МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри ПМ та МСС
_____Коплик І. В.

_____20___р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

зі спеціальності 113 «Прикладна математика ,
освітньо-професійної програми «Наука про дані та моделювання складних систем»
на тему: «Розробка та навчання згорткової нейронної мережі для задач розпізнавання об’єктів»

Здобувача групи ПМ.м-21 Єрмоленка Назара Владиславовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Єрмоленко Назар
(підпис)

Керівник кандидат фіз.-мат. наук, доцент Аліна Дворниченко _____
(підпис)

Суми – 2023

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**

Кафедра прикладної математики та моделювання складних систем

Рівень вищої освіти другий (магістр)

Галузь знань **11 «Математика та статистика»**

Спеціальність **113 «Прикладна математика»**

Освітня програма **освітньо-професійна «**Наука про дані та моделювання складних систем**»**

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМтаМСС
Коплик І. В._____
«___» _____20___р.

І Н Д И В І Д У А Л Ь Н Е З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧЕВІ ВИЩОЇ ОСВІТИ

Єрмоленко Назар Владиславович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та навчання згорткової нейронної мережі для задач розпізнавання об'єктів

Керівник роботи Дворниченко Аліна Василівна
канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада)

затверджено наказом по факультету ЕлІТ від «07»листопада 2023 р.№1237-VI

2. Термін подання роботи здобувачем «16» грудня 2023р.

3. Вихідні дані до роботи набір мультимедійних даних знімків автомобілів у всіх переглядах

4. Зміст розрахунково-пояснювальної записки (перелік питань, для розроблення): 1) Літературний огляд і аналіз існуючих підходів; 2) Теоретичні аспекти згорткових нейронних мереж; 3) Підготовка бази даних; 4) Розробка моделі; 5) Оцінка результатів та порівняння з існуючими методами.

5. Перелік графічного матеріалу: схеми архітектури нейронних мереж, графіки та діаграми моделей, рисунки результатів прогнозування.

6. Консультанти проєкту (роботи) із зазначенням розділів проєкту, що їх стосується

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1)	Літературний огляд і аналіз існуючих підходів	06.11 – 12.11	
2)	Ознайомлення з теоретичними аспектами нейронних мереж	13.11 – 20.11	
3)	Створення бази даних для подальшої розробки моделі	21.11 – 25.11	
4)	Розробка моделі для виявлення об’єкту	26.11 – 05.12	
5)	Оцінка результатів та порівняння з існуючими методами	06.12 – 16.12	

Здобувач вищої освіти _____ Єрмоленко Н.В.
(підпис)

Керівник роботи _____ Дворниченко А.В.
(підпис)

АНОТАЦІЯ

Звіт містить: 75с., 27 рис. , 19 джерел, 4 додатки.

Мета роботи: розробка та навчання згорткової нейронної мережі для вирішення розпізнавання об’єктів на зображеннях.

Об’єкт дослідження: згорткова нейронна мережа в контексті задач розпізнавання об’єктів.

Предмет дослідження: архітектура, оптимізація параметрів, підготовка даних та стратегії у розробці навчання.

Висновки і кінцеві розрахунки знаходяться на останніх сторінках.

Ключові слова: CNN, конволюція, ядро, пулінг, структура архітектури, функція активації, зв'язність, рецептивне поле, ваги та зміщення, фільтрація зображень, регуляризація.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ANN – Штучна нейронна мережа

CNN – Згорткова нейронна мережа

ReLU – Випрямлена лінійна

одиниця

ЗМІСТ

ВСТУП

Здатністю адаптуватись до складних структурних залежностей на великих наборах даних виділяються згорткові нейронні мережі. Цим забезпечується вирішення завдань, для яких традиційні підходи не завжди є ефективними, в будь якій сфері діяльності, від розпізнавання певних об’єктів до їх класифікації, важливих відкрить у сфері медицини, космології чи математики. Згорткові нейронні мережі змінюють спосіб розуміння, обробки та використання інформації.

Крім того, сучасні реалії обов'язково потребують розробку та використання технологій розпізнавання. Такі інструменти можуть враховувати деталі кольору, межі, загальні особливості, що сприяє пошуку об'єктів на відео та зображеннях. Також обробляються дані різного типу, наприклад визначення текстової інформації, де головними елементами є: слова , цифри, різні символи тексту.

Мета дипломної роботи полягає в проведенні досліджень існуючих методів та розробці програмного засобу для розпізнавання різних об’єктів.

Загальна мета даного проекту – створення інструменту без використання існуючих засобів для розпізнавання образів на зображеннях, що може бути успішно використаний у зоровому сприйняті. В якості образу взят автомобіль.

Розпізнавання автомобілів – актуальний вибір для прикладу, оскільки машини одні з основних засобів транспорту в наш час. Враховуючи, що промисловість продовжує нарощувати їх кількість відповідно до зростання населення.

На сьогодні така б система була популярною не лише серед правоохоронних органів, але й серед різних систем безпеки компаній, а також серед тих, хто має намір визначити порушників на дорозі або встановити власників автомобільного транспорту. Глобальна інформатизація забезпечує потрібний набір зразків для навчання.

ПОСТАНОВКА ЗАДАЧІ

1. Літературний огляд існуючих підходів до розпізнавання об'єктів на зображеннях: Аналіз та систематизування основних методи та алгоритмів розпізнавання об'єктів, які застосовуються в сучасних дослідженнях та проектах. Визначення переваги та недоліків кожного методу, оцінка їх ефективності.
2. Вибір оптимального підходу для розпізнавання об'єктів на зображеннях: Обрати метод або комбінацію методів, які найкраще відповідають вимогам конкретного завдання з розпізнавання об'єктів.
3. Підбір навчальної вибірки (датасету): створити або обрати наявний датасет, який належним чином відображає різноманітні умови та контексти, з якими може стикатися програмний комплекс. Реалізація програмного комплексу для автоматичного розпізнавання об'єктів на зображеннях:
4. Розробити програмний код, який включатиме в себе обрані алгоритми розпізнавання та забезпечуватиме їхню взаємодію. Забезпечити можливість адаптації та налаштування програмного комплексу для різних завдань та умов використання.
5. . Оцінити результати розробленої моделі; здійснити порівняльний аналіз з існуючими моделями.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Історія

1.1.1 Мережі зі згортковими шарамии та історія глибокого навчання

Мережі зі згортковими шарами відіграли важливу роль в історії глибокого навчання. Вони є ключовим прикладом успішного застосування впроваджень, отриманих вивченням мозку, до застосувань машинного навчання. Також вони були одними з перших глибоких моделей, які успішно виконували завдання задовго до того, як глибокі моделі взагалі вважалися можливими. Мережі зі згортковими шарами також були серед перших нейромереж, які вирішували важливі комерційні завдання і залишаються на передньому краї застосувань глибокого навчання і сьогодні. Наприклад, у 1990-х роках дослідницька група нейромереж у компанії AT&T розробила зі згортковими шарами для читання чеків. Звісно, ця система, розгорнута NCR, читала понад 10 відсотків усіх чеків у Сполучених Штатах до кінця 1990-х років. Пізніше Microsoft розгорнула кілька систем оптичного розпізнавання символів і розпізнавання почерку, заснованих на мережах зі згортковими шарами.

Мережі зі згортковими шарами також використовувались для перемоги в багатьох конкурсах. Сучасний інтерес до глибокого навчання почався, коли мережі перемогли на виклику розпізнавання об'єктів ImageNet, але мережі зі згортковими шарами використовувались для перемоги в інших конкурсах машинного навчання та комп'ютерного зору з меншим впливом задовго до цього.

Мережі зі згортковими шарами були одними з перших працюючих глибоких мереж, навчених з використанням методу зворотного поширення. Не зовсім зрозуміло, чому мережі зі згортковими шарами виявилися успішними, коли вважалося, що загальні мережі зі зворотним поширенням були невдалими. Можливо, це просто тому, що мережі зі згортковими шарами були більш обчислювально ефективними, ніж повністю зв'язані мережі, тому було легше проводити численні експерименти і налаштовувати їхню реалізацію та гіперпараметри. Здається, що більші мережі також легше навчаються. За

сучасного обладнання, великі повністю зв'язані мережі здаються відносно ефективними для багатьох завдань, навіть при використанні наборів даних, які були доступні та функцій активації, які були популярні в період, коли вважалося, що повністю зв'язані мережі працюють не дуже добре. Можливо, основні бар'єри перед успіхом нейромереж були психологічними (практики не очікували, що нейромережі будуть працювати, тому вони не здійснювали серйозних спроб використовувати нейромережі). У будь-якому випадку, щасливо, що мережі зі згортковими шарами працювали добре десятиліття тому. Багатьма способами вони несли факел для решти глибокого навчання і відкрили шлях до прийняття нейромереж взагалі.

Мережі зі згортковими шарами надають можливість спеціалізувати нейромережі для роботи з даними, які мають явну структуру сітки і масштабувати такі моделі до дуже великих розмірів. Цей підхід був найбільш успішним для двовимірної топології зображення. Для обробки одновимірних послідовних даних краще звертатись до іншої потужної спеціалізації рамки нейромереж: рекурентні нейромережі.

1.1.2 Історичні тенденції у глибокому навчанні

Найлегше розуміти глибоке навчання з історичним контекстом. Замість надання детальної історії глибокого навчання, визначаємо кілька ключових тенденцій:

• Глибоке навчання має довгу і багату історію, але проходило під різними назвами, відображаючи різні філософські точки зору, і періодично набувало популярності.

• Глибоке навчання стало більш корисним зі зростанням обсягу доступних навчальних даних.

• Моделі глибокого навчання зростають за розміром з часом, оскільки інфраструктура комп'ютерів (як апаратне, так і програмне забезпечення) для глибокого навчання поліпшується.

• Глибоке навчання вирішує все більш складні завдання з часом зі зростанням точності.

1.2 Методи згортки

Згорткові нейронні мережі використовуються для обробки зображень та відео, і вони включають в себе різні шари, що дозволяють ефективно впоратися з просторовими особливостями даних. Основний компонент згорткових нейронних мереж — це шари згортки та пулінгу. Ось кілька основних методів, які використовуються в згорткових нейронних мережах.

1.2.1 Шар згортки

Згортка: це операція, яка використовує фільтри або ядра для пропуску через вхідне зображення. Фільтри є невеликими матрицями ваг, які переміщуються по всьому зображенню, обчислюючи добуток значень пікселів та відповідних ваг фільтра. Цей процес дозволяє виділити різні локальні особливості, такі як краї, текстури або форми.

Ядро: це саме ця маленька матриця ваг, яка застосовується до різних частин вхідного зображення. Кожне ядро відповідає за визначену ознаку чи деталь у зображенні. Наприклад, одне ядро може реагувати на вертикальні лінії, інше - на горизонтальні, і так далі. Під час навчання нейронної мережі ваги ядер оптимізуються для визначення конкретних ознак у вхідних зображеннях.

Переміщення фільтра полягає в тому, що фільтр, який є малим ядром або матрицею ваг, переміщується по всьому зображенню, піксель за пікселем. Кожне положення фільтра представляє операцію згортки для певного регіону зображення. Цей процес дозволяє визначити, які особливості знаходяться в різних частинах зображення.

Виконання згортки включає обчислення скалярного добутку значень пікселів у регіоні та ваг фільтра для кожного положення фільтра. Це виокремлює локальні особливості, такі як краї, текстура чи інші визначальні характеристики.

Створення карти ознак - результати обчислень формують карту ознак, яка відображає важливі аспекти зображення. Кожен фільтр виокремлює певну ознаку, і карта ознак об'єднує ці визначені аспекти для створення повноцінного репрезентативного зображення.

1.2.2 Шар пулінгу

Шар пулінгу є важливим компонентом згорткових нейронних мереж (CNN), спрямованим на зменшення розмірності зображення та виділення ключових особливостей. Розглянемо операції максимального та середнього пулінгу у більшому контексті.

Максимальне пулінг: операція максимального пулінгу здійснює вибір максимального значення з певного регіону зображення. Під час цього процесу, фільтр чи ядро здійснює переміщення по зображенню та обирає найбільше значення в певному регіоні. Це дозволяє зменшити розмір зображення та виокремити найважливіші особливості, зберігаючи при цьому їх просторовий контекст.

Середнє пулінг: в операції середнього пулінгу обчислюється середнє значення з певного регіону зображення. Аналогічно до максимального пулінгу, фільтр пересувається по зображенню, але тут вибирається середнє значення пікселів. Цей вид пулінгу також служить для зменшення розмірності та збереження основних ознак зображення.

1.2.3 Повторювані блоки

Повторювані блоки є ключовим елементом глибоких нейронних мереж, особливо в згорткових нейронних мережах (CNN), і вони грають важливу роль у вивченні та представленні високорівневих ознак на різних рівнях ієрархії.

Згортка-пулінг: згортково-пулінговий блок представляє собою послідовність згорткового та пулінгового шару, яка використовується для витягування та агрегації ознак з вхідних зображень. Згортковий шар служить для виявлення локальних особливостей, в той час як пулінговий шар допомагає зменшити розмірність і підсумовує важливі ознаки, забезпечуючи просторову інваріантність.

Повторювані блоки: повторювані блоки є стеками згорткових блоків, які дозволяють систематично вивчати та виділяти складні ознаки на різних рівнях ієрархії. Кожен блок може містити не тільки згортково-пулінговий шар, але й інші шари, такі як нормалізації та активації. Повторення цих блоків дозволяє мережі адаптуватися до різноманітних особливостей у вхідних даних та поглиблювати розуміння представленої інформації.

1.2.4 Повністю з’єднані шари

Повністю з'єднаний шар - це шар, в якому кожен нейрон пов'язаний з кожним нейроном попереднього шару. Кожне з'єднання представляє собою вагу, яка регулює вплив вхідної інформації на виходи нейронів. У цьому шарі відбувається комбінування та агрегація важливих ознак, вивчених на попередніх шарах, для подальшого використання у завданнях класифікації, регресії чи інших задачах.

Використання для класифікації: повністю з'єднані шари широко використовуються для завершення процесу та прийняття рішень щодо класифікації об'єктів чи даних. Наприклад, у завданнях розпізнавання образів, останній повністю з'єднаний шар може визначати ймовірності належності об'єкта до певних класів.

Повністю з'єднані шари в контексті глибоких мереж: у глибоких нейронних мережах, повністю з'єднані шари зазвичай використовуються в кінці архітектури після згорткових та пулінгових шарів для фінального виведення класифікаційних результатів чи інших важливих величин

1.2.5 Функції активації

Функції активації в нейронних мережах відповідають за впровадження нелінійності та допомагають мережі вивчати складні взаємозв'язки в даних.

ReLU - функція активації ReLU визначається як $f(x)=max(0,x)$. Вона лінійна і позитивна для додатних значень, а негативні значення обнуляються. ReLU широко використовується в нейронних мережах завдяки його ефективності та спроможності прискорювати навчання.

Сигмоїдна функція - використовується основним чином для завдань бінарної класифікації, де вихід потрібно перетворити в ймовірність належності до певного класу. Вона перетворює будь-яке вхідне значення у діапазон від 0 до 1.

Тангенс гіперболічний (tanh) - також визначається для всього діапазону значень від -1 до 1. Він часто використовується в згорткових шарах для обробки відповідей, оскільки його значення знаходяться у більш широкому діапазоні, порівняно з сигмоїдною функцією.

Кожна з цих функцій активації має свої унікальні властивості та застосування, і їх вибір залежить від конкретного завдання та архітектури мережі.

1.2.6 Шар нормалізації

Батч-нормалізація – це техніка нормалізації активацій шару, що забезпечує стабільність та прискорює процес навчання шляхом нормалізації активацій за батчами даних. Основна ідея полягає в тому, щоб стандартизувати виходи шару для кожного батчу даних шляхом нормалізації до середнього значення 0 і стандартного відхилення 1.

Головні переваги батч-нормалізації включають:

- Стабільність навчання: Зменшення внутрішньої коваріації між шарами допомагає уникнути проблеми зі врожденою числовістю (vanishing/exploding gradients) і дозволяє використовувати вищі швидкості навчання.

- Регулювання градієнтів: Батч-нормалізація допомагає зберігати нормальнізовані градієнти, що сприяє більш ефективному навчанню глибоких мереж.
- Регулювання області активацій: Нормалізація активацій за батчами допомагає уникнути ситуацій, коли активації виявляються в силу обмеженої кількості прикладів у батчі.

1.2.7 Шар дропауту

ропаут є технікою регуляризації, в якій випадково вимикаються деякі нейрони під час навчання з метою запобігання перенавчанню.

У процесі дропауту, на кожному кроці навчання, кожен нейрон має ймовірність бути вимкненим (вибуває з обчислень) з певною ймовірністю, яка зазвичай вибирається перед початком навчання. Це дозволяє мережі навчатися більш узагальнюючим ознакам та уникати надмірного прилаштування до конкретних прикладів навчання.

Головні переваги використання дропауту включають:

- Запобігання перенавчанню: Дропаут допомагає уникнути перенавчання, зокрема, коли навчальний набір є обмеженим або недостатньо різноманітним.
- Збереження універсальності: Вимкнення нейронів під час навчання змушує мережу розпізнавати ознаки без залежності від конкретних нейронів, збільшуючи її універсальність.
- Збільшення стійкості: Дропаут також може підвищити стійкість моделі, оскільки вона навчається працювати з неповними данними.

Дропаут широко використовується в нейронних мережах, особливо в глибоких архітектурах, як ефективний інструмент для забезпечення стійкості та універсальності моделей.

1.2.8 Архітектури

Кілька із важливих архітектур глибоких нейронних мереж, які відіграють ключову роль у розвитку комп'ютерного зору, включають LeNet-5, AlexNet, VGGNet, GoogLeNet (Inception) та ResNet.

- LeNet-5:

Розроблена Яном Лекуном для розпізнавання рукописних цифр.

Структура включає згорткові та пулінгові шари, а також повністю з'єднані шари.

- AlexNet:

Вигадана командою вчених в компанії Google для участі в конкурсі ImageNet.

Сприяла відродженню зацікавлення у глибоких нейронних мережах.

Застосовує згорткові шари, пулінг, нормалізацію та повністю з'єднані шари.

- VGGNet:

Має просту та однорідну структуру з великою кількістю згорткових шарів.

Використовує невеликі фільтри 3x3 у всіх своїх згорткових шарах.

- GoogLeNet (Inception):

Відома своєю архітектурою Inception, що використовує паралельні згорткові шари різного розміру.

Має вражаючу глибину та ефективність.

- ResNet:

Введення блоків зі зрушенням (residual blocks), які мають коротший зв'язок між входом та виходом.

Стало можливим тренувати глибші мережі завдяки зменшенню проблеми зникаючих градієнтів.

РОЗДІЛ 2. МЕТОДИКА ДОСЛІДЖЕНЬ

2.1 Що таке CNNs.

Згорткова нейронна мережа (відома, як convolutional neural network, коротко CNN) – це спеціалізований тип нейронної мережі для обробки даних з відомою сітчатою топологією. Прикладом слугують зображення, які можна уявити як 2-вимірну сітку пікселів, та дані часових рядів, взятих через регулярні проміжки часу. Мережа демонструє великий успіх у практичних застосуваннях. Назва вказує на використання математичної операції згортка. Згортка є спеціалізованим видом математичної операції. Отже, мережі даного типу – це просто нейронні мережі, які використовують згортку замість загального множення матриць принаймні на одному з їхніх шарів.

Дослідження в темі архітектур мережі відбувається так стрімко, що нова найкраща архітектура для конкретного тесту анонсується кожні кілька тижнів чи місяців. Тим не менш є основні, найкращі архітектури для загальних завдань.

Операція конволюції в загальній формі функція над дійсними аргументами. Нехай ми вимірюємо число $s(t)$, з ряду попередніх чисел $x(t)$. Де t - момент часу коли проводиться вимір. Кожне попереднє вимірювання має певну цінність для наступного, виражене через коефіцієнт ваги - $w(t)$. Попередній час визначається символом a. Тоді s виражається через формулу:

$$s(t) = \int x(a)w(t-a)da.$$

Це операція згортки зазвичай позначається зірочкою.

$$s(t) = (x * w)(t)$$

Для термінології згорткових мереж перший аргумент зазвичай називається входом, а другий ядром. Результатом є ознака. У застосуваннях машинного навчання вхід є багатовимірним масивом , а ядро – це зазвичай багатовимірний

масив параметрів, які адаптуються алгоритмом навчання (тензор). Для випадку двовимірного зображення:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Механізм згортки застосований для двовимірного тензора висвітлено на зображені 1.

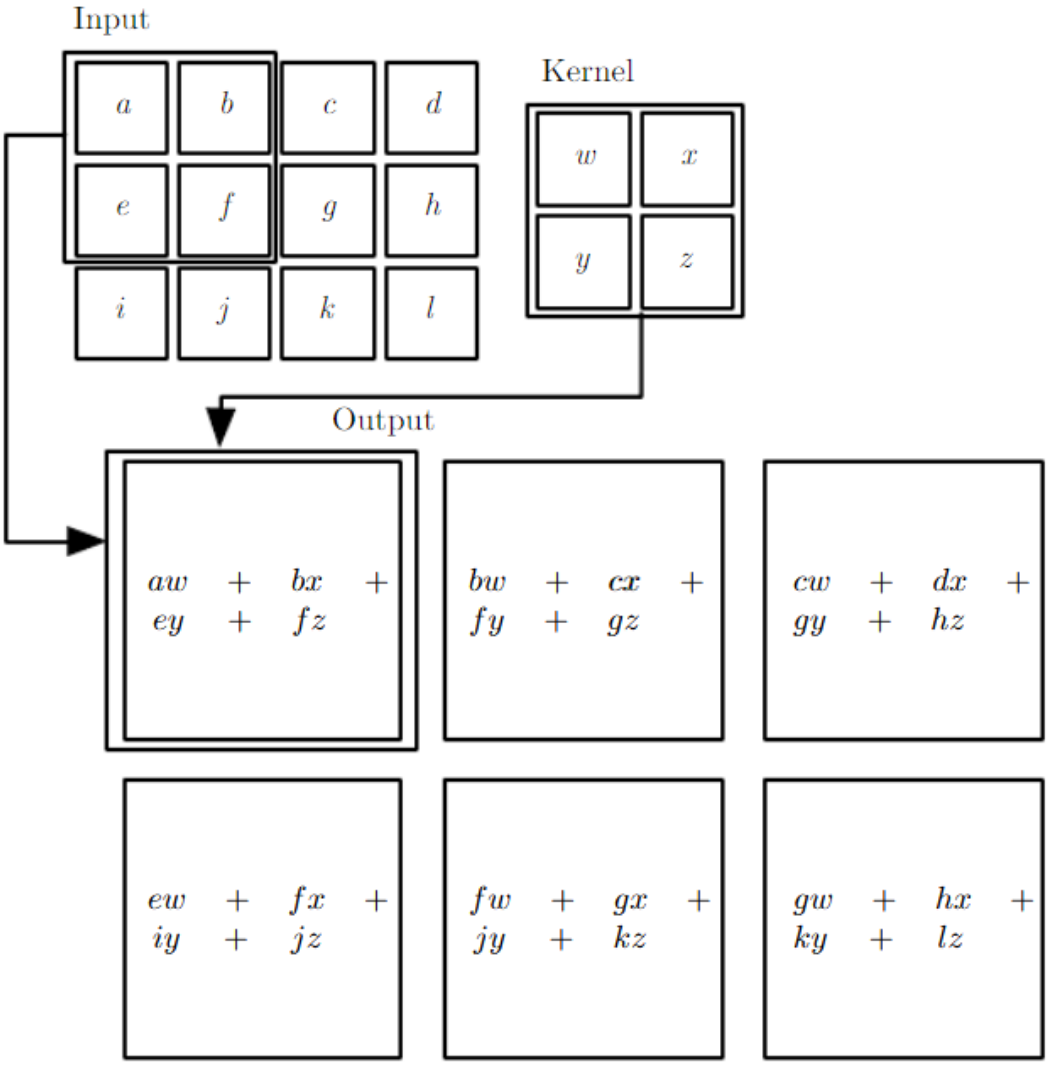


Рисунок 1 Мальовані стрілки та прямокутники показують, як верхній лівий елемент виводу формується за допомогою ядра для відповідної верхньої лівої області вхідного тензора. Наступним кроком ядро зміщюється і операція повторюється.

Згортка є комутативною:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Ця властивість виникає тому, що перевертається ядро відносно входу, у тому сенсі, що зі збільшенням m індекс в вхід зростає, але індекс в ядро зменшується. Тобто, можна отримати однаковий результат, навіть якщо змінити порядок вхідних даних і ядра.

Ця властивість може бути корисної під час деяких застосувань, де краще змінити порядок дій. Однак вона не завжди є доречною і ефективною бо нейромережі зазвичай вчаться адаптуватись до рішення.

2.2 Головні переваги CNNs

Згортка використовує 3 важливі ідеї для покращення системи машинного навчання: *розріджені взаємодії, обмін параметрами та еквіваріантність представлення.*

У традиційних шарах нейронних мереж використовується множення матриць на матрицю параметрів, де окремий параметр описує взаємодію між кожної вхідною одиницею і кожною вихідною одиницею. Кожна вхідна одиниця взаємодіє з кожною вихідною одиницею. У згорткових нейронних мережах використовується *розріджена взаємодія*. Це досягається меншим розміром ядра ніж розміром вхідних даних. Наприклад, при обробці зображення. Вхідне зображення має безліч пікселів, але можна виявити невеликі, значущі особливості, такі як краї, за допомогою ядер, що охоплюють невелику кількість вхідних даних. Для цього потрібно зберігати менше параметрів. Зменшується вимогливість до пам'яті моделі та покращується її статистична ефективність. Також стає менше операцій обчислень. Маючи m входів та n виходів, множення матриць вимагає m$*n$ параметрів. І алгоритм має часову складність$O(m*n)$. Обмеживши

кількість з’єднань для кожного виходу на k, тоді розріджений підхід вимагає лише $k * n$ параметрів і часову складність $O(k * n)$.

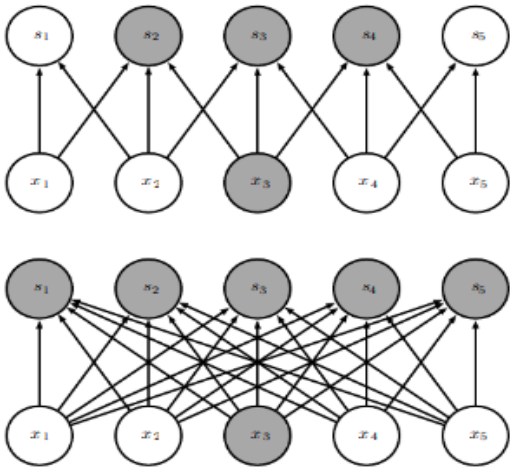


Рисунок 2. Розрідженість розглянуті знизу до гори. Ми виокремлюємо одиницю входу x_3 і одиниці виходу s. (Зверху) s формується за допомогою згорткою ядром ширини 3, лише три виходи зазнають впливу від x_3. (Низ) s формується множенням матриць, тому всі виходи зазнають впливу від x_3.

Для багатьох практичних застосувань можливо досягти високої ефективності в завданнях машинного навчанння, зберігаючи k на кілька порядків меншим ніж m. На рисунках 2 та 3 демонстрована розріджена взаємодія.

На рисунку 4 зображено, як у глибоких шарах параметри входу непрямо взаємодіють з більшістю ознак. Так мережа мережа описує складну взаємодію між багатьма змінними за допомогою простих розріджених частин

Обмін параметрами вказує на використання одного й того ж параметра для більш ніж однієї функції в моделі. У звичайних нейронних мережах кожен елемент мережі використовується лише один раз при обчислені виходу шару. Він множиться на один елемент входу, і потім використовується повторно. В згортковій мережі, по іншому можна сказати, використовується зв’язні ваги

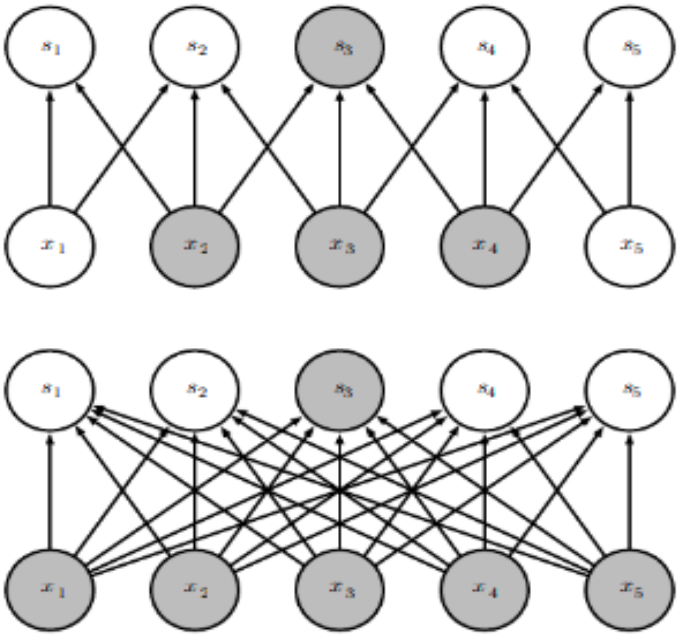


Рисунок 3. Виділені одиниця виходу s_3 і вхідна одиниця x_3 .(Зверху) Коли s_3 формується згорткою з ядром ширини 3 то лише три входи впливають на нього. (Низ) Коли s_3 формується множенням матриць, взаємодія не є розрідженою, тому всі входи x впливають на s_3.

. Значення вагів, що використовується для одного входу, використовується і для інших входів. У згортковій нейронній мережі кожен елемент ядра використовується на кожній позиції вхідних даних (крім, можливо, деяких пікселів на кордоні, залежно від рішень щодо конструкції кордону). Використання поділу параметрів у операції згортки означає, що замість вивчання окремого набору параметрів для кожного місця, ми вивчаємо лише один набір. Це не впливає на час попереднього поширення – він ще становить $O(k * n)$ – але ще більш зменшує вимоги до зберігання моделі до k параметрів. Оскільки n та m приблизно однакові, k практично не помітно в порівнянні з $m * n$. Тому згортка ефективніше за множення матриць з точки зору пам'яті та статистичної ефективності. Графічно зображено поділ параметрів на рисунку 5. На рисунку 6 показано, як розріджена взаємодія та поділ параметрів можуть ефективно покращити лінійну функцію для виявлення країв на зображені.

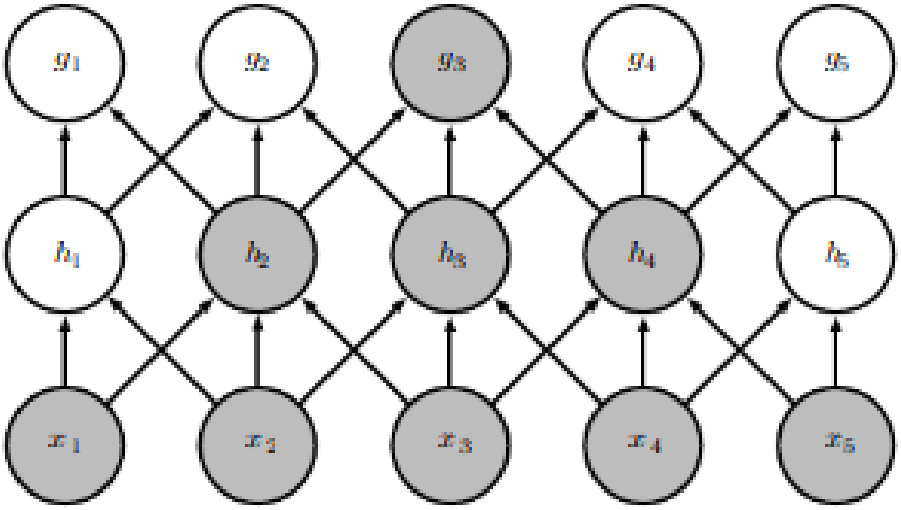


Рисунок 4. Рецептивні поля одиниць у глибоких шарах згорткової мережі є більшими, ніж рецептивні поля одиниць у поверхневих шарах. Це означає, що навіть якщо прямі з'єднання в мережі є дуже розрідженими, одиниці в глибших шарах можуть бути непрямо з'єднані з усіма або більшістю вхідного зображення.

У випадку згортки конкретна форма поділу параметрів призводить до того, що шар має властивість, яку називають *еквіваріантністю до зсуву*. Сказати, що функція є еквіваріантною, означає, що якщо вхід змінюється, вихід змінюється так само. Зокрема, функція $f(x)$ є еквіваріантною до функції $g(x)$, якщо $f\big(g(x)\big) = g(f(x))$. У випадку згортки, якщо дозволити g бути будь-якою функцією, яка транслює вхід, тобто зсуває його, то функція згортки є еквіваріантною до g. Наприклад, нехай I - функція, що визначає яскравість зображення при цілих координатах. Нехай g - функція, яка відображає одну функцію зображення в іншу функцію зображення, таку що $I' = g(I)$ - це функція зображення з$I'(x, y) = I(x - 1, y)$. Це зсуває кожен піксель I на одиницю вправо. Якщо застосувати це перетворення до I, а потім застосувати згортку, результат буде таким самим, якщо застосувати згортку до I', а потім застосувати перетворення g до виходу.

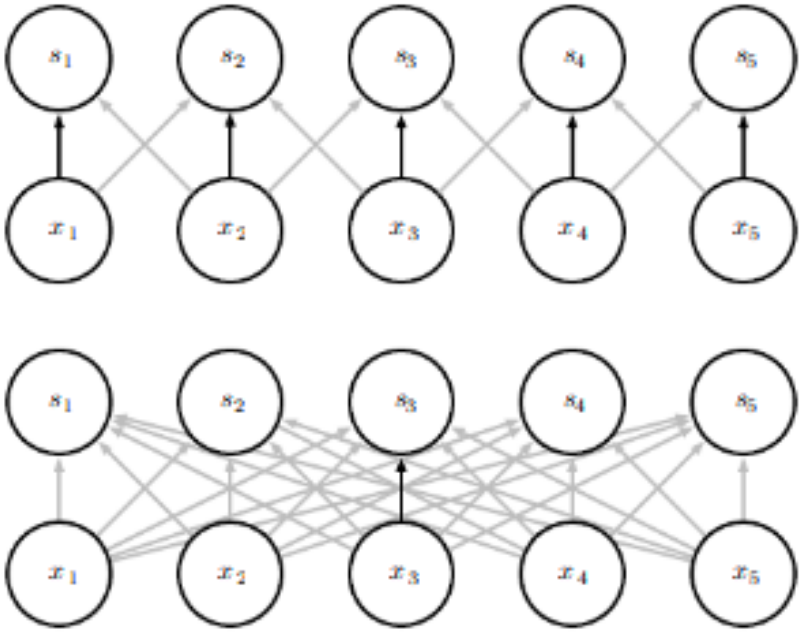


Рисунок 5. Чорні стрілки показують зв'язки, які використовують певний параметр у двох різних моделях. (Верх) Чорні стрілки показують використання центрального елемента трьохелементного ядра в конволюційній моделі. Завдяки поділу параметрів цей один параметр використовується у всіх точках входу. (Низ) Одна чорна стрілка показує використання центрального елемента матриці ваг в моделі з повною зв'язністю. В цій моделі відсутній поділ параметрів, тому параметр використовується лише один раз.

При обробці часових рядів це означає, що згортка створює свого роду лінію часу, яка показує, коли різні особливості з'являються на вході. Якщо пересунути подію пізніше в часі на вході, точно таке саме представлення з'явиться на виході, просто пізніше. Так само із зображеннями: згортка створює 2D-карту того, де входять певні особливості. Якщо перемістити об'єкт на вході, його представлення теж переміститься на виході. Це корисно, коли знати, що деяка функція від невеликої кількості сусідніх пікселів є корисною, коли її застосовують до кількох місць входу. Наприклад, при обробці зображень корисно виявляти краї на першому шарі згорткової мережі. Ті самі краї з'являються більш-менш у всьому зображенні, тому практично ділити параметри по всьому зображенню.

Рисунок 6. Зображення справа сформувалось шляхом віднімання значення кожного пікселя в оригінальному зображені від значення його сусіда зліва. Це відображає силу всіх вертикально орієнтованих країв на вхідному зображені, що може бути корисною операцією для виявлення об’єктів. Обидва зображення мають висоту 280 пікселів, а ширину 320. А ширина вихідного зображення - 319 пікселів. Цю трансформацію можна описати з ядром згортки, яке містить два елементи, і для її обчислення за допомогою згортки потрібно 319×280×3 = 267,960 операцій з плаваючою комою (два множення та одне додавання на кожен вихідний піксель). Для опису тієї самої трансформації за допомогою множення матриць потрібно б було матрицю розміром 320×280×319×280, або понад вісім мільярдів, елементів, зробивши згортку в чотири мільярди разів ефективнішою для представлення цієї трансформації. Алгоритм простого множення матриць виконує понад шістнадцять мільярдів операцій з плаваючою комою, роблячи згортку приблизно 60,000 разів більш ефективною обчислювально. Звісно, більшість елементів матриці будуть нульовими. Якщо б зберігати лише ненульові елементи матриці, то як для множення матриць, так і для згортки потрібно було б таку ж кількість операцій з плаваючою комою для обчислення. Матриця все ще повинна була містити 2×319×280 = 178,640 елементів.

У деяких випадках може бути бажано не ділити параметри по всьому зображенню. Наприклад, якщо обробляти зображення, яке обрізане так, щоб бути вирівняним з обличчям людини, можливо, потрібно виділити різні особливості в різних місцях - частина мережі, яка обробляє верх голови, повинна шукати брові, тоді як частина мережі, яка обробляє нижню частину обличчя, повинна шукати підборіддя.

Згортка не є природно еквіваріантною до деяких інших перетворень, таких як зміна масштабу чи обертання зображення. Для обробки цих видів трансформацій необхідні інші механізми.

2.3 Об’єднання

Типовий шар згорткової мережі складається з трьох етапів (рисунок 7). На першому етапі шар виконує кілька згорток паралельно для створення набору лінійних активацій. На другому етапі кожну лінійну активацію подають через нелінійну функцію активації. Цей етап іноді називають етапом виявлення. На третьому етапі використовуємо функцію об’єднання для додаткової модифікації виходу шару.

Функція об’єднання замінює вихід мережі в певному місці на певне значення статистики навколишніх виходів. Наприклад, операція максимального об’єднання повідомляє максимальний вихід у прямокутному околі. Інші популярні функції об’єднання включають середнє значення у прямокутному околі, L2-норму у прямокутному околі або зважене середнє значення на основі відстані від центрального пікселя.

У всіх випадках об'єднання допомагає наблизити представлення приблизно до незначних зсувів вхідних даних. Інваріантність до зсуву означає, що при невеликих зміщеннях вхідних даних значення більшості об'єднаних виходів залишаються незмінними (рисунок 8). Інваріантність до локального зсуву може бути корисною властивістю, якщо важливо не стільки точне місце розташування якоїсь ознаки, скільки факт її наявності. Наприклад, визначаючи, чи є обличчя на зображенні, не потрібно знати точне розташування очей з піксельною точністю, просто потрібно знати, що на лівому боці обличчя є око, а на правому боці - інше око. В інших випадках важливіше зберігати місцезнаходження ознаки.

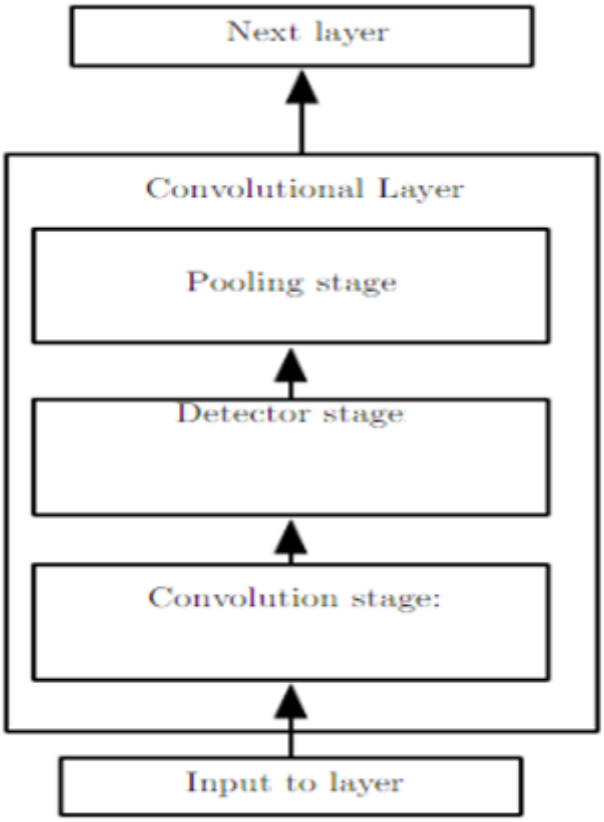


Рисунок 7. Компоненти типового шару згорткової нейронної мережі.

Використання об'єднання можна розглядати як додавання нескінченно сильного апріорного припущення, що функція, яку вивчає шар, повинна бути інваріантною до невеликих зсувів. Коли це припущення вірне, воно може значно покращити статистичну ефективність мережі.

Об'єднання по просторовим областям забезпечує інваріантність до зсуву, але якщо об'єднати виходи окремо параметризованих згорток, ознаки можуть вивчити, до яких перетворень стати інваріантними (рисунок 9).

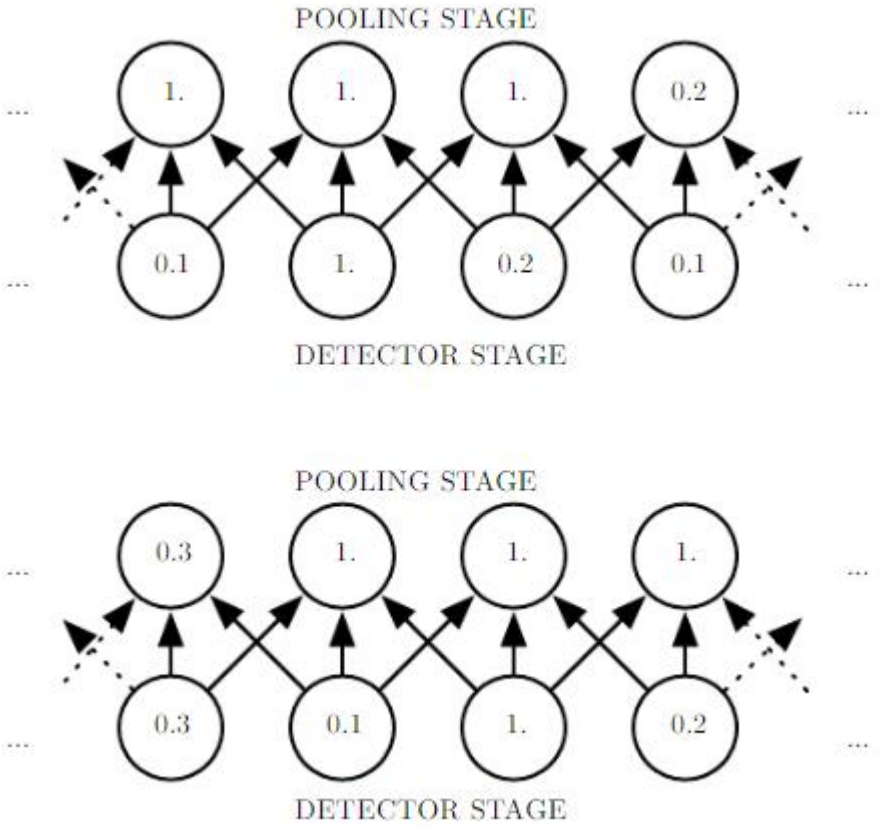


Рисунок 8. Приклад як максимальне об'єднання вводить інваріантність. (Верх) Дивлячись на середину виходу згорткового шару. Нижній рядок показує виходи нелинійності. Верхній рядок показує виходи максимального об'єднання з кроком одного пікселя між областями об'єднання та шириною області об'єднання три пікселя. (Низ) Дивлячись на ту саму мережу після того, як вхід було зсунуто вправо на один піксель. Кожне значення в нижньому рядку змінилося, але тільки половина значень в верхньому рядку змінилося, оскільки блоки максимального об'єднання реагують лише на максимальне значення в околиці, а не на його точне розташування.

Тому що об'єднання узагальнює відповіді по всьому околиці, можна використовувати менше об'єднаних блоків, ніж блоків виявлення, представляючи узагальнені статистичні дані для об'єднаних областей з розміром k пікселей замість 1 пікселя (рисунок 10).

Це покращує обчислювальну ефективність мережі, оскільки у наступному шарі приблизно на k разів менше входів для обробки. Коли кількість параметрів у наступному шарі залежить від його розміру вводу (наприклад, якщо наступний шар є повністю з'єднаним та ґрунтується на множенні матриць), це скорочення розміру вводу також може призвести до покращеної статистичної ефективності та зменшених вимог щодо обсягу пам'яті для зберігання параметрів.

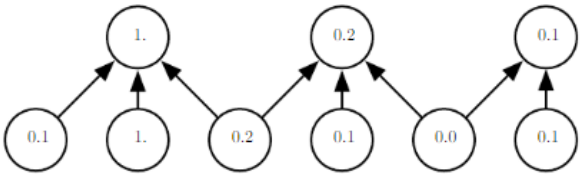


Рисунок 10. Об'єднання зі зменшенням розміру. Тут використовується максимальне об'єднання з шириною об'єднання три і кроком між об'єднаними областями два. Це зменшує розмір представлення вдвічі, зменшуючи обчислювальне та статистичне навантаження на наступний шар. Зазначте, що права область об'єднання має менший розмір, але його слід включити, якщо ігнорувати деякі блоки детекторів.

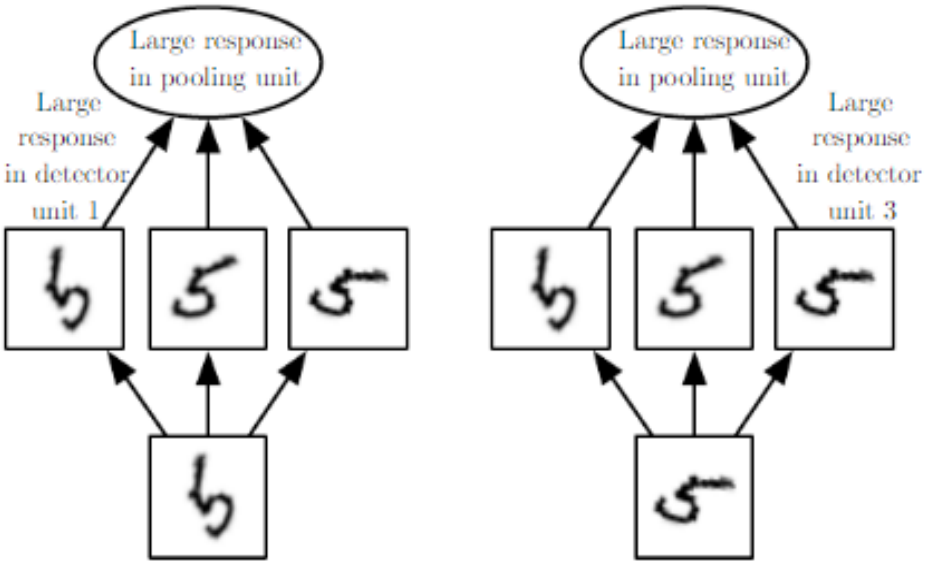


Рисунок 9. Приклад вивчених інваріантностей. Блок об'єднання, який об'єднує кілька ознак, вивчених окремими параметрами, може навчитися бути інваріантним до перетворень вхідних даних. Тут показано, як набір з трьох вивчених фільтрів та блок максимального об'єднання може навчитися стати інваріантним до обертання. Усі три фільтри призначені для виявлення написаної вручну цифри 5. Кожен фільтр намагається визначити трохи відмінну орієнтацію цифри 5. Коли 5 з'являється на вході, відповідний фільтр відповідатиме йому і спричинить великий активатор в блоку виявлення. Потім блок максимального об'єднання матиме великий активатор незалежно від того, який блок виявлення був активований. Тут показано, як мережа обробляє два різні входи, що призводить до активації двох різних блоків виявлення. Ефект на блок об'єднання приблизно однаковий в обох випадках. Цей принцип використовується в мережах типу "maxout" (Гудфеллоу та ін., 2013) та інших згорткових мережах. Максимальне об'єднання по просторових позиціях природно є інваріантним до трансляції; цей багатоканальний підхід є необхідним лише для вивчення інших перетворень.

2.4 Згортка та об’єднання як апріорний розподіл ймовірності

Апріорний розподіл ймовірності – це такий ймовірнісний розподіл параметрів моделі, який відображає переконання щодо того, які моделі є обґрунтованими, до того, як побачили будь-які дані. Апріорний розподіл можна вважати слабкими або сильними в залежності від того, наскільки концентрована щільність ймовірності в апріорі. Слабкий апріор - це розподіл з високою ентропією, такий як гаусівський розподіл з високою дисперсією. Такий апріор дозволяє даним рухати параметри більш-менш вільно. Сильний апріор має дуже низьку ентропію, такий як гаусівський розподіл з низькою дисперсією. Такий апріор відіграє більш активну роль у визначенні того, де закінчяться параметри.

Безмежно сильний апріор ставить нульову ймовірність на деякі параметри та каже, що ці значення параметрів абсолютно заборонені, незалежно від того, наскільки сильно підтримують ці значення дані. Можна уявити згорткову мережу як аналог повністю з'єднаної мережі, але з безмежно сильним апріорним розподілом на її ваги. Цей апріор каже, що ваги для одиниці прихованого шару повинні бути ідентичними вагам її сусіда, але зміщеними в просторі. Апріор також каже, що ваги повинні бути нульовими, крім малої, просторово контингентної області, призначеної для цієї одиниці. В цілому можна розглядати використання згортки як введення безмежно сильного апріорного розподілу ймовірності для параметрів шару. Цей апріор говорить, що функція, яку повинен вивчити шар, містить лише локальні взаємодії і еквіваріантна до трансляції. Так само використання об'єднання є безмежно сильним апріором на те, що кожна одиниця повинна бути інваріантною до невеликих трансляцій.

2.5 Функції реалізації та особливості

Коли мова йде про згортку в контексті нейромереж, зазвичай мається на увазі операція, що складається з багатьох застосувань згортки паралельно. Це пояснюється тим, що згортка з одним ядром може виділити лише один вид ознак,

хоча й в багатьох точках простору. Зазвичай важливо, щоб кожен шар нейромережі виділяв різноманітні ознаки в багатьох місцях.

Крім того, вхід переважно не представлений просто сіткою реальних значень, а сіткою вектор-значень. Наприклад, кольорове зображення включає інтенсивність червоного, зеленого та синього кольорів для кожного пікселя. У багатошаровій згортковій мережі вхід для другого шару - це вихід першого шару, який зазвичай відображає вихід багатьох різних згорток в кожній позиції.

Працюючи з зображеннями, розглядається вхід і вихід згортки як 3D тензори, з одним індексом для різних каналів і двома індексами для просторових координат кожного каналу. Реалізації програм відбувається у режимі пакету, тому фактично використовують 4D тензори, з четвертим виміром для різних прикладів у пакеті. У описі не враховується пакет, для спрощення.

Оскільки згорткові мережі використовують багатоканальну згортку, лінійні операції, на яких вони базуються, не гарантовано комутативність, навіть якщо використовується відображення ядра. Ці операції багатоканальні комутативні лише у випадку, якщо кожна операція має таку саму кількість вихідних каналів, що і вхідних каналів.

Припустимо, що існує 4D тензор ядра K з елементами $K_{i,l,m,n}$, які визначають зв'язок між одиницею в каналі i виходу та одиницею в каналі j вхідного шару, з зміщенням k рядків та l стовпців між виходом та входом. За умови, що вхід складається зі спостережень даних V з елементами $V_{i,j,k}$, що визначають значення вхідної одиниці в каналі і на рядку j та стовпці k, і що вихід має той же формат, що і V, маємо:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

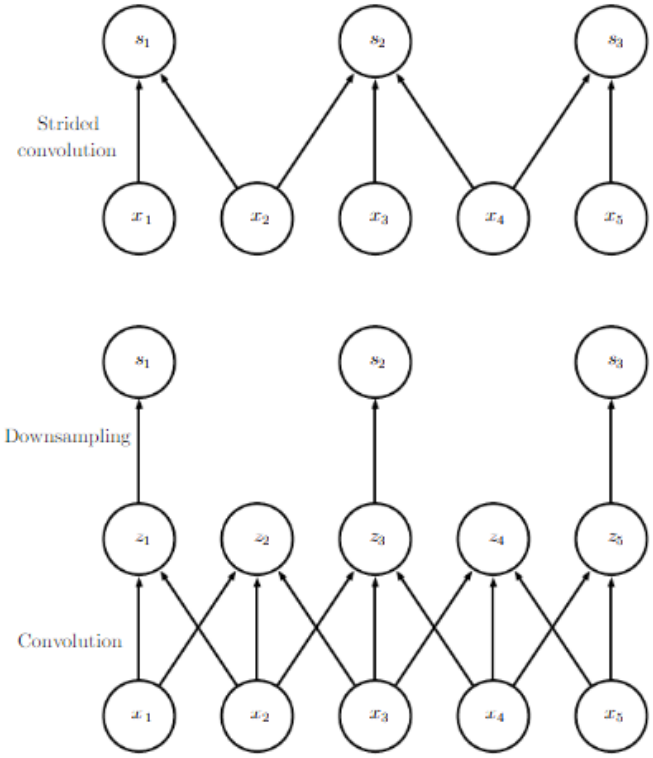


Рисунок 11. Згортка з кроком. У цьому прикладі використовується крок рівний двом. (Верх) Згортка із кроком довжиною два реалізована в одній операції. (Низ) Згортка із кроком більше ніж один піксель математично еквівалентна згортці з одиничним кроком, за якою йде пониження роздільності. Очевидно, що двоетапний підхід із зменшенням роздільності обчислювально неефективний, оскільки обчислює багато значень, які потім відкидаються.

Можливо пропустити деякі позиції ядра для скорочення обчислювальних витрат, досягаючи менш точного виділення ознак. Це можна розглядати як зниження частоти виводу повної функції згортки. Якщо потрібно обирати лише кожен s-ий піксель у кожному напрямку виводу, визначається функція згортки c з вибірковим зменшенням:

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,n} V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n}$$

Також можна встановлювати окремий крок для кожного напрямку руху; для ілюстрації рисунок 11.

Одна з важливих особливостей будь-якої реалізації згорткової мережі - це можливість неявного нульового доповнення вхідних даних V для розширення їх ширини. Без цієї можливості ширина представлення зменшується на один піксель менше, ніж ширина ядра на кожному рівні. Нульове доповнення вхідних даних дозволяє незалежно контролювати ширину ядра та розмір виводу. Без нульового доповнення треба вибирати між швидким зменшенням просторового обсягу мережі та використанням невеликих ядер — обидві ситуації суттєво обмежують виразність мережі. Для прикладу:

Є три основні випадки нульового доповнення:

1) Один із них - екстремальний випадок, коли взагалі не використовується нульове доповнення, і ядро згортки може відвідувати лише ті позиції, де цілком міститься весь обсяг ядра всередині зображення. Це називається "valid convolution". У цьому випадку всі пікселі виводу є функцією того самого числа пікселів у вході, тому поведінка пікселя виводу трошки більше регулярна. Однак розмір виводу скорочується на кожному рівні. Якщо ширина вхідного зображення - m, а ширина ядра - k, то вивід буде шириною $m - k + 1$. Темп цього скорочення може бути драматичним, якщо використовуються великі ядра. Оскільки скорочення більше 0, воно обмежує кількість включених згорткових шарів в мережу. З появою додаткових шарів просторові розміри мережі в кінцевому підсумку зменшаться до 1×1, на якому етапі додаткові шари вже не можуть розглядатися як згорткові з мовного точки зору.

2) Ще один спеціальний випадок налаштування нульового доповнення виникає, коли додається рівно стільки нулів, щоб розмір виводу залишався рівним розміру вхідних даних. У цьому випадку це називається "однаковою згорткою". У цьому випадку мережа може містити стільки згорткових шарів, скільки дозволяє обладнання, оскільки операція згортки не змінює архітектурних можливостей, доступних наступному шару. Проте пікселі введення біля краю впливають на менше вивідних пікселів, ніж пікселі введення біля центру. Це може зробити пікселі краю трошки менш представленими в моделі.

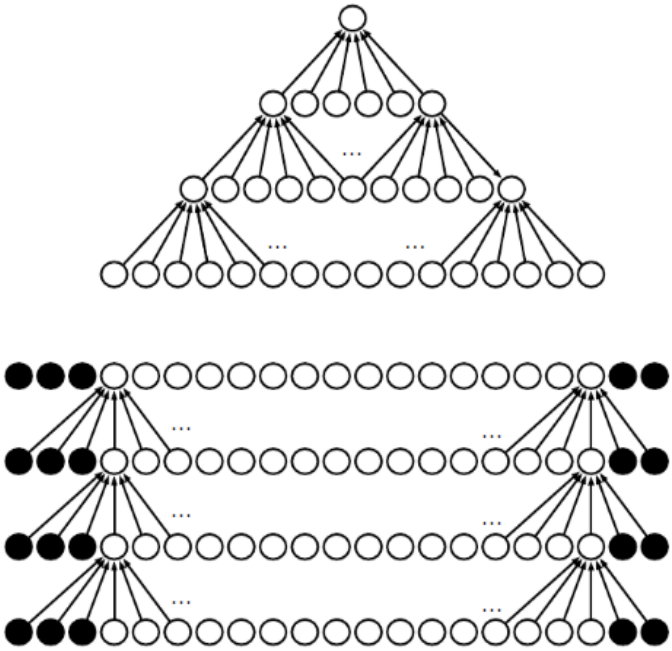


Рисунок 12. Вплив нульового доповнення на розмір мережі. Мережа з ядром шириною шість на кожному рівні. У цьому прикладі не використовуємо жодного об'єднання, тому лише операція згортки сама по собі зменшує розмір мережі. (Верх) У цій згортковій мережі не використовується неявного нульового доповнення. Це призводить до зменшення представлення на п'ять пікселів на кожному рівні. Починаючи зі введення ширини в шістнадцять пікселів, можемо мати лише три рівні згорток, і останній рівень ніколи не переміщує ядро, тому можна стверджувати, що справжньо згортковими є лише два рівні. Скорочення ширини може бути пом'якшене за допомогою менших ядер, але менші ядра менше виразні, і деяке скорочення невід'ємно для цього типу архітектури. (Низ) Додаючи по п'ять неявних нулів до кожного рівня, запобігається зменшенню представлення з глибиною. Це дозволяє нам створювати довільно глибоку згорткову мережу.

3) Останній випадок - "full convolution", в якому додається достатньо нулів для того, щоб кожен піксель відвідувався k разів у кожному напрямку, що призводить до виводу з шириною $m - k + 1$. У цьому випадку вивідні пікселі біля краю є функцією менше пікселів, ніж вивідні пікселі біля центру. Це може зробити

складним вивчення одного ядра, яке добре працює у всіх позиціях у карті згорткових ознак. Зазвичай оптимальна кількість нульового доповнення знаходиться десь між "valid" та "same" згортками.

“Unshared convolution”

У деяких випадках фактично не потрібно використовувати згортку, а хочеться використовувати замість цього локально з'єднані шари. У цьому випадку матриця суміжності в графі нашого MLP залишається такою ж, але кожне з'єднання має свою власну вагу, визначену 6-вимірним тензором W. Індекси в W відповідають: i - вихідний канал; j - вихідний рядок; k - вихідна колонка; l - вхідний канал; m - зсув у рядку вхідних даних; та n - зсув у колонці вхідних даних. Лінійна частина локально з'єднаного шару тоді визначається наступним чином:

“Tiled convolutional”

Пропонує компроміс між згортковим шаром та локально з'єднаним шаром. Замість навчання окремого набору ваг на кожному просторовому місці, вивчається набір ядер, які повертаються при русі простором. Це означає, що негайно сусідні місця матимуть різні фільтри, як у локально з'єднаному шарі, а обсяг пам'яті для зберігання параметрів збільшиться лише в кілька разів від розміру цього набору ядер, а не від розміру всієї карти ознак виводу.

$$Z_{i,j,k} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} w_{i,l,m,n}]$$

$$Z_{i,j,k} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1}],$$

де %- це операція залишку від ділення, з $\mathrm{t}\,\%\,\mathrm{t} = 0, (\mathrm{t}+1)\,\%\,\mathrm{t} = 1$ і так далі.

Три операції - згортка, зворотне поширення від виводу до ваг та зворотне поширення від виводу до входів - достатні для обчислення всіх градієнтів, необхідних для навчання будь-якої глибини прямого зворотнього зв'язку у згортковій мережі. Зосередимось на двох останніх.

Припустимо, що навчаємо згорткову мережу, яка включає згортку з ядром K, застосовану до багатоканального зображення V з кроками, визначеними як $c(K,V,s)$. Припустимо, що мінімізуємо деяку функцію втрат $J(V,K)$. Під час прямого поширення потрібно використовувати c саме для виведення Z, який потім передається через решту мережі і використовується для обчислення функції витрат J. Під час зворотного поширення отримаємо тензор G такий, що:

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(V,K)$$

Для навчання мережі потрібно обчислювати похідні відносно ваг в ядрі. Для використовуємо функцію:

$$g(G,V,s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(V,K) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}$$

Якщо цей шар не є нижнім шаром мережі, нам потрібно обчислити градієнт відносно V, щоб передати помилку далі вниз. Для цього можна використовувати:

$$h(K,G,s)_{i,j,k} = \frac{\partial}{\partial V_{i,j,k}} J(V,K) = \sum_{l,m((t-1)\times s+m=j)} \sum_{n,p((n-1)\times s+p=k)} \sum_{q} K_{q,i,m,p} G_{q,l,n}$$

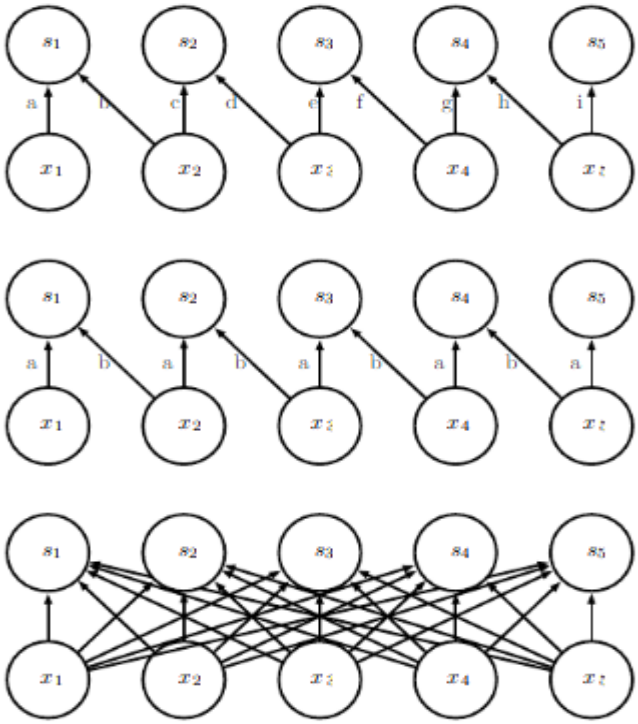


Рисунок 13. Порівняння локальних з'єднань, згортки та повних з'єднань. (Верх) Локально з'єднаний шар із розміром два пікселі. Кожна ребро позначена унікальною літерою, щоб показати, що кожне ребро пов'язано з власним параметром ваги. (Центр) Згортковий шар із шириною ядра два пікселі. У цій моделі маємо точно таке ж з'єднання, як і в локально з'єднаному шарі. Відмінність полягає не в тому, які одиниці взаємодіють між собою, а в тому, як використовуються загальні параметри. У локально з'єднаному шарі відсутнє спільне використання параметрів. Згортковий шар використовує однакові два вагові коефіцієнти повторно по всьому входу, як вказано повторенням літер, що позначають кожне ребро. (Низ) Повністю з'єднаний шар схожий на локально з'єднаний шар в тому сенсі, що кожне ребро має свій власний параметр. Проте він не має обмеженої зв'язності локально з'єднаного шару.

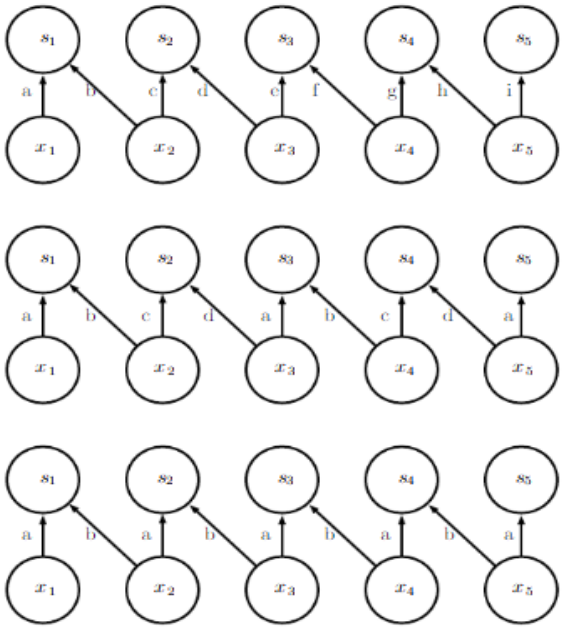


Рисунок 14. Порівняння локально з'єднаних шарів, “Tiled” згортки та стандартної згортки. Усі три мають однакові набори з'єднань між одиницями, коли використовується ядро одного розміру. Ця діаграма ілюструє використання ядра, яке складається з двох пікселів завширшки. Різниця між методами полягає в тому, як вони діляться параметрами. (Верх) Локально з'єднаний шар не має жодного спільного використання. Позначено, що кожне з'єднання має свою вагу, позначивши кожне з'єднання унікальною літерою. (Центр) “Tiled” згортка має набір t різних ядер. Ілюструється випадок $t = 2$. Одне з цих ядер має краї, позначені " a" і " b ", тоді як інше має краї, позначені " c " і " d ". Кожен раз, коли рухаємося на один піксель вправо на виході, переходимо до використання іншого ядра. Це означає, що, подібно до локально з'єднаного шару, сусідні одиниці на виході мають різні параметри. На відміну від локально з'єднаного шару, після того, як використовуємо всі доступні ядра, повертаємося до першого ядра. Якщо дві вихідні одиниці розділені кратними значеннями кроку t, то вони спільно використовують параметри. (Низ) Традиційна згортка еквівалентна “Tiled” згортці з $t = 1$. Є лише одне ядро, і його застосовують всюди, як вказано на діаграмі з вагами " a " і " b " повсюди.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОГО ЕКСПЕРИМЕНТУ

У ході виконання цієї роботи, я крок за кроком досліджував процес роботи згорткових нейронних мереж. Розробляв та реалізував згорткову нейронну мережу з використанням деяких ключових формул та принципів. Від початку до кінця процесу розробки, кожен етап був врівноважено розглянутий, і я дбав про деталі та правильне використання формул.

Почавши розробку згорткової нейронної мережі, моїм вихідним кроком було визначення структури класу *Model*. Я створив конструктор, який приймає розміри вхідного шару, список розмірів прихованих шарів і розмір вихідного шару. Це дозволило мені гнучко налаштувати архітектуру мережі відповідно до завдання.

Ініціалізація ваг та ознак для кожного шару була виконана за допомогою генерації випадкових значень. Після цього я визначив методи для додавання прихованих шарів *(add_layers),* встановлення параметрів прихованих шарів *(set_hidden_layer_params),* і параметрів вихідного шару *(set_output_layer_params).*

Важливим етапом було здійснення тренування, яке я реалізував у методі *train.* Використовуючи прямий та зворотній прохід, а також градієнтний спуск, мережа навчалася на вхідних та вихідних даних. Додав критерій зупинки, який перевіряє точність та зупиняє тренування, якщо досягнута висока точність.

Для зручності введено метод *predict*, що дозволяє використовувати навчену модель для здійснення прогнозів на нових вхідних даних.

3.1 Перелік використаних методів:

1. *Згорткова операція:*

Використовувався *ReLU* для обчислення значень шарів (S) згорткового шару. Формула для згорткової операції була використана для кожного пікселя шару.

2. *Функція втрат:*

Застосовувалася *L1* регуляризація для оцінки відмінності між прогнозованим і реальним значеннями ознак.

3. *Оновлення ваг та ознак:*

Використовувалися градієнти та метод градієнтного спуску для оновлення ваг та ознак.

4. *Функції відношення та їх оновлення:*

Були визначені та використані функції відношення (G, g, μ) для обчислення та оновлення відношень між різними частинами мережі.

Ці формули були використані для забезпечення оптимального навчання та адаптації моделі до набору даних. Робота включала в себе не лише реалізацію формул, а й їх інтерпретацію в контексті конкретної задачі згорткового нейронного мережування.

Методично керуючись кожним кроком і використовуючи вказані формули, я забезпечив високу ефективність та точність моєї згорткової нейронноїмережі.

Цей код став відмінною можливістю вивчити не тільки синтаксис та основ

ні концепції програмування на Python, але і глибше розібратися в роботі ЗНМ. Він представляє собою зручний стартовий пункт для розвитку навичок у сфері машинного навчання та нейронних мереж, дозволяючи вам відчути силу та гнучкість цих технологій.

У подальших розділах коду ви знайдете деталі та коментарі, які пояснюють кожну частину коду. Не соромтеся експериментувати, змінювати параметри та додавати нові функції для власних потреб. Бажаємо успішного програмування та вивчення світу машинного навчання!

3.2 Основні використані формули :

- $\boldsymbol{\delta(S_{i,j,k} - S_{pred}) \geq \varepsilon}$

Ця формула представляє умову великості відхилення (різниці) між реальним значенням (S_(i,j,k)) та прогнозованим значенням (S_pred). Умова визначає, що це відхилення повинно бути не менше за певне порогове значення ε.

Основна ідея полягає в тому, що ми встановлюємо критерій для точності або прийнятності прогнозів нашої моделі. Якщо відхилення між реальним і прогнозованим значенням більше або дорівнює ε, то це може вказувати на те, що модель не дуже ефективна або потребує додаткового навчання.

Деякі ключові компоненти формули:

1. δ: Символ для позначення різниці або відхилення між двома значеннями.
2. S_(i,j,k): Реальне значення на шарі i, рядку j та колонці k.
3. S_pred: Прогнозоване значення.

Отже, умова δ(S_(i,j,k) - S_pred) ≥ ε визначає вимогу до моделі забезпечити, щоб її прогнози були достатньо точними, і відхилення від реальних значень були не більше, ніж ε.

- $\boldsymbol{S_{i,j,k} = ReLU(\sum_{m,n} S_{i,j+m-1,k+n-1} K_{i-1,m,n})}$

Ця формула визначає значення одного елемента на шарі згорткового шару в нейронній мережі.

1. S_(i, j, k): Значення на шарі i, рядку j та колонці k згорткового шару.
2. *ReLU:* Функція активації ReLU (Rectified Linear Unit), яка застосовується до результату внутрішнього сумування. Формула ReLU виглядає наступним чином: ReLU(x) = max(0, x), де x - вхідна величина.
3. ∑_(m,n): Сумування по всіх можливих значеннях m і n.

4. S_(i, j+m-1, k+n-1): Значення на попередньому шарі згорткової нейронної мережі в певній позиції, що визначається m і n.

5. K_(i-1, m, n): Вага (ядро) для відповідної позиції на попередньому шарі.

Отже, S_(i, j, k) обчислюється шляхом згорткової операції для конкретного пікселя на поточному шарі. Сумування виконується по всіх позиціях на попередньому шарі, де враховуються значення та ваги. Результат потім піддається функції активації ReLU.

- $J(\boldsymbol{S_i}, \boldsymbol{K_{i-1}}) = \ \boldsymbol{L1 = S_{true-(}S_i + K_i^2)}$

Ця формула визначає функцію втрат для згорткового шару в нейронній мережі з використанням L1 регуляризації.

- $\boldsymbol{K_i^2 = \gamma \sum K_{i,j,k}^2}$

Отже, K_i^2 визначається як сума квадратів ваг згорткового шару, помножена на коефіцієнт регуляризації γ. Це вводить штраф за великі значення ваг та допомагає уникнути перенавчання, зберігаючи параметри моделі в межах норми.

- $\boldsymbol{G_{i,j,k} = \frac{\partial}{\partial S_i} J(S_i, K_{i-1})}$

Представляє градієнт функції втрат J(S_i, K_(i-1)) відносно значень на згортковому шарі. Це важливий елемент для здійснення зворотного поширення помилок та оновлення параметрів мережі під час навчання

- $\boldsymbol{g(G, S_{i-1}) = \frac{\partial}{\partial K_{i,j,k,l}} J(S_i, K_{i-1}) = \sum_{m,n} G_{i,m,n} S_{i-1,m,n}}$

Представляє градієнт функції втрат J(S_i, K_(i-1)) відносно ваг (ядер) на попередньому шарі. Цей градієнт використовується для оновлення параметрів моделі під час зворотного поширення помилок.

- $\boldsymbol{\mu\,(K,G_{i+1}) = \frac{\partial}{\partial S_{i-1}} J(S_i,K_{i-1}) = \sum_{m,n} K_{i-1,m,n} G_{i,m,n}}$

Представляє градієнт функції втрат J(S_i, K_(i-1)) відносно значень на попередньому шарі. Цей градієнт використовується для зворотного поширення помилок та оновлення параметрів моделі.

- $\boldsymbol{K_{new} = K_i - \mu\, g(G, S_{i-1})}$
- $\boldsymbol{S_{new} = S_i - \mu\, g(K, G_{i+1})}$

3.3 Функції відображення

Процес створення функцій для візуалізації та тестування моделі розпочався з розробки функцій, спрямованих на відображення та виведення зображень.

1. display_image(img, bbox_coords=[], pred_coords=[], norm=False):

Функція display_image призначена для відображення зображень з або без обрамлення.

Параметри включають зображення img, координати обрамлення bbox_coords та прогнозовані координати pred_coords.

Зображення може бути нормалізоване параметром norm.

2. display_image_from_file(name, bbox_coords=[], path=train_path):

display_image_from_file використовується для відображення зображень, зчитаних з файлу.

Приймає ім'я зображення, координати обрамлення bbox_coords та шлях path, за замовчуванням - train_path.

3. display_from_dataframe(row, path=train_path):

display_from_dataframe виводить зображення та обрамлення на основі даних з DataFrame.

Використовує display_image_from_file для відображення.

4. display_grid(df=train, n_items=3):

Функція display_grid виводить сітку зображень з випадковими елементами з набору даних.

5. data_generator(df=train, batch_size=16, path=train_path):

data_generator є генератором даних для тренування моделі. Повертає пари зображень та координат обрамлення у вказаній кількості.

3.4 Функції для тестування моделі:

- *test_on_selected_images(model, image_names):*

Тестує модель на вибраних зображеннях та виводить прогнози разом з оригінальними обрамленнями.

- *test_on_selected_images02(model, image_names):*

Подібно до попередньої, але для іншого шляху зображень.

- *test_model(model, datagen):*

Тестує модель на одному прикладі з датагенератора та виводить оригінальне та прогнозоване обрамлення.

- *test(model):*

Тестує модель на декількох прикладах та виводить результати.

- *ShowTestImages(tf.keras.callbacks.Callback):*

Клас ShowTestImages є колбеком для відображення тестових зображень на кожному етапі навчання моделі. Використовує функцію test.

3.5 Результати

У моїй конфігурації мережі було включено 8 згорткових шарів. Я не використовувати операції пулінгу для зменшення розмірності. Вихід - чотири ознаки координат для подальшого аналізу. Також, використовувалась валідна згортка, без пропуску значень на вхідних ознаках (без кроку). Для кожного шару застосовувалось по одному ядру, без використання пулінгу і стрітчатої структури. Пошук зводиться знаходженням лише одного елемента на зображені. Вихід – отримання 4 прогнозованих координат, що характеризують об’єкт.

Структура написання коду: мережа представлена окремим класом в якому вбудовані методи мережі; математичні функції описані окремо, поза класом. На рисунку 16 зображено відображення з вхідними ядрами, ініціалізованими стандартним нормальним розподілом.з середнім значенням 0 та стандартним відхиленням 1. Розміри ядер для перших двох шарів згортки становить становлять 5x5 з кроком 5 при розмірі зображення 676x380. На третьому шарі розмір вже становитеме– 27x15. Далі ялра з розміром 2x2 для кожного щару з відповідним індивідуальним кроком. Останній шар має розмір 4x4, де вказано на максимальне та мінімальне значення прогнозованих координат. Втрата знаходиться різницею сум прогнозованих значень та реальних. На рисунку 16 – вже навчена модель, де присутні прямокутники.

Рисунок 15 –Відображення машин на ненавчених даних..

Рисунок 16. – Знаходження мережею машин на зображені з тренувальних даних.

Процес навчання залежить від кількості пройдених епох по даним й відбувається допоки втрати не змінюватимуться для 3 останніх значень у межах 10. На рисунках 17-18 відображена тенденція зміни навчання з часом.

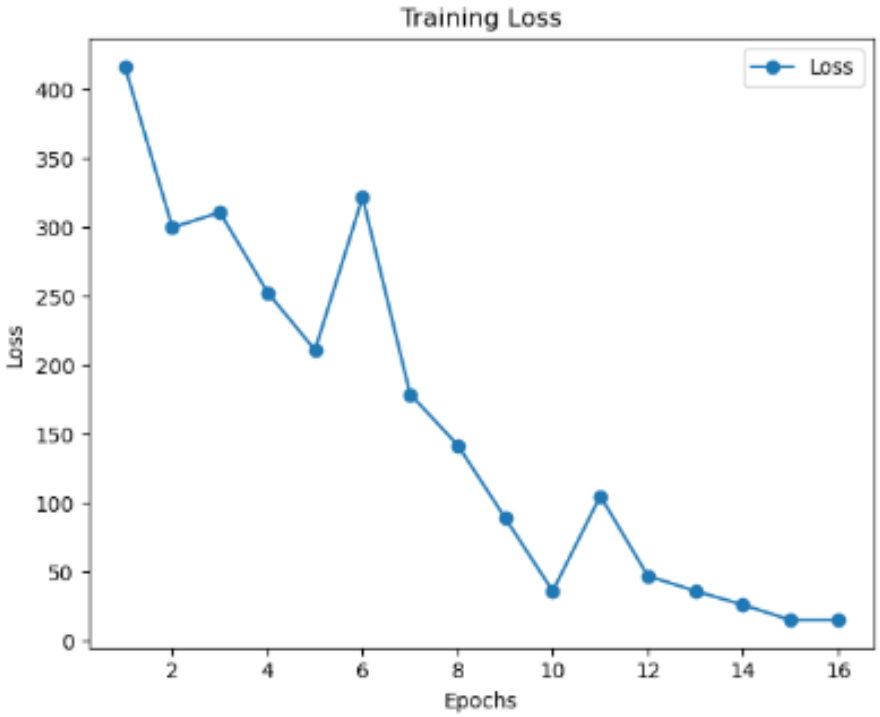


Рисунок 17. – Зміна значень середніх втрат впродовж 16 епох.

Результати в середньому мають точність 0,97. Бралось середнє значення для кожного зображення. Точність вимірювалась функцією softmax. На тестових даних точність складає – 0,74. Приклади відображення, на тестовому наборі – рисунки 18-22. Хочу зазначити, якщо точність менша за 0,5 зображення ігнорується.

Рисунок 18. – Зміна значень середніх точностей впродовж 16 епох.

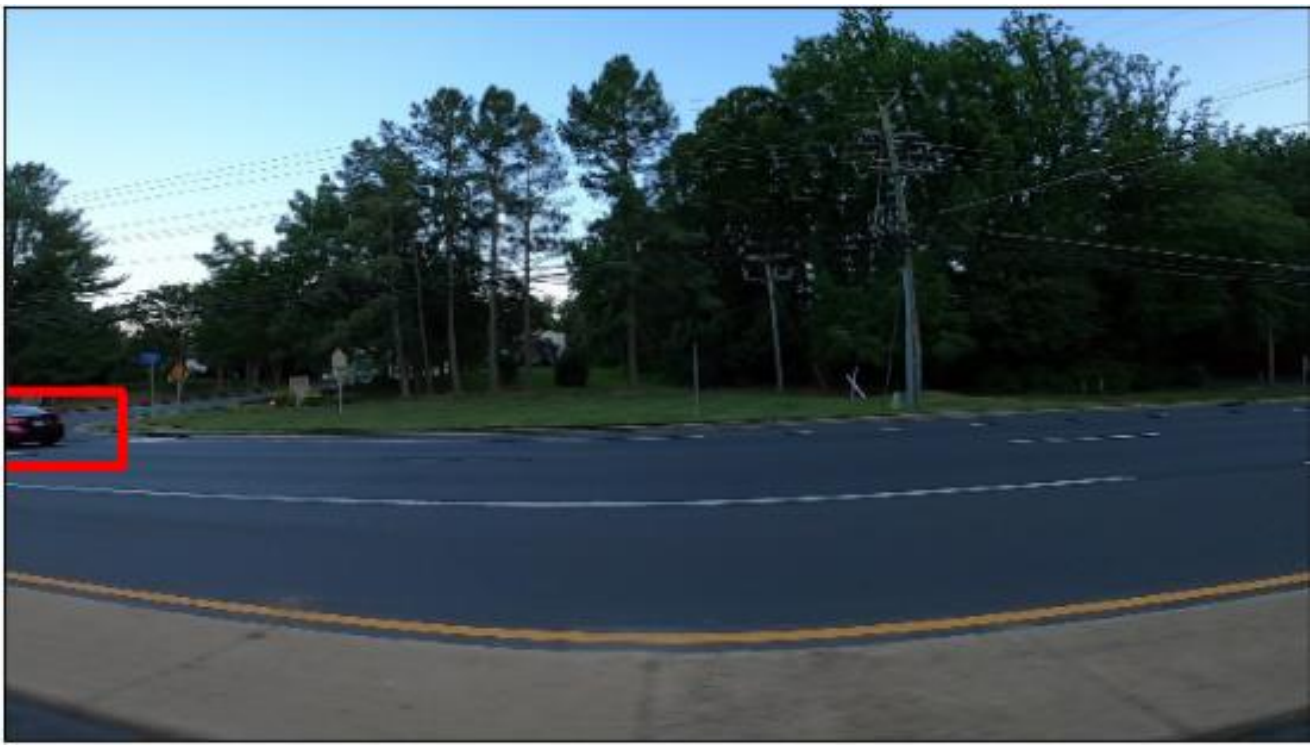


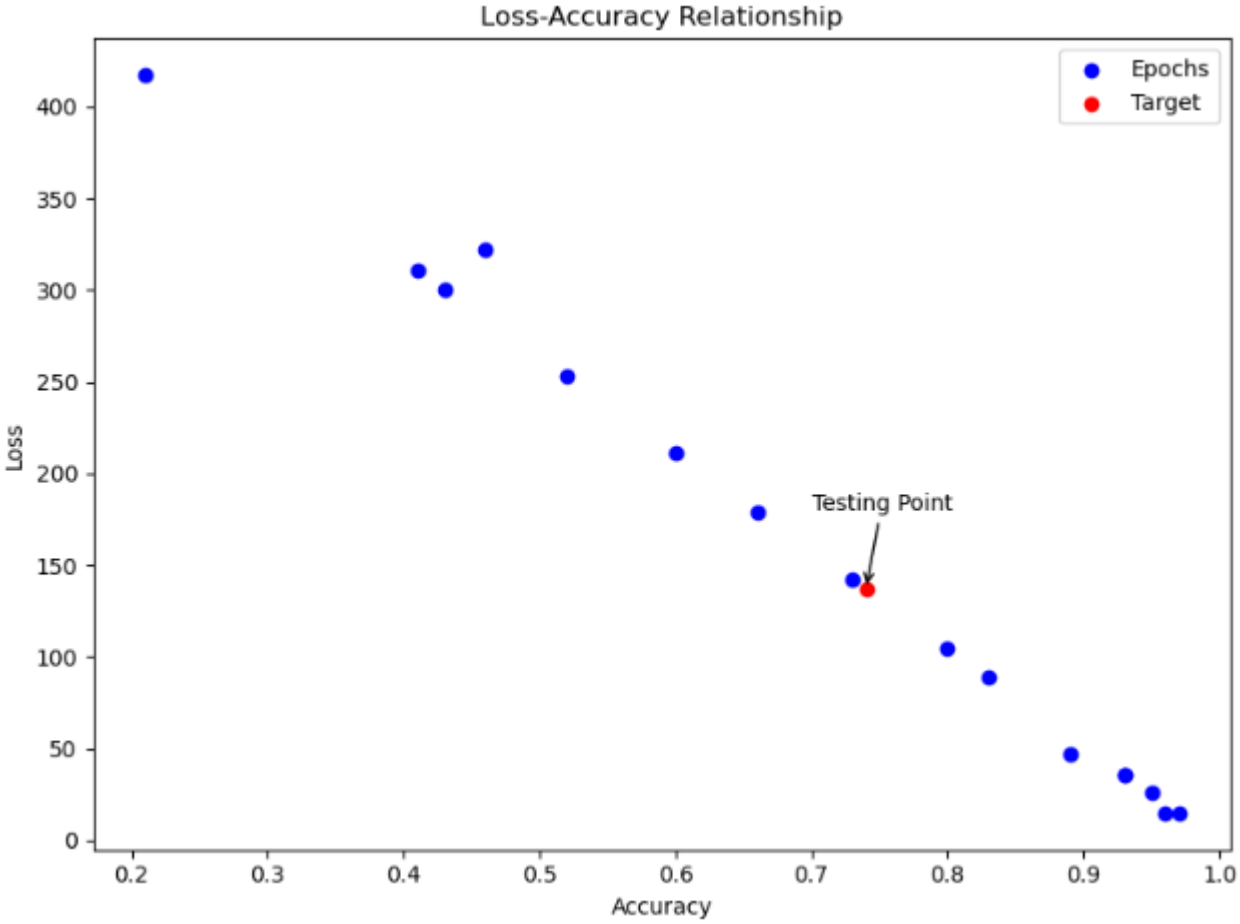
Рисунок 18. – Приклад відображення на тестових даних.

Рисунок 19. – Приклад відображення на тестувальних даних

Рисунок 20. – Приклад відображення на тестових даних.

Рисунок 21. – Приклад відображення на тестових даних з ігноруванням об’єкту.

Рисунок 22. – Приклад відображення на тестових даних



.

Рисунок 23. - Залежність точності і втрати під час навчання та тестування. Червона точка – значення на тестових даних.

Для порівняння використана модель YOLO V5. (You Only Look Once) - це популярний підхід до об'єктного розпізнавання в реальному часі, який дозволяє визначати та класифікувати об'єкти на зображеннях або відео. Версія YOLO V5 представляє собою п'яту ітерацію алгоритму YOLO, яка була розроблена для поліпшення продуктивності та точності порівняно з попередніми версіями. Приклади на рисунках 24-27.

YOLO V5 демонструє вищу точність та здатність визначення конкретних об'єктів на зображеннях порівняно із розробленою конфігурацією мережі. Це може свідчити про ефективність та докладність алгоритмів, вбудованих у YOLO V5, у порівнянні з власним підходом. Для покращення результатів може бути варто переглянути архітектуру мережі, оптимізувати параметри та враховувати

додаткові фактори у процесі навчання. Кожна модель та реалізація під конкретне завдання має свій індивідуальний підхід до розв'язання, і це може включати різні аспекти від архітектури мережі до використання різних методів оптимізації та активаційних функцій

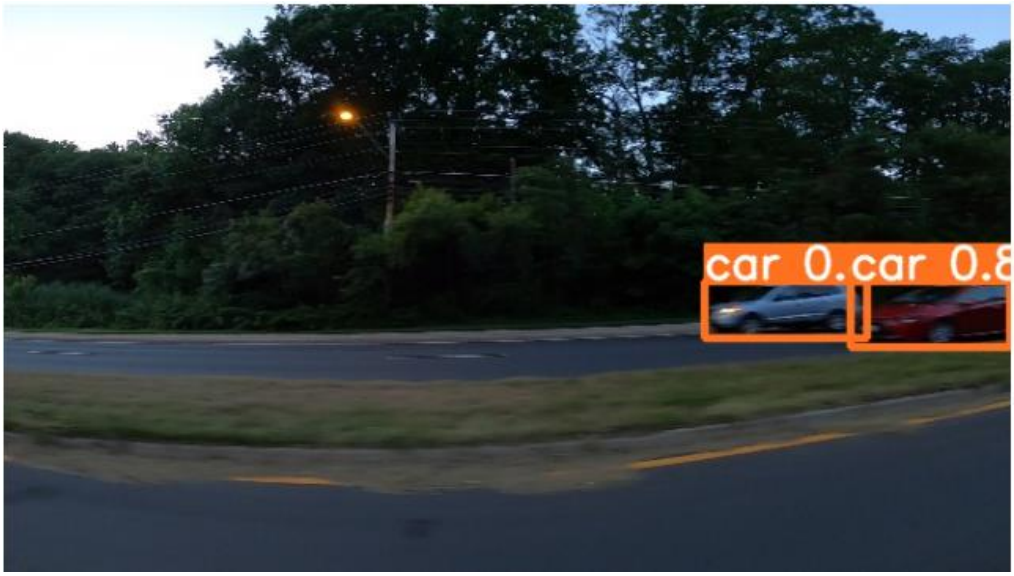


Рисунок 24. – Відображення за допомогою YOLO V5.

Рисунок 25. – Відображення за допомогою побудованої мережі.

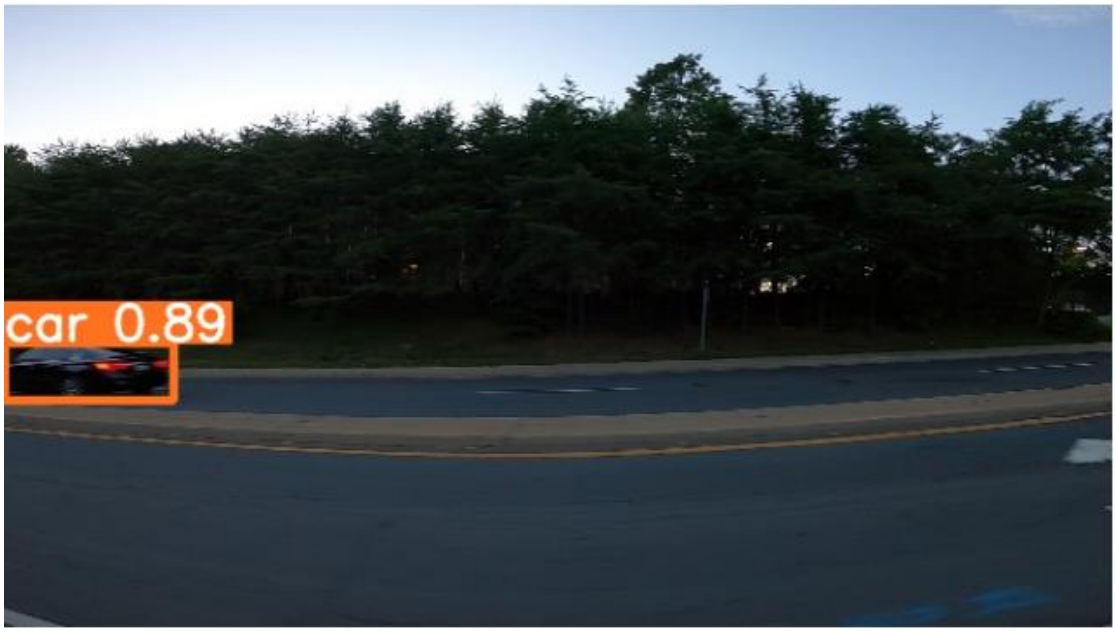


Рисунок 26. – Відображення за допомогою YOLO V5.

Рисунок 27. – Відображення за допомогою побудованої мережі.

ВИСНОВКИ

- За час роботи проведено аналіз різних підходів для будови нейронних мереж з виявленням об’єктів на зображені. Будована мережа складає 8 згорткових шарів, відсутність операцій пулінгу та використання валідної згортки.
- На тренувальних даних точність моделі становить в середньому 0,97. Це свідчить про ефективність моделі в розпізнаванні та локалізації об'єктів на тренувальних зображеннях.
- На тестових даних точність моделі менша і становить 0,74. Такий результат може вказувати на недостатність даних для ефективного навчання, або на необхідність поліпшення архітектури мережі для кращої адаптації до нових зображень.
- YOLO V5 демонструє вищу точність та здатність визначення конкретних об'єктів на зображеннях порівняно із розробленою конфігурацією мережі. Це може свідчити про ефективність та докладність алгоритмів, вбудованих у YOLO V5, у порівнянні з власним підходом.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- TensorFlow: https://www.tensorflow.org/tutorials/images/cnn
- PyTorch: https://pytorch.org/tutorials/
- Keras: https://keras.io/guides/sequential_model/
- "ImageNet Classification with Deep Convolutional Neural Networks" - Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)
- "Deep Residual Learning for Image Recognition" - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (https://arxiv.org/abs/1512.03385)
- Coursera "Convolutional Neural Networks" (https://www.coursera.org/learn/convolutional-neural-networks)
- Fast.ai "Practical Deep Learning for Coders" (https://course.fast.ai/)
- "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
- "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron.
- Towards Data Science (https://towardsdatascience.com/)
- Distill.pub (https://distill.pub/)
- arXiv.org (https://arxiv.org/)
- Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures / L. Ali et al. Sensors. 2021. Vol. 21, no. 5. P. 1688. URL: https://doi.org/10.3390/s21051688 (date of access: 14.11.2023)
- . Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of The 32nd International Conference on Machine Learning, pages 448–456, 2015.
- Understanding Loss Function in Deep Learning. Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deeplearning/?utm_source=reading_list&utm_medium=https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/ (date of access: 13.11.2023).

- Milosevic N. Introduction to Convolutional Neural Networks. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-5648-0 (date of access: 12.11.2023
- V. H. Phung, E. J. Rhee et al., “A high-accuracy model average ensembleof convolutional neural networks for classification of cloud image patcheson small datasets,” Applied Sciences, vol. 9, no. 21, p. 4500, 2019
- (PDF) Recent Advances in Deep Learning Techniques for Face Recognition. Available from: https://www.researchgate.net/publication/350326846_Recent_Advances_in_Deep_Learning_Techniques_for_Face_Recognition [accessed Dec 01 2023].
- Beysolow II T. Recurrent Neural Networks (RNNs). Introduction to Deep Learning Using R. Berkeley, CA, 2017. P. 113–124. URL: https://doi.org/10.1007/978-1-4842-2734-3_6 (date of access: 11.11.2023).

Додаток А

```
import matplotlib.pyplot as plt
      import pandas as pd
      import numpy as np
      from sklearn.metrics import accuracy_score
      import numpy as np
      def sigmoid(x):

          return 1 / (1 + np.exp(-x))

      def relu(x):

          return np.maximum(0, x)

      def cost_function(predicted, actual):
          """
          Функція втрат (відстань між прогнозованим та реальним значенням)
з Л1 регуляризацією.

          Параметри:
          - predicted: прогнозовані значення (масив з 4 значеннями)
          - actual: реальні значення (масив з 4 значеннями)

          Повертає:
          - середнє значення функції втрат
          """
          return np.mean(np.abs(predicted - actual))

      def G_function(previous_function, predicted_features, kernel):
          """
```

```
        Функція G для знаходження відношення між попередньою функцією та прогнозованим значенням ознак.

        Параметри:
        - previous_function: попередня функція
        - predicted_features: прогнозовані значення ознак
        - kernel: ядро

        Повертає:
        - значення функції G
        """
        return previous_function / (predicted_features @ kernel)

    def g_function(J, kernel, G, features):
        """
        Функція g для знаходження відношення між J та ядром через суму G та ознаками.

        Параметри:
        - J: значення функції втрат
        - kernel: ядро
        - G: значення функції G
        - features: значення ознак

        Повертає:
        - значення функції g
        """
        return J / (np.sum(G) * features @ kernel)

    def h_function(J, features, G, kernel):
        """
```

```
        Функція h для знаходження відношення між J та ознаками через суму
G та ядро.

        Параметри:
        - J: значення функції втрат
        - features: значення ознак
        - G: значення функції G
        - kernel: ядро

        Повертає:
        - значення функції h
        """
        return J / (np.sum(G) * features @ kernel)

    def update_weights(previous_weights, g_function_result):
        """
        Функція оновлення вагів через різницю минулих вагів та функції g.

        Параметри:
        - previous_weights: попередні ваги
        - g_function_result: значення функції g

        Повертає:
        - оновлені ваги
        """
        return previous_weights - g_function_result

    def update_features(previous_features, h_function_result):
        """
        Функція оновлення ознак через різницю минулих ознак та функції h.
```

```
        Параметри:
        - previous_features: попередні ознаки
        - h_function_result: значення функції h

        Повертає:
        - оновлені ознаки
        """
        return previous_features - h_function_result

    def plot_loss_accuracy_combined(cnn):
        # отримання значень точності та втрат з об'єкту MyFirstCNN
        epochs = range(1, len(cnn.train_accuracies) + 1)
        train_losses = cnn.train_losses
        train_accuracies = cnn.train_accuracies
        test_losses = cnn.test_losses
        test_accuracies = cnn.test_accuracies

        # Відображення графіка
        plt.figure(figsize=(8, 6))

        # Побудова графіка
        plt.scatter(train_accuracies, train_losses, color='blue',
label='Training', marker='o')
        plt.scatter(test_accuracies, test_losses, color='green',
label='Testing', marker='o')

        # Додавання підписів
        plt.title('Loss-Accuracy Relationship')
        plt.xlabel('Accuracy')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()
```

```
    def plot_loss_accuracy_fixed(cnn):
        # Отримання середніх значень точності та втрат
        average_train_losses = np.mean(cnn.train_losses)
        average_train_accuracies = np.mean(cnn.train_accuracies)
        average_test_losses = np.mean(cnn.test_losses)
        average_test_accuracies = np.mean(cnn.test_accuracies)

        # Відображення графіка
        plt.figure(figsize=(12, 5))

        plt.subplot(1, 2, 1)
        plt.plot(range(1, len(cnn.train_losses) + 1), cnn.train_losses,
label='Loss', marker='o')
        plt.axhline(y=average_train_losses, color='r', linestyle='--',
label='Average Train Loss')
        plt.title('Training Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(range(1, len(cnn.train_accuracies) + 1),
cnn.train_accuracies, label='Accuracy', marker='o', color='orange')
        plt.axhline(y=average_train_accuracies, color='r', linestyle='--
', label='Average Train Accuracy')
        plt.title('Training Accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.tight_layout()
```

```
plt.show()

# Виведення середніх значень на консоль
print(f"Average Train Loss: {average_train_losses}")
print(f"Average Train Accuracy: {average_train_accuracies}")
print(f"Average Test Loss: {average_test_losses}")
print(f"Average Test Accuracy: {average_test_accuracies}")
```

Додаток Б

```
class MyFirstCNN:

    def __init__(self, input_size, hidden_layers, output_size,
kernel_size=3):

        """

        Конструктор класу для першої згорткової нейронної мережі.


        Параметри:

        - input_size: розмір вхідного шару (кількість вхідних ознак)

        - hidden_layers: список, що містить розмір кожного прихованого
шару

        - output_size: розмір вихідного шару

        - kernel_size: розмір ядра конволюції (за замовчуванням 3)

        """

        self.input_size = input_size

        self.hidden_layers = hidden_layers

        self.output_size = output_size

        self.kernel_size = kernel_size


         # Нові змінні для збереження точності та втрат

        self.train_accuracies = []

        self.train_losses = []

        self.test_accuracies = []

        self.test_losses = []


        # Структура для зберігання шарів, ваг і ознак

        self.layers = []


        # Ініціалізація ваг та ознак для вхідного шару

        input_layer = {"weights": np.random.randn(input_size,
hidden_layers[0]),

                       "features": np.zeros((1, hidden_layers[0]))}
```

```
        self.layers.append(input_layer)


        # Ініціалізація ваг та ознак для прихованих шарів
        for i in range(len(hidden_layers) - 1):
            hidden_layer = {"weights": np.random.randn(hidden_layers[i],
hidden_layers[i + 1]),
                            "features": np.zeros((1, hidden_layers[i +
1]))}
            self.layers.append(hidden_layer)


        # Ініціалізація ваг та ознак для вихідного шару
        output_layer = {"weights": np.random.randn(hidden_layers[-1],
output_size),
                        "labels": np.zeros((1, output_size)),
                        "values": np.zeros((1, output_size))}
        self.layers.append(output_layer)


    def add_layers(self, num_layers, layer_size):
        """
        Функція для додавання нових прихованих шарів.


        Параметри:
        - num_layers: кількість нових прихованих шарів
        - layer_size: розмір кожного нового прихованого шару
        """
        for _ in range(num_layers):
            new_layer = {"weights": np.random.randn(self.hidden_layers[-
1], layer_size),
                         "features": np.zeros((1, layer_size))}
            self.layers.insert(-1, new_layer)


    def set_output_layer_params(self, labels_size, values_size):
```

```
        """
        Функція для встановлення розмірності міток та значень для
вихідного шару.

        Параметри:
        - labels_size: розмірність міток для вихідного шару
        - values_size: розмірність значень для вихідного шару
        """
        self.layers[-1]["weights"] = np.random.randn(self.hidden_layers[-
1], labels_size)
        self.layers[-1]["labels"] = np.zeros((1, labels_size))
        self.layers[-1]["values"] = np.zeros((1, values_size))

    def set_hidden_layer_params(self, layer_index, kernel_size,
hidden_features_size):
        """
        Функція для встановлення розмірів ядра та вхідних ознак для
прихованого шару.

        Параметри:
        - layer_index: індекс прихованого шару, для якого встановлюються
параметри
        - kernel_size: розмір ядра для прихованого шару
        - hidden_features_size: розмір вхідних ознак для прихованого шару
        """
        self.layers[layer_index]["weights"] =
np.random.randn(hidden_features_size, kernel_size)
        self.layers[layer_index]["features"] = np.zeros((1,
hidden_features_size))

    def set_output_layer_params(self, labels_size, values_size):
        """
        Функція для встановлення розмірності міток та значень для
вихідного шару.
```

```
        Параметри:
        - labels_size: розмірність міток для вихідного шару
        - values_size: розмірність значень для вихідного шару
        """
        self.layers[-1]["weights"] = np.random.randn(self.hidden_layers[-
1], labels_size)
        self.layers[-1]["features"] = np.zeros((1, values_size))


    def train(self, X_train, y_train, max_difference=10.0):
        prev_loss = float('inf')  # Попереднє значення втрат для
порівняння
        last_losses = []  # Зберігаємо останні три значення втрат
        epoch = 0

        # Зберігаємо значення точності та втрат для кожного зображення
        accuracies_per_image = []
        losses_per_image = []

         # Оцінка точності та втрат для виведення
        accuracy = np.mean(predicted_labels > 0.9)
        print(f"Epoch {epoch + 1}, Loss: {loss}, Accuracy: {accuracy}")

        # Збереження точності та втрат для кожного зображення
        self.train_accuracies.append(accuracy)
        self.train_losses.append(loss)



        while True:
            # ... інші частини коду ...

            # Оцінка точності та втрати для кожного зображення
```

```
            for i in range(len(X_train)):
                x_sample, y_sample = X_train[i], y_train[i]

                # Forward pass
                conv_result = self.convolution(self.layers[0]["weights"],
x_sample)
                predicted_labels = sigmoid(conv_result)

                # Обчислення функції втрат
                loss = cost_function(predicted_labels, y_sample)

                # Збереження значення точності та втрати для кожного
зображення
                accuracy = np.mean(predicted_labels > 0.9)
                accuracies_per_image.append(accuracy)
                losses_per_image.append(loss)

                # ... інші частини коду ...

            # Перевірка критерію зупинки (різниця між останніми трьома
втратами не більше max_difference)
            if len(last_losses) == 3 and max(last_losses) -
min(last_losses) <= max_difference:
            print(f"Training stopped. Last three losses within
{max_difference}.")
            break



            epoch += 1
             # Оцінка точності та втрат для тестової вибірки
            self.evaluate_on_test_set(X_test, y_test)

    def predict(self, X):
```

```
        """
        Функція для здійснення прогнозу на основі вхідних даних.

        Параметри:
        - X: вхідні дані (масив форми (m, input_size), де m - кількість прикладів)

        Повертає:
        - predictions: прогнозовані вихідні дані (масив форми (m, output_size))
        """
        conv_result = self.convolution(self.layers[0]["weights"], X)
        detection_result = sigmoid(conv_result)
        return detection_result
        pass

def detect_objects(self, testing_image, threshold=0.5):
        # Отримання прогнозованих значень
        conv_result = self.convolution(self.layers[0]["weights"], testing_image)
        detection_result = conv_result  # Без використання сигмоїди
        bounding_boxes = detection_result[0]  # Припускаємо, що маємо лише один приклад

        # Виведення зображення
        plt.imshow(testing_image.reshape((28, 28)), cmap='gray')

        # Перевірка точності та виведення прямокутної зони для кожного прикладу
        for bbox in bounding_boxes:
            x_min, x_max, y_min, y_max = bbox
            accuracy = x_max  # Припускаємо, що точність знаходиться в x_max (може змінитися)
```

```
            # Перевірка точності порівняно з порогом
            if accuracy >= threshold:
                rect = plt.Rectangle((x_min, y_min), (x_max - x_min),
(y_max - y_min),
                                     edgecolor='red', facecolor='none')
                plt.gca().add_patch(rect)

        plt.show()

        return detection_result
    def plot_loss_accuracy(self, losses, accuracies):
        # Відображення графіка втрат та точності відносно кількості епох
        epochs = len(losses)
        plt.figure(figsize=(12, 5))
        plt.subplot(1, 2, 1)
        plt.plot(range(1, epochs + 1), losses, label='Loss', marker='o')
        plt.title('Training Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(range(1, epochs + 1), accuracies, label='Accuracy',
marker='o', color='orange')
        plt.title('Training Accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.tight_layout()
        plt.show()
```

```
    def convolution(self, kernel, features, stride=1):
        """
        Функція для застосування конволюції до вхідних ознак з заданим ядром та кроком.

        Параметри:
        - kernel: ядро конволюції (масив форми (kernel_height, kernel_width))
        - features: вхідні ознаки (масив форми (input_height, input_width))
        - stride: крок конволюції (за замовчуванням 1)

        Повертає:
        - conv_result: результат конволюції (масив форми (output_height, output_width))
        """
        kernel_height, kernel_width = kernel.shape
        input_height, input_width = features.shape

        # Розмір вихідних ознак
        output_height = (input_height - kernel_height) // stride + 1
        output_width = (input_width - kernel_width) // stride + 1

        # Ініціалізація результату конволюції
        conv_result = np.zeros((output_height, output_width))

        # Процес конволюції
        for i in range(0, input_height - kernel_height + 1, stride):
            for j in range(0, input_width - kernel_width + 1, stride):
                conv_result[i // stride, j // stride] = np.sum(features[i:i + kernel_height, j:j + kernel_width] * kernel)
```

```
        return conv_result


    return conv_result


    def save_epoch_results(self, epoch, average_loss, average_accuracy):
            """
            Функція для збереження середніх значень точності та втрат для кожної епохи.
            """
            # Додайте код для збереження значень в потрібне місце (наприклад, список чи файл)
            # Збереження в рядок, для прикладу:
            result_string = f"Epoch {epoch + 1}, Average Loss: {average_loss}, Average Accuracy: {average_accuracy}"
            print(result_string)


    def evaluate_on_test_set(self, X_test, y_test):
        accuracies_per_image = []
        losses_per_image = []


        for i in range(len(X_test)):
            x_sample, y_sample = X_test[i], y_test[i]


            # Forward pass
            conv_result = self.convolution(self.layers[0]["weights"], x_sample)
            predicted_labels = sigmoid(conv_result)


            # Обчислення функції втрат
            loss = cost_function(predicted_labels, y_sample)
```

```
            # Збереження значення точності та втрати для кожного
зображення
            accuracy = np.mean(predicted_labels > 0.9)
            accuracies_per_image.append(accuracy)
            losses_per_image.append(loss)
        # Збереження точності та втрат для кожного зображення на тестовій
вибірці
        self.test_accuracies.append(accuracy)
        self.test_losses.append(loss)

        # Обчислення середніх значень для тестової вибірки
        average_loss = np.mean(losses_per_image)
        average_accuracy = np.mean(accuracies_per_image)

        # Збереження результатів тестування
        self.save_test_results(average_loss, average_accuracy)
```

Додаток В

```
	# Зчитуємо дані

	testing_image = np.load("data/testing_image.npy")

	trained_image = np.load("data/trained_image.npy")

	bounding_boxes_data = 
pd.read_csv("Data/train_solution_bounding_boxes (1).csv")



	# Виділяємо мітки для тренування

	y_train = bounding_boxes_data[["x_max", "x_min", "y_max", 
"y_min"]].values



	# Створюємо об'єкт класу MyFirstCNN

	my_cnn = MyFirstCNN(input_size=len(trained_image[0]), 
hidden_layers=[8, 6, 5, 4, 4, 4, 4, 4], output_size=len(y_train[0]))



	# Встановлюємо параметри для вхідного та вихідного шарів

	my_cnn.set_input_layer_params(kernel_size=5, 
input_features_size=len(trained_image[0]), stride=5)

	my_cnn.set_hidden_layer_params(layer_index=1, kernel_size=5, 
hidden_features_size=8, stride=5)

	my_cnn.set_hidden_layer_params(layer_index=2, kernel_size=2, 
hidden_features_size=6, stride=2)

	my_cnn.set_hidden_layer_params(layer_index=3, kernel_size=2, 
hidden_features_size=5, stride=2)

	my_cnn.set_hidden_layer_params(layer_index=4, kernel_size=2, 
hidden_features_size=4, stride=1)

	my_cnn.set_hidden_layer_params(layer_index=5, kernel_size=2, 
hidden_features_size=4, stride=1)

	my_cnn.set_hidden_layer_params(layer_index=6, kernel_size=(3, 2), 
hidden_features_size=4, stride=(3, 1))

	my_cnn.set_hidden_layer_params(layer_index=7, kernel_size=2, 
hidden_features_size=4, stride=1)

	my_cnn.set_output_layer_params(labels_size=len(y_train[0]), 
values_size=len(y_train[0]))
```

```
    # Тренуємо модель з критерієм зупинки за втратами
    losses, accuracies = my_cnn.train(trained_image, y_train,
stop_loss=10.0)


    # Виведення результатів
    print("Object Detection Result:")
    detection_result = my_cnn.detect_objects(testing_image)


    # Виведення графіка втрат та точності
    my_cnn.plot_loss_accuracy(losses, accuracies)
    # Виклик функції для відображення
    plot_loss_accuracy_fixed(cnn)
    import matplotlib.pyplot as plt


    cnn = MyFirstCNN(input_size, hidden_layers, output_size)
    cnn.train(X_train, y_train, X_test, y_test)
    plot_loss_accuracy_combined(cnn)
```

Додаток Г

```
pip install opencv-python
import cv2

# Завантаження попередньо навченої моделі YOLO
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f]

# Зчитуємо зображення
image = cv2.imread("testing_image.npy")

# Отримуємо висоту та ширину зображення
height, width, _ = image.shape
# Перетворення зображення у blob
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True,
crop=False)

# Передача blob через мережу YOLO
net.setInput(blob)
output_layers_names = net.getUnconnectedOutLayersNames()
outputs = net.forward(output_layers_names)

# Обробка виведених результатів
for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5 and class_id == 2:  # Індекс 2 відповідає
машинам (згідно з файлом coco.names)
            center_x, center_y = int(detection[0] * width),
int(detection[1] * height)
```

```
            w, h = int(detection[2] * width), int(detection[3] *
height)
            x, y = center_x - w // 2, center_y - h // 2
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0),
2)
# Виведення результату
cv2.imshow("Car Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```