Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування і економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«Пошук оптимальної архітектури згорткових нейронних мереж за допомогою алгоритмів глобальної оптимізації»

«Optimal architecture search of convolutional neural networks using global optimization algorithms»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Чачко Натан Леонідович

Керівник: канд. фіз.-мат. наук, доц. Страхов Є. М. _____
Рецензент: канд. фіз.-мат. наук, доц. Вербіцький В. В.

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від _____ 2025 р.
Завідувач кафедри
_____ _____

Захищено на засіданні ЕК № _____
Протокол № ___ від _____ 2025 р.
Оцінка _____ / _____ / _____
Голова ЕК
_____ _____

Одеса — 2025 р.

ЗМІСТ

ВСТУП 4

1 ТЕОРЕТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ТА ГЕНЕТИЧНИХ АЛГОРИТМІВ 6

1.1 Штучні нейронні мережі . . . 6
1.1.1 Історичний розвиток . . . 6
1.1.2 Структура штучного нейрона . . . 7
1.1.3 Пряме та зворотне поширення . . . 7
1.2 Архітектура згорткових нейронних мереж . . . 7
1.2.1 Функції активації . . . 8
1.3 Навчання згорткових нейронних мереж . . . 10
1.3.1 Функції втрат . . . 10
1.3.2 Методи оптимізації . . . 11
1.3.3 Батч-навчання та епохи . . . 12
1.3.4 Регуляризація . . . 13
1.3.5 Ініціалізація ваг . . . 14
1.3.6 Валідація та запобігання перенавчанню . . . 14
1.3.7 Класичні архітектури CNN . . . 15
1.4 Методи пошуку оптимальної архітектури CNN . . . 18
1.4.1 Ручне проектування архітектур . . . 18
1.4.2 Підходи до автоматизованого пошуку . . . 18

2 РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ НА JAVA 20

2.1 Загальна структура системи . . . 20
2.1.1 Базовий клас Layer . . . 21
2.2 Реалізація згорткового шару . . . 21
2.2.1 Структура та параметри . . . 22
2.2.2 Прямий прохід згорткового шару . . . 22
2.2.3 Зворотне поширення через згортковий шар . . . 23
2.3 Реалізація повнозв'язного шару . . . 24
2.3.1 Структура та параметри . . . 25
2.3.2 Прямий прохід . . . 25
2.3.3 Зворотне поширення . . . 25
2.3.4 Рекурсивна архітектура . . . 26
2.4 Реалізація шару Max Pooling . . . 27
2.4.1 Структура та параметри . . . 27
2.4.2 Прямий прохід . . . 28
2.4.3 Зворотне поширення . . . 28
2.4.4 Особливості реалізації . . . 29

2.5 Алгоритм навчання мережі 29
2.5.1 Принципи роботи алгоритму 29
2.5.2 Рекурсивне поширення сигналів помилки 29
2.5.3 Особливості обробки батчів 30
2.5.4 Регуляризація та ініціалізація ваг 30

3 ГЕНЕТИЧНИЙ АЛГОРИТМ ДЛЯ ПОШУКУ ОПТИМАЛЬНОЇ АРХІТЕКТУРИ CNN 31
3.1 Еволюційний підхід 31
3.1.1 Генетичні алгоритми 31
3.2 Структура кодування архітектури 32
3.2.1 Представлення хромосоми 32
3.2.2 Типи шарів та гени 32
3.2.3 Обмеження на архітектуру 33
3.3 Популяція та ініціалізація 33
3.3.1 Ініціалізація популяції: 33
3.3.2 Фітнес-функція 34
3.3.3 Механізм кешування 34
3.4 Селекція 35
3.4.1 Елітизм 35
3.4.2 Селекція для кросовера 35
3.5 Кросовер 35
3.5.1 Алгоритм одноточкового кросовера 35
3.6 Мутація 36
3.6.1 Типи мутацій 36
3.7 Формування нового покоління 37
3.8 Загальна схема алгоритму 38
3.9 Переваги та обмеження реалізації 38
3.9.1 Обґрунтування для датасету MNIST 39
3.10 Висновки 39

4 Експериментальна перевірка та аналіз результатів 40
4.1 Умови експерименту 40
4.2 Результати оптимізації 41
4.3 Аналіз результатів 43

Висновки 48

Список літератури 49

ДОДАТОК А 51

ВСТУП

Згорткові нейронні мережі (CNN) є основою сучасних систем комп'ютерного зору та посідають провідне місце серед архітектур глибокого навчання. За останнє десятиліття CNN продемонстрували видатні результати у широкому спектрі завдань: від класифікації та розпізнавання зображень до сегментації об'єктів, виявлення облич та медичної діагностики. Проте ефективність роботи нейронної мережі критично залежить від правильно обраної архітектури — кількості та типів шарів, їхніх параметрів, функцій активації та способів з'єднання.

Традиційно пошук оптимальної архітектури CNN здійснюється вручну експертами у галузі машинного навчання, що вимагає значних часових ресурсів, глибоких теоретичних знань та практичного досвіду. Цей процес часто базується на методі спроб і помилок, інтуїції дослідника та адаптації існуючих успішних архітектур. Така практика не лише є ресурсозатратною, але й не гарантує знаходження глобального оптимуму в просторі можливих архітектур.

Використання алгоритмів глобальної оптимізації, зокрема генетичних алгоритмів, дає змогу автоматизувати цей процес і систематично досліджувати простір архітектур. Генетичні алгоритми, що моделюють процеси природного відбору та еволюції, показують високу ефективність у розв'язанні складних оптимізаційних задач з великою кількістю локальних екстремумів. Застосування еволюційного підходу до пошуку архітектури нейронних мереж (Neural Architecture Search, NAS) є активною областю досліджень і демонструє перспективні результати.

Реалізація повноцінної системи пошуку архітектури CNN з нуля, без використання сторонніх бібліотек машинного навчання, демонструє глибоке розуміння внутрішньої роботи згорткових нейронних мереж, алгоритмів навчання та еволюційних механізмів оптимізації. Така імплементація дозволяє повністю контролювати процес оптимізації, адаптувати алгоритм під специфічні вимоги та отримати цінний досвід у розробці систем машинного навчання.

Метою даної роботи є розроблення та дослідження системи автоматичного пошуку оптимальної архітектури згорткових нейронних мереж за допомогою генетичного алгоритму з подальшою експериментальною перевіркою на класичному датасеті MNIST [1].

Для досягнення поставленої мети необхідно розв'язати наступні завдання:

1) Проаналізувати існуючі підходи до побудови архітектур CNN та методи автоматизованого пошуку архітектури (NAS).

2) Розробити та реалізувати повнофункціональну згорткову нейронну мережу на мові програмування Java.
3) Реалізувати генетичний алгоритм для еволюції та оптимізації архітектури CNN, включаючи механізми кодування архітектури, селекції, кросовера та мутації.
4) Провести навчання та тестування отриманих моделей на датасеті MNIST для задачі розпізнавання рукописних цифр.
5) Виконати експериментальне дослідження, проаналізувати результати оптимізації та оцінити ефективність розробленого підходу.

Об'єкт, предмет і методи дослідження

Об'єкт дослідження: архітектура згорткових нейронних мереж для задач класифікації зображень.

Предмет дослідження: застосування генетичного алгоритма для оптимізації архітектур CNN у просторі можливих конфігурацій мережі.

Методи дослідження: теоретичний аналіз наукової літератури з питань архітектури нейронних мереж та еволюційних обчислень, математичне моделювання процесів навчання та еволюції мереж, об'єктно-орієнтоване програмування, експериментальні дослідження з використанням реальних наборів даних, статистичний аналіз результатів.

Практична значущість

Практична значущість роботи полягає у створенні функціонального програмного забезпечення для автоматизованого пошуку архітектур згорткових нейронних мереж, яке може бути застосоване для розв'язання широкого кола завдань комп'ютерного зору. Розроблений алгоритм дозволяє скоротити час, необхідний для створення ефективної архітектури CNN, та зменшити залежність від експертних знань у галузі глибокого навчання.

Отримане програмне забезпечення, реалізоване на мові Java без використання сторонніх бібліотек, може служити освітньою базою для вивчення внутрішніх механізмів роботи нейронних мереж та еволюційних алгоритмів. Крім того, система може бути основою для подальших досліджень у сферах Neural Architecture Search, еволюційних обчислень та автоматизованого машинного навчання (AutoML). Результати роботи мають потенціал застосування в академічних дослідженнях, промислових системах розпізнавання образів та освітніх проектах з машинного навчання.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ТА ГЕНЕТИЧНИХ АЛГОРИТМІВ

1.1 Штучні нейронні мережі

Штучні нейронні мережі є обчислювальними моделями, натхненними біологічною структурою та функціонуванням нервової системи живих організмів. Історія розвитку нейронних мереж налічує понад сім десятиліть активних досліджень, які супроводжувалися періодами як значного прогресу, так і стагнації.

1.1.1 Історичний розвиток

Початок сучасної епохи штучних нейронних мереж пов'язаний з роботою Френка Розенблатта, який у 1958 році представив перцептрон — першу обчислювальну модель, що імітувала процес навчання біологічного нейрона [2]. Перцептрон був здатний класифікувати лінійно відокремлювані об'єкти, однак його обмеження були виявлені Марвіном Мінськи та Сеймуром Папертом у 1969 році [3], що призвело до першої «зими штучного інтелекту».

Справжній прорив відбувся у 1970-х роках, коли Сеппо Ліннайнмаа у 1970 році розробив математичні основи автоматичного диференціювання в зворотному режимі, а Пол Вербос у 1974 році запропонував застосувати цей метод до навчання нейронних мереж [4]. Проте лише у 1986 році Девід Румельхарт, Джеффрі Хінтон та Рональд Вільямс опублікували знакову роботу [5, 6], яка продемонструвала, що алгоритм зворотного поширення помилки може ефективно навчати багатошарові мережі для розв'язання практичних задач.

Наступний важливий етап настав у кінці 1980-х — на початку 1990-х років з появою згорткових нейронних мереж. Куніхіко Фукушіма представив неокогнітрон у 1980 році [7], а операція max pooling з'явилася у публікації 1982 року. У 1989 році Ян Лекун зі співавторами навчив CNN для розпізнавання рукописних поштових індексів [8], хоча процес навчання тривав три дні. Це призвело до створення архітектури LeNet-5 у 1998 році [13], яка успішно використовувалася банками для розпізнавання цифр на чеках.

Епоха глибокого навчання почалася у 2010-х роках, коли використання графічних процесорів (GPU) дозволило тренувати значно глибші мережі. Створення великих датасетів, таких як ImageNet [9], та розробка нових архітектур CNN [10] призвели до революції у комп'ютерному зорі.

1.1.2 Структура штучного нейрона

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

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(w^T x + b),$$

де x_i — вхідні сигнали, w_i — вагові коефіцієнти, b — зміщення (bias), f — функція активації, y — вихідний сигнал нейрона.

1.1.3 Пряме та зворотне поширення

Основою навчання нейронної мережі є процеси прямого (forward) та зворотного (backward) поширення сигналу. Прямий прохід обчислює вихід мережі для заданого входу, а зворотне поширення використовує градієнти функції втрат для оновлення параметрів.

Для покращення стабільності градієнти усереднюються за батчем навчаючих зразків, а оновлення ваг здійснюється за допомогою алгоритмів градієнтного спуску з заданою швидкістю навчання.

Детальний опис алгоритмів прямого і зворотного поширення, а також їх реалізація у вигляді програмних класів представлено у другій главі.

1.2 Архітектура згорткових нейронних мереж

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є спеціалізованим типом глибоких нейронних мереж, розроблених для ефективної обробки даних з просторовою структурою, зокрема зображень. На відміну від повнозв'язних мереж, CNN використовують локальні зв'язки та спільні ваги, що значно зменшує кількість параметрів та забезпечує інваріантність до зсувів.

CNN складається з послідовності основних типів шарів: згорткових (Convolutional), підвибірки (Pooling) та повнозв'язних (Fully Connected)

шарів. Кожен із цих шарів виконує специфічні функції для обробки та трансформації вхідних даних з метою добування корисних ознак та ефективної класифікації.

Детальний опис структури, механізмів роботи та алгоритмів реалізації кожного типу шару наведено у другій главі цієї роботи.

1.2.1 Функції активації

Функції активації вносять нелінійність у модель, дозволяючи мережі апроксимувати складні залежності.

Sigmoid — це S-подібна функція, яка перетворює вхідні значення до діапазону $(0; 1)$:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Похідна sigmoid для backpropagation:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

Вона часто використовується для задач бінарної класифікації, однак має проблему зникаючих градієнтів у глибоких мережах.

ReLU (Rectified Linear Unit) є найпоширенішою функцією активації у сучасних CNN:

$$\mathrm{ReLU}(x) = \max(0, x).$$

Похідна ReLU:

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & \text{якщо } x > 0 \\ 0, & \text{якщо } x \leq 0 \end{cases}.$$

ReLU прискорює навчання та зменшує проблему зникаючих градієнтів.

Leaky ReLU усуває проблему «мертвих» нейронів, дозволяючи невеликий градієнт для від'ємних значень:

$$\mathrm{LeakyReLU}(x) = \begin{cases} x, & \text{якщо } x > 0 \\ \alpha x, & \text{якщо } x \leq 0 \end{cases},$$

де α — малий коефіцієнт (зазвичай 0.01).

Похідна Leaky ReLU:

$$\text{LeakyReLU}'(x) = \begin{cases} 1, & \text{якщо } x > 0 \\ \alpha, & \text{якщо } x \leq 0 \end{cases}.$$

Softmax використовується на вихідному шарі для багатокласової класифікації, перетворюючи логіти у розподіл ймовірностей:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

де K — кількість класів.

Для числової стабільності при реалізації рекомендується віднімати максимальне значення:

$$\text{Softmax}(z_i) = \frac{e^{z_i - \max(z)}}{\sum_{j=1}^{K} e^{z_j - \max(z)}}.$$

Tanh (гіперболічний тангенс) — це сигмоїдна функція з діапазоном від -1 до 1:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Похідна tanh:

$$\tanh'(x) = 1 - \tanh^2(x).$$

ELU (Exponential Linear Unit) — функція, що поєднує властивості ReLU для додатніх входів і експоненційну форму для від'ємних:

$$\text{ELU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases},$$

де α — гіперпараметр (зазвичай 1.0).

Похідна ELU:

$$\text{ELU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x \leq 0 \end{cases}.$$

Використання цих різних функцій активації дозволяє покращити навчання мережі, підвищити її здатність до узагальнення та подолати проблеми, пов'язані з традиційними функціями.

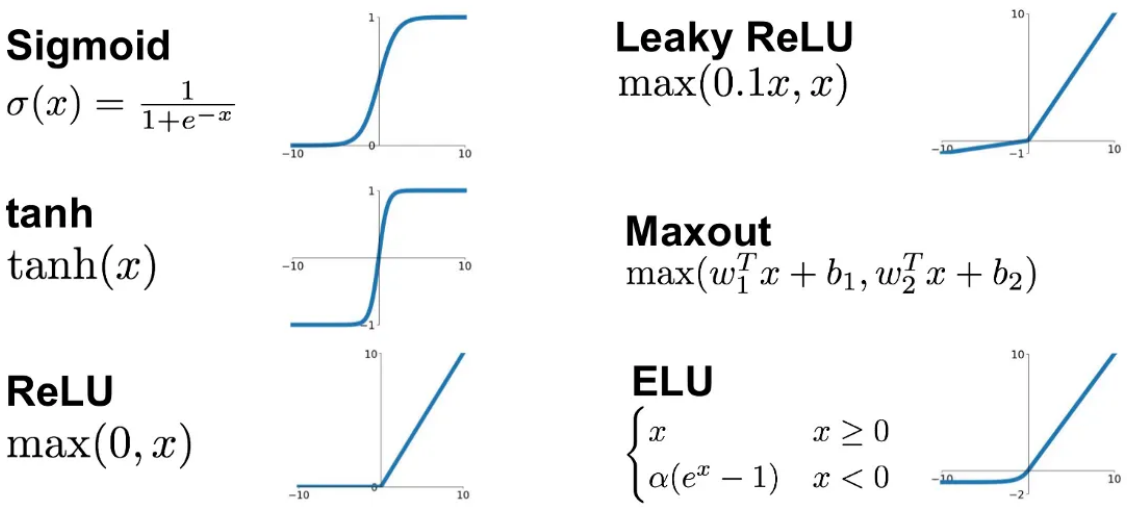


Рис. 1.1. Функції активації

В цій роботі було вирішено здійснювати підбір функцій активації з трьох найпоширеніших: Sigmoid, ReLU та Leaky ReLU.

1.3 Навчання згорткових нейронних мереж

Навчання згорткових нейронних мереж є процесом ітеративної оптимізації параметрів моделі з метою мінімізації помилки на навчальних даних. Цей процес включає обчислення градієнтів функції втрат відносно ваг мережі та їх оновлення за допомогою методів градієнтної оптимізації.

1.3.1 Функції втрат

Функція втрат (loss function) кількісно оцінює різницю між передбаченнями моделі та справжніми мітками. Вибір функції втрат залежить від типу задачі.

Cross-Entropy Loss є стандартною функцією втрат для задач класифікації. Для багатокласової класифікації використовується категоріальна крос-енентропія:

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{i,j} \log(\hat{y}_{i,j}),$$

де N — розмір батчу, K — кількість класів, $y_{i,j}$ — справжня мітка (one-hot encoded), $\hat{y}_{i,j}$ — передбачена ймовірність класу j для зразка i.

Для випадку, коли мітки представлені індексами класів (а не one-hot

векторами), формула спрощується:

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\log(\hat{y}_{i,c_i}),$$

де c_i — індекс правильного класу для зразка i. Для датасету MNIST з 10 класами цифр ця функція втрат є найбільш придатною.

Mean Squared Error (MSE) використовується для задач регресії:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

1.3.2 Методи оптимізації

Методи оптимізації визначають, як саме використовувати обчислені градієнти для оновлення параметрів мережі.

Стохастичний градієнтний спуск (SGD)

Найпростіший метод оптимізації, що оновлює параметри у напрямку, протилежному градієнту:

$$\theta_{t+1} = \theta_t - \eta\nabla_\theta\mathcal{L}(\theta_t),$$

де θ — параметри моделі (ваги та зміщення), η — швидкість навчання (learning rate), $\nabla_\theta\mathcal{L}$ — градієнт функції втрат.

SGD з імпульсом (momentum) покращує збіжність, додаючи частку попереднього оновлення:

$$v_{t+1} = \gamma v_t + \eta\nabla_\theta\mathcal{L}(\theta_t),$$
$$\theta_{t+1} = \theta_t - v_{t+1},$$

де γ — коефіцієнт імпульсу (зазвичай 0.9), v_t — вектор швидкості, $v_0 = 0$.

Adaptive Learning Rate методи

AdaGrad адаптує швидкість навчання для кожного параметра:

$$G_t = G_{t-1} + (\nabla_\theta\mathcal{L}(\theta_t))^2,$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta_t),$$

де G_t — сума квадратів градієнтів до моменту t, ϵ — мала константа для числової стабільності (зазвичай 10^{-8}), $G_0 = 0$.

RMSprop використовує експоненційно зважене ковзне середнє квадратів градієнтів:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)(\nabla_\theta \mathcal{L}(\theta_t))^2,$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta_t),$$

де β — коефіцієнт згладжування (зазвичай 0.9), $E[g^2]_0 = 0$.

Adam (Adaptive Moment Estimation) комбінує переваги momentum та RMSprop:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta \mathcal{L}(\theta_t),$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2,$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t},$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t,$$

де $\beta_1 = 0.9$, $\beta_2 = 0.999$ — типові значення, $m_0 = 0$, $v_0 = 0$. Корекція зміщення ($\hat{m}_t$, $\hat{v}_t$) є критичною на перших кроках навчання. Adam є одним з найпопулярніших оптимізаторів для навчання CNN завдяки своїй робастності та ефективності.

1.3.3 Батч-навчання та епохи

Батч (batch) — це підмножина навчальних даних, що використовується для одного оновлення параметрів. Розмір батчу впливає на швидкість збіжності та стабільність навчання:

- Малі батчі (16-32): більше шуму в градієнтах, краща генералізація
- Великі батчі (128-256): стабільніші градієнти, швидше навчання, але гірша генералізація

Епоха — один повний прохід через весь навчальний датасет. Типово CNN навчаються десятки або сотні епох.

Mini-batch gradient descent є компромісом між повним батчем (весь датасет) та стохастичним градієнтним спуском (один зразок):

$$\theta_{t+1} = \theta_t - \eta \frac{1}{|B|} \sum_{i \in B} \nabla_\theta \mathcal{L}(\theta_t; x_i, y_i),$$

де B — батч розміром $|B|$.

В цій роботі реалізовано mini-batch gradient descent: всі експерименти проводилися з розміром батчу $|B| = 16$, що забезпечило компроміс між швидкістю збіжності та здатністю моделі до узагальнення на нових даних.

1.3.4 Регуляризація

Регуляризація є ключовим методом запобігання перенавчанню (overfitting) — ситуації, коли модель добре працює на навчальних даних, але погано генералізує на нові дані.

L2-регуляризація (Weight Decay)

Додає штраф за великі значення ваг до функції втрат:

$$\mathcal{L}_{total} = \mathcal{L}_{data} + \lambda \sum_l ||W^{(l)}||_2^2 = \mathcal{L}_{data} + \lambda \sum_l \sum_{i,j} (W_{i,j}^{(l)})^2,$$

де λ — коефіцієнт регуляризації (зазвичай 10^{-4} - 10^{-3}). Це призводить до модифікованого оновлення ваг:

$$W^{(l)} \leftarrow W^{(l)} - \eta \left(\frac{\partial \mathcal{L}_{data}}{\partial W^{(l)}} + 2\lambda W^{(l)} \right) = W^{(l)}(1 - 2\eta\lambda) - \eta \frac{\partial \mathcal{L}_{data}}{\partial W^{(l)}}.$$

Саме L2-регуляризація використовується в рамках цієї роботи.

L1-регуляризація

Використовує абсолютні значення ваг, що призводить до розрідженості:

$$\mathcal{L}_{total} = \mathcal{L}_{data} + \lambda \sum_l ||W^{(l)}||_1 = \mathcal{L}_{data} + \lambda \sum_l \sum_{i,j} |W_{i,j}^{(l)}|.$$

Dropout

Dropout випадково «вимикає» частину нейронів під час навчання з ймовірністю p (зазвичай 0.5 для повнозв'язних шарів, 0.1-0.3 для згорткових). У даній роботі використовується inverted dropout:

$$\tilde{a}^{(l)} = \frac{1}{1-p} \cdot (a^{(l)} \odot m), \quad m \sim \text{Bernoulli}(1-p),$$

де m — бінарна маска, що визначає, які нейрони залишаються активними.

Масштабування на $\frac{1}{1-p}$ під час навчання гарантує, що під час тестування (коли dropout не застосовується) очікувані значення активацій залишаються незмінними. Dropout діє як ансамбль багатьох підмереж і значно покращує генералізацію.

1.3.5 Ініціалізація ваг

Правильна ініціалізація ваг критична для успішного навчання глибоких мереж. Погана ініціалізація може призвести до зникнення або вибуху градієнтів.

Xavier/Glorot ініціалізація [19] для шарів з tanh або sigmoid:

$$W_{ij} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right),$$

де $\mathcal{U}(a,b)$ — рівномірний розподіл на інтервалі $[a,b]$.

He ініціалізація [18] для шарів з ReLU та Leaky ReLU активацією:

$$W_{ij} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right),$$

де $\mathcal{N}(\mu,\sigma)$ — нормальний розподіл з середнім μ та стандартним відхиленням σ, n_{in} — кількість входів до шару, n_{out} — кількість виходів.

Зміщення b зазвичай ініціалізуються нулями: $b = 0$.

1.3.6 Валідація та запобігання перенавчанню

Для коректної оцінки продуктивності моделей датасет MNIST розділяється на три незалежні множини:

- **Навчальна вибірка (training set)** — використовується для оновлення ваг мережі під час тренування кожної архітектури (50,000

зображень)
- **Валідаційна вибірка (validation set)** — використовується для обчислення фітнес-функції генетичного алгоритму та моніторингу процесу еволюції (10,000 зображень)
- **Тестова вибірка (test set)** — використовується виключно для фінальної незалежної оцінки найкращої знайденої архітектури (10,000 зображень)

Такий розподіл забезпечує коректність експериментальної оцінки: генетичний алгоритм оптимізує архітектури на основі валідаційної точності, не маючи доступу до тестової вибірки, що запобігає витоку інформації (data leakage) та гарантує об'єктивність фінальних результатів.

Early stopping застосовується під час навчання кожної архітектури: процес зупиняється, якщо точність на валідаційній вибірці не покращується протягом заданої кількості епох (patience parameter), що запобігає перенавчанню та економить обчислювальні ресурси.

На етапі розробки та налагодження системи для прискорення експериментів використовується зменшена вибірка, що становить 1% від повного датасету (600 зображень), розподілена у тих самих пропорціях. Фінальні результати та висновки базуються на експериментах з повним датасетом MNIST.

1.3.7 Класичні архітектури CNN

LeNet-5 (1998) [13]— піонерська архітектура, розроблена Яном Лекуном для розпізнавання рукописних цифр. Складається з семи шарів: два згорткових шари з subsampling, за якими слідують три повнозв'язних шари. Вхідне зображення має розмір 32×32 пікселі (градації сірого). LeNet-5 продемонструвала ефективність CNN для задач розпізнавання образів, але була обмежена малою глибиною та обчислювальними ресурсами того часу

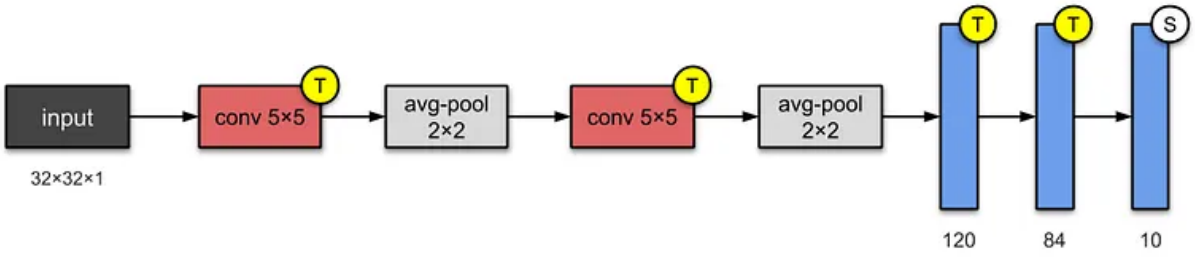


Рис. 1.2. Архітектура LeNet-5 [11]

AlexNet (2012) [14] стала революційною архітектурою, яка виграла

змагання ImageNet-2012 з величезним відривом. Вона містить 8 шарів (5 згорткових і 3 повнозв'язних) і близько 60 мільйонів параметрів. AlexNet популяризувала використання ReLU замість sigmoid/tanh активацій, застосувала dropout для запобігання перенавчанню та продемонструвала ефективність навчання на GPU.

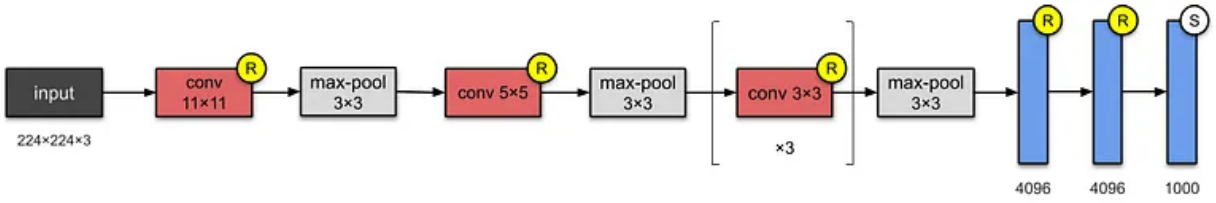


Рис. 1.3. Архітектура AlexNet [11]

VGG-16 (2014) [15] досліджувала вплив глибини мережі на її продуктивність. Архітектура використовує виключно маленькі 3×3 згорткові фільтри, які застосовуються послідовно для досягнення більшого рецептивного поля. VGG-16 містить 16 шарів з вагами (13 згорткових і 3 повнозв'язних) та близько 138 мільйонів параметрів. Простота та однорідність архітектури зробили VGG популярною базовою моделлю.

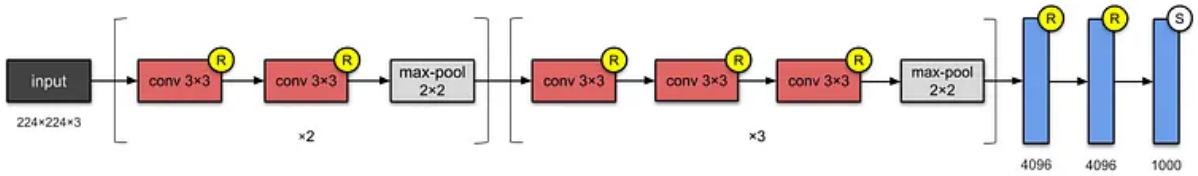


Рис. 1.4. Архітектура VGG-16 [11]

ResNet (2015) [16] вирішила проблему деградації градієнтів у дуже глибоких мережах через запровадження залишкових з'єднань (residual connections). Ці з'єднання дозволяють градієнтам поширюватися безпосередньо через шари, що уможливило створення мереж з сотнями шарів. ResNet досягла рекордної точності на ImageNet і стала стандартом для багатьох завдань комп'ютерного зору.

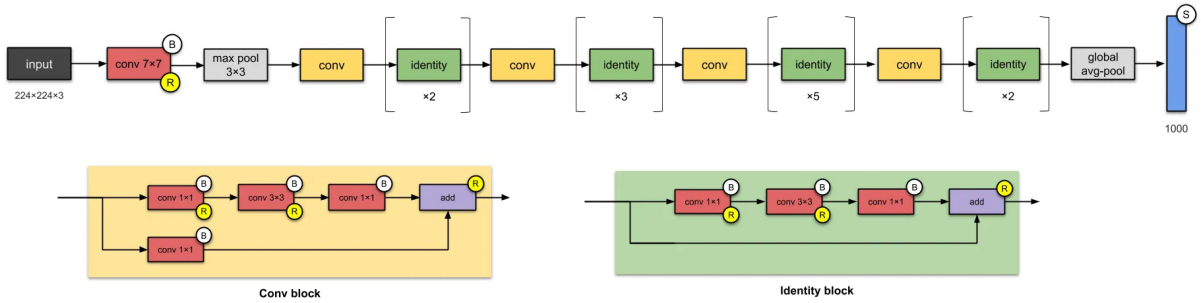


Рис. 1.5. Архітектура ResNet [11]

U-Net (2015) [17] — архітектура, розроблена Олафом Роннебергером та його колегами для задач семантичної сегментації зображень, особливо у біомедичних застосуваннях. На відміну від попередніх архітектур, призначених для класифікації, U-Net створена для попіксельного передбачення, де кожен піксель вихідного зображення має бути класифікований. Архітектура має характерну U-подібну форму, що складається з двох основних частин: encoder (стискаючий шлях) та decoder (розширюючий шлях). Encoder послідовно зменшує просторову розмірність через згорткові шари та max pooling, виділяючи високорівневі ознаки. Decoder відновлює просторову розмірність через операції upsampling та згортки. Ключовою особливістю є skip connections — прямі з'єднання між відповідними шарами encoder та decoder, які передають просторову інформацію з високою роздільною здатністю. Це дозволяє зберегти деталі оригінального зображення та досягти точної локалізації. U-Net продемонструвала видатні результати у сегментації медичних зображень, навіть з невеликими навчальними датасетами, завдяки використанню аугментації даних. Архітектура містить близько 31 мільйона параметрів та стала стандартом для задач семантичної сегментації в медичній візуалізації.

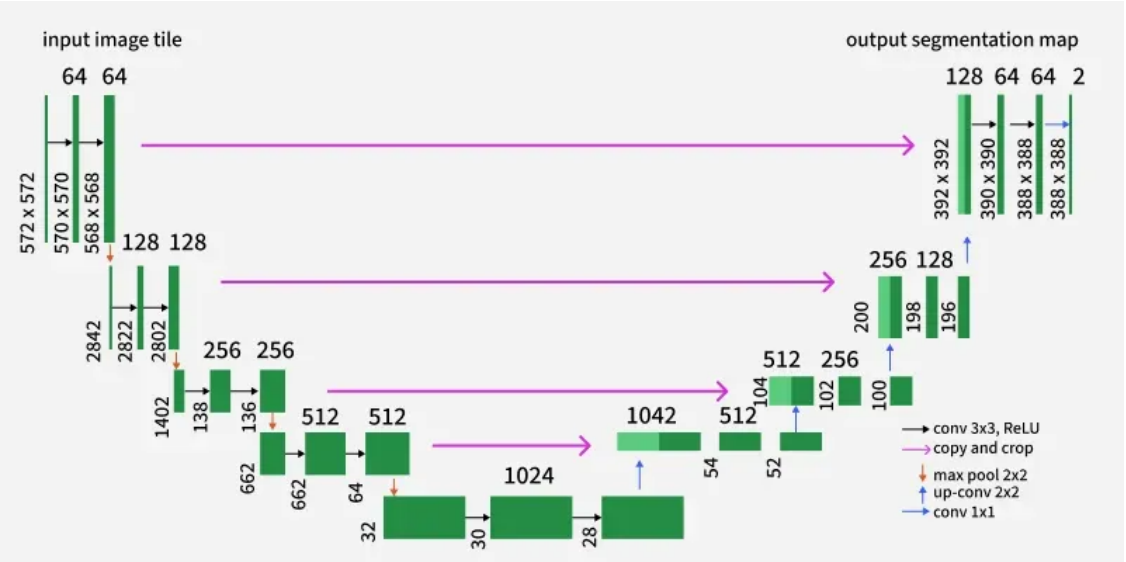


Рис. 1.6. Архітектура U-Net [12]

1.4 Методи пошуку оптимальної архітектури CNN

Процес проектування архітектури нейронної мережі традиційно вимагав значної експертизи та емпіричних експериментів. Автоматизований пошук архітектури (Neural Architecture Search, NAS) виник як відповідь на цю проблему, дозволяючи алгоритмічно досліджувати простір можливих архітектур.

1.4.1 Ручне проектування архітектур

Традиційний підхід до створення CNN полягає у ручному визначенні структури мережі на основі інтуїції, досвіду та емпіричного тестування. Експерти визначають кількість шарів, типи операцій, розміри фільтрів та інші гіперпараметри. Цей процес є часозатратним, вимагає глибоких знань та часто не гарантує оптимального результату. Більшість сучасних архітектур (LeNet, VGG, ResNet) були створені саме таким чином.

1.4.2 Підходи до автоматизованого пошуку

NAS складається з трьох основних компонентів: простору пошуку (search space), що визначає можливі архітектури; стратегії пошуку (search strategy), яка досліджує цей простір; та методу оцінювання продуктивності (performance evaluation), що ранжує кандидати.

Підходи на основі навчання з підкріпленням (Reinforcement Learning) трактують пошук архітектури як задачу послідовного прийняття рішень. Агент (часто реалізований як рекурентна нейронна мережа) генерує архітектури крок за кроком, приймаючи рішення про шари, операції та з'єднання. Після створення архітектури вона навчається та оцінюється на цільовій задачі, а результуюча продуктивність (наприклад, точність валідації) використовується як сигнал винагороди. Класичний приклад — NASNet, що використовував RL для виявлення високопродуктивних згорткових комірок, які могли бути перенесені на більші датасети, такі як ImageNet.

Градієнтні методи (Gradient-based) трактують простір пошуку як неперервний граф, де вибір архітектури представлений у вигляді ймовірностей. DARTS (Differentiable Architecture Search) оптимізує ці ймовірності за допомогою градієнтного спуску, що значно зменшує обчислювальну вартість пошуку. Цей підхід дозволяє знаходити архітектури за лічені години замість днів або тижнів.

Еволюційні методи імітують процес природного відбору для виявлення оптимальних архітектур. Вони підтримують популяцію кандидатів, оцінюють їхню продуктивність (фітнес) та селектують найкращі моделі. Нові архітектури створюються шляхом мутації (зміна шарів, розмірів ядер або з'єднань) та кросовера (комбінування частин двох архітектур). Протягом ітерацій популяція еволюціонує до кращих дизайнів.

Відомим прикладом є AmoebaNet-A — класифікатор зображень, що перевершив багато архітектур, спроектованих вручну. Дослідження Large-Scale Evolution of Image Classifiers продемонструвало, що чисті еволюційні методи можуть конкурувати з моделями, створеними людиною, за умови достатніх обчислювальних ресурсів.

РОЗДІЛ 2

РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ НА JAVA

2.1 Загальна структура системи

Система автоматичного пошуку архітектури CNN реалізована на мові Java без використання сторонніх бібліотек машинного навчання. Така архітектура рішення забезпечує повний контроль над процесом навчання та дозволяє детально дослідити внутрішні механізми роботи нейронної мережі.

Основу системи складає об'єктно-орієнтована архітектура з чіткою ієрархією класів. Абстрактний базовий клас `Layer` визначає спільний інтерфейс для всіх типів шарів нейронної мережі. Від нього успадковуються три конкретні реалізації: `ConvolutionLayer` для згорткових операцій, `MaxPoolLayer` для операцій підвибірки та `FullyConnectedLayer` для повнозв'язних шарів.

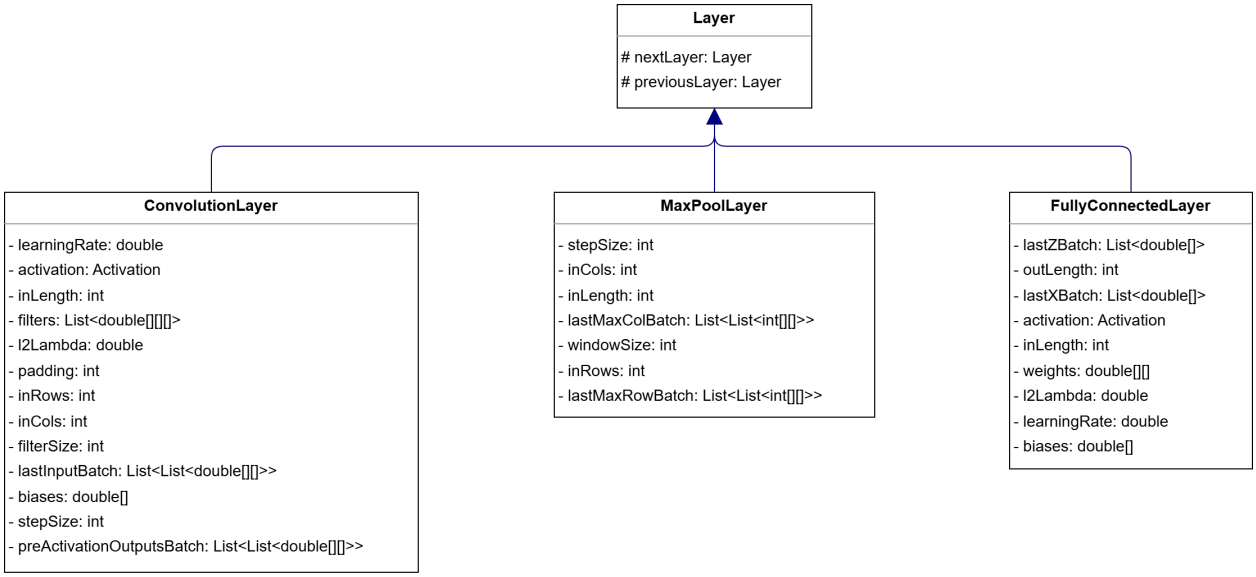


Рис. 2.1. Ієрархічна структура класів шарів

Інтерфейс `Activation` визначає загальну структуру функцій активації, з конкретними реалізаціями `ReLU`, `LeakyReLU` та `Sigmoid`.

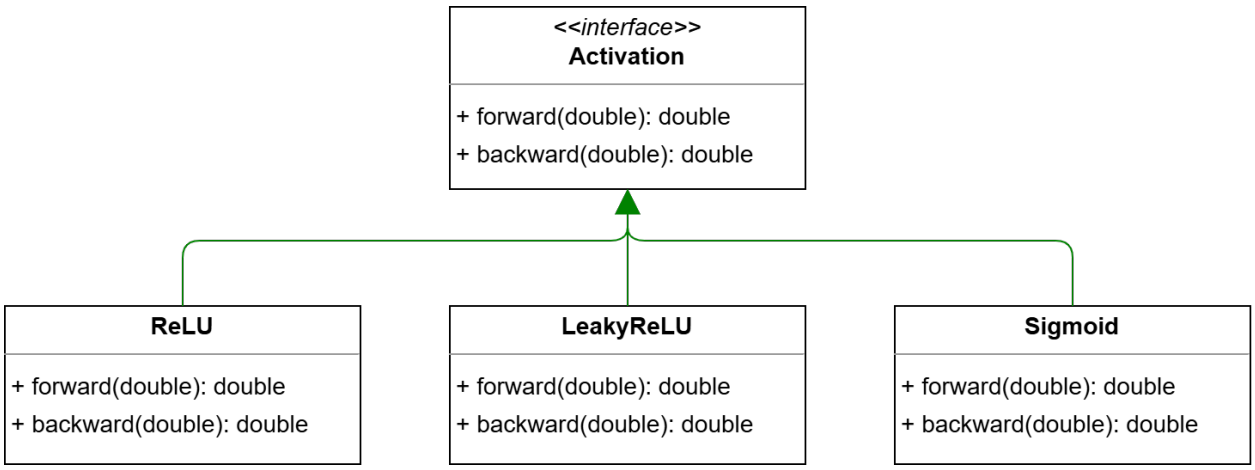


Рис. 2.2. Ієрархічна структура класів функцій активації

Головним класом, який об'єднує ці шари у мережу та керує навчанням, є клас `NeuralNetwork`, а для зручного конструювання архітектури використовується клас-будівельник `NetworkBuilder`.

2.1.1 Базовий клас Layer

Клас `Layer` реалізує патерн двозв'язного списку через атрибути `nextLayer` та `previousLayer`, що дозволяє ефективно передавати дані в обох напрямках: вперед під час прямого проходу та назад під час зворотного поширення помилки. Основні абстрактні методи включають `getOutput()` для реалізації forward propagation та `backPropagation()` для backward propagation.

Для забезпечення сумісності між різними типами шарів клас містить допоміжні методи `matrixToVector()` та `vectorToMatrix()` , які перетворюють дані між матричним представленням (для згорткових шарів) та векторним (для повнозв'язних шарів). Метод `matrixToVector()` послідовно копіює елементи всіх матриць у одновимірний масив, зберігаючи просторову структуру даних. Зворотна операція `vectorToMatrix()` відновлює матричне представлення з вектора, використовуючи задані розміри.

2.2 Реалізація згорткового шару

Згортковий шар є основним будівельним блоком CNN, який виконує операцію згортки — обчислення скалярного добутку між ядром (фільтром) та локальною областю входу.

2.2.1 Структура та параметри

Клас `ConvolutionLayer` реалізує операцію згортки для багатоканальних зображень. Основними параметрами є розмір фільтрів $k \times k$, кількість фільтрів F, крок зсуву (stride) s, padding p, а також функція активації. Кожен фільтр містить ваги у вигляді тривимірної матриці розмірності $[inLength \times k \times k]$ та скаляр зміщення (bias).

Розмір вихідної карти ознак після операції згортки обчислюється таким чином:

$$H_{out} = \left\lfloor \frac{H_{in} - k + 2p}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - k + 2p}{s} \right\rfloor + 1,$$

де H_{in}, W_{in} — висота та ширина вхідних карт ознак, k — розмір фільтра, p — padding, s — stride.

2.2.2 Прямий прохід згорткового шару

Під час прямого проходу для кожного вхідного зразка і для кожного фільтра виконується операція згортки відповідно до формули:

$$(Y * W_f)_{i,j} = \sum_{c=0}^{C-1} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} Y_{c,i \cdot s+m, j \cdot s+n} \cdot W_{f,c,m,n},$$

де $Y \in \mathbb{R}^{C \times H_{in} \times W_{in}}$ — вхідні карти ознак з C каналами, а W_f — фільтр номер f.

Після згортки додається зміщення b_f, а потім застосовується функція активації f:

$$A_{f,i,j} = f\left((Y * W_f)_{i,j} + b_f\right).$$

Цей процес повторюється для усіх фільтрів і для усіх зразків батчу.

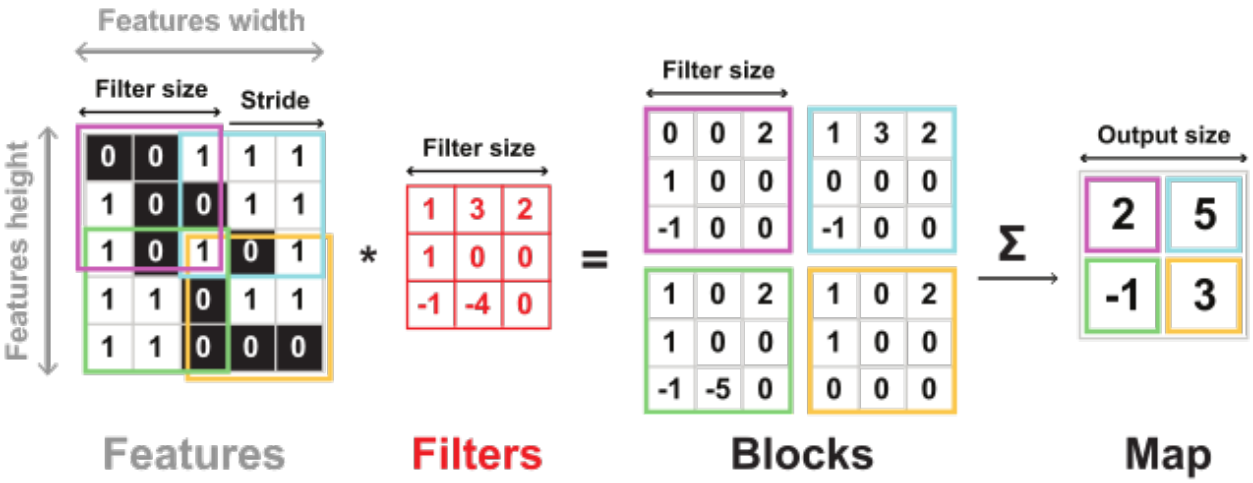


Рис. 2.3. Операція згортки

Згорткові шари виявляють локальні паттерни, такі як краї, текстури та форми. Перші шари розпізнають прості ознаки, а глибші шари комбінують їх у складніші представлення.

2.2.3 Зворотне поширення через згортковий шар

Зворотне поширення реалізовано для батчів розміром N. Для кожного зразка обчислюється локальний градієнт відносно виходу шару δ, що є похідною втрат по активації після застосування функції активації:

$$\delta^{(l)}_{f,i,j} = \frac{\partial \mathcal{L}}{\partial z^{(l)}_{f,i,j}} = \frac{\partial \mathcal{L}}{\partial a^{(l)}_{f,i,j}} \cdot f'(z^{(l)}_{f,i,j}),$$

де $z^{(l)}_{f,i,j}$ — зважена сума перед активацією.

Градієнти по фільтрах обчислюються як кореляція між вхідними картами ознак та сигнальним градієнтом помилки:

$$\frac{\partial \mathcal{L}}{\partial W_{f,c,m,n}} = \frac{1}{N} \sum_{b=1}^{N} \sum_{i,j} \delta^{(l,b)}_{f,i,j} \cdot X^{(b)}_{c,i\cdot s+m,j\cdot s+n},$$

де $X^{(b)}$ — вхідні карти ознак b-го зразка батчу.

Градієнт відносно входу шару передається на попередній шар через повну згортку градієнта помилки з перевернутими за просторовими осями фільтрами:

$$\frac{\partial \mathcal{L}}{\partial X^{(b)}_{c,i,j}} = \sum_{f} \sum_{m,n} \delta^{(l,b)}_{f,m,n} \cdot W_{f,c,(i-m\cdot s),(j-n\cdot s)},$$

підсумовування здійснюється по обмеженій області, де фільтр накладається

на позицію (i,j).

Градієнти по зміщенням обчислюються як:

$$\frac{\partial \mathcal{L}}{\partial b_f} = \frac{1}{N} \sum_{b=1}^{N} \sum_{i,j} \delta_{f,i,j}^{(l,b)}.$$

Після обчислення цих градієнтів параметри фільтрів і зміщень оновлюються за правилом градієнтного спуску із врахуванням коефіцієнта швидкості навчання η і L2-регуляризації зі змінною λ:

$$W := W - \eta \left(\frac{\partial \mathcal{L}}{\partial W} + \lambda W \right), \quad b := b - \eta \frac{\partial \mathcal{L}}{\partial b}.$$

Для оптимізації обчислень у процесі зворотного проходу використовуються допоміжні методи для обробки паддінгу, розвороту фільтрів і розширення вхідних даних.

Завдяки рекурсивній архітектурі класу, градієнти для попереднього шару передаються викликом методу `backPropagationBatch()`, що забезпечує поширення помилки крізь усю мережу.

2.3 Реалізація повнозв'язного шару

Після декількох згорткових і pooling шарів, які виконують функцію екстракції ознак, зазвичай використовуються повнозв'язні (fully connected) шари для класифікації або регресії. У цих шарах кожен нейрон з'єднаний з усіма нейронами попереднього шару, що дозволяє інтегрувати інформацію з різних частин зображення для прийняття фінального рішення.

Перед першим повнозв'язним шаром багатовимірний вихід згорткових шарів перетворюється у одновимірний вектор (операція flatten). Повнозв'язні шари працюють аналогічно до класичних багатошарових перцептронів:

$$a^{(l)} = f(W^{(l)} a^{(l-1)} + b^{(l)}),$$

де всі елементи матриці $W^{(l)}$ є навчальними параметрами.

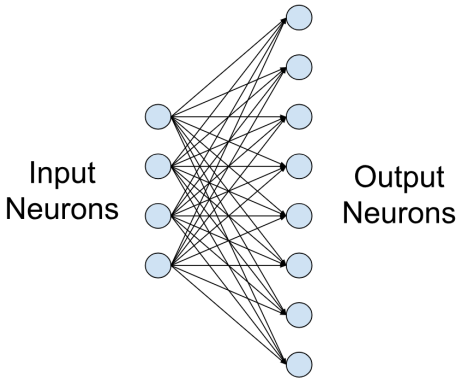


Рис. 2.4. Повнозв'язний шар

2.3.1 Структура та параметри

Клас `FullyConnectedLayer` реалізує повнозв'язний (fully connected) шар згідно з класичною матричною моделлю. Основними параметрами є розміри вхідного вектору n_{in} і кількість вихідних нейронів n_{out}, матриця ваг $W \in \mathbb{R}^{n_{in} \times n_{out}}$, вектор зміщень $b \in \mathbb{R}^{n_{out}}$, а також об'єкт функції активації.

Для підтримки навчання з батчами клас зберігає послідовності останніх вхідних векторів (`lastXBatch`) і проміжних зважених сум (`lastZBatch`), що використовуються під час зворотного проходу.

Ініціалізація ваг вибирається залежно від функції активації: для `ReLU`, `LeakyReLU` та лінійної активації застосовується ініціалізація за Хе (He initialization), а для сигмоїдної функції — ініціалізація Ксав'єра (Xavier initialization).

2.3.2 Прямий прохід

Прямий прохід для кожного вхідного вектора $a^{(l-1)} \in \mathbb{R}^{n_{in}}$ та ваг $W^{(l)}$ реалізується формулою:

$$z^{(l)} = W^{(l)T} a^{(l-1)} + b^{(l)},$$

де $z^{(l)} \in \mathbb{R}^{n_{out}}$ — зважена сума на шарі l.

Застосовується функція активації:

$$a^{(l)} = f^{(l)}(z^{(l)}),$$

де $f^{(l)}$ — обрана функція активації для шару.

У реалізації оптимізовано матричний добуток, пропускаючи нульові елементи вхідного вектора, що є особливо ефективним при зустрічі розріджених даних (наприклад, після ReLU активації чи пулінгу).

2.3.3 Зворотне поширення

Зворотне поширення реалізоване для батчів розміром N. Для кожного зразка батчу обчислюється локальна похибка як поелементний добуток градієнта втрат по виходу шару на похідну функції активації:

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial a^{(l)}} \odot f'^{(l)}(z^{(l)}),$$

де $\odot$ — поелементне множення (Hadamard product).

У наступному кроці обчислюється градієнт відносно вхідних даних шару:

$$\frac{\partial \mathcal{L}}{\partial a^{(l-1)}} = W^{(l)}\delta^{(l)}.$$

Градієнти відносно ваг обчислюються для всього батчу як середнє арифметичне по зразках:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{1}{N}(a^{(l-1)})^T\delta^{(l)},$$

де $a^{(l-1)}$ — матриця з вхідними векторами батчу, $\delta^{(l)}$ — матриця з векторами локальної похибки.

Градієнт по зміщеннях:

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \frac{1}{N}\sum_{i=1}^{N}\delta_i^{(l)}.$$

Оновлення параметрів виконується за правилом градієнтного спуску з урахуванням L2-регуляризації:

$$W^{(l)} := W^{(l)} - \eta\left(\frac{\partial \mathcal{L}}{\partial W^{(l)}} + \lambda W^{(l)}\right),$$

$$b^{(l)} := b^{(l)} - \eta\frac{\partial \mathcal{L}}{\partial b^{(l)}},$$

де η — швидкість навчання, λ — коефіцієнт L2-регуляризації.

Після оновлення ваг, локальний градієнт $\frac{\partial \mathcal{L}}{\partial a^{(l-1)}}$ передається попередньому шару для продовження процесу backpropagation.

2.3.4 Рекурсивна архітектура

Метод `getOutputBatch()` реалізує рекурсивний forward pass по мережі, передаючи вихідні значення поточного шару на вхід наступного.

Метод `backPropagationBatch()` виконує рекурсивний backward pass, отримуючи пачку градієнтів і поширюючи помилку назад через всі шари мережі послідовно.

2.4 Реалізація шару Max Pooling

Pooling (субдискретизація або підвибірка) — це операція зменшення просторової розмірності карт ознак, що забезпечує інваріантність до невеликих трансформацій та зменшує обчислювальну складність. Найпоширенішим типом є max pooling, який вибирає максимальне значення у кожному вікні:

$$y_{i,j} = \max_{0 \le m < p, 0 \le n < p} x_{i \cdot s + m, j \cdot s + n},$$

де p — розмір вікна pooling (зазвичай 2×2), s — stride (зазвичай дорівнює p).

Розмір виходу після max pooling:

$$H_{out} = \left\lfloor \frac{H_{in} - p}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - p}{s} \right\rfloor + 1.$$

Інші поширені варіанти — average pooling, який обчислює середнє значення у вікні:

$$y_{i,j} = \frac{1}{p^2} \sum_{m=0}^{p-1} \sum_{n=0}^{p-1} x_{i \cdot s + m, j \cdot s + n}.$$

Операції pooling не мають навчальних параметрів, але відіграють важливу роль у досягненні просторової інваріантності та контролі перенавчання (overfitting).

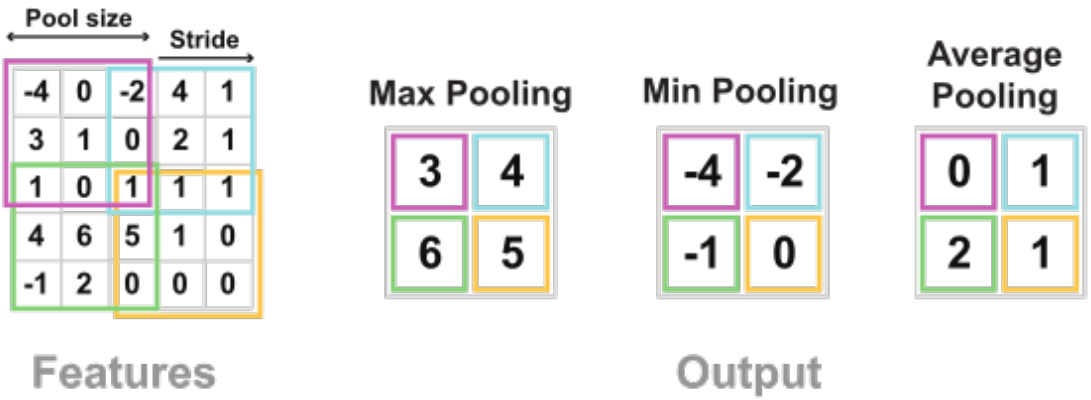


Рис. 2.5. Операції pooling

2.4.1 Структура та параметри

Клас `MaxPoolLayer` реалізує операцію підвибірки (max pooling) для вхідних карт ознак з параметрами: розмір вікна пулінгу $w \times w$ (windowSize)

та крок зсуву (stride) s. Вхідні дані представлені у вигляді багатоканальних карт ознак з розмірами $C \times H_{in} \times W_{in}$.

Розміри вихідної карти ознак обчислюються за формулами:

$$H_{out} = \left\lfloor \frac{H_{in} - w}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - w}{s} \right\rfloor + 1,$$

де H_{in}, W_{in} — висота і ширина вхідних карт, w — розмір вікна, s — stride.

Для підтримки зворотного проходу зберігаються індекси максимальних елементів у кожному вікні пулінгу для кожного зразка батчу, що значно спрощує та оптимізує обчислення градієнтів.

2.4.2 Прямий прохід

При прямому проході для кожного вікна розміром $w \times w$ по кожній карті ознак обирається максимальне значення всередині цього вікна. Ці максимуми формують відповідні елементи вихідної карти ознак $A \in \mathbb{R}^{C \times H_{out} \times W_{out}}$.

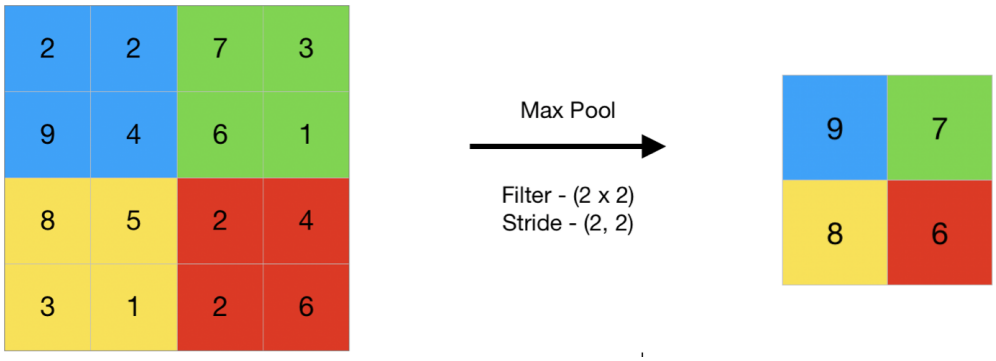


Рис. 2.6. Операція Max Pooling

Для батча розміром N, операція виконується послідовно для кожного зразка, для кожного каналу, та кожної позиції вхідної карти ознак.

2.4.3 Зворотне поширення

Під час зворотного проходу градієнт втрат по виходу шару $\delta^{(l)}$ поширюється лише до позицій, де був обраний максимум під час forward pass. Формально для кожного зразка батча:

$$\frac{\partial \mathcal{L}}{\partial X_{c,i,j}} = \begin{cases} \delta^{(l)}_{c,r,s}, & \text{якщо } (i,j) \text{ є позицією максимуму у } w \times w \text{ вікні,} \\ 0, & \text{інакше,} \end{cases}$$

де (r, s) — позиція елемента у вихідній карті ознак, що відповідає вікну, у якому знаходиться (i, j).

Завдяки збереженню індексів максимумів у під час прямого проходу, зворотний поширення виконується ефективно, коректно передаючи градієнти попередньому шару.

Для батчевої обробки повертаються батчі карт з градієнтами відповідних розмірів.

2.4.4 Особливості реалізації

- Реалізація використовує рекурсивні виклики методів `getOutputBatch()` та `backPropagationBatch()`, що дозволяє легко інтегрувати шар у мережу будь-якої глибини.
- Збережені масиви індексів максимумів забезпечують модульність та ефективність обчислень.
- Кількість параметрів в шарі відсутня, оскільки max pooling не має навчальних ваг.

2.5 Алгоритм навчання мережі

2.5.1 Принципи роботи алгоритму

Навчання згорткової нейронної мережі базується на послідовному виконанні трьох основних етапів: прямого проходу (forward pass), обчислення функції втрат, а потім зворотного проходу (backward pass) для оновлення ваг.

Під час прямого проходу мережею пропускається батч вхідних зразків, після чого отримується набір вихідних значень. Функція втрат (наприклад, крос-ентропія) обчислюється шляхом порівняння передбачень мережі з істинними мітками.

Зворотне поширення використовує градієнти функції втрат для кожного параметра мережі, які обчислюються рекурсивно шарами методом `backPropagationBatch()`, що дозволяє ефективно оновлювати параметри.

2.5.2 Рекурсивне поширення сигналів помилки

Кожен шар реалізує методи `getOutputBatch()` для прямого проходу батчу та `backPropagationBatch()` для зворотного проходу. Ці методи ви-

кликають аналогічні методи наступного або попереднього шару, створюючи рекурсивний ланцюжок передачі даних та сигналів помилок.

Такий підхід дозволяє модульно реалізувати довільну кількість шарів та гнучко формувати мережеву архітектуру без жорсткого зв'язування зовнішнього коду.

2.5.3 Особливості обробки батчів

Навчання відбувається на батчах розміром N, що забезпечує баланс між якістю оцінки градієнтів та розмірністю пам'яті.

Градієнти для параметрів усереднюються по всіх зразках батчу, що стабілізує процес навчання та сприяє кращій узагальнювальній здатності мережі.

Модель підтримує обробку батчів на рівні основних класів шарів, зокрема, збереження проміжних значень для всіх зразків використовується для точного обчислення локальних градієнтів.

2.5.4 Регуляризація та ініціалізація ваг

Для запобігання перенавчанню використовується L2-регуляризація ваг зі змінною коефіцієнта регуляризації λ: під час обчислення нового значення ваг до градієнта додається додатковий доданок, пропорційний самим вагам.

$$W := W - \eta \left(\frac{\partial \mathcal{L}}{\partial W} + \lambda W \right).$$

Завдяки цьому ваги не можуть надмірно збільшуватися, і модель краще узагальнює дані.

Ініціалізація ваг здійснюється залежно від типу шару та функції активації: для згорткових та повнозв'язних шарів з `ReLU`-подібними активаціями використовується ініціалізація за Хе (He initialization [18]), яка базується на розподілі з урахуванням кількості вхідних зв'язків.

Для шарів з сигмоїдною активацією застосовується ініціалізація Ксав'єра (Xavier initialization [19]), яка оптимізує розподіл початкових ваг для стабільності сигналів у мережі.

РОЗДІЛ 3

ГЕНЕТИЧНИЙ АЛГОРИТМ ДЛЯ ПОШУКУ ОПТИМАЛЬНОЇ АРХІТЕКТУРИ CNN

3.1 Еволюційний підхід

У даній роботі обрано еволюційний підхід на основі генетичних алгоритмів з наступних причин:

- Глобальний пошук: генетичні алгоритми ефективно досліджують великі дискретні простори пошуку та менш схильні до застрягання в локальних оптимумах порівняно з градієнтними методами.
- Дискретна природа: архітектура CNN складається з дискретних компонентів (типи шарів, розміри фільтрів), що природно відповідає еволюційному представленню.
- Незалежність від диференційовності: на відміну від градієнтних методів, еволюційні алгоритми не вимагають, щоб простір пошуку був диференційованим.
- Паралелізація: популяційний характер дозволяє одночасно оцінювати декілька архітектур.
- Гнучкість: легко адаптуються під різні цільові функції та обмеження.

Хоча еволюційні методи можуть бути обчислювально затратними через необхідність навчання багатьох архітектур, вони демонструють хорошу збіжність та здатність знаходити несподівані, високоефективні рішення.

3.1.1 Генетичні алгоритми

Генетичні алгоритми (Genetic Algorithms, GA) є метаевристичними методами оптимізації, натхненими процесами природного відбору та еволюції [20]. Вони належать до класу еволюційних обчислень та ефективно розв'язують складні оптимізаційні задачі, де традиційні методи можуть бути неефективними.

Генетичний алгоритм працює з популяцією потенційних розв'язків (особин), кожен з яких представлений у вигляді хромосоми - закодованого представлення архітектури CNN. Алгоритм ітеративно покращує популяцію, застосовуючи оператори, що імітують генетичні процеси: селекцію, кросовер та мутацію.

3.2 Структура кодування архітектури

3.2.1 Представлення хромосоми

На відміну від простого лінійного представлення, реальна реалізація використовує структурований підхід, де хромосома складається з набору генів шарів (LayerGene), кожен з яких повністю описує один шар мережі з усіма його параметрами.

Хромосома представляється як впорядкований список генів:

```
Conv(8,3×3,same,ReLU) → Pool → Conv(16,5×5,valid,Sigmoid) →
→ FC(128,ReLU) → FC(10,Linear)

Conv(4,5×5,valid,ReLU) → Conv(8,3×3,same,ReLU) → Pool →
→ Conv(16,3×3,same,ReLU) → FC(10,Linear)

Conv(16,3×3,same,LeakyReLU) → Pool → Conv(32,5×5,valid,ReLU) →
→ Pool → FC(256,ReLU) → FC(10,Linear)
```

3.2.2 Типи шарів та гени

У реалізації передбачені три основних типи шарів, кожен з яких має відповідний набір параметрів:

Згорткові шари (CONVOLUTION):

- Кількість фільтрів (filters): допустимі значення {8, 16, 32, 64, 128}.
- Розмір ядра (kernel size): допустимі значення $\{3, 5, 7\}$.
- Функція активації (activation): {ReLU, LeakyReLU, Sigmoid}.
- Padding: Same (з розширенням на (filterSize - 2) / 2) або Valid (без розширення)
- Stride: $\{1, 2\}$.

Шари max-pooling (MAX_POOL):

- Розмір вікна (pool window): {2, 3}.
- Stride: $\{1, 2\}$.
- Максимальна кількість шарів pooling: 3 (для забезпечення достатнього зменшення просторових розмірів).

Повнозв'язні шари (FULLY_CONNECTED):

- Розмір шару (fc size) для прихованих шарів: $\{64, 128, 256, 512\}$.
- Функція активації: {ReLU, LeakyReLU, Sigmoid}.

- Вихідний шар завжди має 10 нейронів (для класифікації 10 цифр MNIST) з лінійною активацією.

3.2.3 Обмеження на архітектуру

Генерація архітектури підлягає наступним обмеженням для забезпечення валідності мереж:

Конволюційні блоки:

- Кількість згорткових блоків: від 0 до 3.
- Кожен блок складається з одного або декількох згорткових шарів та опціонально шару max-pooling.
- Монотонне зростання кількості фільтрів: кожен наступний згортковий шар має кількість фільтрів, не меншу за попередній.

Просторові розміри:

- Вхід: 28×28 (MNIST).
- Після кожної операції обраховується новий розмір: $newSize = (inputSize - filterSize + 2 \times padding)/stride + 1$.
- Мінімальний допустимий розмір: 3×3 (щоб уникнути надмірного зменшення).
- Max-pooling застосовується лише коли просторовий розмір > 6.

Повнозв'язні шари:

- Кількість прихованих FC шарів: від 0 до 2
- Останній вихідний шар завжди присутній.

3.3 Популяція та ініціалізація

Популяція - це множина особин (кандидатів-рішень), що еволюціонує протягом поколінь. Розмір популяції N є важливим гіперпараметром: мала популяція може не охопити достатню різноманітність, тоді як велика вимагає більше обчислювальних ресурсів.

3.3.1 Ініціалізація популяції:

Кожна нова архітектура генерується випадково з дотриманням всіх обмежень:

1) Випадково вибирається кількість згорткових блоків (від 0 до 4)
2) Для кожного блоку:

- Генерується згортковий шар з випадковими параметрами в допустимих межах
- З ймовірністю, визначеною просторовими розмірами, додається max-pooling

3) Генерується від 0 до 2 прихованих повнозв'язних шарів
4) Вихідний шар з 10 нейронів та лінійною активацією

Така стратегія ініціалізації гарантує, що все початкове населення складається з валідних архітектур, готових до тренування.

3.3.2 Фітнес-функція

На відміну від простої точності, ця реалізація використовує багатокритеріальну фітнес-функцію, що враховує як точність, так і складність моделі:
$F = E + \lambda \cdot C$,
де:

- $E = 100$ - Accuracy $\times$ 100 - процент помилок на валідаційному наборі
- $C = \frac{\text{TotalParameters}}{100000}$ - нормалізована кількість параметрів
- $\lambda = 1$ - коефіцієнт, що визначає важливість складності

Таким чином, фітнес-функція мінімізується генетичним алгоритмом, балансуючи між точністю та ефективністю моделі.

Приклад обчислення:

- Архітектура з 98% точністю та 250,000 параметрів: $F = 2+1\times2.5 = 4.5$
- Архітектура з 97% точністю та 50,000 параметрів: $F = 3+1\times0.5 = 3.5$

Друга архітектура вважається краще незважаючи на меншу точність, оскільки її менша складність виправдовує незначну втрату точності.

3.3.3 Механізм кешування

Для оптимізації обчислювальних затрат реалізована система кешування. Якщо архітектура (представлена як рядок) вже була оцінена, її фітнес береться з кешу без повторного тренування. Це дозволяє значно скоротити час еволюції, оскільки популяція часто переобчислює однакові архітектури. Також, якщо нова згенерована архітектура уже є в кеші, то генерується нова, ще не тренована раніше архітектура.

3.4 Селекція

Селекція визначає, які особини передадуть свої гени наступному поколінню. У реалізації використовується детерміністична селекція на основі елітизму:

3.4.1 Елітизм

Найкращі N_{elite} особин (ELITE_COUNT = 10% від населення) автоматично копіюються до наступного покоління без змін. Це гарантує, що найкращі знайдені архітектури не будуть втрачені та слугуватимуть основою для подальшої еволюції.

3.4.2 Селекція для кросовера

Батьки для кросовера вибираються випадково з групи елітних особин (перші ELITE_COUNT особин після сортування за фітнесом). Алгоритм гарантує, що дві батьківські особини завжди різні:

```
Individual p1 = population.get(RANDOM.nextInt(ELITECOUNT));
Individual p2 = population.get(RANDOM.nextInt(ELITECOUNT));
while (p1.equals(p2)) {
    p2 = population.get(RANDOM.nextInt(ELITECOUNT));
}
```

Якщо після 10 спроб вони все ще однакові, другий з батьків вибирається з усього покоління.

3.5 Кросовер

Кросовер (схрещування) комбінує дві батьківські хромосоми для створення нащадків. Реалізація використовує одноточковий кросовер на рівні шарів:

3.5.1 Алгоритм одноточкового кросовера

1) З батьківських хромосом видаляється останній FC шар з 10 нейронів (він завжди присутній).

2) Визначається точка розділу: випадково вибрана позиція в межах меншої з двох хромосом.
3) Нащадок складається з:
    - Шарів від першої батьківської хромосоми до точки розділу;
    - Шарів від другої батьківської хромосоми від точки розділу до кінця;
    - Вихідного FC шару з 10 нейронів.

Приклад:

```
Parent1: [Conv(32,3), Pool, Conv(64,3), FC(128), FC(10)]
Parent2: [Conv(16,5), Conv(32,3), Pool, FC(256), FC(10)]
    == точка розділу після Conv (32,3) ==
Child1: [Conv(32,3), Conv(32,3), Pool, FC(256), FC(10)]
Child2: [Conv(16,5), Conv(64,3), FC(128), FC(10)]
```

Такий підхід забезпечує:

- Наслідування найкращих згорткових структур від обох батьків;
- Збереження топології мережі;
- Природну комбінацію різних архітектурних характеристик.

3.6 Мутація

Мутація вносить випадкові зміни у хромосому, підтримуючи генетичну різноманітність та дозволяючи досліджувати нові області простору пошуку.

3.6.1 Типи мутацій

Реалізація передбачає чотири типи мутацій, кожна з яких застосовується з певною ймовірністю:

Модифікація згорткового шару (ймовірність 30%)

- Випадково вибирається згортковий шар.
- Змінюються його параметри: кількість фільтрів, розмір ядра, padding, stride, функція активації.
- При виборі нової кількості фільтрів дотримується обмеження монотонного зростання.

Додавання згорткового шару (ймовірність 20%)

- Випадково вибирається позиція в архітектурі після наявного згоркового шару.

- Додається новий згортковий шар з випадковими параметрами.
- Перевіряється валідність просторових розмірів.

Модифікація max-pooling (ймовірність 20%)

- З ймовірністю 50% видаляється наявний шар max-pooling (якщо є більш ніж один).
- З ймовірністю 50% додається новий шар max-pooling після випадково вибраного згорткового шару.

Видалення згорткового шару (ймовірність 30%)

- Видаляється випадково вибраний згортковий шар (якщо залишається більше 2 шарів).
- Гарантується збереження мінімальної структури мережі.

Умови для мутацій:

- Останній вихідний FC шар з 10 нейронів ніколи не змінюється.
- Всі мутовані архітектури перевіряються на валідність просторових розмірів.
- Якщо операція призводить до не валідної архітектури, мутація відхиляється та створюється нова архітектура.

3.7 Формування нового покоління

Нове покоління формується за наступною схемою:

1) Елітизм: перші 10% від населення найкращих особин копіюються без змін.
2) Кросовер: 50% нових особин створюються шляхом кросовера елітних особин.
3) Мутація: 30% нових особин створюються шляхом мутації випадково вибраних особин.
4) Випадкові іммігранти: залишок популяції (10%) заповнюється новими, повністю випадковими архітектурами.

Розподіл популяції в процесі реального експерименту (50 особин):

- Елітні особини: 5 (10%)
- Нащадки кросовера: 25 (50%)
- Мутанти: 15 (30%)
- Випадкові іммігранти: 5 (10%)

Випадкові іммігранти допомагають запобігти передчасній збіжності та дозволяють алгоритму дослідити нові області простору пошуку.

3.8 Загальна схема алгоритму

Генетичний алгоритм працює за наступною схемою:

1) Ініціалізація: створення початкової популяції N випадкових архітектур, що задовольняють всі обмеження.
2) Перевірка кешу: видалення особин, архітектури яких вже були оцінені (попередження дублювання).
3) Оцінювання фітнесу: паралельне тренування та тестування кожної архітектури на тренувальному та валідаційному наборах, обчислення фітнесу.
4) Сортування: впорядкування популяції за фітнесом (від меншого до більшого).
5) Селекція: вибір елітних особин
6) Кросовер: створення нащадків шляхом кросовера елітних особин.
7) Мутація: внесення випадкових змін у вибрані особини.
8) Випадкові іммігранти: заповнення залишку популяції новими архітектурами.
9) Формування нового покоління: заміна старої популяції новою (зберігаючи елітні особини).
10) Перевірка критерію зупинки: якщо досягнуто максимальну кількість поколінь, зупинитися; інакше повернутися до кроку 2.

Паралелізація: На кроку 3 оцінювання фітнесу виконуються паралельно з використанням всіх доступних процесорних ядер (IntStream.range().parallel Кешування дозволяє уникнути повторного оцінювання однакових архітектур.

3.9 Переваги та обмеження реалізації

Переваги:

- Багатокритеріальна оптимізація: баланс між точністю та складністю моделі дозволяє знаходити компромісні рішення.
- Паралелізація: одночасне оцінювання декількох архітектур прискорює пошук.
- Кешування: запобігання повторному обчисленню однакових архітектур значно скорочує час еволюції.
- Гнучкість архітектури: обмеження є м'якими, що дозволяє досліджувати широкий простір можливих мереж.
- Елітизм з випадковими іммігранти: забезпечує баланс між експлуатацією кращих рішень та дослідженням нових областей.

Обмеження:

- Обчислювальна вартість: кожен індивід потребує повного тренування мережі, що робить еволюцію часозатратною.
- Гіперпараметри: велика кількість налаштованих параметрів (розмір популяції, кількість поколінь, коефіцієнти мутацій) впливає на результати.
- Відсутність теоретичних гарантій: немає гарантії збіжності до глобального оптимуму.
- Залежність від простору пошуку: якість результатів залежить від визначення допустимих значень параметрів та типів шарів.
- Стохастичність: результати можуть варіюватися через випадкову природу алгоритму.

3.9.1 Обґрунтування для датасету MNIST

Для експериментів на датасету MNIST, який є відносно простим, обрана стратегія із такими компромісами:

- Монотонне зростання фільтрів: гарантує, що мережа прогресивно збільшує перцептивне поле та складність.
- Багатокритеріальна фітнес-функція: з $\lambda = 1$ забезпечує штраф за надмірно складні мережі, але дозволяє знаходити високоточні рішення.
- Невеликі популяція та кількість поколінь: прискорення процесу еволюції при обмежених обчислювальних ресурсах.

3.10 Висновки

Реалізований генетичний алгоритм поєднує класичні техніки еволюційної оптимізації з сучасними вдосконаленнями (паралелізація, кешування, багатокритеріальна оптимізація). Структурована система обмежень забезпечує, що всі генеровані архітектури є валідними та придатними для ефективного тренування на MNIST. Баланс між дослідженням та експлуатацією досягається поєднанням мутації, кросовера та випадкових імігрантів, що дозволяє алгоритму знаходити як локально оптимальні, так і глобально конкурентоспроможні архітектури.

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Умови експерименту

Експериментальна перевірка проводилася з використанням генетичного алгоритму для еволюційного пошуку архітектур згорткових нейронних мереж. Усі налаштування, що керують генерацією, мутацією та навчанням моделей, були винесені у модуль `Constants`. Таблиця 4.1 містить повний перелік використаних параметрів.

Параметр	**Значення**
Розмір вхідного зображення	28×28
Кількість класів	10
Дозволені розміри FC-шарів	$\{64, 128, 256, 512\}$
Коефіцієнт нормалізації (scale factor)	255
Параметр LeakyReLU (leak)	0.01
Дозволені розміри фільтрів	$\{3, 5, 7\}$
Дозволена кількість фільтрів	$\{4, 8, 16, 32, 64\}$
Активації	ReLU, LeakyReLU, Sigmoid
Мінімум блоків Convolution	0
Максимум блоків Convolution	4
Максимум FC-шарів	3
Пулінгові вікна	$\{2, 3\}$
Крок пулінгу	$\{1, 2\}$
Крок згортки	$\{1, 2\}$
Learning rate (Sigmoid)	0.1
Learning rate (ReLU/LeakyReLU)	0.01
Learning rate (Fully Connected)	0.01
L2-регуляризація λ	0.01

Розмір популяції	50
Кількість поколінь	20
Еліта	10% (5 моделей)
Кросовер	50% (25 моделей)
Мутації	30% (15 моделей)
Частка датасету для швидкого фітнесу	1%
Кількість епох	5
Batch size	16

Табл. 4.1. Основні константи та параметри експерименту

Для прискорення еволюційного процесу використовувалася паралельна обробка: навчання кожної архітектури виконувалося у багатопоточному режимі на 16 потоках CPU.

Наприкінці еволюції всі моделі були збережені у кеш. Для подальшої перевірки передбачено окремий фінальний етап: додаткове навчання 20 найкращих архітектур уже на **повному навчальному наборі** (100% даних замість 1%), що дозволить визначити глобально найкращу архітектуру серед знайдених.

4.2 Результати оптимізації

У процесі роботи еволюційного алгоритму було згенеровано загалом 1020 архітектур згорткових нейронних мереж. Серед них 96 моделей виявилися не валідними (некоректні комбінації шарів, неможливість побудувати вихідний тензор після послідовності операцій тощо), тому вони були автоматично відсіяні ще до етапу навчання. Решта 924 архітектур були повноцінно навчені та протестовані на заданій вибірці.

Еволюційний процес тривав 20 поколінь. Для оцінки фітнесу кожна архітектура тренувалася на 1% датасету (MNIST) протягом 5 епох з batch size = 16. Такий режим дозволив швидко порівнювати різні архітектури, зосереджуючись не на абсолютній точності, а на їх відносній придатності до подальшої оптимізації.

Для візуалізації динаміки еволюції побудовано графіки зміни фітнесу та кількості параметрів по поколіннях (див. рис. 4.1, 4.2).

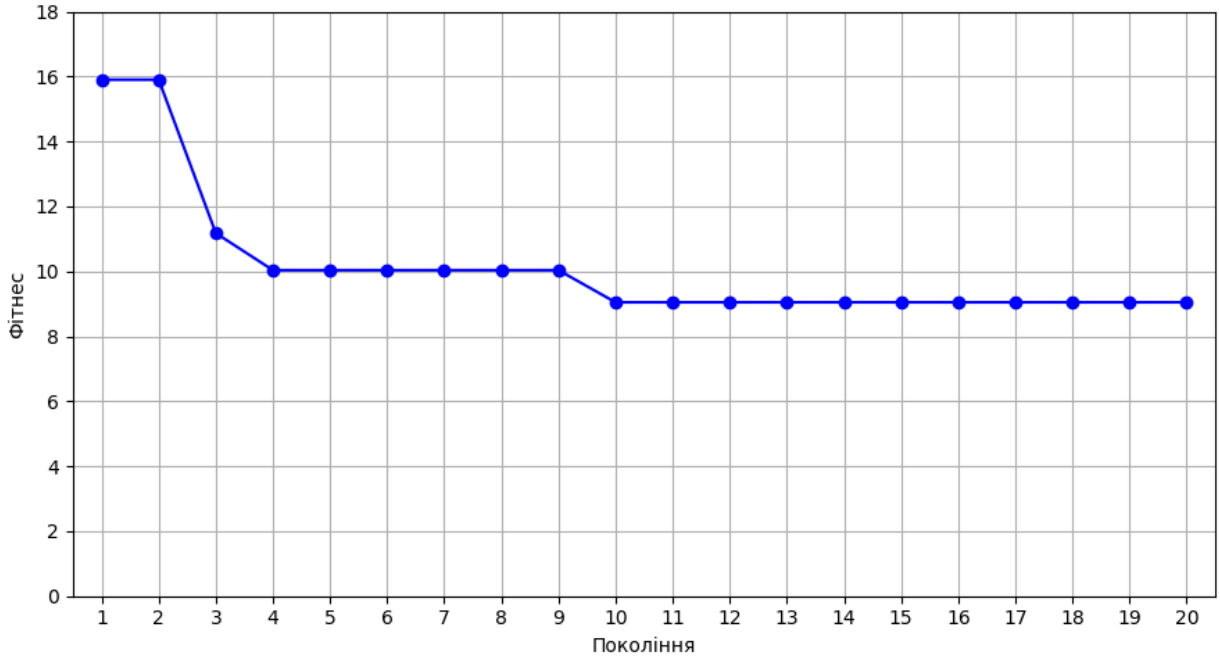


Рис. 4.1. Фітнес за поколіннями

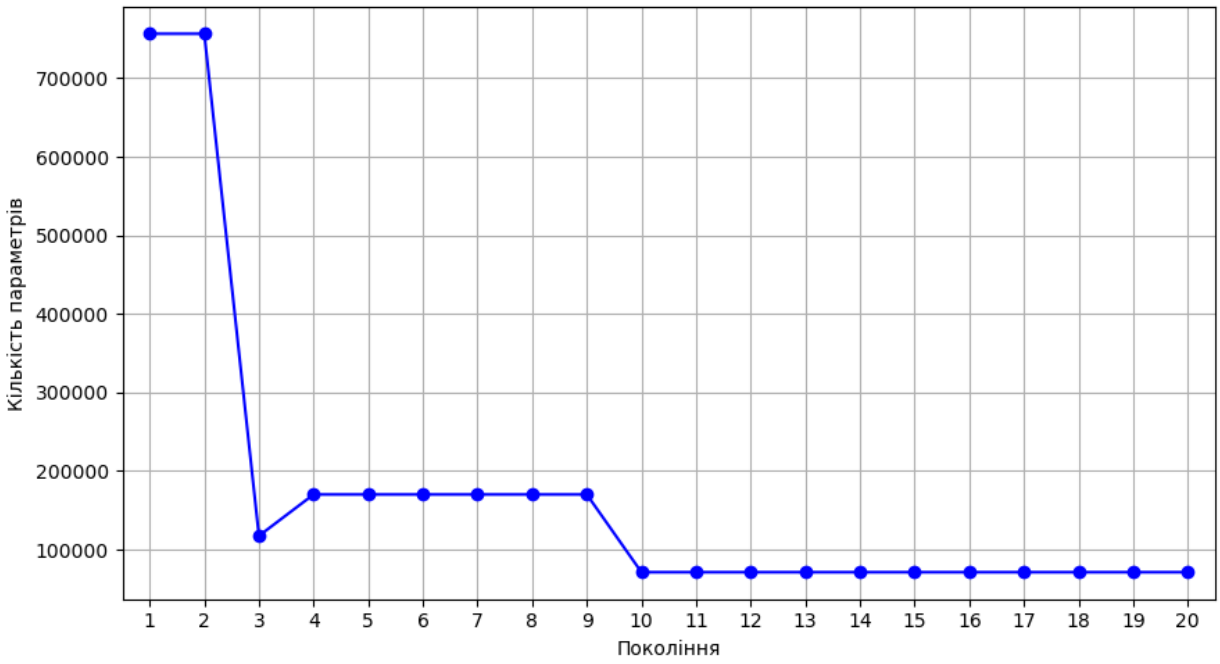


Рис. 4.2. Кількість параметрів за поколіннями

Для глибшого розуміння поведінки еволюційного алгоритму було проаналізовано розподіл значень фітнесу серед усіх згенерованих і валідних архітектур. Такий розподіл дозволяє оцінити,наскільки рівномірно алгоритм досліджував простір архітектур і наскільки рідкісними були високоякісні рішення:

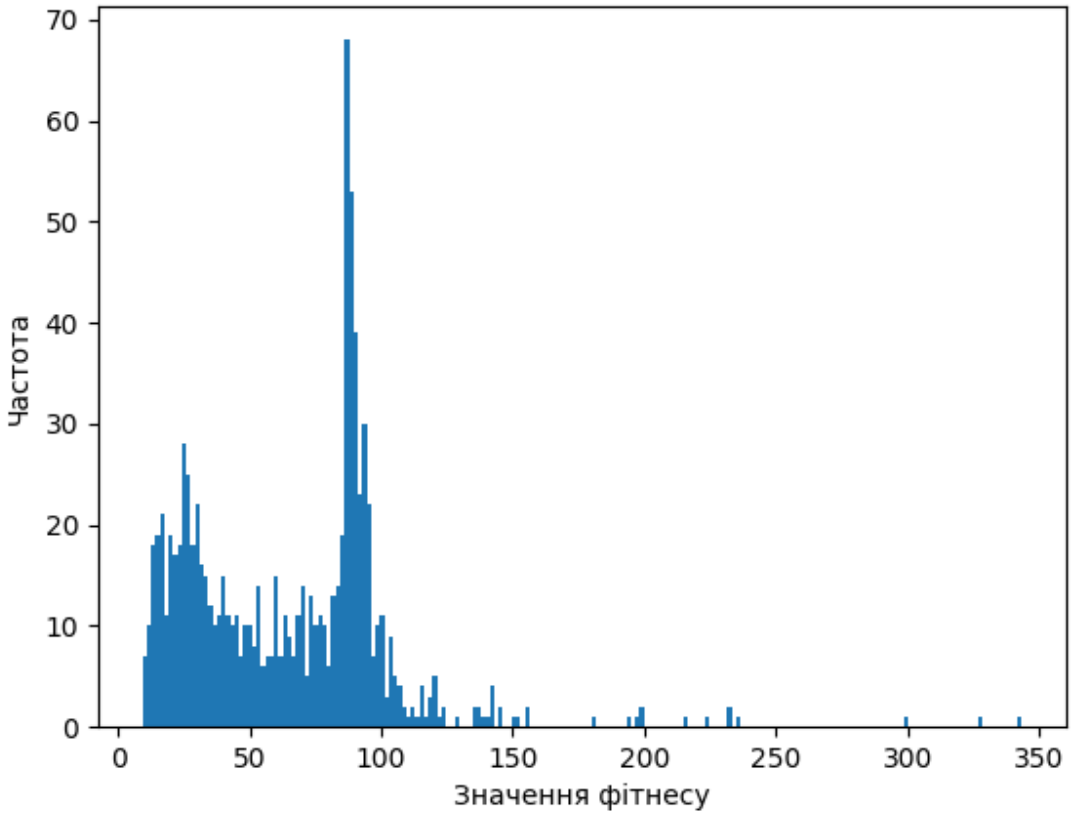


Рис. 4.3. Розподіл значень фітнесу

На гістограмі видно, що основна частина архітектур зосереджена в діапазоні 80–90, що відповідає початковим та невдалим архітектурам. Найкращі (оптимізовані) архітектури мають значно нижчі значення фітнесу (ліворуч на графіку, $F < 20$). Окремі віддалені значення пов'язані з дуже неефективними мережами (з великою кількістю параметрів).

4.3 Аналіз результатів

Після завершення еволюційного пошуку було відібрано 20 найкращих архітектур згідно з досягнутим значенням фітнесу. Для незалежної оцінки їхньої узагальнювальної здатності заплановано повторне тренування кожної моделі на повному навчальному наборі MNIST (100% даних, 60 000 зображень). Через обмеження обчислювальних ресурсів вдалося повністю або частково навчити 18 моделей на одному комп'ютері та 14 моделей на іншому. Решта архітектур виявилися надто "важкими": час однієї епохи сягав десятків годин, тому їх доцільно розглядати як неефективні з точки зору мети цієї роботи – пошуку не лише точної, але й відносно легкої для навчання моделі.

Графік 4.4 ілюструє зміну тестової точності для топових архітектур у процесі навчання протягом 10 епох. Через стохастичний характер навчання (рандомна ініціалізація ваг, перемішування батчів) результати з двох комп'ютерів дещо відрізняються: для окремих моделей різниця тестової точності сягає 2–3%, а інколи й більше. Проте загальна картина зберігається: більшість архітектур сходяться до адекватних значень (приблизно 0.95), а декілька моделей демонструють стабільне наближення до 0.98.

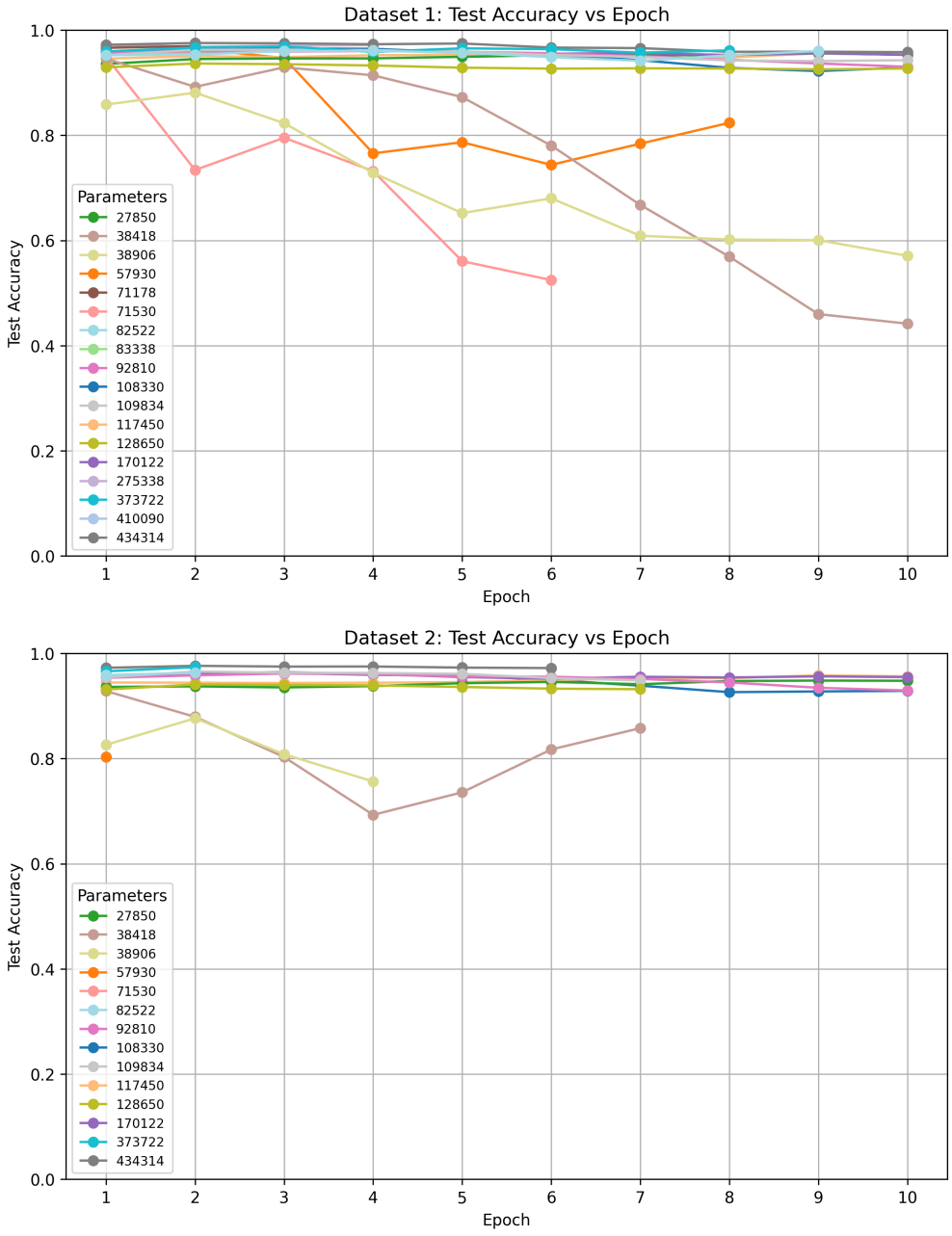


Рис. 4.4. Тестова точність найкращих архітектур за епохами на 100% датасету

Цікавим спостереженням стала поява архітектури, яка під час еволюції мала доволі хороший фітнес, але при донавчанні на повному датасеті показала лише близько 10% точності, тобто рівень випадкового вгадування

для задачі з 10 класами. При цьому час однієї епохи для цієї моделі становив близько 17 годин. Така поведінка свідчить, що навіть низьке значення фітнесу на зменшеній вибірці не завжди гарантує успіх при масштабуванні на повні дані, а також підкреслює важливість врахування обчислювальної вартості при виборі архітектури.

Попри зазначені проблеми, обидва незалежні запуски показали узгоджену тенденцію: найвищу тестову точність демонструє модель з найбільшою кількістю параметрів серед досліджених топових архітектур, але з помірним часом навчання. Її структура виявилася досить простою:

```
CONVOLUTION (64 фільтри 5×5, stride = 1, same padding, LeakyReLU) →
→ MAX_POOL (3×3, stride = 1) → FC output (10)
```

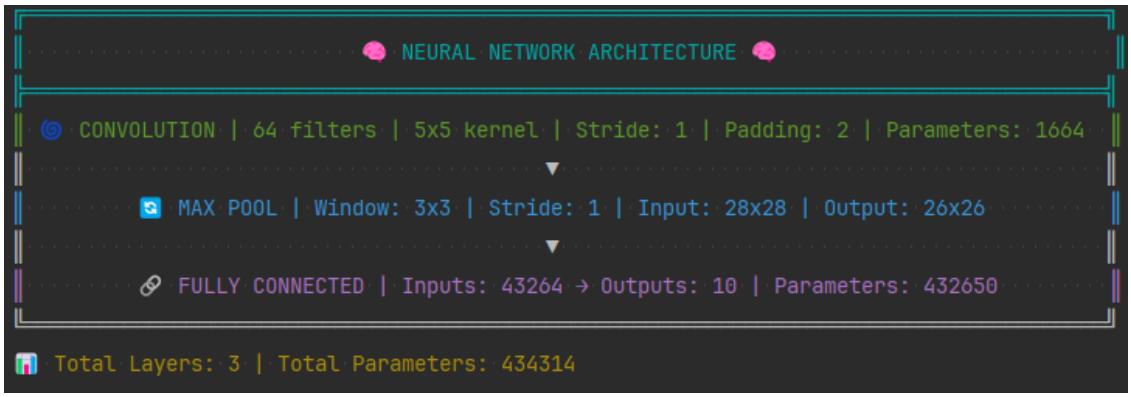


Рис. 4.5. Найкраща архітектура

Ця архітектура досягає тестової точності близько 98% (рис. 4.6) при адекватному часі однієї епохи, тому саме її було обрано як “найкращу” для подальшого аналізу та побудови графіка точності по класах (рис. 4.7).

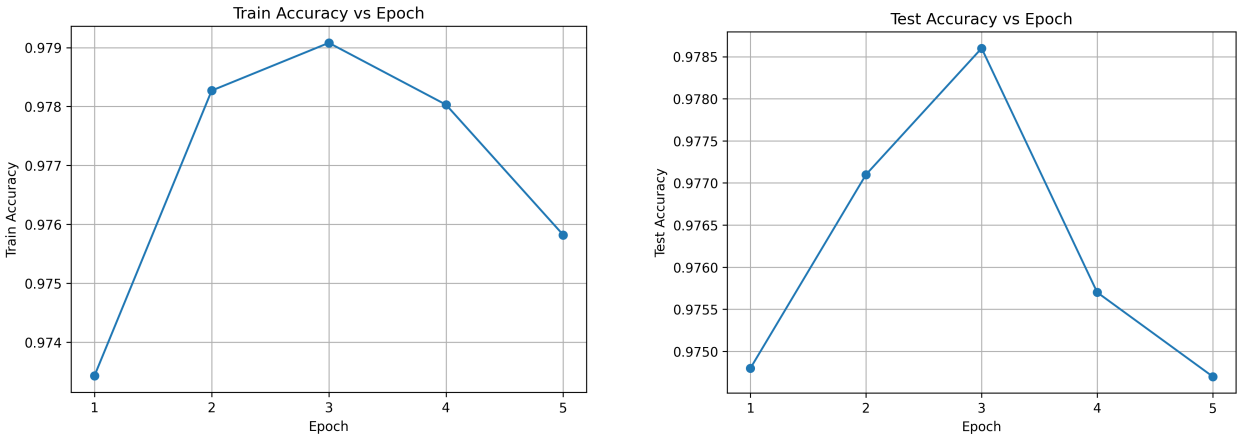


Рис. 4.6. Точність найкращої моделі за епохами

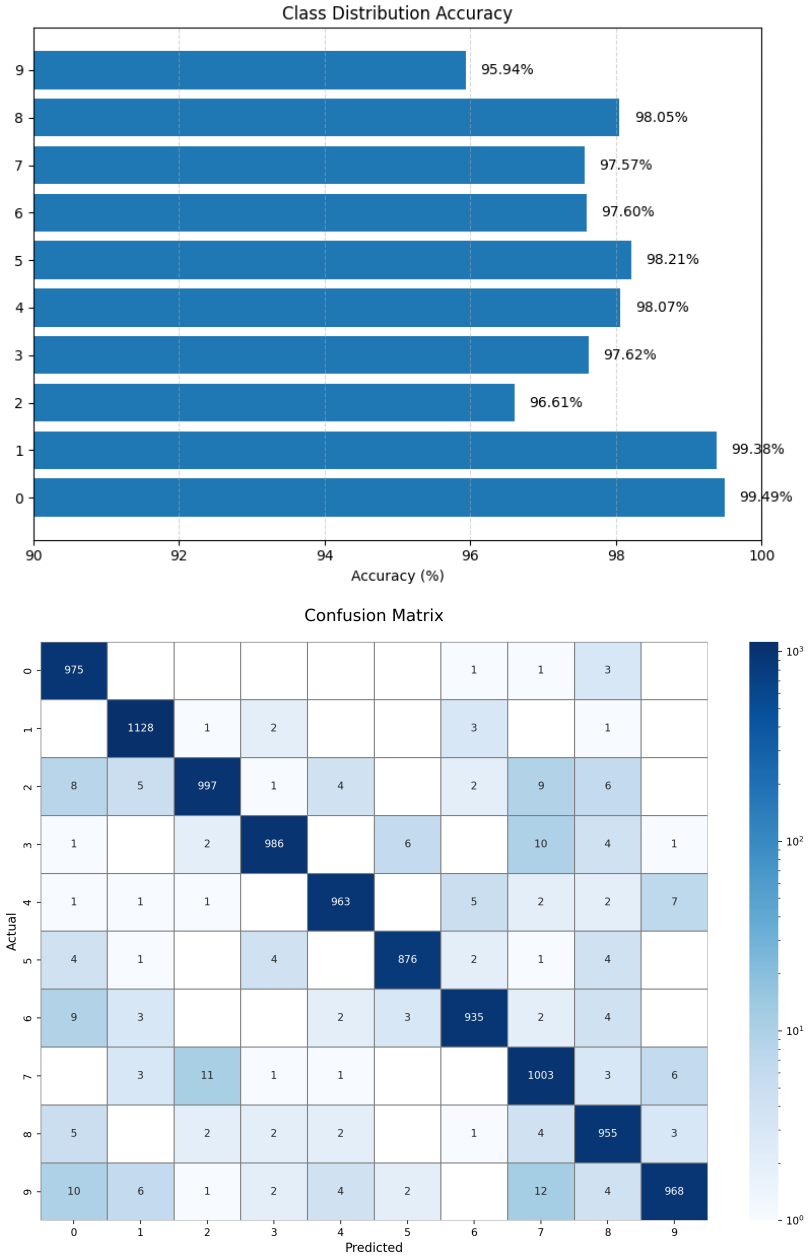


Рис. 4.7. Точність розпізнавання класів для найкращої архітектури

Як видно з рис. 4.7, найкраща архітектура демонструє високу та доволі рівномірну якість розпізнавання для всіх класів: для більшості цифр точність перевищує 97–98%. Найкращі результати досягнуті для класів 0 (99,49%) та 1 (99,38%), що є типовим для датасету MNIST, оскільки ці цифри мають найменшу внутрішню варіативність написання. Відносно нижчі значення спостерігаються для класів 2 (96,61%) та 9 (95,94%), де частина помилок пов'язана зі схожістю контурів із іншими цифрами та більшою різноманітністю рукописних стилів. Водночас жоден клас не опускається нижче 95%, що підтверджує здатність моделі надійно узагальнювати для всіх типів рукописних цифр.

Для практичної валідації було протестовано найкращу модель у режимі реального часу за допомогою інтерактивного модуля visualization,

що дає змогу користувачу вводити рукописні цифри. На рис. 4.8 наведено приклади результатів розпізнавання цифрового вводу.



Рис. 4.8. Приклади роботи інтерактивного демо для рукописних цифр

Таким чином, еволюційний пошук дозволяє автоматично знаходити архітектури згорткових нейронних мереж, що мають високу точність і оптимізовану складність.

Основні результати:

- алгоритм дослідив великий простір архітектур (924 валідних моделей);
- значення фітнесу систематично зменшувалися протягом 20 поколінь, при цьому частина архітектур відсіялася як надто повільні для практичного використання;
- найкращі моделі підтвердили свою якість при повноцінному навчанні на 100% даних;
- глобально найкраща архітектура досягла високої точності не лише на тестовому наборі, але й у режимі реального часу.

Отже, підхід еволюційної оптимізації показав себе як дієвий інструмент для автоматичного пошуку архітектур CNN та може бути розширений для складніших задач і датасетів.

ВИСНОВКИ

У цій роботі був створений і протестований повноцінний генетичний алгоритм для пошуку оптимальної архітектури згорткових нейронних мереж на датасеті MNIST. В ході роботи вдалося розробити власну реалізацію CNN і всіх необхідних еволюційних операторів, а також провести серію експериментів для оцінки якості знайдених рішень.

Основні результати:

- Здійснено автоматичний пошук архітектур CNN, який дозволив знайти моделі з тестовою точністю до 98% без ручного підбору гіперпараметрів.
- Згенеровано понад тисячу архітектур, із яких відібрано кращі для повторного навчання на всьому датасеті.
- Доведено, що генетичні алгоритми можуть ефективно автоматизувати вибір структури нейромережі.
- Створене програмне ядро легко розширити для інших датасетів та задач комп'ютерного зору.

Практичний результат роботи — готовий інструмент для автоматизованої оптимізації нейронних мереж, який може бути корисним для дослідників та розробників без великих знань у глибокому навчанні.

В майбутньому логічно масштабувати підхід для складніших датасетів і додати більш «економічні» критерії оптимізації (наприклад, час навчання чи споживану пам'ять). Цікаво також протестувати систему на реальних користувацьких даних і інтегрувати з мобільними або онлайн-рішеннями.

Загалом мета роботи досягнута, а результати підтверджують доцільність застосування еволюційних алгоритмів для задач автоматичного пошуку архітектури нейронних мереж.

Результати, викладені в цій роботі, були представлені на міжнародній науково-практичній конференції «Інформаційні технології і автоматизація – 2025» [21] і опубліковані у вигляді тез.

СПИСОК ЛІТЕРАТУРИ

1. LeCun, Y., Cortes, C., & Burges, C. J. C. (2010). The MNIST database of handwritten digits.
   http://yann.lecun.com/exdb/mnist/
2. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain
   Psychological Review. — 1958. — Vol. 65, No. 6. — P. 386–408. DOI: 10.1037/h0042519
3. Minsky M., Papert S. Perceptrons: An Introduction to Computational Geometry. —
4. Werbos P. J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences: Ph.D. thesis. — Cambridge, MA: Harvard University, 1974.Cambridge, MA: MIT Press, 1969. — 258 p.
5. Rumelhart D. E., Hinton G. E., Williams R. J. Learning representations by back-propagating errors
   Nature. — 1986. — Vol. 323. — P. 533–536. DOI: 10.1038/323533a0
6. Rumelhart D. E., Hinton G. E., Williams R. J. Learning internal representations by error propagation
   Parallel Distributed Processing: Explorations in the Microstructure of Cognition / Eds. D. E. Rumelhart, J. L. McClelland. — Cambridge, MA: MIT Press, 1986. — Vol. 1. — P. 318–362.
7. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position
   Biological Cybernetics. — 1980. — Vol. 36, No. 4. — P. 193–202. DOI: 10.1007/BF00344251
8. LeCun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., Jackel L. D. Backpropagation applied to handwritten zip code recognition
   Neural Computation. — 1989. — Vol. 1, No. 4. — P. 541–551. DOI: 10.1162/neco.1989.1.4.541
9. Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L. ImageNet: A large-scale hierarchical image database
   IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2009. — P. 248–255. DOI: 10.1109/CVPR.2009.5206848
10. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks
    Advances in Neural Information Processing Systems (NeurIPS). — 2012. — Vol. 25. — P. 1097–1105.
11. Raimi Karim Illustrated: 10 CNN Architectures — Medium, 2019
    medium.com/data-science/illustrated-10-cnn-architectures

12. U-Net Architecture Explained — GeeksforGeeks, October 9, 2025
geeksforgeeks.org/machine-learning/u-net-architecture-explained
13. 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.
yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
14. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks
Advances in Neural Information Processing Systems (NeurIPS). — 2012. — Vol. 25. — P. 1097–1105.
NeurIPS 2012
15. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition
International Conference on Learning Representations (ICLR). — 2015.
arXiv:1409.1556
16. 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.
arXiv:1512.03385
17. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional networks for biomedical image segmentation
Medical Image Computing and Computer-Assisted Intervention (MICCAI). — 2015. — Vol. 9351. — P. 234–241.
arXiv:1505.04597
18. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
arXiv:1502.01852
19. Understanding the difficulty of training deep feedforward neural networks // Xavier Glorot, Yoshua Bengio Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.
http://proceedings.mlr.press/v9/glorot10a.html
20. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми – К.:«Корнійчук», 2008. – 446 с. SBN 978-966-7599-50
https://pdf.lib.vntu.edu.ua/books/2016/Kononyk_2008_470.pdf
21. Страхов Є.М., Чачко Н.Л. Пошук оптимальної архітектури згорткових нейронних мереж за допомогою алгоритмів глобальної оптимізації
Інформаційні технології і автоматизація – 2025 / Матеріали XVIII міжнародної науково-практичної конференції. Одеса, 30-31 жовтня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. – 1316 с.
https://ontu.edu.ua/download/konfi/2025/Collection-of-abstracts-of-the-conference-ITIA-2025.pdf

ДОДАТОК А

Вихідний код та всі дані: https://github.com/Natanius18/CNN_evolution

