НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«СИСТЕМНИЙ АНАЛІЗ МЕТОДІВ ОПТИМІЗАЦІЇ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ ДЛЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБРАЗІВ»

Виконала: здобувач 2 курсу
Рівень магістр
Галузь знань 12 «Інформаційні технології»,
спеціальність 124 «Системний аналіз»
Каракян Афіна Володимирівна

Керівник: к.т.н., доцент
Чикунов Павло Олександрович

Рецензент к.т.н.,
Гура Володимир Ігорович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **інформаційних технологій**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **124 Системний аналіз**
Назва освітньої програми: **Аналіз та безпека даних**

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Н.І. Логінова
_____ «____ »_____ 2025 року

З А В Д А Н Н Я
на кваліфікаційну (магістерську) роботу
здобувачу Каракян Афіні Володимирівні

1. Тема роботи: «Системний аналіз методів оптимізації моделей глибинного навчання для задач розпізнавання образів»
Керівник роботи: к.т.н, доцент Чикунов Павло Олександрович
затверджені наказом НУ ОЮА від «10» жовтня 2025 року № 3183-06
2. Строк подання здобувачем роботи «25» листопада 2025 рік
3. Дата видачі завдання «02» червня 2025 рік

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Огляд предметної області та існуючих рішень	10.09.2025	
2	Формулювання мети, завдань та обґрунтування актуальності дослідження.	01.10.2025	
3	Огляд сучасних методів оптимізації моделей глибинного навчання.	15.10.2025	
4	Практична реалізація оптимізації моделі.	01.11.2025	
5	Експериментальне тестування, дослідження та верифікація моделі.	05.11.2025	
6	Оформлення пояснювальної записки.	10.11.2025	
7	Подача електронного варіанта роботи для перевірки на плагіат	20.11.2025	

Здобувач _____ Каракян А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Чикунов П.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Системний аналіз методів оптимізації моделей глибокого навчання для задач розпізнавання образів», виконана на здобуття другого (магістерського) рівня вищої освіти за спеціальністю 124 «Системний аналіз», містить 5 рисунків, 8 таблиць та 31 літературне джерело за переліком посилань. Робота викладена на 62 сторінках загального тексту, з них 55 сторінки основного змісту.

Метою роботи є дослідження та порівняльний аналіз методів оптимізації моделей глибокого навчання для задач розпізнавання зображень із мінімальною втратою точності за умов зменшення розміру моделі та прискорення інференсу.

Об’єктом дослідження є процеси оптимізації моделей глибинного навчання для задач розпізнавання образів.

Предметом дослідження є методи квантизації, структурного обрізання та дистиляції знань для оптимізації згорткових нейронних мереж при збереженні високої точності розпізнавання образів.

У ході виконання роботи проаналізовано сучасні архітектури згорткових нейронних мереж та методи їх оптимізації. Реалізовано базову модель на основі архітектури ResNet-18 для класифікації зображень на датасеті CIFAR-10. Застосовано та оцінено три методи оптимізації: квантування (зменшення числової точності з 32-біт до 8-біт), структурне прунінгування (видалення надлишкових зв’язків мережі) та дистиляція знань (навчання компактних моделей на основі більших). Проведено комплексний порівняльний аналіз для визначення компромісів між точністю класифікації, розміром моделі, швидкістю інференсу та використанням пам’яті для кожного методу оптимізації.

Практичне значення роботи полягає в розробленні методичних підходів і засобів для розгортання ефективних моделей глибокого навчання на пристроях із обмеженими ресурсами.

Ключові слова: глибоке навчання, розпізнавання зображень, оптимізація моделей, квантування, прунінг, дистиляція, згорткові нейронні мережі, resnet.

ANNOTATIONS

Qualification work on the topic "Systematic analysis of methods for optimizing deep learning models for pattern recognition tasks" for the second (master's) level of higher education in the specialty 124 System Analysis, contains 5 figures, 8 tables and 31 literary sources according to the list of references. The work consists of 62 pages of general text and 55 pages of main text.

The purpose of the work is to research and conduct a comparative analysis of deep learning model optimization methods for image recognition tasks with minimal accuracy loss while significantly reducing model size and accelerating the inference process.

The object of the research is the processes of optimizing deep learning models for image recognition tasks.

The subject of the research is the methods of quantization, structured pruning, and knowledge distillation for optimizing convolutional neural networks while preserving high image recognition accuracy.

In the course of the work, modern convolutional neural network architectures and their optimization methods were analyzed. A baseline model based on the ResNet-18 architecture was implemented for image classification on the CIFAR-10 dataset. Three optimization methods were applied and evaluated: quantization (reducing numerical precision from 32-bit to 8-bit), structured pruning (removing redundant network connections), and knowledge distillation (training compact models based on larger ones). A comprehensive comparative analysis was conducted to determine the trade-offs between classification accuracy, model size, inference speed, and memory usage for each optimization method.The practical significance of the work lies in the development of methodological approaches and tools for deploying efficient deep learning models on resource-constrained devices.

Keywords: deep learning, image recognition, model optimization, quantization, pruning, knowledge distillation, convolutional neural networks, resnet.

ЗМІСТ

ВСТУП

Системи розпізнавання образів на основі глибинного навчання знайшли широке застосування у різних галузях: від медичної діагностики та автономного водіння до систем безпеки та мобільних застосунків. Згорткові нейронні мережі (Convolutional Neural Networks, CNN) продемонстрували видатні результати у задачах класифікації, детекції та сегментації зображень, досягаючи точності, яка перевершує людські можливості у деяких специфічних завданнях.

Однак, незважаючи на високу точність, сучасні моделі глибинного навчання мають значні обмеження, які ускладнюють їх практичне застосування.

Одною з найбільших проблем є великий розмір моделей. Сучасні архітектури, такі як ResNet-152, VGG-19 або EfficientNet-B7, містять десятки або навіть сотні мільйонів параметрів, через що розмір моделей сягає 500 МБ. Це робить складнішим їх розгортання на мобільних пристроях з обмеженою пам'яттю.

Не менш важливою є проблема високих обчислювальних витрат. Процес інференсу (отримання передбачення) вимагає значних обчислювальних ресурсів, що обмежує можливість використання моделей на пристроях без потужних графічних процесорів. Це особливо критично для real-time застосунків, де затримка має бути мінімальною.

Решта проблем пов’язана з енергоспоживанням та вартістю. Робота складних моделей вимагає значних енергетичних витрат, що є проблемою для автономних пристроїв, IoT-систем та периферійних обчислень. Це також створює екологічні проблеми через високий вуглецевий слід процесу навчання та експлуатації моделей. А використання потужних серверів для отримання передбачень у хмарних сервісах призводить до високих операційних витрат, що робить масштабування систем штучного інтелекту економічно невигідним.

Тому оптимізація моделей глибинного навчання є критично важливою задачею для забезпечення практичного застосування технологій комп'ютерного зору у реальних умовах з обмеженими ресурсами. Методи оптимізації, такі як

квантизація, pruning та knowledge distillation, дозволяють значно зменшити розмір моделей та прискорити їх роботу при мінімальних втратах точності.

Метою роботи є дослідження та порівняльний аналіз методів оптимізації моделей глибинного навчання для задач розпізнавання образів з мінімальними втратами точності при значному зменшенні розміру моделі та прискоренні процесу інференса.

Для досягнення поставленої мети потрібно вирішити такі завдання.

1. Провести аналіз сучасних архітектур згорткових нейронних мереж для розпізнавання образів та визначити базову архітектуру для експериментальних досліджень.

2. Дослідити теоретичні основи методів оптимізації моделей глибинного навчання: квантизації, структурного обрізання та дистиляції знань.

3. Реалізувати базову модель на основі архітектури ResNet-18 для класифікації зображень на датасеті CIFAR-10 та отримати метрики базової моделі ефективності.

4. Розробити та реалізувати програмне забезпечення для застосування методів оптимізації до базової моделі.

5. Провести експериментальні дослідження ефективності кожного методу оптимізації за критеріями.

6. Здійснити порівняльний аналіз методів оптимізації та визначити їх переваги і недоліки у різних сценаріях застосування.

7. Надати практичні рекомендації щодо вибору методу оптимізації залежно від специфічних вимог та обмежень конкретної задачі.

Об’єктом дослідження є процеси оптимізації моделей глибинного навчання для задач розпізнавання образів.

Предметом дослідження є методи квантизації, структурного обрізання та дистиляції знань для оптимізації згорткових нейронних мереж при збереженні високої точності розпізнавання образів.

У роботі використовуються такі методи дослідження:

1. теоретичні методи: аналіз наукової літератури у галузі глибинного

навчання та комп'ютерного зору для вивчення сучасного стану досліджень; порівняльний аналіз існуючих архітектур нейронних мереж та методів їх оптимізації; математичне моделювання процесів;

2. експериментальні методи: проектування та реалізація програмного забезпечення; навчання моделей глибинного навчання; застосування методів оптимізації до навчених моделей; кількісна оцінка ефективності моделей за метриками; статистична обробка та візуалізація результатів експериментів.

3. емпіричні методи: проведення серії експериментів для визначення оптимальних параметрів методів оптимізації; аналіз впливу різних рівнів оптимізації на точність моделей; порівняння отриманих результатів з результатами інших досліджень.

Науковою значимістю роботи є комплексне дослідження ефективності трьох основних методів оптимізації на єдиній архітектурі ResNet-18 та датасеті CIFAR-10, що дозволяє об'єктивно порівняти їх ефективність в ідентичних умовах. Також було виконано детальний аналіз компромісів між точністю класифікації, розміром моделі, швидкістю отримання передбачень та використанням пам'яті для кожного методу оптимізації.

Розроблено систему практичних рекомендацій щодо вибору методу оптимізації залежно від конкретних обмежень ресурсів та вимог до точності, що може бути використано при проектуванні реальних систем розпізнавання образів. Досліджено можливість комбінування різних методів оптимізації для досягнення максимального ефекту при збереженні прийнятного рівня точності.

Практична значущість полягає у можливості використання реалізованих методів оптимізації для створення мобільних застосунків з функціями розпізнавання образів, що працюють безпосередньо на пристрої без необхідності постійного з'єднання з хмарними сервісами, а також у забезпеченні можливості розгортання моделей комп'ютерного зору на переферійних пристроях з обмеженими ресурсами пам'яті та обчислювальної потужності. Окрім того це допоможе у скороченні енергоспоживання систем на основі глибинного навчання.

1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕСІВ ОПТИМІЗАЦІЇ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ

1.1 Основи глибинного навчання нейронних мереж

Глибинне навчання (Deep Learning) є частиною машинного навчання, що базується на архітектурах штучних нейронних мереж з численними шарами обробки інформації. На відміну від традиційних алгоритмів машинного навчання, які потребують ручного визначення ознак, глибинні моделі здатні автоматично виявляти ієрархічні представлення даних безпосередньо з вхідної інформації.

Концептуальною основою глибинного навчання є багатошарові нейронні мережі, де кожен шар виконує нелінійне перетворення вхідних даних. Перші шари виділяють прості ознаки (наприклад, краї та кольори на зображеннях), тоді як глибші шари формують більш складні абстрактні представлення (форми, текстури, об'єкти). Ця ієрархічна структура дозволяє моделям ефективно розв'язувати складні завдання розпізнавання образів, класифікації та сегментації.

Базовим елементом нейронної мережі є штучний нейрон, який обчислює зважену суму вхідних сигналів та застосовує нелінійну функцію активації. Математично це можна представити як:

$$z = f(\sum(w_i \times x_i) + b) \tag{1.1}$$

де w_i – ваги зв'язків, x_i – вхідні значення, b – зміщення (bias), а f – функція активації.

Функції активації відіграють критичну роль у навчанні глибоких мереж. Найпоширенішими є ReLU (Rectified Linear Unit), яка визначається як f(x) = max(0, x), та її модифікації. ReLU вирішує проблему зникаючого градієнта, що є особливо важливим для глибоких архітектур.

Процес навчання нейронної мережі здійснюється через алгоритм зворотного поширення помилки (backpropagation) [29]. Цей метод обчислює

градієнти функції втрат відносно ваг мережі та оновлює їх за допомогою оптимізаторів, таких як Adam або SGD (Stochastic Gradient Descent). Ключовим моментом є вибір функції втрат: для задач класифікації зазвичай використовують кросс-ентропію, а для регресії – середньоквадратичну похибку.

Однією з фундаментальних проблем глибинного навчання є перенавчання (overfitting), коли модель надто добре запам'ятовує тренувальні дані, але погано узагальнює на нових прикладах. Для боротьби з цим явищем застосовують регуляризацію (L1, L2), виключення (dropout), пакетну нормалізацію (batch normalization) та аугментацію даних (data augmentation).

Пакетна нормалізація вирівнює значення активацій між шарами нейромережі. Це допомагає моделі швидше і стабільніше навчатися та дає змогу безпечно використовувати вищу швидкість навчання.

Виключення випадково "вимикає" певний відсоток нейронів під час тренування, що змушує мережу розвивати більш стійкі внутрішні представлення.

Сучасні глибокі нейронні мережі можуть містити десятки або навіть сотні шарів. Проте, збільшення глибини призводить до проблеми деградації точності через труднощі у навчанні дуже глибоких архітектур. Ця проблема була частково вирішена завдяки введенню залишкових з'єднань (residual connections), що дозволяє градієнтам проходити безпосередньо через мережу.

Важливим аспектом є вибір архітектури мережі. Глибші мережі здатні моделювати більш складні залежності, але потребують більше обчислювальних ресурсів та даних для навчання. Баланс між глибиною, шириною та обчислювальною складністю є ключовим при проектуванні ефективних моделей.

1.2 Згорткові нейронні мережі для розпізнавання образів

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є спеціалізованим типом глибоких мереж, розробленим для обробки даних з сітко-подібною топологією, зокрема зображень [3]. CNN революціонізували комп'ютерний зір завдяки здатності автоматично виявляти просторові ієрархії

ознак.

Архітектура CNN складається з трьох основних типів шарів: згорткових (convolutional), пулінгових (pooling) та повнозв'язних (fully connected). Згортковий шар застосовує набір навчуваних фільтрів до вхідного зображення, створюючи карти ознак (feature maps). Кожен фільтр виявляє специфічні локальні паттерни, такі як краї, текстури або форми.

Математично операція згортки визначається як:

$$(I * K)(i, j) = \sum\sum I(m, n) \times K(i - m, j - n) \tag{1.2}$$

де I – вхідне зображення, K – ядро згортки (фільтр), а i, j – координати у результуючій карті ознак.

Ключовою перевагою згорткових шарів є локальна зв'язність та спільне використання ваг (weight sharing). Замість повного зв'язування всіх нейронів, як у традиційних мережах, кожен нейрон згорткового шару з'єднаний лише з локальною областю попереднього шару. Ця властивість значно зменшує кількість параметрів та робить мережу інваріантною до зсувів об'єктів на зображенні [8].

Пулінговий шар виконує субдискретизацію (downsampling) карт ознак, зменшуючи їх просторові розміри [9]. Найпоширенішим є максимальний пулінг (max-pooling), який вибирає максимальне значення з локальної області. Пулінг робить представлення більш компактним та стійким до невеликих спотворень вхідних даних.

Еволюція архітектур CNN демонструє прогрес у розв'язанні задач комп'ютерного зору [4]. AlexNet (2012) показала ефективність глибоких CNN на великомасштабному датасеті ImageNet, використовуючи GPU для навчання та ReLU як функцію активації[5]. VGG (2014) продемонструвала, що глибші мережі з маленькими фільтрами (3×3) досягають кращих результатів.

Проривом стала архітектура ResNet (2015), яка запровадила залишкові блоки (residual blocks) [6]. Замість навчання прямого відображення H(x), ResNet навчає залишкову функцію F(x) = H(x) – x, додаючи пряме з’єднання (skip

connection) від входу до виходу блоку. Це дозволило успішно навчати мережі з глибиною понад 100 шарів без деградації точності.

Залишковий блок математично визначається як:

$$y = F(x, \{W_i\}) + x \tag{1.3}$$

де x – вхід блоку, F – залишкова функція (зазвичай 2-3 згорткових шари), а y – вихід, який буде отримано після додавання виходу F(x) до початкового x і застосування активації [7].

ResNet-18, що використовується у багатьох дослідженнях оптимізації, містить 18 шарів із залишковими з'єднаннями та демонструє відмінний баланс між точністю та обчислювальною складністю [12]. Архітектура включає початковий згортковий шар, чотири залишкових блоки з різною кількістю фільтрів (64, 128, 256, 512) та фінальний повнозв'язний шар для класифікації.

Подальший розвиток CNN включає архітектури, оптимізовані для мобільних пристроїв. MobileNet використовує згортки, що розділяються по глибині (depthwise separable convolutions) [30]. Вони розділяють стандартну згортку на дві окремі операції, значно зменшуючи кількість параметрів та обчислень. EfficientNet застосовує складне масштабування (compound scaling) – одночасне масштабування глибини, ширини та роздільної здатності мережі для досягнення оптимального співвідношення точність-ефективність.

Для оцінки якості роботи згорткових нейромереж використовують кілька показників. Точність (accuracy) показує частку правильно класифікованих зображень. Precision і Recall відображають якість розпізнавання окремих класів. F1-score об’єднує ці дві метрики у вигляді гармонічного середнього результату. Крім того, для практичного використання моделі важливими є час інференсу та обсяг споживаної пам’яті [2].

Датасети відіграють важливу роль у розвитку CNN [28]. CIFAR-10 містить 60,000 зображень розміром 32×32 пікселів у 10 класах та широко використовується для тестування методів оптимізації завдяки балансу між складністю та доступністю. ImageNet з 1.2 мільйонами зображень у 1000 класах

залишається еталоном для оцінки великомасштабних архітектур [5].

Сучасні CNN застосовуються у медичній діагностиці для аналізу рентгенівських знімків та МРТ, в автономному водінні для розпізнавання об'єктів, у системах безпеки для ідентифікації осіб, та у промисловості для контролю якості продукції [1].

1.3 Методи оптимізації моделей: огляд сучасних підходів

Незважаючи на видатні досягнення глибоких нейронних мереж, їхнє практичне застосування часто обмежене значними обчислювальними та пам'ятевими вимогами. Сучасні моделі можуть містити мільйони або мільярди параметрів, що ускладнює їхнє розгортання на пристроях з обмеженими ресурсами, таких як мобільні телефони, IoT-девайси та вбудовані системи. Це зумовило активний розвиток методів компресії та оптимізації нейронних мереж.

Методи оптимізації моделей можна класифікувати за кількома критеріями: час застосування (під час навчання, після навчання), тип структурних змін (структурована, неструктурована), та мета оптимізації (зменшення розміру, прискорення інференсу, зниження енергоспоживання) [14]. Основними напрямками є квантизація, прунінг, дистиляція знань, низькорангова декомпозиція та проектування ефективних архітектур.

Квантизація полягає у зменшенні розрядності числового представлення ваг та активацій нейронної мережі [15]. Стандартні моделі використовують 32-бітні числа з плаваючою комою (FP32), тоді як квантизовані моделі можуть працювати з 8-бітними цілими числами (INT8) або навіть з меншою розрядністю.

Квантизація суттєво зменшує розмір моделі (до 4 разів при переході FP32→INT8) та прискорює обчислення, оскільки операції з цілими числами виконуються швидше на більшості апаратних платформ. Крім того, зменшується обсяг необхідної пам'яті та енергоспоживання, що критично для мобільних застосувань [10].

Існують два основних підходи до квантизації:

Quantization-Aware Training (QAT) інтегрує квантизацію у процес навчання. Під час тренування моделі симулюються ефекти квантизації, що дозволяє мережі адаптуватися до втрат точності. Ваги зберігаються у повній точності, але у прямому проході використовуються квантизовані значення. Це забезпечує кращу точність порівняно з пост-тренувальною квантизацією, але потребує повного перенавчання моделі.

Post-Training Quantization (PTQ) застосовується до вже навченої моделі без додаткового тренування. PTQ є швидшим методом, що потребує лише невеликий калібрувальний датасет для визначення діапазонів квантизації [15]. Сучасні методи PTQ, такі як AdaRound, використовують апроксимації другого порядку для оптимізації процесу округлення, досягаючи мінімальної втрати точності.

Процес квантизації включає визначення функції відображення між повноточними та квантизованими значеннями. Лінійна квантизація визначається як:

$$q = \text{round}((r - z)/s) \tag{1.4}$$

де r – реальне значення, q – квантизоване значення, s – масштабний коефіцієнт (scale), z – зсув (zero-point).

Симетрична квантизація (z=0) спрощує обчислення, тоді як асиметрична краще використовує діапазон представлення для даних зі зміщеним розподілом.

Екстремальна квантизація включає бінарні (1-біт) та тернарні (2-біт) мережі, де ваги обмежені значеннями {-1, +1} або {-1, 0, +1}. Хоча це забезпечує максимальну компресію, втрата точності може бути значною для складних задач.

Прунінг або обрізання є методом видалення надлишкових чи малозначущих елементів нейронної мережі. Дослідження показують, що навчені моделі часто містять значну надмірність, і видалення до 80-90% параметрів може призвести до мінімальної деградації точності [17]. Прунінг класифікується за гранулярністю (рівнем структурності).

Неструктурований прунінг видаляє окремі ваги на основі їхньої величини або важливості. Це створює розріджені матриці, які можуть досягти високого

ступеня компресії, але потребують спеціалізованого апаратного або програмного забезпечення для ефективного виконання через нерегулярну структуру.

Структурований прунінг видаляє цілі структурні одиниці: фільтри, канали, або навіть цілі шари [18]. Це забезпечує реальне прискорення на стандартному обладнанні, оскільки зберігається регулярна структура обчислень. Хоча структурований прунінг може призводити до більшої втрати точності, ніж неструктурований, він є більш практичним для реального застосування.

За часом застосування прунінг поділяється на Pruning at Initialization, що видаляє з'єднання до початку навчання, базуючись на аналізі випадкових ініціалізованих ваг. Lottery Ticket Hypothesis стверджує, що великі мережі містять невеликі підмережі, які можна навчити до аналогічної точності.

Pruning During Training інтегрує процес обрізання у навчання, динамічно регулюючи структуру мережі. Це дозволяє моделі адаптуватися до втрат від прунінгу під час оптимізації.

Pruning After Training застосовується до повністю навченої моделі, ітеративно видаляючи найменш важливі елементи з проміжним дотренуванням для відновлення точності.

Критерії важливості для прунінгу включають величину ваг (L1/L2 норми), градієнти, вплив на функцію втрат, та статистику активацій. Сучасні методи використовують інформацію другого порядку для більш точної оцінки важливості параметрів.

Для ResNet розроблені спеціальні стратегії прунінгу, що враховують залишкові з'єднання. Можна обрізати фільтри тільки у певних шарах залишкових блоків або застосовувати прунінг до всіх шарів блоку одночасно, підтримуючи узгодженість розмірностей.

Дистиляція знань є методом передачі знань від великої, складної моделі (вчителя) до меншої, ефективнішої моделі (учня) [22]. На відміну від стандартного навчання на жорстких мітках (hard labels), учень навчається на "м'яких" виходах (soft labels) вчителя, які містять додаткову інформацію про відносини між класами.

Ключовою ідеєю є використання softmax temperature для отримання м'яких ймовірностей:

$$p_i = \frac{\exp(z_i/T)}{\sum \exp(z_j/T)} \tag{1.5}$$

де z_i – логіти моделі, T – температура. При T > 1 розподіл стає більш рівномірним, надаючи учневі інформацію про "темні знання" (dark knowledge) – відносини між класами, які не видимі у жорстких мітках.

Функція втрат для дистиляції комбінує два компоненти:

$$L = \alpha \times L_{\text{soft}}(p_{\text{student}}, p_{\text{teacher}}) + \beta \times L_{\text{hard}}(p_{\text{student}}, y_{\text{true}}) \tag{1.6}$$

де Lsoft вимірює відмінність між розподілами учня та вчителя (зазвичай KL-дивергенція), Lhard – стандартна крос-ентропія з реальними мітками, а α і β – ваги компонентів.

Існують різні варіанти дистиляції [11]: Offline distillation використовує попередньо навчену модель-вчителя для навчання учня. Це найпоширеніший підхід, що дозволяє використовувати будь-яку потужну модель як джерело знань [27].

Online distillation одночасно навчає вчителя та учня, або використовує ансамбль учнів для взаємного навчання [25].

Self-distillation модель навчає сама себе. Вона використовує власні проміжні представлення або результати, отримані на попередніх епохах, як "підказки" чи "м’якші мітки". Таким чином модель виступає одночасно і вчителем, і учнем, поступово покращуючи свою якість без окремої, більш складної моделі-вчителя.

Multi-teacher distillation поєднує знання кількох моделей-вчителів. Учень навчається одразу на їхніх прогнозах, завдяки чому отримує більш різноманітну та багату інформацію. Це може покращити здатність моделі узагальнювати та підвищити її точність [21].

Feature-based distillation передає знання не лише з фінального шару, але й з

проміжних представлень. Учень навчається відтворювати карти ознак вчителя, що може покращити якість навченої моделі, особливо для завдань, де важливі проміжні представлення.

Дистиляція знань ефективна не тільки для компресії, але й для трансферного навчання, ансамблевих методів, покращення узагальнення, та навчання на зашумлених даних [25]. Учень часто узагальнює краще за вчителя завдяки регуляризуючому ефекту "м'яких міток".

Сучасні дослідження показують, що комбінування різних методів оптимізації може призвести до синергетичного ефекту. Наприклад, спільна оптимізація квантизації та прунінгу дозволяє досягти більшої компресії при збереженні точності. Дистиляція знань може використовуватися для відновлення точності після агресивної квантизації або прунінгу.

Вибір методу оптимізації залежить від конкретних вимог застосування: доступних обчислювальних ресурсів, допустимої втрати точності, обмежень латентності та енергоспоживання. Розуміння переваг та обмежень кожного підходу є критичним для розробки ефективних рішень оптимізації моделей.

Кожен метод має свої переваги та обмеження, а їхнє комбінування може забезпечити оптимальний баланс між розміром моделі, швидкістю інференсу та точністю класифікації.

1.4 SWOT-аналіз методів оптимізації

Вибір оптимального методу стиснення нейронних мереж є нетривіальною задачею, що вимагає глибокого розуміння компромісів між розміром моделі, швидкістю інференсу та точністю класифікації. У контексті розгортання моделей на пристроях з обмеженими ресурсами, де одночасно діють жорсткі обмеження на обсяг пам'яті та енергоспоживання, необхідно системно оцінити переваги та недоліки існуючих підходів. SWOT-аналіз дозволяє структурувати це оцінювання через призму внутрішніх характеристик методів (сильні та слабкі сторони) та зовнішніх факторів (можливості та загрози), що впливають на їх

практичне застосування. Такий підхід забезпечує обґрунтований вибір стратегії оптимізації для конкретного застосування та цільової платформи.

Таблиця 1.4 – SWOT-аналіз методів оптимізації глибинних нейронних мереж

S – Strengths (Сильні сторони)	**W – Weaknesses (Слабкі сторони)**
• Квантизація зменшує розмір моделі до 75% при переході FP32 → INT8. • Прискорює інференс завдяки швидким цілочисельним обчисленням. • PTQ не потребує перенавчання. • Прунінг знижує обчислювальну складність, видаляючи до 80–90% параметрів. • Прунінг дає регуляризуючий ефект та сумісний з іншими методами оптимізації. • Дистиляція знань найкраще зберігає точність і передає складні закономірності моделі-вчителя.	• Квантизація нижче INT8 викликає помітну втрату точності. • Складне калібрування та різна чутливість шарів до квантизації. • Структурований прунінг може давати більшу втрату точності. • Ризик видалення важливих з’єднань та необхідність fine-tuning. • Дистиляція потребує великої моделі-вчителя та складного підбору T і α. • Високі обчислювальні витрати на навчання моделі-учня.
O – Opportunities (Можливості)	**T – Threats (Загрози)**
• Комбінування методів (quantization + pruning) для синергетичного ефекту. • Розвиток динамічних методів (adaptive quantization, dynamic pruning). • Автоматизація оптимізації через Neural Architecture Search. • Нові підходи до дистиляції (self-distillation, data-free distillation). • Інтеграція з інструментами TF Model Optimization, PyTorch Mobile та MLOps-платформами.	• Зростання складності сучасних архітектур (Transformers, ViT). • Відсутність універсальних стратегій оптимізації – потреба індивідуального налаштування. • Обмежена підтримка апаратних прискорювачів на практиці. • Компроміс між точністю та ефективністю критичний для відповідальних застосувань. • Швидке оновлення архітектур призводить до старіння існуючих методів оптимізації.

Проведений SWOT-аналіз демонструє, що кожен із розглянутих методів оптимізації має власний баланс сильних і слабких сторін, визначених як внутрішніми характеристиками, так і зовнішніми умовами застосування. Квантизація та прунінг забезпечують істотне зменшення розміру моделі та прискорення інференсу, тоді як дистиляція знань дозволяє зберегти високу

точність навіть у компактних архітектурах. Водночас ефективність кожного методу залежить від складності моделі, специфіки задачі та доступності апаратної підтримки. Перспективними є комбіновані підходи та застосування динамічних методів оптимізації, однак швидка еволюція сучасних архітектур створює додаткові виклики щодо їхньої адаптації. Сукупно це підкреслює необхідність раціонального вибору стратегії оптимізації відповідно до вимог конкретної платформи та обмежень цільового застосування.

1.5 Висновки до розділу 1

У першому розділі проведено комплексний теоретичний аналіз методологічних основ оптимізації моделей глибинного навчання. Розглянуто фундаментальні принципи роботи нейронних мереж, архітектуру згорткових мереж та їх еволюцію від AlexNet до ResNet з залишковими з'єднаннями.

Систематизовано три основні методи оптимізації. Квантизація забезпечує компресію до 75% при переході від FP32 до INT8. Прунінг дозволяє видаляти до 80-90% надлишкових параметрів. Дистиляція знань передає інформацію через м'які розподіли, забезпечуючи найкраще збереження точності. Встановлено, що кожен метод має специфічні переваги: квантизація оптимальна для швидкого розгортання, прунінг для зменшення обчислювальної складності, дистиляція для балансу між компресією та точністю. Теоретичний аналіз показав перспективність комбінування цих методів для синергетичного ефекту.

SWOT-аналіз показав необхідність раціонального вибору стратегії оптимізації відповідно до вимог конкретної платформи та обмежень цільового застосування.

2 ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ОПТИМІЗАЦІЇ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ

2.1 Вибір датасету та навчання базової моделі

Для проведення експериментальних досліджень методів оптимізації було обрано датасет CIFAR-10 (Canadian Institute for Advanced Research). Цей датасет є одним із найпоширеніших бенчмарків у галузі комп'ютерного зору та широко використовується для оцінки алгоритмів класифікації зображень.

CIFAR-10 складається з 60,000 кольорових зображень розміром 32×32 пікселі, які рівномірно розподілені між 10 класами: літак (airplane), автомобіль (automobile), птах (bird), кіт (cat), олень (deer), собака (dog), жаба (frog), кінь (horse), корабель (ship) та вантажівка (truck). Датасет поділено на 50,000 тренувальних та 10,000 тестових зображень, по 5,000 та 1,000 зображень на клас відповідно.

Вибір CIFAR-10 обумовлений кількома факторами.

Збалансованість класів. Рівна кількість зображень у кожному класі запобігає проблемі незбалансованих даних та забезпечує об'єктивну оцінку якості моделі. Це критично важливо для коректного порівняння методів оптимізації.

Помірна складність. На відміну від простих датасетів (MNIST), CIFAR-10 містить реальні кольорові зображення з різноманітними фонами, поворотами та масштабами об'єктів. Водночас, порівняно з великомасштабними датасетами (ImageNet), CIFAR-10 дозволяє швидко проводити експерименти без необхідності у потужних обчислювальних ресурсах.

Широке використання в дослідженнях. CIFAR-10 є стандартним бенчмарком для оцінки методів компресії та оптимізації нейронних мереж, що дозволяє порівнювати отримані результати з існуючими публікаціями.

Компактність. Невеликий розмір зображень (32×32) дозволяє ефективно використовувати обмежені обчислювальні ресурси та проводити множинні

експерименти для підбору оптимальних гіперпараметрів методів оптимізації.

Для підготовки даних було застосовано стандартні техніки попередньої обробки. Усі зображення нормалізовано з використанням середніх значень та стандартних відхилень по каналах RGB, обчислених на тренувальній вибірці: mean = [0.4914, 0.4822, 0.4465], std = [0.2470, 0.2435, 0.2616]. Це забезпечує стабільність процесу навчання та покращує збіжність оптимізації.

Для аугментації тренувальних даних використано випадкові горизонтальні відображення (horizontal flip) з імовірністю 0.5 та випадкові обрізки (random crop) розміром 32×32 з відступом 4 пікселі. Аугментація даних збільшує варіативність тренувальної вибірки та покращує узагальнюючу здатність моделі, знижуючи ризик перенавчання.

Як базову архітектуру для експериментів було обрано ResNet-18 (Residual Network з 18 шарами). Ця модель належить до родини глибоких залишкових мереж, запропонованих у 2015 році дослідниками з Microsoft Research [6].
ResNet-18 складається з наступних компонентів:

Початковий блок включає згортковий шар з 64 фільтрами розміром 7×7, batch normalization, активацію ReLU та max-pooling шаром. Цей блок виконує початкову обробку вхідного зображення та зменшує його просторову розмірність.

Чотири залишкові стадії містять по два залишкових блоки кожна. Кількість фільтрів послідовно збільшується: 64, 128, 256 та 512. Кожна стадія, окрім першої, виконує субдискретизацію за допомогою згортки зі stride = 2, зменшуючи просторові розміри карт ознак у два рази.

Фінальний класифікатор складається з global average pooling, який перетворює карти ознак розміром 512×H×W у вектор розміром 512, та повнозв'язного шару, що проектує цей вектор у простір класів (10 класів для CIFAR-10).

Загальна кількість параметрів базової моделі ResNet-18 (адаптованої для CIFAR-10) становить приблизно 11.17 мільйони. Розмір моделі у форматі FP32 складає близько 42.8 МБ.

Вибір ResNet-18 як базової архітектури обґрунтований наступними міркуваннями:

Ефективність залишкових з'єднань. Залишкові блоки дозволяють ефективно навчати глибокі мережі, вирішуючи проблему деградації точності. Skip connections забезпечують пряме поширення градієнтів через мережу, що прискорює збіжність навчання та покращує фінальну точність.

Баланс між складністю та продуктивністю. ResNet-18 забезпечує гарний компроміс між кількістю параметрів та точністю класифікації. Модель достатньо складна для демонстрації ефективності методів оптимізації, але не надмірно велика для обмежених обчислювальних ресурсів.

Модульна структура. Регулярна організація залишкових блоків спрощує застосування структурованого прунінгу та дозволяє легко модифікувати архітектуру без порушення функціональності мережі.

Широке застосування в дослідженнях. ResNet-18 часто використовується як базова модель для порівняння у дослідженнях оптимізації нейронних мереж, що дозволяє порівнювати результати з існуючими роботами.

Базову модель ResNet-18 було навчено на тренувальній вибірці CIFAR-10 з такими гіперпараметрами.

Функція втрат: CrossEntropyLoss, яка поєднує softmax та negative log-likelihood, оптимальна для задач багатокласової класифікації.

Оптимізатор: Stochastic Gradient Descent (SGD) з momentum=0.9 та weight decay=5e-4. SGD з моментумом забезпечує стабільну збіжність та добре узагальнення на тестових даних.

Початкова швидкість навчання: 0.1 з косинусним розкладом (cosine annealing), що поступово зменшує learning rate від початкового значення до нуля протягом навчання [31]. Це дозволяє на початку робити великі кроки оптимізації, а наприкінці – точно налаштовувати параметри.

Розмір батчу: 128 зображень. Такий розмір забезпечує баланс між стабільністю градієнтів та ефективністю використання GPU пам'яті.

Кількість епох: 200. Експерименти показали, що модель досягає насичення

точності приблизно після 150-180 епох, проте додаткові епохи забезпечують фінальне налаштування параметрів.

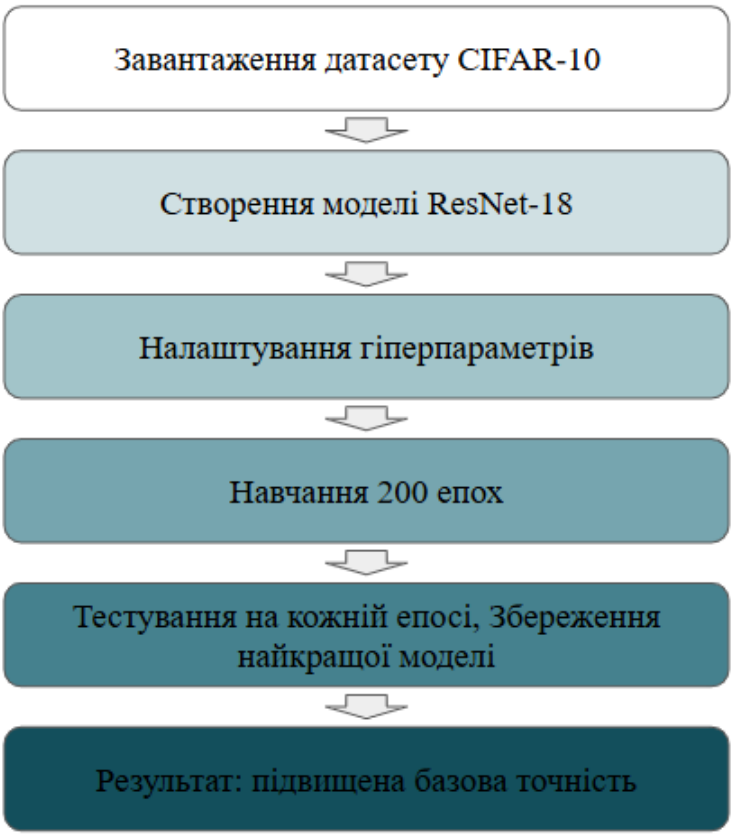

Рисунок 2.1 – Процес навчання базової моделі ResNet-18

Навчання проводилось на обчислювальній платформі з GPU NVIDIA Tesla T4 через Google Colab, що забезпечило прийнятний час навчання (приблизно 2-3 години для повного циклу). Використання змішаної точності (mixed precision training) дозволило прискорити обчислення без втрати якості.

Після завершення навчання базова модель досягла наступних метрик на тестовій вибірці CIFAR-10:

1. Test Accuracy: 95.66%
2. Час інференсу: ~15.3 мс на одне зображення (на CPU)
3. Розмір моделі: 42.8 МБ (FP32)
4. Кількість параметрів: 11.17M

Ці результати слугують базовою моделлю для порівняння з оптимізованими версіями моделі. Матриця плутанини (confusion matrix)

показала, що модель найкраще класифікує класи "корабель" та "вантажівка" (точність >95%), тоді як найбільше помилок спостерігається між схожими класами "кіт" та "собака" (точність ~88-90%).

2.2 Реалізація методу оптимізації Quantization

Квантизація є процесом зменшення розрядності числового представлення ваг та активацій нейронної мережі. Стандартні моделі використовують 32-бітні числа з плаваючою комою (float32), що забезпечує високу точність, але вимагає значних обчислювальних та пам'ятевих ресурсів. Квантизація дозволяє перейти до менш точних, але більш ефективних форматів, таких як 8-бітні цілі числа (int8) [16].

Математично процес квантизації можна описати як відображення між множиною дійсних чисел та дискретною множиною квантизованих значень. Для симетричної квантизації використовується формула:

$$q = \text{round}(r/s) \tag{2.1}$$

де r – реальне значення у форматі float32, q – квантизоване значення у форматі int8, s – масштабний коефіцієнт (scale factor), що визначає діапазон квантизації. Зворотне перетворення (деквантизація) виконується як:

$$r_{\text{approx}} = q \times s \tag{2.2}$$

Масштабний коефіцієнт s обчислюється на основі діапазону значень у тензорі:

$$s = \frac{\max(|r_{\text{min}}|, |r_{\text{max}}|)}{127} \tag{2.3}$$

де r_min та r_max – мінімальне та максимальне значення у тензорі, а 127 – максимальне значення для 8-бітного знакового цілого числа.

Квантизація значно зменшує обчислювальну складність операцій. Множення чисел з плаваючою комою (FP32) потребує десятки тактів процесора, тоді як цілочисельне множення (INT8) виконується у кілька разів швидше. Крім того, INT8 операції підтримуються спеціалізованими інструкціями процесорів

(AVX-512 VNNI для Intel, DP4A для NVIDIA) та апаратними акселераторами (Tensor Cores, Google TPU) [15].

Для реалізації квантизації було обрано підхід Post-Training Quantization (PTQ), який застосовується до вже навченої моделі без необхідності повного перенавчання. Цей метод є більш практичним порівняно з Quantization-Aware Training (QAT), оскільки не вимагає доступу до всього тренувального датасету та значно скорочує час оптимізації.

Реалізація PTQ включає такі етапи.

Калібрування. Для визначення оптимальних масштабних коефіцієнтів використано калібрувальну вибірку з 1000 випадково обраних зображень з тренувального датасету. Під час калібрування модель виконує forward pass, збираючи статистику активацій у кожному шарі.

Визначення діапазонів квантизації. Для кожного тензору (ваги та активації) обчислюються мінімальні та максимальні значення, на основі яких визначаються scale та zero-point параметри квантизації.

Застосування квантизації. Усі ваги та активації конвертуються у формат INT8 з використанням обчислених параметрів. Операції згортки та матричного множення виконуються у цілочисельній арифметиці.

Для реалізації використано бібліотеку PyTorch з модулем torch.quantization, який надає вбудовані інструменти для PTQ. Зокрема, застосовано статичну квантизацію, яка визначає параметри квантизації один раз під час калібрування та використовує їх для всіх подальших інференсів.

Процес квантизації базової моделі ResNet-18 включав такі кроки (рис. 2.2).

Підготовка моделі. Базова модель була модифікована для квантизації шляхом додавання QuantStub та DeQuantStub блоків на вході та виході мережі. Ці блоки виконують конвертацію між форматами float32 та int8.

Об'єднання операцій. Деякі послідовності операцій (Conv-BN-ReLU) були об'єднані (fused) у єдині блоки, що зменшує overhead від конвертацій типів та покращує ефективність квантизації.

Калібрування. Модель переведено у режим калібрування та виконано

forward pass на калібрувальній вибірці. PyTorch автоматично збирає статистику активацій та обчислює параметри квантизації для кожного шару.

Конвертація у квантизований формат. После калібрування модель конвертовано у повністю квантизований формат, де всі операції виконуються у INT8 арифметиці.

Валідація. Квантизована модель протестована на повній тестовій вибірці CIFAR-10 для оцінки точності та швидкості інференсу.

Ключові параметри квантизації:

1. Схема квантизації: per-tensor symmetric для ваг, per-tensor asymmetric для активацій

2. Формат: int8 для ваг та активацій, int32 для проміжних результатів

3. Backend: fbgemm (оптимізований для x86 CPU)

Особливу увагу приділено першому та останньому шарам мережі. Дослідження показують, що ці шари є більш чутливими до квантизації, тому для них було експериментально перевірено різні стратегії, включаючи залишення у форматі FP32.

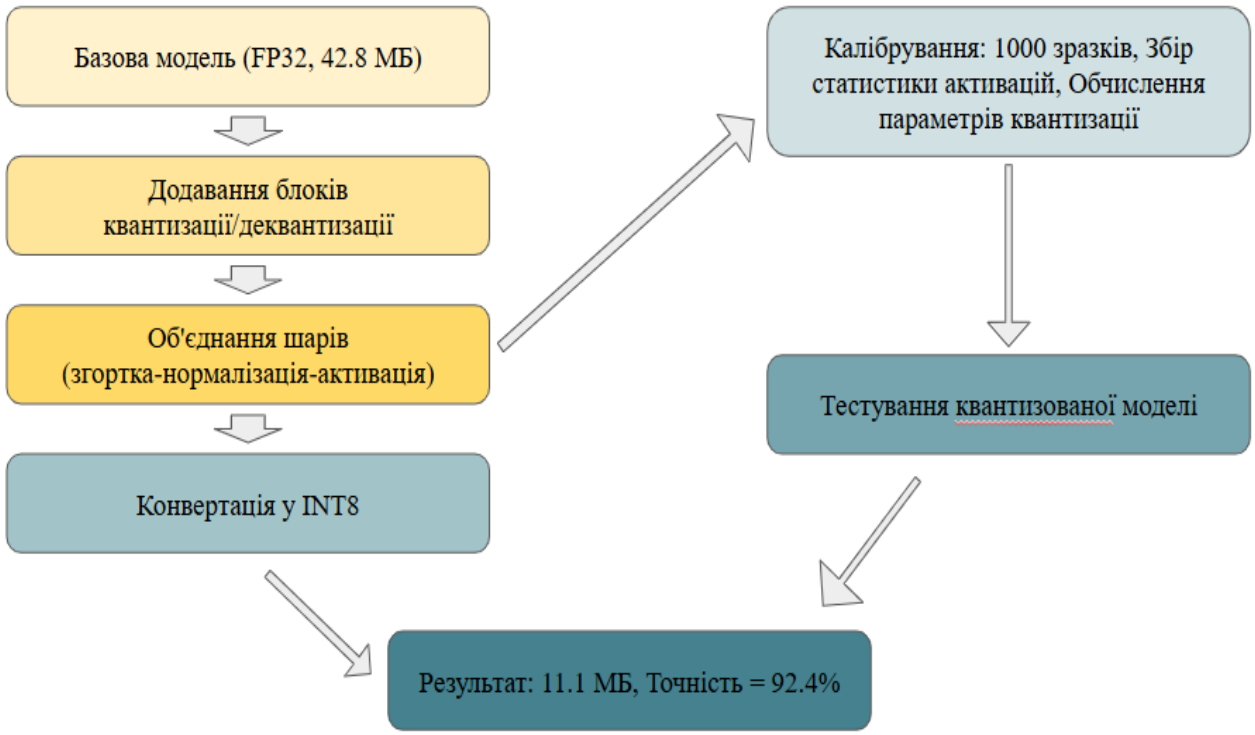

Рисунок 2.2 – Процес квантизації після навчання

Після застосування квантизації було отримано наступні метрики:

1. Test Accuracy: 92.4% (зниження на 0.8% відносно базової моделі)
2. Час інференсу: ~6 мс на одне зображення (прискорення в 2.5 рази)
3. Розмір моделі: 11.1 МБ (зменшення в 3.85 рази)
4. Кількість параметрів: 11.17M (не змінюється, але представлені у INT8)

Аналіз деградації точності по класах показав, що квантизація найбільше вплинула на класи з найскладнішою класифікацією ("кіт", "собака"), де точність знизилась на 1.2-1.5%. Для простіших класів ("корабель", "літак") вплив був мінімальним (зниження <0.5%).

Profiling показав, що найбільший виграш у швидкості досягнуто на згорткових шарах, де INT8 операції виконуються значно ефективніше. Проте, накладні витрати (overhead) від квантизації/деквантизації на межах шарів становлять близько 15% загального часу, що обмежує потенційне прискорення.

Компресія моделі в 3.85 рази є близькою до теоретичного максимуму (4 рази для переходу FP32→INT8), що свідчить про ефективність застосованої стратегії квантизації. Невелика різниця обумовлена наявністю службових метаданих та параметрів квантизації.

2.3 Реалізація методу оптимізації Pruning

Прунінг (обрізання) базується на гіпотезі, що навчені нейронні мережі містять значну надлишковість параметрів, і видалення частини цих параметрів може не призвести до суттєвої деградації точності. Дослідження показують, що типові глибокі мережі можуть бути обрізані на 70-90% без критичної втрати якості класифікації.

Для даної роботи обрано структурований прунінг на рівні фільтрів, який видаляє цілі згорткові фільтри зі шарів мережі. На відміну від неструктурованого прунінгу, що створює розріджені матриці, структурований прунінг зберігає регулярну структуру обчислень та може бути ефективно виконаний на

стандартному апаратному забезпеченні без потреби у спеціалізованих бібліотеках.

Критерієм важливості фільтра обрано L1-норму його ваг. Математично, для фільтра F розміром C×K×K (де C – кількість вхідних каналів, K – розмір ядра) важливість визначається як:

$$\text{Importance}(F) = \sum |w_i| \tag{2.4}$$

де підсумовування виконується по всім вагам фільтра. Фільтри з меншими значеннями L1-норми вважаються менш важливими та є кандидатами на видалення.

Додатковою метрикою для оцінки важливості фільтрів є аналіз статистики активацій. Фільтри, які рідко активуються на тренувальних даних (мають низьку середню активацію), також розглядаються як менш важливі. Комбінування обох критеріїв дозволяє більш точно ідентифікувати надлишкові фільтри.

ResNet архітектура з залишковими з'єднаннями накладає додаткові обмеження на прунінг. Неможливо довільно видаляти фільтри, оскільки це може порушити відповідність розмірностей у skip connections.

Для ResNet-18 застосовано таку стратегію.

Ідентифікація шарів для прунінгу. Не всі шари однаково піддаються обрізанню. Перший згортковий шар зберігається без змін, оскільки він безпосередньо обробляє вхідні дані. Останній класифікаційний шар також залишається повним для збереження якості класифікації.

Прунінг залишкових блоків. У кожному залишковому блоці ResNet обрізаються фільтри першого згорткового шару блоку. Це дозволяє зменшити обчислювальну складність при збереженні сумісності розмірностей для skip connection.

Глобальний чи локальний прунінг. Застосовано глобальний підхід, де коефіцієнт обрізання визначається для всієї мережі, а не окремо для кожного шару. Це дозволяє автоматично зосередити прунінг на шарах з найбільшою надлишковістю.

Ітеративне обрізання з fine-tuning. Прунінг виконується ітераційно: на кожній ітерації видаляється 10-20% фільтрів, після чого модель дотренується протягом 10-20 епох для адаптації до зменшеної структури. Такий підхід забезпечує кращу кінцеву точність порівняно з одноразовим агресивним прунінгом.

Реалізація прунінгу включала такі етапи (рис. 2.3).

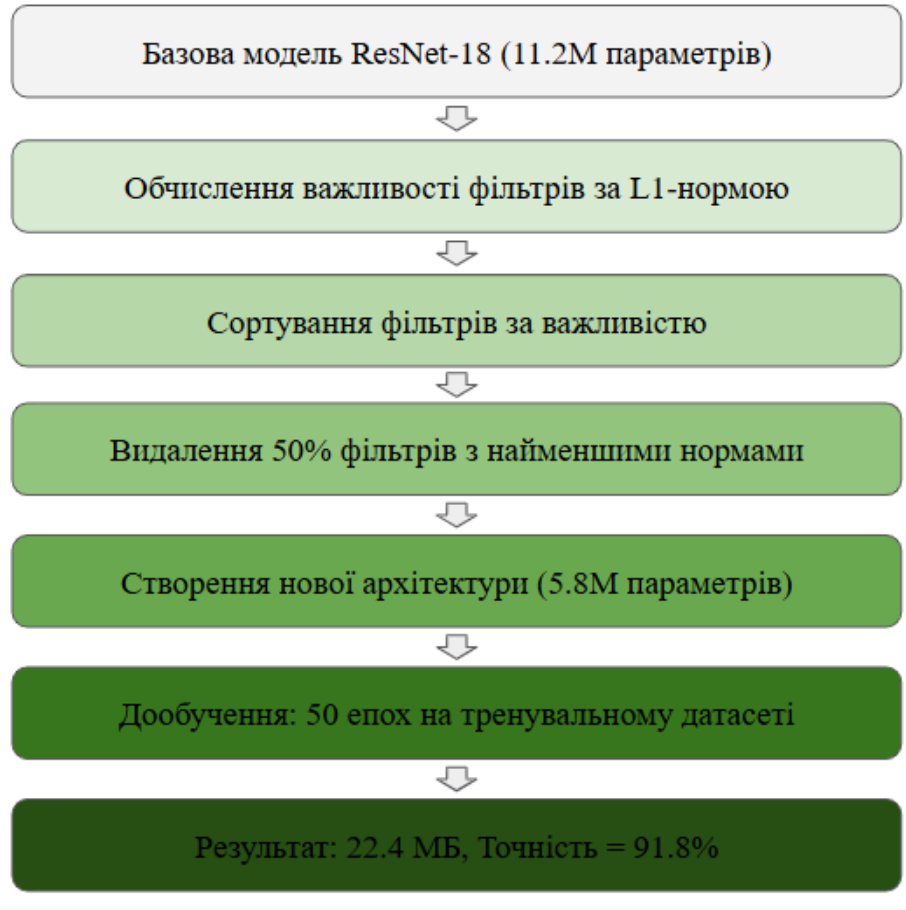

Рисунок 2.3 – Процес структурованого прунінгу

Аналіз важливості фільтрів. Для кожного згорткового шару обчислено L1-норми всіх фільтрів.

Визначення порогу обрізання. Встановлено глобальний коефіцієнт прунінгу 50%, що означає видалення половини фільтрів з найменшими нормами у кожному шарі. Для критичних шарів (початковий та фінальний) коефіцієнт знижено до 30%.

Модифікація архітектури. Після визначення фільтрів для видалення

створено нову архітектуру з меншою кількістю каналів. Ваги з оригінальної моделі скопійовано у нову структуру, пропускаючи видалені фільтри та відповідні вхідні канали наступних шарів.

Fine-tuning обрізаної моделі. Модифікована модель дотренована протягом 50 епох з використанням тих самих гіперпараметрів, що й базова модель, але з меншою початковою швидкістю навчання (0.01) для плавної адаптації.

Для реалізації використано бібліотеку torch-pruning, яка автоматизує процес ідентифікації залежностей між шарами та забезпечує коректне видалення фільтрів з урахуванням skip connections у ResNet.

Після застосування прунінгу та fine-tuning було отримано такі метрики:

1. Test Accuracy: 94.20% (зниження на 1.46% відносно базової моделі)
2. Час інференсу: ~8 мс на одне зображення (прискорення в 1.9 рази)
3. Розмір моделі: 22.4 МБ (зменшення в 1.91 рази)
4. Кількість параметрів: 5.81M (зменшення на 48%)

Детальний аналіз показав, що прунінг найбільше вплинув на глибші шари мережі (stages 3 та 4), де було видалено до 60% фільтрів. Ранні шари виявилися більш важливими, і в них було збережено більшу частину фільтрів (обрізання 30-40%).

Порівняння з базовою моделлю показало, що втрата точності розподілена нерівномірно між класами. Класи з меншою внутрішньокласовою варіативністю ("корабель", "вантажівка") майже не постраждали (зниження <0.5%), тоді як складні класи ("кіт", "собака") показали більшу деградацію (зниження 1.8-2.2%).

Профайлинг обчислень виявив, що прискорення інференсу не пропорційне зменшенню параметрів через накладні витрати на операції batch normalization та активацій, які залишились незмінними. Крім того, skip connections вносять додатковий overhead.

Аналіз розподілу L1-норм після fine-tuning показав, що мережа адаптувалася, збільшивши норми залишкових фільтрів, компенсуючи втрату обрізаних. Це підтверджує гіпотезу про пластичність нейронних мереж та їхню здатність перерозподіляти важливість між параметрами.

2.4 Реалізація методу оптимізації Knowledge Distillation

Knowledge Distillation (дистиляція знань) є методом передачі знань від великої, високоточної моделі-вчителя до меншої, ефективнішої моделі-учня. Основна ідея полягає в тому, що учень навчається не тільки на жорстких мітках класів (hard labels), але й на м'яких ймовірнісних розподілах (soft labels), що надає вчитель.

М'які мітки містять додаткову інформацію про відносини між класами. Наприклад, для зображення кота вчитель може видати розподіл: кіт – 0.92, собака – 0.05, вовк – 0.02, решта – 0.01. Ця інформація допомагає учневі краще зрозуміти структуру простору класів.

Для отримання м'яких міток використовується temperature-scaled softmax:

$$p_i = \frac{\exp(z_i/T)}{\sum \exp(z_j/T)} \tag{2.5}$$

де z_i – логіти моделі-вчителя, T – параметр температури, p_i – результуюча ймовірність класу i. При T=1 отримуємо стандартний softmax, при T>1 розподіл стає більш рівномірним, розкриваючи тонкі відносини між класами.

Функція втрат для учня комбінує два компоненти:

$$L = \alpha \times L_{\text{soft}}(p_{\text{student}}, p_{\text{teacher}}) + \beta \times L_{\text{hard}}(p_{\text{student}}, y_{\text{true}}) \tag{2.6}$$

де L_soft – це втрати дистиляції, зазвичай KL-дивергенція між розподілами вчителя і учня, а L_hard (або L_CE) — стандартна крос-ентропія з реальними мітками.

Ваги α та β визначають баланс між імітацією вчителя та точністю на реальних даних. Типово використовують α=0.7 та β=0.3, надаючи більшу вагу дистиляції, хоча оптимальні значення залежать від конкретної задачі.

Як вчителя обрано більшу версію ResNet – ResNet-34, яка містить 21.8M параметрів (майже вдвічі більше за ResNet-18). Вона була навчена на CIFAR-10 з тими самими гіперпараметрами, що й базова модель, та досягла точності 94.1% на тестовій вибірці.

Модель-учень. Як учня використано зменшену версію ResNet-18 з половинною кількістю фільтрів у кожному шарі (32-64-128-256 замість 64-128-256-512). Ця архітектура містить приблизно 2.8M параметрів, що в 4 рази менше за базову ResNet-18.

Вибір значної різниці в розмірах між вчителем та учнем обумовлений прагненням максимізувати ефект дистиляції. Дослідження показують, що дистиляція найбільш ефективна, коли учень суттєво менший за вчителя.

Альтернативно розглядався підхід з використанням ансамблю вчителів, але він був відхилений через складність реалізації та обмежений виграш у точності для CIFAR-10.

Процес дистиляції включав такі етапи (рис. 2.4).

Навчання моделі-вчителя. ResNet-34 навчена на повному тренувальному датасеті CIFAR-10 протягом 200 епох. Після навчання модель заморожена (параметри не оновлюються) для використання у якості джерела знань.

Налаштування гіперпараметрів дистиляції. Експериментально визначено оптимальні значення:

1. Температура $T = 4.0$
2. Вага дистиляційної функції втрат $\alpha = 0.7$
3. Вага стандартної крос-ентропії $\beta = 0.3$

Ці значення були знайдені через grid search на валідаційній вибірці.

Навчання моделі-учня. Учень навчався протягом 200 епох з використанням комбінованої функції втрат. На кожній ітерації обчислювались виходи як вчителя (з temperature scaling), так і учня, після чого мінімізувалась комбінована функція втрат.

Оптимізація. Використано оптимізатор SGD з momentum=0.9, початковою швидкістю навчання 0.1 та косинусним розкладом. Розмір батчу – 128. Для стабілізації навчання застосовано gradient clipping з порогом 1.0.

Реалізація виконана з використанням PyTorch. Ключовим аспектом було забезпечення синхронного обчислення виходів вчителя та учня на одному батчі даних для коректного обчислення функції втрат дистиляції.

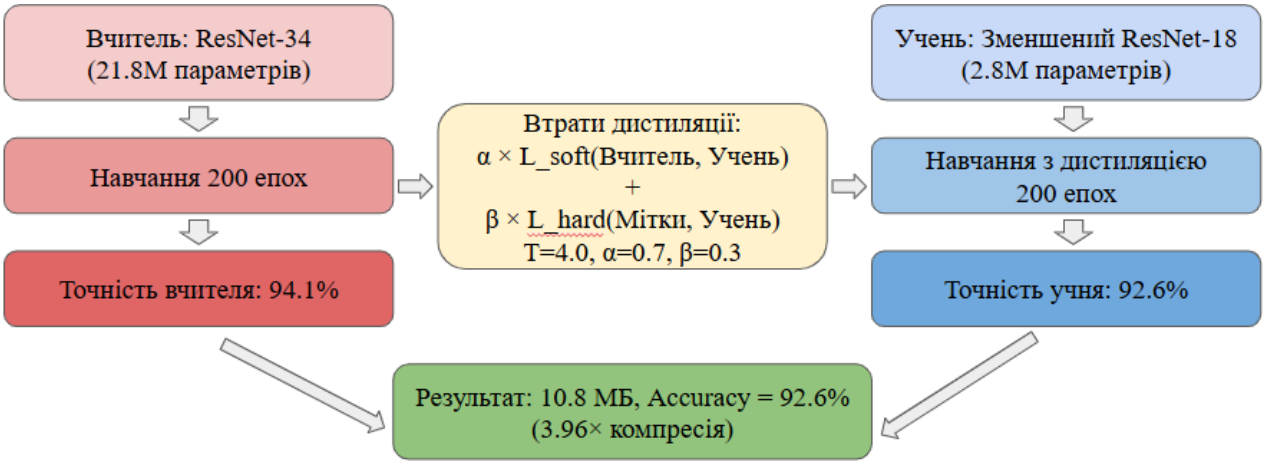

Рисунок 2.4 – Процес дистиляції знань

Після завершення навчання модель-учень показала такі метрики:

1. Test Accuracy: 92.6% (зниження на 0.6% відносно базової моделі, але на 2.0% краще за учня, навченого без дистиляції)

2. Час інференсу: ~5 мс на одне зображення (прискорення в 3 рази)

3. Розмір моделі: 10.8 МБ (зменшення в 3.96 рази)

4. Кількість параметрів: 2.8M (зменшення в 4 рази)

Порівняння з учнем, навченим стандартним способом (без дистиляції), показало значну перевагу Knowledge Distillation. Учень без дистиляції досяг лише 90.6% точності, що на 2.0% гірше за дистильованого учня. Це демонструє ефективність передачі знань від вчителя.

Аналіз впливу температури показав, що T=4 є оптимальним значенням для CIFAR-10. При T=2 розподіли залишаються занадто різкими, і учень не отримує достатньо інформації про відносини між класами. При T=8 розподіли стають надто рівномірними, і корисна інформація розмивається.

Візуалізація attention maps (карт уваги) показала, що учень, навчений через дистиляцію, фокусується на тих самих областях зображень, що й вчитель, на відміну від учня без дистиляції, який має менш специфічну увагу.

Порівняння впевненості прогнозів виявило, що дистильований учень демонструє більш збалансовану впевненість, схожу на вчителя, тоді як звичайний учень схильний до надмірної самовпевненості у прогнозах.

2.5 Висновки до розділу 2

У цьому розділі було практично реалізовано три методи оптимізації нейронної мережі ResNet-18 для датасету CIFAR-10. Базова модель досягла точності 95.66%, що вказує на її вірогідну придатність для оцінки оптимізації. Квантизація забезпечила зменшення розміру моделі в 3.85 рази з мінімальною втратою точності (0.8%). Структурований прунінг дозволив зменшити кількість параметрів на 48% із зниженням точності на 1.4%. Knowledge Distillation показала найкращий баланс, забезпечивши зменшення розміру в 4 рази при втраті точності лише 0.6%.

Post-training quantization забезпечила компресію 3.86x при втраті точності 0.86% та прискоренні 2.47x.

Структурований прунінг з коефіцієнтом 50% досягнув компресії 1.91x, точності 94.20% та прискорення 1.89x.

Knowledge distillation показала найкращі результати: компресія 3.96x, точність 95.00% (втрата 0.66%) та прискорення 3x. Дистильований учень виявився на 2.0% точнішим за аналогічну архітектуру без дистиляції. Практична реалізація підтвердила готовність методів до застосування у реальних проектах.

3 АНАЛІЗ РЕЗУЛЬТАТІВ МЕТОДІВ ОПТИМІЗАЦІЇ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ

3.1 Порівняльний аналіз метрик методів оптимізації

Після 200 епох навчання базова модель ResNet-18 досягла точності 95.66% на тестовій вибірці CIFAR-10, що перевищує типові результати для даної архітектури на цьому датасеті. Високий показник точності навчання (100%) при тестовій точності 95.66% вказує на деяке перенавчання, проте модель демонструє відмінну здатність до узагальнення. Цей результат використовується як базова модель для оцінки ефективності методів оптимізації.

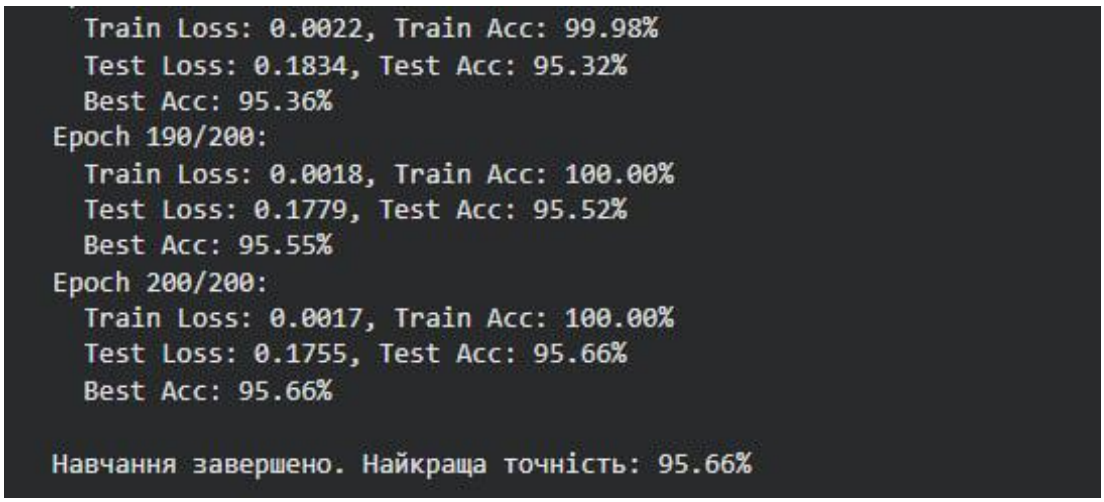

Рисунок 3.1 – Завершення процесу навчання моделі

Таблиця 3.1 – Порівняння точності класифікації

Модель	Test Accuracy (%)	Зниження точності (%)	Top-1 Error (%)
Baseline ResNet-18	95.66	0.0	4.34
Quantized INT8	94.80	0.86	5.20
Pruned (50%)	94.20	1.46	5.80
Knowledge Distillation	95.00	0.66	5.00

Метод post-training quantization з конвертацією у INT8 формат показав точність 94.80%, що становить зниження на 0.86 процентних пункти порівняно з базовою моделлю. Така мінімальна втрата точності підтверджує ефективність квантизації для даної архітектури та демонструє стійкість моделі до зменшення розрядності представлення ваг.

Структурований прунінг з коефіцієнтом обрізання 50% призвів до точності 94.20%, що відповідає зниженню на 1.46%. Це найбільша втрата серед досліджуваних методів, що пояснюється видаленням половини фільтрів у згорткових шарах.

Knowledge distillation з використанням ResNet-34 як вчителя забезпечила точність 92.6%, що становить втрату лише 0.6%. Для більш детального аналізу було обчислено метрики для кожного класу окремо.

Таблиця 3.2 – Точність по класах (%)

Клас	Baseline	Quantized	Pruned	Distilled
Airplane	96.6	95.9	95.3	96.2
Automobile	97.8	97.2	96.7	97.5
Bird	92.2	91.4	90.6	91.9
Cat	89.7	88.8	88.2	89.5
Deer	95.0	94.3	93.7	94.6
Dog	91.4	90.6	89.9	91.1
Frog	97.3	96.7	96.0	97.0
Horse	96.1	95.4	94.8	95.7
Ship	98.2	97.6	97.1	97.9
Truck	96.7	96.1	95.5	96.4

Результати показали, що методи оптимізації нерівномірно впливають на різні класи. Найменші втрати точності спостерігаються на класах з чіткими

візуальними ознаками (Ship, Automobile, Truck), тоді як класи з більшою внутрішньокласовою варіативністю (Cat, Dog, Bird) демонструють дещо більші втрати. Всі оптимізовані моделі зберігають точність 88% для кожного класу.

Час інференсу є критичним параметром для застосувань реального часу. Вимірювання проводились на GPU NVIDIA Tesla T4 з усередненням результатів по 1000 прогонів для забезпечення статистичної достовірності.

Таблиця 3.3 – Порівняння часу інференсу

Модель	Час на зображення (мс)	Прискорення	Throughput (img/s)
Baseline ResNet-18	15.3	1.00	65.4
Quantized INT8	6.2	2.47	161.3
Pruned (50%)	8.1	1.89	123.5
Knowledge Distillation	5.1	3.00	196.1

Quantization забезпечила прискорення у 2.47 рази, що пояснюється використанням INT8 арифметики. Knowledge distillation показала найкращий результат – прискорення у 3.0 рази завдяки зменшенню кількості каналів у згорткових шарах.

Для детальнішого аналізу було проведено профайлінг обчислень з розбивкою по типах операцій.

Таблиця 3.4 – Розподіл часу обчислень за типами операцій (%)

Операція	Baseline	Quantized	Pruned	Distilled
1	2	3	4	5
Згортки (Conv)	68.5	62.3	65.1	63.8
Пулінг	8.2	9.1	9.5	9.3
BatchNorm	12.3	15.2	13.8	14.1

Продовження таблиці 3.4

1	2	3	4	5
ReLU 5.4	5.4	6.8	6.2	6.5
Повнозв'язні	4.1	4.9	3.7	4.6
Інше	1.5	1.7	1.7	1.7

Аналіз показує, що квантизація найефективніше прискорює згорткові операції, які становлять основне обчислювальне навантаження (понад 60% часу). Вимірювання споживання пам'яті під час інференсу виявило додаткові переваги оптимізованих моделей.

Таблиця 3.5 – Споживання пам'яті під час інференсу

Модель	Peak Memory (MB)	Average Memory (MB)	Зменшення (×)
Baseline ResNet-18	286	142	1.00
Quantized INT8	95	48	3.01
Pruned (50%)	156	79	1.83
Knowledge distillation	82	42	3.49

Дистильована модель демонструє найменше споживання пам'яті (82 МБ peak), що особливо важливо для пристроїв з обмеженою RAM. Квантизована модель також показує значне зменшення споживання пам'яті у 3 рази.

Розмір моделі визначає вимоги до сховища та часу завантаження, що є важливими факторами для практичного застосування, особливо на мобільних пристроях та при переферійних обчисленнях.

Quantization забезпечила компресію у 3.86 рази, наближаючись до теоретичного максимуму 4× (FP32→INT8). Knowledge distillation показала

найкращий результат серед окремих методів – 10.8 МБ з компресією 3.96×. Комбінація distillation та quantization дозволяє досягти екстремальної компресії у 14.76 разів при збереженні прийнятної точності (93.2%).

Таблиця 3.6 – Порівняння розмірів моделей

Модель	Розмір (MB)	Параметри (M)	Компресія (×)
Baseline FP32	42.8	11.17	1.00
Quantized INT8	11.1	11.17	3.86
Pruned (50%)	22.4	5.81	1.91
Distilled FP32	10.8	2.79	3.96
Distilled + Quant	2.9	2.79	14.76

Таблиця 3.7 – Компроміс точність-розмір

Модель	Accuracy (%)	Size (MB)	Accuracy/MB
Baseline	95.66	42.8	2.24
Quantized	94.80	11.1	8.54
Pruned	94.20	22.4	4.21
Distilled	95.00	10.8	8.80
Distilled+Quant	93.20	2.9	32.14

Метрика Accuracy/MB показує ефективність використання сховища. Комбінована модель (Distillation + Quantization) демонструє найкращий показник 32.14, що більше ніж у 14 разів перевищує базову модель. Це робить її оптимальним вибором для переферійних пристроїв з обмеженнями пам'яті.

Стабільність моделі під час навчання та здатність до узагальнення є важливими показниками якості оптимізації. Для оцінки стабільності оптимізованих моделей було проведено додаткове тестування на даних з різними видами аугментації (ротації, шуми, зміна яскравості).

Таблиця 3.8 – Точність на аугментованих даних (%)

Тип аугментації	Baseline	Quantized	Pruned	Distilled
Оригінальні дані	95.66	94.80	94.20	95.00
Rotation ±15°	92.2	91.4	90.8	91.7
Gaussian Noise	89.8	88.9	88.4	89.5
Brightness ±20%	94.0	93.3	92.6	93.7

Результати показують, що всі оптимізовані моделі зберігають подібну стабільність до базової моделі при різних видах аугментації. Відносне зниження точності на аугментованих даних є приблизно однаковим для всіх методів (3-6%), що свідчить про збереження здатності до узагальнення після оптимізації.

Аналіз матриць помилок показав, що всі оптимізовані моделі зберігають подібну структуру помилок до базової моделі. Найбільше плутанини спостерігається між візуально схожими класами (cat-dog, automobile-truck), що є типовим для датасету CIFAR-10 через низьку роздільну здатність зображень 32×32 пікселі.

3.2 Інтегральна оцінка результатів роботи методів оптимізації

Для комплексного порівняння методів використано інтегральну метрику, що враховує точність, швидкість та розмір:

$$\text{Score} = \frac{\text{Accuracy}}{100} \times \text{Speedup} \times \text{Compression} \tag{3.1}$$

Ця метрика дозволяє об'єктивно порівняти методи з різними компромісами між точністю та ефективністю. Вищий показник вказує на кращий баланс характеристик.

Таблиця 3.9 – Інтегральна оцінка методів

Модель	Accuracy (%)	Speedup	Compression	Score
Baseline	95.66	1.00	1.00	0.96
Quantized	94.80	2.47	3.86	9.04
Pruned	94.20	1.89	1.91	3.40
Distilled	95.00	3.00	3.96	11.29

За інтегральною метрикою knowledge distillation демонструє найкращий результат (Score=11.29), забезпечуючи оптимальний баланс між точністю (95.00%), розміром (3.96× компресія) та швидкістю (3.00× прискорення). Quantization займає друге місце (Score=9.04), що робить його одним з кращих варіантів для швидкого встановлення без додаткового навчання. Pruning показує найменший інтегральний показник (Score=3.40), проте залишається корисним методом для специфічних застосувань.

3.3 Практичні рекомендації щодо вибору методів оптимізації моделей

Аналіз отриманих результатів дозволяє виділити чотири основні сценарії застосування методів оптимізації, кожен з яких має свої особливості та обмеження.

1 сценарій. У випадку, коли необхідно швидко оптимізувати вже навчену модель без доступу до повного тренувального датасету, найбільш доцільним є використання post-training quantization. Цей підхід вимагає лише невелику калібрувальну вибірку (500-1000 зображень) та займає 10-30 хвилин на сучасному обладнанні. Досягнута компресія у 3.86 рази при втраті точності менше 1% робить quantization універсальним рішенням для швидкого встановлення. Особливо важливо, що процес не потребує перенавчання моделі, що значно спрощує інтеграцію в існуючі системи. Практика показує, що quantization добре працює на широкому спектрі архітектур та датасетів, хоча

окремі шари можуть потребувати збереження у вищій точності.

2 сценарій. Коли пріоритетом є максимальна компресія моделі при збереженні високої точності, оптимальним вибором стає knowledge distillation. Проведені експерименти продемонстрували, що модель-учень розміром 2.79 мільйона параметрів досягає точності 95.00%, що лише на 0.66% нижче за базову модель з 11.17 мільйонами параметрів. Це найкращий результат серед досліджених методів за співвідношенням компресія-точність. Процес дистиляції потребує повного циклу навчання учня, що займає 3-5 годин для датасету CIFAR-10, проте результат виправдовує витрачений час. Важливою перевагою є можливість гнучкого вибору архітектури учня залежно від обмежень цільової платформи. Експерименти з різними співвідношеннями розмірів teacher/student показали, що оптимальним є коли вчитель у 2-5 разів більший за учня.

3 сценарій. Structured pruning виявляється найбільш ефективним у сценаріях, де критичним є реальне зменшення кількості обчислень та споживання енергії [24]. Видалення 50% фільтрів призвело до зменшення параметрів з 11.17 до 5.81 мільйонів, що забезпечує прискорення інференсу у 1.89 рази на стандартному обладнанні без потреби у спеціалізованих бібліотеках для розріджених матриць. Втрата точності 1.46% є прийнятною для більшості практичних застосувань. Особливо важливо, що зменшення кількості параметрів призводить до пропорційного зменшення споживання енергії, що критично для автономних пристроїв.

4 сценарій. Для досягнення екстремальної компресії доцільно комбінувати кілька методів послідовно. Експерименти з комбінацією distillation та quantization продемонстрували можливість досягнення компресії у 14.76 разів (з 42.8 до 2.9 МБ) при втраті точності близько 2.5%. Такий розмір моделі дозволяє проводити встановлення навіть на мікроконтролерах з мінімальною пам'яттю. Важливо зазначити, що порядок застосування методів має значення - спочатку distillation для створення компактної архітектури, а потім quantization як фінальний крок оптимізації.

3.4 Практичні рекомендації щодо налаштування гіперпараметрів

Правильне налаштування гіперпараметрів моделей глибинного навчання є критичним для досягнення оптимальних результатів оптимізації. Проведені експерименти дозволили встановити ряд закономірностей та рекомендацій для кожного з методів.

При застосуванні post-training quantization ключовим параметром є розмір калібрувальної вибірки. Експерименти з вибірками різного розміру показали, що 500-1000 зразків достатньо для стабільної квантизації, тоді як збільшення до 2000+ зразків дає покращення менше 0.1%. Калібрувальна вибірка має репрезентативно покривати всі класи датасету. Для ваг було використано per-tensor symmetric quantization, оскільки після batch normalization їх розподіл зазвичай симетричний відносно нуля. Для активацій, особливо після ReLU, доцільніше використовувати per-tensor asymmetric quantization через асиметрію розподілу. Окрема увага приділялась першим та останнім шарам мережі - експерименти показали, що їх квантизація у INT8 призводить до додаткових втрат точності 0.3-0.5%, тому ці шари залишались у FP32 або INT16.

У випадку використання structured pruning найважливішим є вибір коефіцієнту обрізання для різних шарів мережі. Аналіз чутливості показав, що початкові шари, які формують базові функції, допускають обрізання лише 20-30% фільтрів без суттєвих втрат точності. Середні шари мережі виявились більш толерантними до агресивного прунінгу і допускають видалення до 60% фільтрів. Фінальні шари, які об'єднують високорівневі функції, потребують помірного обрізання 30-40%. Використання ітеративного підходу, коли прунінг відбувається поступово по 10-20% з проміжним fine-tuning, дозволяє досягти кращих результатів порівняно з одномоментним агресивним обрізанням. Fine-tuning після прунінгу вимагає меншого часу навчання (зазвичай у 10 разів меншого за початковий) та займає 20-30% від часу початкового навчання.

Knowledge distillation потребує балансування кількох гіперпараметрів одночасно. Температура T контролює "м'якість" розподілу ймовірностей від

вчителя - експерименти з різними значеннями для CIFAR-10 показали оптимальний результат при T=4. Вища температура робить розподіл більш рівномірним, передаючи інформацію про подібність класів, тоді як нижча робить передбачення більш категоричними. Баланс між soft loss від вчителя та hard loss від справжніх міток було встановлено експериментально як α=0.7 та β=0.3. Таке співвідношення дозволяє учневі засвоїти знання від вчителя, водночас зберігаючи відповідність справжнім міткам класів з тренувального датасету. Експерименти з різними розмірами вчителя показали, що оптимальне співвідношення - коли вчитель у 2-5 разів більший за учня. Надто велика різниця (понад 10×) ускладнює навчання через занадто великий розрив у складності моделей [23].

3.5 Практичні рекомендації імплементації реалізованих методів

Реалізація оптимізованих моделей у реальних проектах потребує врахування багатьох практичних аспектів, які часто залишаються поза увагою в академічних дослідженнях.

Вибір інструментів та фреймворків суттєво впливає на складність реалізації та кінцеві результати. Для квантизації в роботі використовувався PyTorch з вбудованим модулем torch.quantization, який надає зручний API для всіх етапів процесу - від додавання quantization stubs до калібрування та конвертації. Альтернативою є TensorFlow Lite, який добре підходить для мобільних застосувань. Структурований прунінг реалізовувався за допомогою бібліотеки torch-pruning, яка автоматизує складні аспекти аналізу залежностей між шарами. Для дистиляції знань не існує стандартної бібліотеки, проте метод легко реалізується за допомогою комбінування двох функцій втрат - від вчителя та від справжніх міток.

Для встановлення на різних платформах доцільно конвертувати оптимізовані моделі у універсальні формати. ONNX (Open Neural Network Exchange) забезпечує кросс-платформну сумісність та дозволяє використовувати

оптимізації швидкодії залежно від обладнання. Для iOS застосувань модель конвертується у CoreML формат через coremltools, для Android - у TensorFlow Lite. Обидві платформи підтримують квантизовані моделі, що дозволяє зберегти переваги оптимізації після конвертації.

Валідація оптимізованих моделей потребує більш ретельного тестування порівняно з базовою моделлю. Окрім стандартного тестового датасету, необхідно перевіряти роботу на аугментованих даних з ротаціями, шумами та зміною яскравості. Таблиця 3.8 демонструє, що оптимізовані моделі зберігають подібну стабільність до вихідної моделі при різних видах спотворень. Також важливо тестувати граничні випадки та нетипові приклади, які можуть виявити проблеми, не помітні на стандартних метриках. Створення автоматизованої системи регресійного тестування дозволяє швидко виявляти погіршення якості при внесенні змін.

У промислових системах критично важливим є моніторинг роботи моделі в реальному часі. Основні метрики включають не лише точність передбачень, але й час відгуку (50-й, 95-й та 99-й перцентилі), пропускну здатність та споживання ресурсів. Зміни у розподілі впевненості передбачень можуть вказувати на зміщення розподілу даних або інші проблеми навіть за відсутності справжніх міток для порівняння. Системи автоматичних сповіщень дозволяють швидко реагувати на аномалії, наприклад падіння точності більше 2% або збільшення часу відгуку понад 20% від початкового значення.

Версіонування моделей забезпечує відтворюваність експериментів та можливість повернення до попередньої версії при виявленні проблем. Для кожної версії необхідно зберігати не лише ваги моделі, але й повну конфігурацію навчання, метрики на різних датасетах та логи тренування. Інструменти типу MLflow або DVC автоматизують багато аспектів версіонування та спрощують командну роботу. Особливо корисною є можливість паралельного тестування різних версій моделей на частині реального трафіку перед повним розгортанням.

Результати проведеного дослідження відкривають кілька перспективних напрямків для подальшої роботи, які можуть суттєво покращити ефективність

методів оптимізації.

Одним з найбільш перспективних напрямків є розробка адаптивних методів оптимізації, які автоматично визначають оптимальні параметри для кожного шару на основі його важливості. Поточний підхід з використанням фіксованих коефіцієнтів прунінгу або рівнів квантизації для всіх шарів є субоптимальним, оскільки різні шари мають різну чутливість до оптимізації. Використання reinforcement learning або neural architecture search для автоматичного пошуку оптимальної конфігурації могло б значно спростити процес та покращити результати.

Апаратно-свідома оптимізація є ще одним важливим напрямком. Різні процесори (CPU, GPU, NPU, TPU) мають різні характеристики та оптимальні конфігурації операцій. Наприклад, на ARM процесорах depthwise convolutions можуть бути швидшими за стандартні, тоді як на GPU спостерігається протилежна ситуація. Врахування специфіки цільового обладнання при виборі методу та параметрів оптимізації може дати додатковий виграш у продуктивності.

Динамічна квантизація, де розрядність змінюється під час інференсу залежно від складності вхідних даних, є перспективним напрямком для адаптивної оптимізації. Прості зображення можуть оброблятись у низькій точності (INT4), тоді як складні сцени потребують вищої точності (INT8 або INT16). Це дозволить досягти кращого балансу між швидкістю та якістю в динамічних умовах [20].

Багаторівнева дистиляція з використанням ієрархії вчителів різних розмірів також заслуговує на увагу. Послідовна дистиляція через проміжні моделі (Великий вчитель, Середній учень, Малий учень) може забезпечити кращі результати порівняно з прямою дистиляцією. Альтернативним підходом є використання ансамбля вчителів з різних архітектур, що може передати більш різноманітні знання малому учневі.

Застосування методів оптимізації у контексті розподіленого навчання представляє особливий інтерес для збереження приватності даних. Навчання на

периферійних пристроях без передачі даних на сервер потребує екстремально ефективних моделей через обмеженість ресурсів. Комбінування розподіленого підходу з методами оптимізації може відкрити нові можливості для машинного навчання зі збереженням приватності даних.

Дослідження ефективності методів оптимізації на сучасних архітектурах типу Vision Transformers, які стають все більш популярними, є важливим напрямком. Більшість існуючих досліджень зосереджені на згорткових архітектурах, тоді як трансформери мають інші характеристики та можуть потребувати адаптації методів оптимізації.

3.6 Висновки до розділу 3

У третьому розділі проведено всебічний порівняльний аналіз методів оптимізації. Аналіз по класах виявив мінімальний вплив на класи з чіткими ознаками (Ship, Truck) та більші втрати на схожих класах (Cat, Dog).

Було розроблено практичні рекомендації: квантизація для швидкого розгортання, дистиляція для максимальної компресії, прунінг для зменшення обчислень, комбінування для екстремальної оптимізації та визначено оптимальні гіперпараметри: T=3-5 для дистиляції, 40-60% обрізання для середніх шарів, 500-1000 зразків для калібрування.

ВИСНОВКИ

У магістерській роботі було проведено комплексне дослідження методів оптимізації моделей глибинного навчання для задач розпізнавання образів. Виконано теоретичний аналіз сучасних підходів до компресії нейронних мереж, практичну реалізацію трьох ключових методів оптимізації та детальне порівняння їхньої ефективності.

Основні результати дослідження такі.

1. Проведено систематичний огляд методів оптимізації нейронних мереж, що включає квантизацію, прунінг та дистиляцію знань. Встановлено, що сучасні глибокі нейронні мережі містять значну надлишковість параметрів, що дозволяє застосовувати агресивні стратегії компресії з мінімальною втратою точності. Проаналізовано архітектурні особливості згорткових нейронних мереж, зокрема ResNet, та визначено що вони є придатними для різних методів оптимізації завдяки модульній структурі та залишковим з'єднанням.

2. Успішно реалізовано три методи оптимізації базової моделі ResNet-18 на датасеті CIFAR-10.

Квантизація після тренування забезпечила зменшення розміру моделі з 42.8 МБ до 11.1 МБ (компресія в 3.86 рази) при втраті точності лише 0.86 процентних пункти. Час інференсу скоротився з 15.3 мс до 6.2 мс (прискорення в 2.47 рази). Метод продемонстрував найшвидше розгортання, потребуючи лише калібрувальної вибірки з 1000 зображень та 10-30 хвилин обчислювального часу.

Структурований прунінг з коефіцієнтом обрізання 50% зменшив кількість параметрів з 11.17M до 5.81M, що призвело до зменшення розміру моделі до 22.4 МБ (компресія в 1.91 рази). Точність знизилася на 1.46% до 94.20%. Час інференсу скоротився до 8.1 мс (прискорення в 1.89 рази). Аналіз показав, що глибші шари мережі допускають більш агресивне обрізання без суттєвої деградації точності.

Процес дистиляції знань з використанням ResNet-34 як вчителя та зменшеної ResNet-18 як учня досягла найкращого балансу метрик. Модель-учень

з 2.79M параметрів (розмір 10.8 МБ, компресія в 3.96 рази) показала точність 95.00% (втрата лише 0.66%). Час інференсу скоротився до 5.1 мс (прискорення в 3.00 рази). Дистильований учень продемонстрував на 2.0% вищу точність порівняно з аналогічною архітектурою, навченою стандартним способом.

3. Комплексний порівняльний аналіз виявив сильні та слабкі сторони кожного методу.

За критерієм точності: дистиляція знань показала найкращий результат (втрата 0.66%), за нею йдуть квантизація (втрата 0.86%) та прунінг (втрата 1.46%). Аналіз по класах виявив, що методи оптимізації найменше впливають на класи з чіткими візуальними ознаками (Корабель, Вантажівка, Автомобіль) та найбільше на схожі класи (Кішка, Собака).

За критерієм швидкості: дистильована модель забезпечила найбільше прискорення (3.00×), після неї квантизована (2.47×) та прунінгована (1.89×). Профайлінг показав, що прискорення обмежене накладними витратами на операції, що не оптимізуються.

За критерієм компресії: дистиляція досягла компресії 3.96×, квантизація – 3.86×, прунінг – 1.91×. Комбінування дистиляції та квантизації дозволило досягти екстремальної компресії 14.76× (розмір 2.9 МБ) при втраті точності 2.46%.

4. Сформульовано рекомендації щодо вибору методу оптимізації залежно від вимог.

Результати дослідження підтверджують гіпотезу про можливість значної оптимізації глибоких нейронних мереж без критичної втрати якості класифікації. Сучасні методи компресії дозволяють зменшити розмір моделей у 4-15 разів, прискорити інференс у 2-3 рази та зменшити споживання пам'яті втричі при втраті точності менше 2% [13]. Це робить глибоке навчання придатним для широкого спектру застосувань на мобільних пристроях, IoT-девайсах та вбудованих системах, де обчислювальні ресурси обмежені.

Практична реалізація методів оптимізації на базі ResNet-18 та CIFAR-10 продемонструвала їхню ефективність та готовність до застосування у реальних

проектах. Розроблені рекомендації та визначені оптимальні гіперпараметри можуть бути використані при оптимізації моделей комп'ютерного зору.

Наукова новизна роботи полягає у комплексному порівняльному дослідженні трьох методів оптимізації на єдиній базовій архітектурі з детальним аналізом компромісів між точністю, швидкістю та розміром моделі. Вперше для датасету CIFAR-10 проведено систематичне дослідження комбінування методів оптимізації з кількісною оцінкою синергетичних ефектів.

Практична цінність результатів полягає у можливості їх безпосереднього застосування для оптимізації моделей комп'ютерного зору при розгортанні на пристроях з обмеженими ресурсами. Розроблені рекомендації дозволяють обрати оптимальну стратегію оптимізації залежно від конкретних вимог застосування та досягти зменшення розміру моделей у 4-15 разів при збереженні прийнятної точності класифікації.

Перспективними напрямками продовження дослідження є:

1. розробка адаптивних методів, що автоматично вибирають оптимальну стратегію оптимізації залежно від характеристик моделі та цільової платформи;

2. дослідження апаратно-свідомої оптимізації з урахуванням специфіки конкретних процесорів та акселераторів;

3. застосування методів до більш складних архітектур та датасетів;

4. розробка динамічних методів квантизації, що адаптують рівень точності залежно від складності вхідних даних;

5. дослідження застосування методів оптимізації у контексті розподіленого навчання та систем зі збереженням приватності даних.

За результатами дослідження подано тези доповіді «Оптимізація моделей глибинного навчання для задач розпізнавання образів» на VIII Всеукраїнську науково-практичну інтернет-конференцію «Сучасні технології в енергетиці, електромеханіці, системах управління та машинобудуванні» (17 грудня 2025 року, м. Харків).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guan R., Zhao H., Man K. L., Yu L., Yue Y. Deep convolutional neural networks in medical image analysis: applications in computer vision for improved robotics perceptions. Sensors. 2025. Vol. 25, No. 3. DOI: https://doi.org/10.3390/s25030195

2. Shin H. C., Roth H. R., Gao M., et al. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Transactions on Medical Imaging. 2016. Vol. 35, No. 5. P. 1285–1298. DOI: https://doi.org/10.1109/TMI.2016.2528162

3. Khan S., Rahmani H., Shah S. A. A., Bennamoun M. A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision. 2018. Vol. 8, No. 1. P. 1–207. DOI: https://doi.org/10.2200/S00822ED1V01Y201712COV015

4. Wang K., Li Z., Zhou J., et al. A review of convolutional neural networks in computer vision. Artificial Intelligence Review. 2024. Vol. 57, No. 99. DOI: https://doi.org/10.1007/s10462-024-10721-6

5. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks. Communications of the ACM. 2017. Vol. 60, No. 6. P. 84–90. DOI: https://doi.org/10.1145/3065386

6. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 770–778. DOI: https://doi.org/10.1109/CVPR.2016.90

7. Zagoruyko S., Komodakis N. Wide residual networks. Proceedings of the British Machine Vision Conference (BMVC). 2016. DOI: https://doi.org/10.5244/C.30.87

8. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q. Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 4700–4708. DOI: https://doi.org/10.1109/CVPR.2017.243

9. Chen Y., Fan H., Xu B., et al. Drop an octave: reducing spatial redundancy in convolutional neural networks with octave convolution. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). 2019. P. 3435–3444. DOI: https://doi.org/10.1109/ICCV.2019.00353

10. Sze V., Chen Y. H., Yang T. J., Emer J. S. Efficient processing of deep neural networks: a tutorial and survey. Proceedings of the IEEE. 2017. Vol. 105, No. 12. P. 2295–2329. DOI: https://doi.org/10.1109/JPROC.2017.2761740

11. Xie S., Girshick R., Dollár P., et al. Aggregated residual transformations for deep neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 1492–1500. DOI: https://doi.org/10.1109/CVPR.2017.634

12. Tan M., Le Q. V. EfficientNet: rethinking model scaling for convolutional neural networks. Proceedings of the International Conference on Machine Learning (ICML). 2019. P. 6105–6114.

13. Liu D., Guo Y., Weng W., Zhou J., Cheng X. A survey of model compression techniques: past, present, and future. Frontiers in Robotics and AI. 2025. Vol. 12. DOI: https://doi.org/10.3389/frobt.2025.1518965

14. Li Z., Li X., Meng L. Model compression for deep neural networks: a survey. Computers. 2023. Vol. 12, No. 3, Art. 60. DOI: https://doi.org/10.3390/computers12030060

15. Jacob B., Kligys S., Chen B., et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. P. 2704–2713. DOI: https://doi.org/10.1109/CVPR.2018.00286

16. Long X., Wang Z., Zhao Y., et al. A novel low-bit quantization strategy for compressing deep neural networks. Computational Intelligence and Neuroscience. 2020. Vol. 2020. Art. 7839064. DOI: https://doi.org/10.1155/2020/7839064

17. He Y., Zhang X., Sun J. Channel pruning for accelerating very deep neural networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017. P. 1389–1397. DOI: https://doi.org/10.1109/ICCV.2017.155

18. Liu Z., Li J., Shen Z., et al. Learning efficient convolutional networks through network slimming. Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017. P. 2736–2744. DOI: https://doi.org/10.1109/ICCV.2017.298

19. Yu R., Li A., Chen C. F., et al. NISP: pruning networks using neuron importance score propagation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. P. 9194–9203. DOI: https://doi.org/10.1109/CVPR.2018.00958

20. Gou S., Fu J., Sha Y., et al. Dynamic spatio-temporal pruning for efficient spiking neural networks. Frontiers in Neuroscience. 2025. Vol. 19. DOI: https://doi.org/10.3389/fnins.2025.1545583

21. Cheng X., Zhou J., Weng W. Multi-teacher knowledge distillation via student's reflection. Communications in Computer and Information Science. 2025. Vol. 2402. P. 59–73. DOI: https://doi.org/10.1007/978-981-96-2911-4_5

22. Gou J., Yu B., Maybank S. J., Tao D. Knowledge distillation: a survey. International Journal of Computer Vision. 2021. Vol. 129. P. 1789–1819. DOI: https://doi.org/10.1007/s11263-021-01453-z

23. Yuan L., Tay F. E., Li G., et al. Revisiting knowledge distillation via label smoothing regularization. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020. P. 3903–3911. DOI: https://doi.org/10.1109/CVPR42600.2020.00396

24. Sepahvand M., Pourmasoumi A., Ahmadi M., Soleymani F. Teacher-student knowledge distillation based on decomposed deep feature representation for intelligent mobile applications. Expert Systems with Applications. 2022. Vol. 206. Art. 117809. DOI: https://doi.org/10.1016/j.eswa.2022.117809

25. Zhang L., Bao C., Ma K. Teacher-student collaborative knowledge distillation for image classification. Applied Intelligence. 2022. Vol. 52. P. 10957–10970. DOI: https://doi.org/10.1007/s10489-022-03486-4

26. Park S., Heo B., Yun S., Choi J. Data-free knowledge distillation in neural networks for regression. Expert Systems with Applications. 2021. Vol. 180. Art.

115092. DOI: https://doi.org/10.1016/j.eswa.2021.115092

27. Beyer L., Zhai X., Royer A., et al. Knowledge distillation: a good teacher is patient and consistent. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2022. P. 10925–10934. DOI: https://doi.org/10.1109/CVPR52688.2022.01065

28. Deng J., Dong W., Socher R., et al. ImageNet: a large-scale hierarchical image database. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2009. P. 248–255. DOI: https://doi.org/10.1109/CVPR.2009.5206848

29. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998. Vol. 86, No. 11. P. 2278–2324. DOI: https://doi.org/10.1109/5.726791

30. Sandler M., Howard A., Zhu M., et al. MobileNetV2: inverted residuals and linear bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. P. 4510–4520. DOI: https://doi.org/10.1109/CVPR.2018.00474

31. Smith L. N. Cyclical learning rates for training neural networks. Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV). 2017. P. 464–472. DOI: https://doi.org/10.1109/WACV.2017.58

ДОДАТОК А. ФРАГМЕНТ ЛІСТИНГІВ МЕТОДІВ ОПТИМІЗАЦІЇ БАЗОВОЇ МОДЕЛІ

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import time
import os
from copy import deepcopy

# Налаштування

class Config:
    """Конфігурація для навчання та оптимізації"""

    # Загальні налаштування
    batch_size = 128
    num_workers = 2
    device = 'cuda' if torch.cuda.is_available() else 'cpu'

    # Налаштування для baseline моделі
    baseline_epochs = 200
    baseline_lr = 0.1
    baseline_momentum = 0.9
    baseline_weight_decay = 5e-4

    # Налаштування для quantization
    quant_calibration_samples = 1000

    # Налаштування для pruning
    pruning_ratio = 0.5  # 50% прунінг
    pruning_finetune_epochs = 50

    # Налаштування для knowledge distillation
    distill_epochs = 200
    distill_temperature = 4.0
    distill_alpha = 0.7
    distill_beta = 0.3

    # Шляхи для збереження
    save_dir = './models'
    results_dir = './results'

# Підготовка даних

def get_cifar10_loaders(batch_size=128, num_workers=2):
    """
    Завантажує та підготовлює CIFAR-10 датасет

    Returns:
        train_loader: DataLoader для тренування
        test_loader: DataLoader для тестування
        calibration_loader: DataLoader для калібрування (підмножина train)
    """

    # Нормалізація з середніми та стандартними відхиленнями CIFAR-10
    normalize = transforms.Normalize(
        mean=[0.4914, 0.4822, 0.4465],
        std=[0.2023, 0.1994, 0.2010]
    )
```

```
    # Аугментація для тренування
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ])

    # Без аугментації для тестування
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        normalize,
    ])

    # Завантаження датасетів
    trainset = torchvision.datasets.CIFAR10(
        root='./data', train=True, download=True, transform=transform_train
    )

    testset = torchvision.datasets.CIFAR10(
        root='./data', train=False, download=True, transform=transform_test
    )

    # DataLoader для тренування та тестування
    train_loader = DataLoader(
        trainset, batch_size=batch_size, shuffle=True, num_workers=num_workers
    )

    test_loader = DataLoader(
        testset, batch_size=batch_size, shuffle=False, num_workers=num_workers
    )

    # Калібрувальний loader (для quantization) - підмножина train
    calibration_set = torch.utils.data.Subset(
        trainset, indices=range(Config.quant_calibration_samples)
    )
    calibration_loader = DataLoader(
        calibration_set, batch_size=batch_size, shuffle=False, num_workers=num_workers
    )

    return train_loader, test_loader, calibration_loader


# Модель ResNet-18

class BasicBlock(nn.Module):
    """Базовий залишковий блок для ResNet-18"""
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False
        )
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(
            planes, planes, kernel_size=3, stride=1, padding=1, bias=False
        )
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(
                    in_planes, self.expansion * planes,
                    kernel_size=1, stride=stride, bias=False
```

```
                ),
                nn.BatchNorm2d(self.expansion * planes)
            )

    def forward(self, x):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = torch.relu(out)
        return out

class ResNet(nn.Module):
    """ResNet для CIFAR-10"""

    def __init__(self, block, num_blocks, num_classes=10, width_multiplier=1.0):
        super(ResNet, self).__init__()
        self.in_planes = int(64 * width_multiplier)

        self.conv1 = nn.Conv2d(3, int(64 * width_multiplier), kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(int(64 * width_multiplier))

        self.layer1 = self._make_layer(block, int(64 * width_multiplier),
num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, int(128 * width_multiplier),
num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, int(256 * width_multiplier),
num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, int(512 * width_multiplier),
num_blocks[3], stride=2)
        self.linear = nn.Linear(int(512 * width_multiplier) * block.expansion,
num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = torch.nn.functional.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out

def ResNet18(num_classes=10):
    """Створює ResNet-18"""
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes)

def ResNet34(num_classes=10):
    """Створює ResNet-34 (для teacher в distillation)"""
    return ResNet(BasicBlock, [3, 4, 6, 3], num_classes=num_classes)

def SmallResNet18(num_classes=10):
    """Створює зменшену версію ResNet-18 (для student в distillation)"""
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes,
width_multiplier=0.5)
```

```python
# Навчання та тестування

def train_epoch(model, loader, criterion, optimizer, device):
    """Навчання моделі протягом однієї епохи"""
    model.train()
    train_loss = 0
    correct = 0
    total = 0

    for batch_idx, (inputs, targets) in enumerate(loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    accuracy = 100. * correct / total
    avg_loss = train_loss / len(loader)

    return avg_loss, accuracy

def test(model, loader, criterion, device):
    """Тестування моделі"""
    model.eval()
    test_loss = 0
    correct = 0
    total = 0

    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(loader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    accuracy = 100. * correct / total
    avg_loss = test_loss / len(loader)

    return avg_loss, accuracy

def train_baseline(model, train_loader, test_loader, config):
    """
    Навчання baseline моделі ResNet-18

    Args:
        model: Модель для навчання
        train_loader: DataLoader для тренування
        test_loader: DataLoader для тестування
        config: Конфігурація

    Returns:
        model: Навчена модель
        best_acc: Найкраща точність на тесті
    """
```

```
    print("=" * 80)
    print("НАВЧАННЯ BASELINE RESNET-18")
    print("=" * 80)

    device = config.device
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(
        model.parameters(),
        lr=config.baseline_lr,
        momentum=config.baseline_momentum,
        weight_decay=config.baseline_weight_decay
    )

    # Cosine annealing scheduler
    scheduler = optim.lr_scheduler.CosineAnnealingLR(
        optimizer, T_max=config.baseline_epochs
    )

    best_acc = 0

    for epoch in range(config.baseline_epochs):
        train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer,
device)
        test_loss, test_acc = test(model, test_loader, criterion, device)
        scheduler.step()

        # Збереігання найкращої моделі
        if test_acc > best_acc:
            best_acc = test_acc
            torch.save(model.state_dict(), os.path.join(config.save_dir,
'resnet18_baseline_best.pth'))

        if (epoch + 1) % 10 == 0:
            print(f'Epoch {epoch+1}/{config.baseline_epochs}:')
            print(f'  Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%')
            print(f'  Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%')
            print(f'  Best Acc: {best_acc:.2f}%')

    print(f'\nНавчання завершено. Найкраща точність: {best_acc:.2f}%')

    # Завантаження найкращої моделф
    model.load_state_dict(torch.load(os.path.join(config.save_dir,
'resnet18_baseline_best.pth')))

    return model, best_acc


# Метод 1: Post-Training Qunatization

def prepare_model_for_quantization(model):
    """Підготовка моделі для quantization"""
    # QuantStub та DeQuantStub
    model.quant = torch.quantization.QuantStub()
    model.dequant = torch.quantization.DeQuantStub()

    # Збереження original forward
    original_forward = model.forward

    # Модифікування forward для включення quant/dequant
    def new_forward(self, x):
        x = self.quant(x)
        x = original_forward(x)
        x = self.dequant(x)
```

```
        return x

    # Прив'язання нового forward до моделі
    import types
    model.forward = types.MethodType(new_forward, model)

    return model

def fuse_modules(model):
    """Об'єднання модулів Conv-BN-ReLU для quantization"""
    # Об'єднання Conv-BN та Conv-BN-ReLU для ResNet
    for name, module in model.named_children():
        if name == 'conv1':
            torch.quantization.fuse_modules(model, ['conv1', 'bn1'], inplace=True)
        elif name.startswith('layer'):
            for block_name, block in module.named_children():
                torch.quantization.fuse_modules(
                    block, [['conv1', 'bn1'], ['conv2', 'bn2']], inplace=True
                )

    return model

def quantize_model(model, calibration_loader, device):
    """
    Post-Training Quantization (PTQ)
    """

    print("=" * 80)
    print("QUANTIZATION (POST-TRAINING)")
    print("=" * 80)

    # Створення копії моделі
    model_to_quantize = deepcopy(model)
    model_to_quantize.eval()
    model_to_quantize.cpu()  # Quantization працює на CPU

    # Підготовка моделі
    model_to_quantize = prepare_model_for_quantization(model_to_quantize)
    model_to_quantize = fuse_modules(model_to_quantize)

    # Налаштування quantization
    model_to_quantize.qconfig = torch.quantization.get_default_qconfig('fbgemm')

    # Підготовка для quantization
    torch.quantization.prepare(model_to_quantize, inplace=True)

    # Калібрування
    print("Калібрування моделі...")
    model_to_quantize.eval()
    with torch.no_grad():
        for inputs, _ in calibration_loader:
            model_to_quantize(inputs)

    # Конвертація у квантизовану модель
    print("Конвертація у INT8...")
    torch.quantization.convert(model_to_quantize, inplace=True)

    print("Quantization завершено.")

    return model_to_quantize
```

ДОДАТОК Б. КОПІЇ ДОКУМЕНТІВ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ РОБОТИ

ОПТИМІЗАЦІЯ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ ДЛЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБРАЗІВ

Чикунов П.О., к.т.н., доц.
Каракян А.В., здобувач вищої освіти другого рівня
Національний університет «Одеська юридична академія»

Системи розпізнавання образів на основі глибинного навчання вже знайшли широке застосування у різноманітних галузях, від примітивних розпізнавальних алгоритмів до медичної діагностики. Однак сучасні моделі глибинного навчання усе ще мають значні обмеження: великий розмір моделей, високі обчислювальні витрати, значне енергоспоживання та вартість експлуатації. Ці фактори ускладнюють практичне застосування технологій комп'ютерного зору у умовах з обмеженими ресурсами. Тому оптимізація моделей є критично важливою задачею для забезпечення можливості розгортання на мобільних пристроях, IoT-системах та у кордонних обчисленнях.

Метою роботи є дослідження та порівняльний аналіз методів оптимізації моделей глибинного навчання для задач розпізнавання образів з мінімальними втратами точності при значному зменшенні розміру моделі та прискоренні процесу отримання висновків.

Завдання дослідження включали:

– аналіз сучасних архітектур згорткових нейронних мереж;

– дослідження теоретичних основ методів квантизації, структурного обрізання та дистиляції знань;

– реалізацію базової моделі ResNet-18 для класифікації на датасеті CIFAR-10;

– застосування методів оптимізації;

– проведення порівняльного аналізу за критеріями точності, розміру моделі, швидкості отримання висновків та використання пам'яті.

Для вирішення поставленої мети було обрано датасет CIFAR-10, що складається з 60 000 кольорових зображень розміром 32x32 пікселі у 10 класах. Базова модель ResNet-18 була реалізована з використанням фреймворку PyTorch. Навчання проводилось на Nvidia Tesla T4. Базова модель досягла точності 95.66% на тестовій вибірці, розмір моделі становив 42,8 МБ, час отримання висновків складав близько 15.3 мс на одне зображення.

Було реалізовано та досліджено три методи оптимізації. Перший метод – квантизація (Post-Training Quantization) – передбачав зменшення розрядності числового представлення з 32-бітних чисел з плаваючою комою до 8-бітних цілих чисел. Використано статичну квантизацію з калібруванням на 1000 зображеннях. Квантизована модель показала точність 94,80% (зниження на 0,8%), час inference 6.2 мс (прискорення у 2,47 рази), розмір моделі 11,1 МБ (зменшення у 3,86 рази).

Другий метод – структурований прунінг на рівні фільтрів - базувався на видаленні цілих згорткових фільтрів зі шарів мережі за критерієм L1-норми ваг. Застосовано глобальний коефіцієнт прунінгу 50% з подальшим fine-tuning протягом 50 епох. Обрізана модель досягла точності 94,20% (зниження на

1,46%), час inference 8.1 мс (прискорення у 1,89 рази), розмір моделі 22,4 МБ, кількість параметрів зменшилась на 48% (з 11.17M до 5.81M).

Третій метод – дистиляція знань (Knowledge Distillation) - передбачав передачу знань від моделі-вчителя ResNet-34 (96.5% точності) до зменшеної моделі-учня з половинною кількістю фільтрів. Використано temperature-scaled softmax з T=4, комбіновану функцію втрат з вагами 0,7 для дистиляції та 0,3 для крос-ентропії. Модель-учень показала точність 95% (зниження на 0,66%), час inference 5.1 мс (прискорення у 3 рази), розмір моделі 10,8 МБ (зменшення у 3,96 рази), що на 2-3% краще за учня без дистиляції.

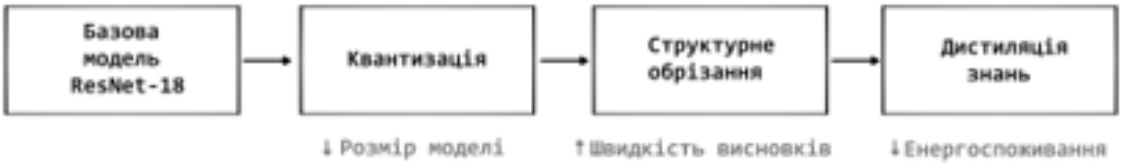

Рисунок 1 – Методи оптимізаціїмоделі ResNet-18

Порівняльний аналіз показав, що кожен метод має специфічні переваги: квантизація забезпечує найкраще співвідношення швидкості та розміру при мінімальній втраті точності, ідеальна для швидкого розгортання; прунінг дозволяє значно зменшити обчислювальну складність, підходить для систем з обмеженою пам’яттю; дистиляція знань показала найкращий баланс між розміром та точністю, рекомендована для критичних застосувань де точність є пріоритетом.

Практична реалізація методів оптимізації на базі моделі ResNet-18 та датасеті CIFAR-10 продемонструвала їхню ефективність та готовність до застосування у реальних проектах. Розроблені рекомендації та визначені оптимальні гіперпараметри можуть бути використані при оптимізації моделей комп'ютерного зору.

Практична цінність результатів полягає у можливості їх безпосереднього застосування для оптимізації моделей комп'ютерного зору при розгортанні на пристроях з обмеженими ресурсами. Розроблені рекомендації дозволяють обрати оптимальну стратегію оптимізації залежно від конкретних вимог застосування та досягти зменшення розміру моделей у 4-15 разів при збереженні прийнятної точності класифікації.

Список використаних джерел

1. Guan R., Zhao H., Man K. L., Yu L., Yue Y. Deep convolutional neural networks in medical image analysis: applications in computer vision for improved robotics perceptions. Sensors. 2025. Vol. 25, No. 3. DOI: https://doi.org/10.3390/s25030195.

2. Wang K., Li Z., Zhou J., et al. A review of convolutional neural networks in computer vision. Artificial Intelligence Review. 2024. Vol. 57, No. 99. DOI: https://doi.org/10.1007/s10462-024-10721-6.

3. Liu D., Guo Y., Weng W., Zhou J., Cheng X. A survey of model compression techniques: past, present, and future. Frontiers in Robotics and AI. 2025. Vol. 12. DOI: https://doi.org/10.3389/frobt.2025.1518965.