою задачею комп'ютерного бачення, яка включає широкий спектр завдань, кожне з яких спрямоване на вдосконалення, аналіз чи інтерпретацію візуальної інформації. Одним із найпоширеніших завдань є виділення меж об'єктів на зображенні, яке передбачає знаходження границь між об'єктами. Для цього застосовують різноманітні алгоритми, серед яких розпізнавання країв, згорткові нейронні мережі та методи сегментації.

Такі підходи дозволяють точно ідентифікувати структуру об'єктів, їх контури та взаєморозташування. У поєднанні з машинним навчанням та глибокими нейромережами ці методи стають ще більш ефективними, дозволяючи автоматизувати розпізнавання та аналіз зображень у реальних додатках [22].

Ще одним важливим завданням є підвищення різкості зображення. Його мета – покращити чіткість і деталізацію, що досягається за допомогою фільтрів підвищення різкості, які акцентують увагу на дрібних деталях і покращують сприйняття зображення. Натомість розмиття зображення використовується для згладжування деталей, зменшення шуму або створення художнього ефекту. Це завдання зазвичай виконується шляхом застосування фільтрів розмиття, які забезпечують плавні переходи між кольорами та формами, усуваючи різкі переходи.

Однією з передових задач є автоматичне розпізнавання облич, що потребує використання складних алгоритмів, таких як згорткові нейронні мережі, здатні навчатися на великих наборах даних для виявлення особливостей обличчя. Ця технологія широко застосовується у сфері безпеки, а також в інтерактивних програмах і соціальних мережах. У випадку з видаленням шуму зображення основна мета полягає в усуненні небажаних артефактів, які можуть виникати через низьку якість зйомки чи зовнішні фактори, наприклад, недостатнє освітлення або вібрацію камери. Для цього використовуються як класичні алгоритми фільтрації, так і сучасні нейромережеві підходи.

Доповнюючи ці завдання, можна виділити також відновлення зображень, яке полягає у відтворенні втрачених або пошкоджених частин зображення, наприклад, у реставрації старих фотографій. Ця задача може включати складні методи інтерполяції, моделі глибокого навчання або комбіновані підходи.

Для виконання всіх перелічених завдань активно застосовуються згорткові фільтри, які стали стандартним інструментом у сучасних графічних редакторах. Вони дозволяють автоматизувати обробку зображень, роблячи її

швидкою та ефективною навіть для складних операцій. Завдяки поєднанню таких методів із нейронними мережами вдалося досягти суттєвих успіхів у комп'ютерному баченні, що відкриває нові можливості для досліджень і практичних застосувань [23].

Навчання нейронної мережі з використанням PyTorch включає етапи створення моделі, визначення функції втрат та оптимізатора, а також процеси передобробки даних і тренування моделі на вибраних даних.

PyTorch – фреймворк машинного навчання для мови Python з відкритим початковим кодом, створений на базі Torch. Використовується для вирішення різних завдань в області комп'ютерного зору і обробки мови.

На суперкомп'ютері cHARISMa підготовлене оточення з PyTorch версії 1.8 та 1.11. Також кожен користувач може самостійно створити персональне оточення з необхідною версією PyTorch.

У цьому прикладі розглядається завдання навчання нейронної мережі по класифікації зображень з датасета CIFAR-10. Використовувана модель нейронної мережі забезпечує точність класифікації до 95%. Запуск навчання нейронної мережі можна робити через веб-інтерфейс Jupyter Notebook або з використанням sbatch-файлів черги завдань. Процес конвертації коду з Jupyter-ноутбуків для запуску в пакетному режимі.

Навчання в інтерфейсі Jupyter Notebook:

– запустіть Jupyter Notebook на обчислювальному вузлі суперкомп'ютера з використанням сервісу JupyterHub;

– викачайте файл ноутбука і завантажте його в Jupyter Notebook train_cifar10 (IPYNB, 12 Кб);

– відредагуйте змінну max_epoch, чим більше епох навчання, тим точніше будуть результати класифікації.

Навчання з використанням черги завдань полягає в розподілі обчислювальних задач серед кількох робочих одиниць, що дозволяє ефективно обробляти великі обсяги даних паралельно.

При запуску обчислень через чергу завдань не потрібно чекати онлайн звільнення ресурсів. Планувальник автоматично запустить і виконає завдання у міру появи ресурсів, а всі результати збережуться в домашній директорії. У такому режимі користувачі можуть ставити в чергу сотні завдань відразу з різними вхідними параметрами і не турбуватися про необхідність ручного запуску наступного завдання після завершення попереднього – все виконується автоматично. Приклад sbatch-файлу запуску розташований на суперкомп'ютері. Завдання навчання можна запустити командою sbatch torch_cifar.sbatch, а виведення процесу навчання буде відображатися у файлі ~/torch_cifar_xxxxxx.log, де xxxxxx – це присвоєний номер задачі. Процес навчання нейронної мережі займе деякий час. Зі збільшенням змінної max_epoch збільшується час навчання, але і підвищується точність [24].

Після кожної епохи модель проходить тестування на тестових даних. Якщо точність моделі в цій епосі перевищує попередні значення, то модель записується у файл ~/checkpoint/ckpt.pth (рис.3.2).

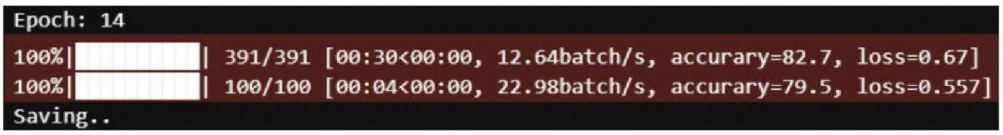

Рисунок 3.2 – Вигляд виконання епох

Приклад роботи навченої нейронної мережі можна отримати з використанням наступного коду (рис.3.3).

```
import numpy as np import matplotlib.pyplot as
plt
img = testset[14][0] # Зображення з тестової
вибірки, змініте номер для іншого зображення
label = testset[14][1] # Клас зображення з
тестової вибірки, номер повинен відповідати
номеру зображення вище img_np = img.numpy()
img_np = np.transpose(img_np, (1, 2, 0))
plt.imshow(img_np) plt.show() img =
img.reshape(1, 3, 32, 32) with
torch.no_grad(): logits = net(img)
predicted_label = torch.argmax(logits)
print(f"Label: {classes[label]}")
print(f"Predicted:
{classes[predicted_label.item()]}")
```

Рисунок 3.3 – Навчена нейронна мережа

Вибирається зображення з тестової вибірки і мережа класифікує його. Як видно на зображенні нижче, результат (Predicted) співпадає з початковим класом (Label). Для класифікації інших зображень змініть його номер в тестовій вибірці – testset [14] (рис.3.4).

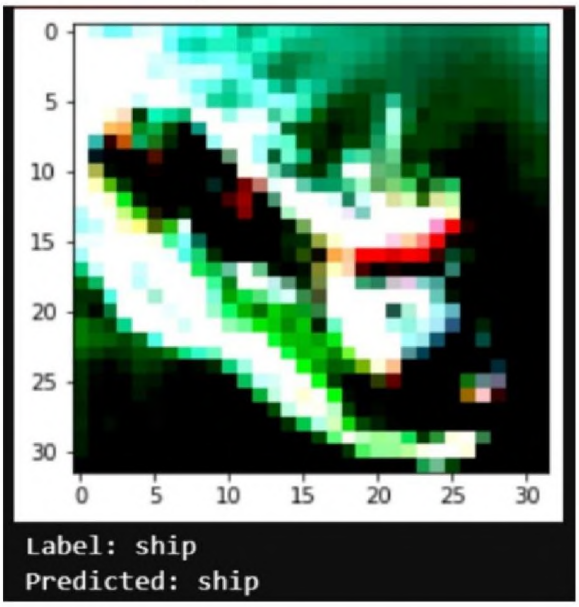

Рисунок 3.4 – Зображення з тестової вибірки

Сучасні методи навчання нейронних мереж, такі як згорткові нейронні мережі, PyTorch, та алгоритми AutoML, відкривають нові горизонти в обробці зображень та машинному навчанні. Від простих операцій, як видалення шуму чи підвищення різкості, до складних задач класифікації й розпізнавання об'єктів – ці технології дозволяють вирішувати завдання з небаченою раніше точністю [25].

CIFAR-10, як і багато інших популярних наборів даних, став своєрідним майданчиком для експериментів і вдосконалення алгоритмів, демонструючи, як невеликі покращення у підходах здатні суттєво вплинути на кінцевий результат. Використання суперкомп'ютерів і спеціалізованих середовищ, таких як Jupyter Notebook у поєднанні з планувальниками завдань, підкреслює масштабність і автоматизацію сучасних обчислень, дозволяючи дослідникам і розробникам зосередитися на ключових аспектах моделювання.

Можливо, невдовзі алгоритми, які сьогодні тестуються на наборах даних на кшталт CIFAR-10, знайдуть застосування у нових сферах – від медицини до автономного транспорту. Це лише підтверджує, що ми стоїмо на порозі нової ери, де обчислювальна потужність і алгоритми штучного інтелекту радикально змінюють спосіб, у який ми взаємодіємо зі світом [26].

Кожен внесок у розвиток цієї галузі може стати кроком до нових революційних відкриттів.

3.2 Порівняльній аналіз моделей нейромережного класифікатору зображень з різними гіперпараметрами

У машинному навчанні гіперпараметрами називають параметри моделі, значення яких встановлюються перед запуском процесу її навчання. Ними можуть бути, як параметри самого алгоритму, наприклад, глибина дерева в random forest, число сусідів в knn, ваги нейронів в нейронний мережах, так і способи обробки ознак, пропусків і так далі. Вони використовуються для управління процесом навчання, тому підбір оптимальних гіперпараметрів – дуже важливий етап в побудові ML-моделей, що дозволяє підвищити точність, а також боротися з перенавчанням. На сьогодні існують декілька популярних підходів до рішення задачі [27].

Пошук по гратках. У цьому способі значення гіперпараметрів задаються вручну, потім виконується їх повний перебір. Популярною реалізацією цього методу є Grid Search з sklearn. Незважаючи на свою простоту цей метод має і серйозні недоліки:

Дуже повільний оскільки потрібно перебрати усі комбінації усіх параметрів. Притому перебір триватиме навіть при свідомо невдалих комбінаціях.

Часто в цілях заощадження часу доводиться укрупнювати крок перебору, що може привести до того, що оптимальне значення параметра не

буде знайдено, якщо заданий діапазон значень від 100 до 1000 з кроком 100 (прикладом такого параметра може бути кількість дерев у випадковому лісі, або градієнтному бустингу), а оптимум знаходиться близько 550, то GridSearch його не знайде.

Випадковий пошук. Тут параметри беруться випадковим чином з вибірки з вказаним розподілом. У sklearn цей метод реалізований як Randomized Search. У більшості випадків він швидше GridSearch, до того ж значення параметрів не обмежені сіткою. Проте, навіть це не завжди дозволяє знайти оптимум і не захищає від перебору свідомо невдалих комбінацій.

Баєсова оптимізація. Тут значення гіперпараметрів в поточній ітерації вибираються з урахуванням результатів на попередньому кроці. Основна ідея алгоритму полягає в наступному – на кожній ітерації підбору знаходиться компроміс між дослідженням з найвдалішими зі знайдених комбінацій гіперпараметрів і дослідженням з великою невизначеністю (де можуть знаходитися ще вдаліші комбінації). Це дозволяє у багатьох випадках знайти кращі значення параметрів моделі за меншу кількість часу [28].

Hyperopt – популярна python-бібліотека для підбору гіперпарметрів. У ній реалізовано 3 алгоритми оптимізації, класичний Random Search, метод баєсової оптимізації Tree of Parzen Estimators (TPE), та Simulated Annealing – метод імітації. Hyperopt може працювати з різними типами гіперпараметрів неперериваючими, дискретними, категоріальними, що є важливою перевагою цієї бібліотеки. Бібліотека також підтримує розподілене обчислення, де потрібна висока продуктивність. Крім того, Hyperopt інтегрується з популярними бібліотеками, такими як TensorFlow і PyTorch, дозволяючи ефективно налаштовувати моделі глибокого навчання.

Встановити hyperopt дуже просто (рис. 3.5).

```
pip install hyperopt
```

Рисунок 3.5 – Встановлення hyperopt

Приклад роботи цієї бібліотеки в реальному завданні – передбачити, чи заробляє людина більше за $50 тис. Завантажимо необхідні бібліотеки, і підготуємо дані на вхід моделі (рис.3.6-3.9).

```
from functools import partial
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
```

Рисунок 3.6 – Приклад застосування hyperopt

```
df = pd.read_csv('adult.data.csv')
```

Рисунок 3.7 – Загружаємо данні

```
df.drop_duplicates(inplace=True, ignore_index=True)
```

Рисунок 3.8 – Видаляємо дублікати

```
X = df.drop(labels=['salary', 'native-country'], axis=1).copy()
y = df['salary'].map({'<=50K':0,'>50K':1}).values
```

Рисунок 3.9 – Готуємо ознаки і цільову змінну

У даних є ознаки різних типів, які, відповідно, вимагають і різної обробки. Для цього скористаємося методом ColumnTransformer з бібліотеки sklearn, який дозволяє задати свій спосіб обробки для кожної групи ознак. Для категоріальних ознак (тип object) використовуватимемо методи SimpleImputer (замінює пропуски, які позначені символом ""?) та OneHotEncoder (виконує dummy-кодування). Числові ознаки (інші типи) масштабуватимемо за

допомогою StandardScaler. В якості моделі виберемо логістичну регресію (рис.3.10-3.14).

```
num_columns = np.where(X.dtypes != 'object')[0]
cat_columns = np.where(X.dtypes == 'object')[0]
```

Рисунок 3.10 – Вибераємо категоріальні та числові данні

```
cat_pipe = Pipeline([('imputer', SimpleImputer(missing_values='?',
                                strategy='most_frequent')),
                     ('ohe', OneHotEncoder(sparse=False,
                        handle_unknown='ignore'))])
```

Рисунок 3.11 – Пайплайн для категоріальних ознак

```
num_pipe = Pipeline([('scaler', StandardScaler())])
```

Рисунок 3.12 – Пайплайн для числових ознак

```
model = Pipeline([('transformer', transformer),
                  ('lr', LogisticRegression(random_state=1, n_jobs=-1,
                           solver='liblinear'))])
```

Рисунок 3.13 – Підсумкова модель

```
search_space = {
                'lr__penalty' : hp.choice(label='penalty',
                          options=['l1', 'l2']),
                'lr__C' : hp.loguniform(label='C',
                        low=-4*np.log(10),
                        high=2*np.log(10))
                }
```

Рисунок 3.14 – Сформуємо простір пошуку параметрів для hyperopt

Виведемо результати в pandas DataFrame за допомогою спеціальної функції і візуалізуємо (рис.3.15-3.16).

```
    results = pd.DataFrame([{**x, **x['params']} for x in  hp_results])
    results.drop(labels=['status', 'params'], axis=1, inplace=True)
    results.sort_values(by=['loss'], ascending=False, inplace=True)
    return results

results = df_results(trials.results)
sns.set_context("talk")
plt.figure(figsize=(8, 8))
ax = sns.scatterplot(x='lr__C', y='loss', hue='lr__penalty',
                                                    data=results);
ax.set_xscale('log')
ax.set_xlim(1e-4, 2e2)
ax.grid()
```

Рисунок 3.15 – Приклад застосуваннябібліотеки hyperopt

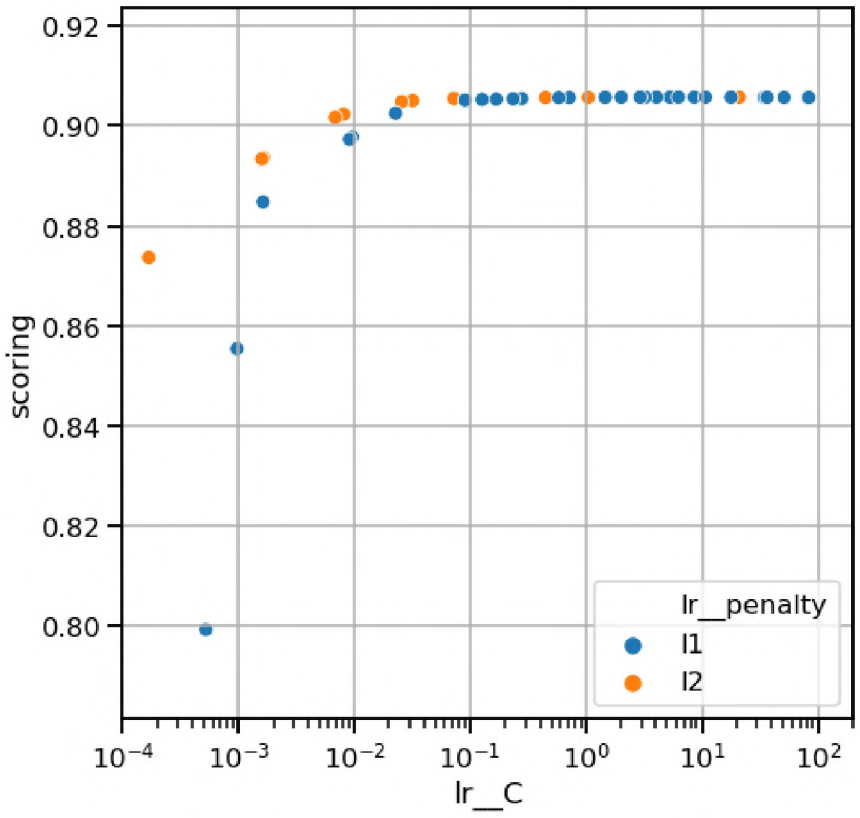

Рисунок 3.16 – Графік результатів hyperopt

На графіці видно, що hyperopt майже не дослідив, де виходили низькі значення roc auc, а зосередився на районі з найбільшими значеннями цієї метрики [29].

hyperopt – це лише один із багатьох інструментів, які дозволяють спростити та прискорити підбір гіперпараметрів у машинному навчанні. В залежності від завдання, можна комбінувати різні методи оптимізації для досягнення кращих результатів. Наприклад, спершу використовувати випадковий пошук для грубого оцінювання простору параметрів, а потім перейти до баєсової оптимізації для уточнення.

Варто зазначити, що підбір гіперпараметрів – це лише один із кроків у побудові моделі. Інші аспекти, такі як обробка даних, вибір відповідних ознак, аналіз отриманих результатів та перевірка надійності моделі, є не менш важливими.

Інструменти, подібні до hyperopt, активно інтегруються з іншими бібліотеками, такими як TensorFlow, PyTorch, XGBoost, що дозволяє працювати з найсучаснішими моделями та досягати найкращих результатів у різних галузях – від прогнозування продажів до медичних досліджень [30].

Таким чином, hyperopt – потужний інструмент для налаштування моделі, яким легко і зручно користуватися.

3.3 Економічне обґрунтування застосування AutoKeras

Впровадження AutoKeras дозволяє скоротити трудомісткість розробки завдяки автоматизації створення та оптимізації нейромереж. Зменшення трудомісткості передбачається на рівні 30%, що скорочує загальний обсяг роботи до:

$$344 \times 0{,}7 = 240{,}8 \text{ годин} \approx 241 \text{ годину}$$

Для команди з 3-х працівників (аналітик, розробник, тестувальник) годинні ставки залишаються незмінними:

Отже розрахуємо годинну ставку 3-ох працівників:

ЧС1 = 12 500,00 / 8 * 22 = 71,02 (грн/год);

ЧС2 = 22 000,00/ 8 * 22 = 125,00 (грн/год);

ЧС3 =10 000,00 / 8 * 22 = 55,82 (грн/год).

Отже, можемо розрахувати загальну суму витрат на оплату праці $З_{тр}$, яка визначається по формулі:

$$Зтр = \sum_{i=1}^{n} ЧС_i \times Ti \quad , \tag{3.1}$$

де $ЧС_i$ – годинна ставка *i*-го працівника (грн);

Ti – трудомісткість розробки ПП (чол × год);

i – категорія працівника;

n – кількість працівників, зайнятих розробкою ПП.

Таким чином, загальна сума на оплату праці складає:

Зтр = (71,02 + 125,00 + 55,82) * 241 = 62928,89 грн.

Порівняння витрат із використанням Keras та AutoKeras:

– витрати з використання Keras 86632,96 грн.;

– витрати з використанням AutoKeras 62928,89 грн.

Економія:

$$E=86632,96-62928,89=23704,07 \text{грн.}$$

Відносна економія:

$$E_{відн} = \frac{23\,704,07}{86\,632,96} \times 100 \approx 27,37\%.$$

Впровадження AutoKeras у процес розробки програмного забезпечення дозволяє знизити витрати на оплату праці команди на 27,37%, що становить 23704,07 грн. Це підтверджує економічну ефективність автоматизації проектування нейронних мереж за допомогою сучасних інструментів, таких як AutoKeras.

Таким чином, загальна сума витрат на оплату праці при застосуванні AutoKeras (аналітик, розробник, тестувальник) становить 62928,89 грн [31].

Висновки до розділу 3

У третьому розділі було проведено експериментальне дослідження, спрямоване на аналіз ефективності нейромережних класифікаторів за різних умов навчання. Основна увага була приділена оцінці якості моделей, побудованих із використанням різних конфігурацій і гіперпараметрів, а також їх практичній ефективності.

Результати навчання з різними конфігураціями продемонстрували значний вплив архітектури мережі на кінцеву точність і швидкість обробки даних. Випробування різних підходів дозволило виявити оптимальні конфігурації, які забезпечують баланс між продуктивністю та обчислювальними ресурсами.

Порівняння моделей із різними гіперпараметрами виявило, що параметри навчання, такі як швидкість навчання, кількість шарів і кількість нейронів у кожному шарі, мають вирішальне значення для досягнення найкращих результатів.

Економічне обґрунтування застосування AutoKeras підтвердило доцільність використання автоматизованих інструментів для створення нейромереж.

У підсумку, проведені дослідження підтвердили, що ретельне налаштування архітектури моделей, таких як AutoKeras, є ефективними підходами для побудови високоякісних нейромережних класифікаторів, що відповідають сучасним вимогам точності, швидкості та економічності.

ВИСНОВКИ

Розгляд основ нейромережних класифікаторів, методів їх оптимізації, а також технологій автоматизованого проєктування нейромереж (AutoML) дозволяє отримати глибше розуміння сучасних підходів у машинному навчанні, зокрема в задачах класифікації. Ключову роль нейромережних класифікаторів у розв'язанні різноманітних задач, починаючи від медичної діагностики і закінчуючи фінансовими прогнозами. Нейромережі дозволяють знаходити складні залежності між даними та успішно застосовуються в тих сферах, де традиційні методи машинного навчання не здатні досягти таких результатів.

Дослідження підтвердило, що використання AutoKeras значно спрощує процес вибору оптимальних гіперпараметрів, оскільки інструмент застосовує сучасні методи пошуку, зокрема баєсову оптимізацію. Це дозволяє ефективно обирати найкращі параметри для моделі, зменшуючи потребу в ручному налаштуванні, що є складним і часовитратним завданням. AutoKeras забезпечує автоматизований підхід до оптимізації, адаптуючись до різноманітних наборів даних та специфіки задач.

Крім того, автоматизоване налаштування архітектури нейромереж, яке забезпечується AutoKeras, є важливим досягненням у галузі автоматизації машинного навчання. Завдяки використанню алгоритмів AutoML, система здатна будувати оптимальні архітектури навіть для складних задач класифікації, що дозволяє досягати високої точності моделі без потреби в глибоких знаннях у галузі машинного навчання. Це особливо корисно для спеціалістів, які лише починають працювати з нейромережами, оскільки усуває необхідність у ручному дизайні та дозволяє швидко знаходити рішення високої якості.

Порівняння моделей із різними гіперпараметрами виявило, що параметри навчання, такі як швидкість навчання, кількість шарів і кількість нейронів у кожному шарі, мають вирішальне значення для досягнення

найкращих результатів. Зокрема, адаптивний підхід до налаштування гіперпараметрів забезпечив поліпшення продуктивності без значного збільшення витрат часу на оптимізацію.

Економічне обґрунтування застосування AutoKeras підтвердило доцільність використання автоматизованих інструментів для створення нейронних мереж.

З метою підтвердження можливості практичної реалізації моделі глибокого навчання класифікації транспорту в роботі визначені економічні показники запропонованих рішень. Наприклад, була розрахована загальна сума витрат на оплату праці для розробки програмного забезпечення. Загалом, витрати складають 62928,89 грн.

Таким чином, результатами роботи є створення архітектур нейронних мереж для класифікації зображень, рекомендації щодо застосування AutoKeras для синтезу класифікаторів. Вони може бути використані для подальших досліджень за даною тематикою та при проектуванні хмарних сервісів.