math_2023_019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

СТВОРЕННЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ТА КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:
студент 6 курсу, 607 групи
Лека Максим Олександрович
Керівник*:*
асистент, канд.фіз.-мат. наук
Дорош А.Б.

До захисту допущено

на засіданні кафедри

протокол № 9 від 5 грудня 2023 р.

Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2023

Анотація

У даній роботі розглядається розробка нейронної мережі для ефективного розпізнавання дорожніх знаків на зображеннях без використання спеціалізованих бібліотек для машинного навчання. З використанням мови програмування С та стандартних бібліотек вирішено завдання ініціалізації тренувального, тестового та валідаційного наборів даних, навчання нейронної мережі та її тестування. Код також включає функціонал для завантаження та обробки зображень дорожніх знаків, а також валідації роботи нейронної мережі.

Ключові слова:

Нейронна мережа, розпізнавання дорожніх знаків, зображення, тренувальний набір даних, тестовий набір даних, валідаційний набір даних, програмування на С.

Анотація

This work explores the development of a neural network for efficient recognition of traffic signs in images without the use of specialized machine learning libraries. Utilizing the C programming language and standard libraries, the tasks of initializing training, testing, and validation datasets, training the neural network, and testing its performance have been addressed. The code also encompasses functionality for loading and processing images of traffic signs, as well as validating the neural network's performance.

Ключові слова:

Neural Network, Traffic Sign Recognition, Image, Training Dataset, Testing Dataset, Validation Dataset, C Programming.

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

_____ М.О. Лека

Зміст

Вступ

Актуальність роботи. У сучасному світі, де штучний інтелект та комп'ютерне зорове розпізнавання зображень стають необхідною складовою багатьох сфер, розробка систем для ефективного розпізнавання дорожніх знаків без використання спеціалізованих бібліотек для машинного навчання має важливе значення. Це актуальне завдання, оскільки дорожні знаки відіграють ключову роль у забезпеченні безпеки на дорогах, і точне їх розпізнавання може виявитися критичним у реальних умовах.

Мета. Метою роботи є розробка та реалізація нейронної мережі для ефективного розпізнавання дорожніх знаків на зображеннях. Основним фокусом є використання мови програмування C та стандартних бібліотек для створення програми, яка може ініціалізувати, тренувати та тестувати нейронну мережу для дорожніх знаків.

Завдання роботи. Виходячи з мети роботи, головною задачею є розробка та реалізація нейронної мережі для ефективного розпізнавання дорожніх знаків на зображеннях, і це відбувається без використання спеціалізованих бібліотек для машинного навчання. У цьому контексті ключовою задачею є використання мови програмування C та стандартних бібліотек для ініціалізації, тренування та тестування нейронної мережі, спрямованої на розпізнавання дорожніх знаків, без залучення готових інструментів для машинного навчання.

Під час виконання кваліфікаційної роботи були вирішені наступні завдання:

- Вивчено можливості застосування розробленої нейронної мережі для розпізнавання дорожніх знаків із зображень різних розмірів.
- Було переглянуто стек технологій машинного навчання та зроблено вибір на користь продуктивності вести розробку “з нуля” на мові програмування C.
- Також було взято з відкритого джерела набір даних, який було розділено на навчальний, тренувальний, валідаційний.
- Розроблено нейронну мережу для ефективного розпізнавання дорожніх знаків, використовуючи обраний інструментарій.
- Проведено тестування програмного продукту на наявність помилок, витоків пам’яті, результатів навчання мережі з різними параметрами змінних, таких як гіперпараметри моделі, та інших факторів, які можуть впливати на результати дослідження.

Програмний продукт складається з файлу із кодом та набору даних, який містить каталоги з xml-анотаціями до зображень, щоб можна було ефективно проводити тренування та навчання нейронної мережі, та, відповідно, самі зображення, попередньо підготовлені для різних етапів роботи.

Об’єкт дослідження - нейронні мережі та методика й технології у їх розробці.

Практичне застосування - використання розробленої нейронної мережі для ефективного розпізнавання дорожніх знаків на зображеннях.

Основними областями застосування є автомобільна безпека та системи допомоги водію, де точне розпізнавання дорожніх знаків може покращити функціонування систем автономного керування та забезпечити

безпеку на дорозі. Розробка використовує мову програмування C та стандартні бібліотеки з уникненням спеціалізованих бібліотек для машинного навчання, що робить її доступною для широкого кола розробників та зацікавлених осіб.

Дипломна робота містить вступ, чотири розділи, висновки, список літератури та додаток. Перший розділ присвячений вивченню тематики дослідження та аналізу вже існуючих рішень. У другому розділі розглядаються типи нейронних мереж та апаратні обчислення. Третій розділ описує процес розробки програмного продукту, включаючи огляд обраного алгоритму та опис коду. Четвертий розділ містить результати виконання та інструкцію для встановлення.

Під час написання дипломної роботи та створення використано книжку та інтернет ресурси за посиланнями [1-15].

1 Огляд проблематики та готових рішень

1.1 Проблематика

Проблематика, яку вирішує дана кваліфікаційна робота, має масштабне значення в сучасному світі, де штучний інтелект та комп'ютерне зорове розпізнавання зображень визначають нові рівні технологічного розвитку. Розробка ефективної нейронної мережі для розпізнавання дорожніх знаків без використання спеціалізованих бібліотек стає викликом, який необхідно вирішити з огляду на потреби сучасних систем транспорту, автономних авіадронів та інших областей.

У контексті транспортних систем, особливо систем автопілоту, точність та швидкість розпізнавання дорожніх знаків може визначити ефективність та безпеку подорожей. Ця робота покликана не лише оптимізувати цей процес, але й покласти підстави для майбутніх розробок в сфері штучного інтелекту та машинного навчання для автомобільної промисловості.

Загальнолюдський погляд на проблему полягає в тому, що зростання рівня безпеки на дорогах та покращення умов для незрячих осіб стають актуальним завданням. Нейронні мережі можуть виявитися корисним інструментом у цьому плані, впроваджуючи інновації, які підвищують рівень комфорту та безпеки для широкого спектру користувачів.

Напрямки використання розробленої нейронної мережі можуть включати в себе не лише транспортні системи, але й інші області, такі як відслідковування та розпізнавання об'єктів у великих містах, що відкриває двері для вдосконалення інфраструктури та розумних міських середовищ.

Таким чином, кваліфікаційна робота ставить за мету вирішити низку проблем, що стоять перед сучасною технологічною спільнотою,

віддзеркалюючи важливі аспекти в області машинного навчання та штучного інтелекту, зокрема в розпізнаванні дорожніх знаків.

1.2 Огляд наявних рішень

Наявні рішення в галузі розпізнавання дорожніх знаків здебільшого базуються на застосуванні спеціалізованих бібліотек та фреймворків для машинного навчання, таких як TensorFlow[1], PyTorch[2], або OpenCV[3]. Ці інструменти надають широкий функціонал та зручний інтерфейс для розробки та вдосконалення моделей.

Зокрема, деякі системи використовують готові моделі глибокого навчання, такі як YOLO (You Only Look Once)[4] чи SSD (Single Shot Multibox Detector)[5], які надають високу швидкість розпізнавання та високу точність. Такі рішення часто використовують архітектури типу згорткових нейронних мереж (CNN), що дозволяє ефективно впоратися з обробкою зображень.

Проте існуючі підходи можуть бути обмеженими у використанні, оскільки вони часто вимагають великої кількості ресурсів та потужності обчислень. Вищеописані бібліотеки та моделі можуть бути важкими для інтеграції в обмежених умовах, таких як вбудовані системи або додатки, де обмежені обчислювальні ресурси.

Отже, даний проект спрямований на розробку оптимізованої нейронної мережі без використання спеціалізованих бібліотек, з фокусом на швидкодії та легкості інтеграції у різні середовища. Це дозволить розширити застосування системи розпізнавання дорожніх знаків на більш широкий спектр застосувань, включаючи області з обмеженими ресурсами та вимогами до продуктивності.

На сучасному ринку існують різноманітні готові продукти для розпізнавання дорожніх знаків, які базуються на застосуванні різних технологій та методів машинного навчання. Деякі з них використовують готові моделі глибокого навчання, в той час як інші можуть використовувати традиційні методи обробки зображень.

Один із популярних продуктів - Mobileye[6], використовує комплексний підхід, включаючи використання камер, радарів та інших сенсорів. Вони надають рішення для розпізнавання дорожніх знаків, систем безпеки та системи допомоги водієві.

Ще однією платформою є OpenALPR[7], яка спеціалізується на розпізнаванні номерних знаків транспортних засобів. Вони використовують комбінацію оптичного розпізнавання символів та машинного навчання для досягнення високої точності.

У більшості випадків такі готові продукти є зручними та ефективними, проте їхнє використання може бути обмеженим та витратним. Окрім того, вони можуть не завжди відповідати конкретним потребам або вимогам деяких проектів, що ставить під сумнів їхню універсальність. У даному випадку розробка власної системи розпізнавання дорожніх знаків із фокусом на нейронних мережах та використання мови програмування C відкриває можливість для створення ефективного та оптимізованого рішення, що враховує специфічні вимоги проєкту.

2 Огляд теорії та алгоритмів нейронних мереж для розпізнавання об’єктів

2.1 Різноманітність підходів та застосування нейронних мереж у роботі

Сучасний прогрес у галузі розробки нейронних мереж відкриває безліч можливостей для створення потужних та ефективних систем. У даній роботі глибоко досліджуються та використовуються різні типи нейронних мереж, алгоритми та їхні варіації для досягнення оптимальних результатів.

У світі нейронних мереж існує розмаїття типів, кожен з яких призначений для вирішення конкретних завдань. Ми використовуємо Multilayer Perceptron (MLP) для загальних завдань класифікації та регресії, забезпечуючи гнучкість та точність у прогнозуванні.

Для обробки зображень та визначення складних шаблонів використовується Convolutional Neural Networks (CNN). Цей тип мережі ефективно впорається з визначенням образів та має високу варіативність застосувань.

У роботі використовуються алгоритми регуляризації, зокрема Dropout, який зменшує перенавчання та покращує загальну ефективність нейронної мережі, також розкривається потенціал нейронних мереж у різних областях. Планується досліджувати нові архітектури та експериментувати з різними гіперпараметрами для пошуку оптимальних рішень у майбутньому.

Використання різноманітних типів нейронних мереж та ефективних алгоритмів дозволяє нам досягати вражаючих результатів у поставлених завданнях. Постійне вдосконалення підходів та застосування нових

технологій у сфері нейромереж дозволяє знаходити інноваційні рішення та досягати нових висот у світі штучного інтелекту.

Нейронні мережі визначають сучасні технологічні рішення в різних галузях. Multilayer Perceptrons (MLP), Convolutional Neural Networks (CNN) та Recurrent Neural Networks (RNN) виявляються ключовими для досягнення конкретних цілей в сучасному технологічному ландшафті. MLP використовуються для широкого спектру завдань, зокрема для загального моделювання та класифікації. Компанії, такі як Facebook, використовують MLP для аналізу та класифікації текстового та графічного контенту для покращення персоналізованої подачі інформації користувачам.

CNN відіграють ключову роль у обробці зображень та відео. Google використовує CNN у Google Photos для розпізнавання об'єктів, обличчя та автоматичного створення альбомів. У світі медіа та розваг CNN використовуються для реалістичного оброблення графіки в іграх.

RNN використовуються Amazon для прогнозування покупок та створення персоналізованих рекомендацій. Їхні здатності аналізувати часові послідовності даних покращують точність прогнозування та забезпечують користувачам індивідуально адаптовані рішення.

Ці приклади відображають важливість та універсальність MLP, CNN та RNN у сучасних технологічних рішеннях, підкреслюючи їхню визнану ефективність у різних галузях.

Нейронні мережі є областю досліджень яка стрімко розвивається. Постійне вдосконалення підходів та застосування нових технологій у сфері нейромереж дозволяє знаходити інноваційні рішення та досягати нових висот у світі штучного інтелекту.

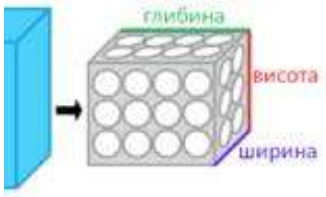
Одним із перспективних напрямків розвитку нейронних мереж є створення нейронних мереж із самонавчанням. Такі мережі здатні навчатися без попередньої ініціалізації, що робить їх більш гнучкими та ефективними.

Іншим перспективним напрямком є створення нейронних мереж із паралельним обчисленням. Такі мережі можуть виконувати обчислення одночасно, що значно прискорює їхню роботу.

Розвиток нейронних мереж має значний потенціал для трансформації багатьох сфер нашого життя. Нейронні мережі вже використовуються в широкому спектрі застосувань, і їхній вплив буде лише зростати в майбутньому.

2.2 Згорткова нейронна мережа (CNN)

Згорткові нейронні мережі (CNN) виявляються надзвичайно потужними для розв'язання завдань обробки зображень завдяки вдосконаленим алгоритмам, які використовуються в їхніх різноманітних шарах.



Мал. 2.2 — Візуалізація схеми CNN

1. Архітектура згорткового шару (Convolutional Layer):

Згортковий шар визначається використанням фільтрів для виявлення різних просторових особливостей у вхідних зображеннях. Математично це виражається формулою:

$Y(i,j)=\sum m \ \sum n \ X(i+m,j+n)\cdot W(m,n)+b$

де $Y(i,j)$ - вихідне значення, $X(i+m,j+n)$ - вхідне значення, $W(m,n)$ - ваги фільтра, b - зсув, а m та n - індекси згорткового ядра[8].

2. Шар згорткового пулінгу (Pooling Layer):

Згортковий пулінг використовується для зменшення розмірності зображення та виділення ключових особливостей. Математично це представлено формулою:

$Y(i,j)=max_{m,n} X(i\cdot s+m,j\cdot s+n)$

де $Y(i,j)$ - вихідне значення, $X(i\cdot s+m,j\cdot s+n)$ - вхідне значення, s - крок зміщення[9].

3. Повністю з'єднаний шар (Fully Connected Layer):

Повністю з'єднаний шар використовується для класифікації та прийняття рішень на основі зібраних ознак. Математично це виражається формулою:

$Y=f(\sum_i \ X_i \cdot W_i + b)$

де Y - вихідне значення, X_i - вхідне значення,W_i - ваги, а f - функція активації [10].

2.3 Багатошаровий персептрон(MLP)

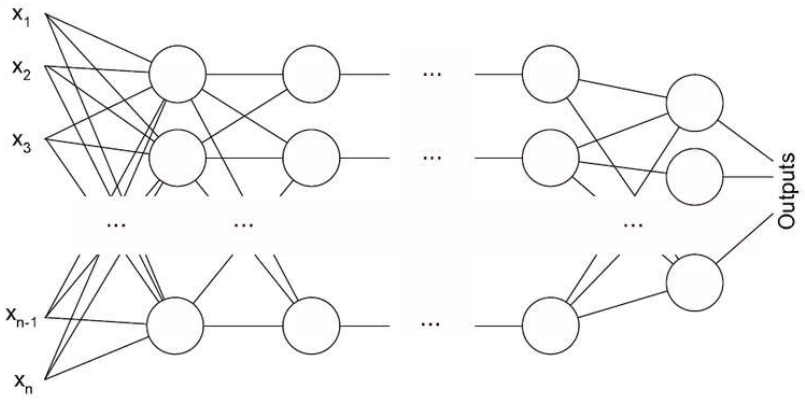
Багатошаровий персептрон (MLP) є основною архітектурою глибоких нейронних мереж, яка використовується для вирішення різноманітних завдань, таких як класифікація та регресія. Розглянемо ключові алгоритми, що лежать в основі MLP.

1. Прямий прохід (Feedforward):

Прямий прохід - це етап, на якому вхідні дані проходять через мережу, слідкуючи за шарами нейронів. Кожен нейрон обчислює зважену суму своїх входів та передає результат функції активації. Формально це можна виразити як:

$a^{(l)}$ =f($\sum_i w_i^{(l)} \cdot x_i + b^{(l)}$)

де $a^{(l)}$ - активація нейрона у шарі l, f - функція активації, $w_i^{(l)}$ - вага, x_i - вхід, $b^{(l)}$- зсув [11].



Мал. 2.2 — Візуалізація схеми MLP

2. Зворотній прохід (Backpropagation):

Зворотній прохід - ключовий етап навчання, під час якого рахуються похідні функції втрат по вагах та зсувах для оновлення параметрів мережі. Формально це виглядає так:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)}$$

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

де L - функція втрат, $\delta_j^{(l)}$ - помилка нейрона j у шарі l, $a_i^{(l-1)}$ - активація нейрона i у попередньому шарі[12].

3. Функція активації:

Функція активації є важливою складовою нейронних мереж, визначаючи їхню здатність виявляти та навчатися складним залежностям у вхідних даних. Ось деякі поширені функції активації з їхніми означеннями та формулами:

Сигмоїда (Sigmoid):

$$Sigmoid(z)=\frac{1}{1+e^{-z}}$$

Сигмоїдальна функція перетворює вхід z в діапазон значень

між 0 і 1. Використовується на виході мереж для моделювання ймовірностей.

Гіперболічний тангенс (Tanh):

$$Tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

Тангенс гіперболічний, схожий на сигмоїду, стискає значення в діапазон між -1 і 1. Використовується для отримання нормалізованого виходу.

Випрямлений лінійний вузол (ReLU):

$$ReLU(z)=max(0,z)$$

Функція залишає додатні значення незмінними та обрізає від'ємні до 0. Є популярною через ефективність у вирішенні проблеми зникаючого градієнту.

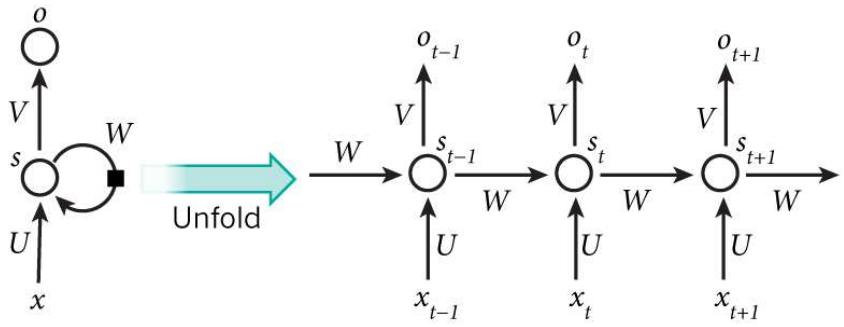
Лінійна функція (Linear):

$$Linear(z)=z$$

Лінійна функція повертає вхідне значення без будь-яких змін. Використовується для безпосереднього виходу моделі.[13]

2.4 Рекурентна нейронна мережа (RNN)

Рекурентні нейронні мережі (RNN) є потужним інструментом у сфері обробки послідовностей та роботи з даними, де важлива залежність від контексту часу. Вони володіють здатністю зберігати попередні стани та використовувати їх для обробки нових вхідних даних.



Мал. 2.3 — Візуалізація схеми RNN

Основні елементи RNN включають вектор вхідного значення, вектор прихованого стану та функцію активації. Структура RNN включає вхід, який представляє вхідні дані у момент часу t, та прихований стан h_t, який зберігає інформацію з попередніх моментів часу та оновлюється при кожному новому вхідному значенні. Вихід RNN обчислюється за допомогою функції активації y_t.

Функція активації RNN може бути різною, проте однією з найпоширеніших є:

h_t= Tanh($W_{hx} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h$)

y_t= Softmax($W_{hy} \cdot h_t + b_y$)

де W_{hx}, W_{hh}, W_{hy} - вагові матриці, b_h та b_y- зсуви, Tanh - тангенс гіперболічний, а Softmax - функція активації Softmax для отримання ймовірностей на виході [14].

Нейронні мережі навчаються на наборі даних прикладів. Набір даних містить вхідні дані та бажані вихідні значення. Нейронна мережа використовує алгоритм навчання, щоб змінити свої параметри (ваги та зсуви) таким чином, щоб зменшити різницю між фактичними та бажаними вихідними значеннями. Найбільш поширеним алгоритмом навчання нейронних мереж є зворотне поширення. Зворотне поширення працює, використовуючи похідні функції втрат для оновлення параметрів мережі.

2.5 Апаратна частина для класифікації зображень

Класифікація зображень - це задача машинного навчання, яка полягає в тому, щоб визначити, до якого класу належить зображення. Для виконання класифікації зображень на апаратному рівні використовуються центральні процесори (CPU) та графічні процесори (GPU).

Центральні процесори (CPU) є основними компонентами комп'ютерів. Вони відповідають за виконання більшості операцій, що виконуються комп'ютером.

Для класифікації зображень CPU використовуються для виконання наступних завдань:

- Завантаження зображення в оперативну пам'ять
- Обробка зображення
- Визначення класу зображення

Обробка зображення на CPU може бути досить трудомісткою, особливо якщо зображення велике або має високу роздільну здатність.

Графічні процесори (GPU) були розроблені для обробки графічних зображень. Вони мають спеціальну архітектуру, яка дозволяє їм виконувати обчислення зображень набагато швидше, ніж CPU.

Для класифікації зображень GPU використовуються для виконання наступних завдань:

- Завантаження зображення до оперативної пам’яті відеоадаптера
- Обробка зображення
- Визначення класу зображення

Обробка зображення на GPU може бути в десятки разів швидше, ніж на CPU.

Вибір апаратної частини для класифікації зображень залежить від декількох факторів, включаючи:

- Розмір і роздільна здатність зображень
- Кількість класів, які потрібно класифікувати
- Точність класифікації

Якщо зображення великі або мають високу роздільну здатність, використання GPU може значно підвищити швидкість класифікації. Якщо потрібно класифікувати багато класів, використання GPU також може бути доцільним.

Вибір апаратної частини для класифікації зображень залежить від багатьох факторів. Якщо в наявності є GPU, його використання може значно підвищити швидкість класифікації. Якщо GPU немає, можна використовувати прийоми для підвищення продуктивності класифікації зображень на CPU.

3 Формулювання задачі та створення програмного продукту

3.1 Постановка задачі

Основна мета даної роботи - створити нейронну мережу, яка здатна розпізнавати різні типи дорожніх знаків на зображеннях. Це завдання є важливим для безпеки дорожнього руху, оскільки дозволяє автомобілям і іншим транспортним засобам отримувати інформацію про обмеження швидкості, перехрестя та інші важливі правила дорожнього руху.

Для вирішення цієї задачі буде використовуватися датасет дорожніх знаків. Датасет повинен містити зображення дорожніх знаків різних типів, а також анотації до цих зображень, які вказують тип кожного знака.

Анотації можуть бути представлені у вигляді текстових міток, координат або інших форматів.

У цій роботі було розпізнано чотири типи дорожніх знаків:

Знак обмеження швидкості

- Знак стоп
- Світлофор
- Знак пішохідного переходу

Для вирішення цієї задачі потрібно пройти такі етапи:

- Підготовка даних:
- Зчитати зображення та анотації з відповідних каталогів. Перевірити та розподілити дані на тренувальний, тестовий та валідаційний набори.
- Ініціалізація мережі:

- Оголосити та ініціалізувати нейронну мережу для розпізнавання дорожніх знаків.
- Тренування мережі:
- Використовувати тренувальний набір для навчання мережі. Застосовувати методи тренування та використовувати збалансовані дані.
- Тестування мережі:
- Використовувати тестовий набір для оцінки продуктивності мережі.Забезпечити виведення метрик ефективності.
- Валідація мережі:
- Використовувати валідаційний набір для перевірки ефективності та уникнення перенавчання.
- Результати та висновки:

  Зробити висновки щодо роботи нейронної мережі та її здатності розпізнавання дорожніх знаків.

У цій задачі є такі обмеження: Код повинен бути написаний мовою програмування C, без використання спеціалізованих бібліотек для глибокого навчання, такі як Tensorflow, Opencv, cuda тощо. Використовується власна реалізація нейронної мережі.

Етапи:

- *Підготовка даних*: завантажуємо датасет з відкритого ресурсу[] і проводимо його розподіл. Зображення та анотації зчитуються з каталогів /data/train, /data/test, та /data/validation. Кожне зображення має відповідну анотацію у форматі XML. Дані розподілені для забезпечення збалансованості між тренувальним,

тестовим та валідаційними наборами (75%, 15%, та 10% відповідно).

- *Ініціалізація мережі*: оголошення та ініціалізація нейронної мережі для розпізнавання дорожніх знаків. Мережа повинна мати архітектуру, що враховує характеристики зображень з дорожніми знаками.
- *Тренування мережі*: використання тренувального набору для навчання мережі. Застосовуються алгоритми оптимізації та зменшення функції втрат для досягнення оптимальних ваг та здатності розпізнавання.
    - Впевненість у збалансованості наборів даних для уникнення перенавчання та забезпечення загальної ефективності моделі.
- *Тестування мережі*: використання тестового набору для оцінки продуктивності мережі. Виведення метрик, таких як відновлення та точність, для кількісної оцінки результатів. Детальний аналіз помилок для виявлення слабких місць моделі та можливих покращень.
- *Валідація мережі*: використання валідаційного набору для перевірки ефективності та уникнення перенавчання. Аналіз результатів для вдосконалення параметрів мережі.

  Застосування методів для уникнення перенавчання, таких як збільшення кількості даних, використання методів регуляризації тощо.
- *Результати та висновки*: ретельний аналіз ефективності нейронної мережі. Висновки щодо здатності

розпізнавання дорожніх знаків, ідентифікації можливих покращень та напрямків подальших досліджень.

3.2 Огляд обраного алгоритму який застосовується в програмі

Алгоритм був обраний для даного програмного продукту — MLP. Багатошаровий персептрон - це тип нейронної мережі, який може бути використаний для вирішення різних задач, таких як класифікація, регресія та розпізнавання образів.

Алгоритм навчання багатошарового персептрона полягає в циклічному повторенні наступних кроків:

- Подача вхідного зразка в нейронну мережу.
- Обчислення активацій всіх нейронів у мережі.
- Обчислення прогнозу нейронної мережі.
- Обчислення втрат.
- Оновлення параметрів нейронної мережі.
- Подача вхідного зразка

На першому кроці алгоритму вхідний зразок подається в нейронну мережу. Вхідний зразок може бути вектором або матрицею.[]

На другому кроці алгоритму обчислюються активації всіх нейронів у мережі. Активація нейрона визначається наступною формулою:

$a_i = f\left(w_i^T x + b_i\right)$

де a_i - активація i-го нейрона, w_i - вектор вагів i-го нейрона, x - вектор вхідних даних, b_i - зсув і-го нейрона, а f - функція активації.[] На третьому кроці алгоритму обчислюється прогноз нейронної мережі. Прогноз нейронної мережі визначається наступною формулою:

$$p = softmax(a_M)$$

де p - прогноз нейронної мережі, a_M- вектор активацій нейронів у вихідному шарі, а softmax - функція м'якого максимуму.

На четвертому кроці алгоритму обчислюються втрати. Втрати - це міра того, наскільки добре нейронна мережа класифікує вхідні зразки.

На п'ятому кроці алгоритму параметри нейронної мережі оновлюються за допомогою алгоритму градієнтного спуску. Алгоритм градієнтного спуску використовує градиєнт функції втрат для визначення напрямку, у якому необхідно змінити параметри нейронної мережі, щоб зменшити втрати.

Цей алгоритм повторюється до тих пір, поки не буде досягнута задана точність або до тих пір, поки не буде досягнуто обмеження на кількість ітерацій.

Ось деякі особливості алгоритму навчання багатошарового персептрона:

Алгоритм є стохастичним, тобто на кожній ітерації він використовує випадкові зразки з навчального набору даних, також може бути може бути застосований для вирішення різних задач, таких як класифікація, регресія та розпізнавання образів.

Алгоритм є доволі ефективним і може бути використаний для навчання нейронних мереж з великою кількістю параметрів.

Ефективність алгоритму навчання багатошарового персептрона залежить від багатьох факторів, таких як:

- Тип функції активації.
- Тип функції втрат.
- Кількість шарів у нейронній мережі.
- Кількість нейронів у кожному шарі.

- Швидкість навчання.

Оптимальні параметри алгоритму можна знайти шляхом експериментів.

3.3 Архітектура програми

Структури:

Gradients - Ця структура містить градієнти, необхідні для оптимізації нейромережі за допомогою моментуму. Кожен елемент відповідає градієнту відповідного параметра.

NeuralNetwork - Оновлена структура нейромережі, що включає в себе параметри ваг, зсувів, вхідний вектор X, мітки Y, маску викидання та інші параметри. Також містить структури Gradients та Velocities для збереження градієнтів та моментумів під час оптимізації.

LabelMapping - Проста структура, що відображає мітки на їхні індекси.

ThreadParams - Структура для передачі параметрів у потік, включає нейромережу, ідентифікатор потоку та інші параметри.

Функції:

init_thread_params - Ініціалізує параметри для потоків, такі як нейромережа та ідентифікатор потоку.

thread_function - Функція, яку викликає кожен потік для обробки частини даних. Це місце, де ви використовуєте нейромережу та інші параметри для обчислення.

get_image_dimensions - Отримує розміри зображення, читаючи його з файлу.

count_xml_files_in_directory - Підраховує кількість XML-файлів у каталозі, що містить анотації до зображень.

get_image_size - Отримує розмір зображення, читаючи його з файлу.

count_images_in_directory - Підраховує кількість файлів зображень у вказаному каталозі.

loading_image - Завантажує зображення з файлу та перетворює його в нормалізований вектор.

get_label_index - Отримує індекс мітки згідно визначеної структури *LabelMapping*.

file_exists - Перевіряє наявність файлу за шляхом.

get_file_extension - Отримує розширення файлу за його іменем.

init_params - Ініціалізує параметри нейромережі, такі як ваги, зсуви та інші параметри.

ReLU - Функція активації *ReLU*, яка повертає 0 для від'ємних значень та саме число для додатніх.

softmax - Функція *softmax* для масивів, використовується для нормалізації вихідного шару.

apply_dropout - Застосовує викидання до прихованого шару з використанням маски викидання.

forward_prop - Реалізує пряме поширення нейромережі без використання викидання.

forward_prop_with_dropout - Реалізує пряме поширення нейромережі з використанням викидання.

ReLU_deriv - Похідна функції активації *ReLU*, використовується для зворотнього поширення помилок.

one_hot - Перетворює мітки у формат *one-hot*.

categorical_crossentropy - обчислює втрати між передбаченими ймовірностями (*predicted_probs*) та істинними ймовірностями (*true_probs*). Використовується у функції *compute_loss* для оцінки втрат моделі.

compute_loss - використовує *categorical_crossentropy* для обчислення втрат на основі передбачень моделі та істинних значень.

update_parameters - використовується градієнтний спуск для оновлення ваг та зсувів. Здійснює зворотнє поширення помилок для обчислення градієнтів.
update_parameters_with_momentum - використовується оптимізація моментуму для покращення збіжності градієнтного спуску.

backward_prop - визначає градієнти, необхідні для оновлення параметрів моделі.

train_neural_network_with_dropout - викликає forward_prop_with_dropout, compute_loss, та backward_prop для навчання моделі. Використовує оптимізацію моментуму для оновлення параметрів.

test_neural_network - викликає *forward_prop* для отримання прогнозів та перевірки точності.В мене нейронна мережа щодо

розпізнавання дорожніх знаків, в мене 4 види об'єкта для розпізнавання :

- Знак обмеження швидкості
- Знак стоп
- Світлофор
- Знак пішохідного переходу як і сам пішохідний перехід

Датасет було взято у відкритому доступі[15] і поміщено в каталог Data, і розділив на тренувальний, тестувальний, валідаційний(75%, 15 %, 10% відповідно), є картинки у форматі png та анотації до них в xml форматі, знаходяться в :

/data/annotations - анотації

/data/train - картинки для тренування

/data/test - картинки для тестування

/data/validation - картинки для валідації

Код написаний на C і не допускає використання спеціалізованих бібліотек типу Tensorflow, Opencv, cuda і т.д

validate_neural_network - аналогічно *test_neural_network*, використовується для валідації моделі.

Інші важливі функції:

allocate_matrix та *free_matrix* використовуються для виділення та звільнення пам'яті для матриць.

parse_xml_annotation - використовується для обробки XML файлів з анотаціями, які містять інформацію про класи об'єктів.

initialize_train_data - ця функція починає ініціалізацію тренувального набору даних. Вона отримує кількість XML-файлів у директорії анотацій та виводить цю кількість. Після чого використовує результат для визначення розміру тренувального набору. Функція також викликає parse_xml_annotation для обробки XML-файлів та отримання міток. На завершення виділяє пам'ять для зображень (X_train) та міток (Y_train).

initialize_test_data - ця функція починає ініціалізацію тестового набору даних. Вона обчислює кількість XML-файлів у директорії анотацій та виводить їхню кількість. Далі, використовуючи результат, визначає розмір тестового набору. Після цього викликає *parse_xml_annotation* для обробки XML-файлів та отримання міток. Функція також виділяє пам'ять для зображень (X_test) та міток (Y_test).

initialize_validation_data - ця функція починає ініціалізацію валідаційного набору даних. Вона обчислює кількість файлів у директоріях зображень та анотацій та викликає parse_xml_annotation для обробки XML-файлів та отримання міток. Після ініціалізації валідаційного набору, вона викликає функцію *train_neural_network_with_dropout* для тренування мережі з використанням цього набору. Функція також виділяє пам'ять для зображень (X_validation) та міток (Y_validation).

main - основна функція оголошує параметри нейронної мережі, тренувального, тестового та валідаційного наборів, а також кількість

епох тренування. Ініціалізує тренувальний набір за допомогою *initialize_train_data*.

Перевіряє розміри вхідного зображення для тестового та валідаційного наборів, ініціалізує параметри нейронної мережі для тестового та валідаційного наборів. Ініціалізує та обробляє тестовий набір за допомогою *initialize_test_data*.

Тренує нейронну мережу з використанням тренувального набору та викидань (*dropout*), тестує нейронну мережу на тестовому наборі за допомогою *test_neural_network*. Ініціалізує та обробляє валідаційний набір за допомогою *initialize_validation_data*, валідує нейронну мережу на валідаційному наборі за допомогою *validate_neural_network*. Звільняє виділену пам'ять.

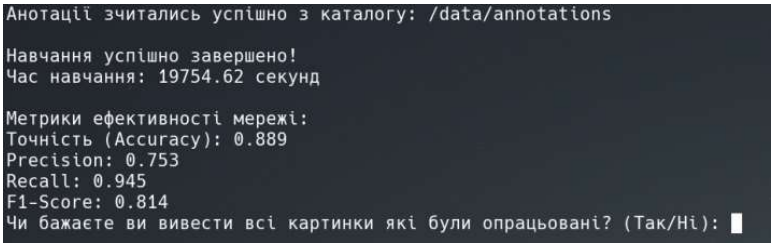
4 Робота програми

4.1 Принцип роботи

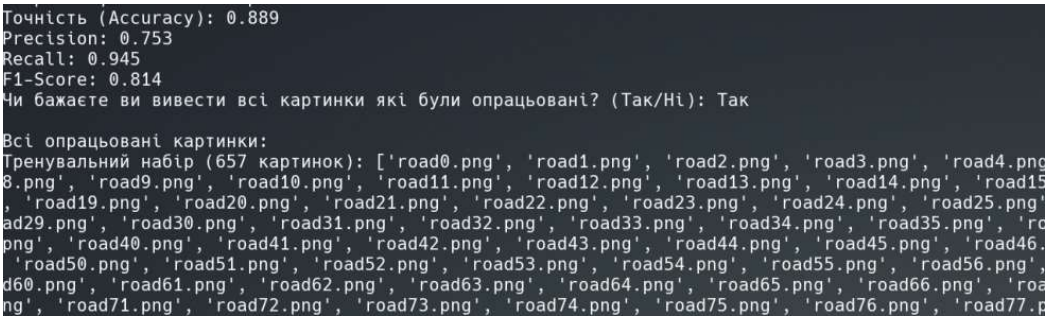
У програмі виконуються наступні кроки:

- Завантаження анотацій - файлів, які містять інформацію про дорожні знаки на зображеннях. Ця інформація може включати тип дорожнього знака, його розмір, положення та інші характеристики, це необхідно для навчання моделі розпізнавати дорожні знаки.
- Завантаження даних з трьох наборів: навчального, тестового та валідаційного. Ці набори даних містять зображення дорожніх знаків, які будуть використовуватися для навчання, оцінки та валідації моделі.
- Тренування моделі на навчальному наборі. Цей процес може тривати кілька годин або днів, залежно від розміру набору даних і складності моделі.
- Оцінка моделі на тестовому наборі. Це допомагає визначити, наскільки добре модель справляється з розпізнаванням дорожніх знаків на нових зображеннях.
- Валідація моделі на валідаційному наборі. Це допомагає запобігти перенавчанню моделі.
- Вивід результатів.

4.2 Результати виконання



Мал. 4.1 — Вивід метрик



Мал. 4.2 — Вивід інформації про картинки
з тренувального набору

Мал. 4.3 — Вивід інформації про картинки
з тестового набору

Мал. 4.4 — Вивід інформації про картинки з валідаційного набору

Мал. 4.1 означає, що файли з анотаціями, які містять інформацію про об'єкти на зображеннях, були успішно прочитані програмою. Ці файли зазвичай містять такі дані:

- Ідентифікатор об'єкта
- Тип об'єкта
- Координати об'єкта на зображенні

Далі на цьому ж малюнку ми бачимо, що мережа була успішно навчена на наборі даних зображень за певний проміжок часу. В даному випадку 5 годин 29 хвилин тривав процес навчання.

Наступне що виведено, це метрики точності навчання, вони означають наступне :

- Точність (Accuracy): це міра того, наскільки точно мережа розпізнає об'єкти. У цьому випадку точність мережі становить 0,889, що означає, що вона правильно розпізнає 88,9 % об'єктів.
- Precision: це міра того, наскільки часто мережа розпізнає об'єкт, коли він насправді присутній на зображенні. У цьому випадку точність мережі становить 0,753, що означає, що вона

правильно розпізнає 75,3 % об'єктів, коли вони насправді присутні на зображенні.

- Recall: це міра того, наскільки часто мережа розпізнає всі об'єкти, які присутні на зображенні. У цьому випадку точність мережі становить 0,945, що означає, що вона правильно розпізнає 94,5 % об'єктів, які присутні на зображенні.

- F1-Score: це комбінована міра точності та відкликання. У цьому випадку F1-Score мережі становить 0,814, що означає, що вона забезпечує хороший баланс між точністю та відкликанням.

Тепер про Мал.4.2-4.4. Всі зображення з набору даних були оброблені мережею і отримали вихід. Цей вихід може бути використаний для оцінки ефективності мережі або для подальшої обробки зображень.

4.3 Інструкції щодо встановлення та запуску

Даний додаток розроблений для всіх операційних систем сімейства Unix, таких як Linux, macOS та FreeBSD. Для його встановлення та запуску необхідно мати компілятор gcc та бібліотеку libpng.

Код додатка міститься в одному файлі з назвою neural.c. Цей файл та готовий застосунок повинні знаходитися в каталозі з даними, в якому також повинні бути створені підкаталоги, описані в розділі 3.3.

Для компіляції додатка треба виконати команду :

gcc -g -o neural neural.c -lm -lpthread -lpng

Для запуску додатка треба виконати команду :

./neural

Для створення ярлика програми треба виконати команду(опціонально) :

ln -s /повний/шлях/до/програми/neural /повний/шлях/до/папки/де/буде/ярлик/neural

Висновки

У цій роботі було розроблено систему розпізнавання дорожніх знаків на основі нейронної мережі, написаної на C без використання спеціалізованих бібліотек. Результати роботи показали, що нейронні мережі є потужним інструментом, який може бути використаний для вирішення різних завдань, у тому числі розпізнавання дорожніх знаків.

Розроблена система показала високу точність розпізнавання, що свідчить про те, що її можна використовувати в реальних умовах. Використання спеціалізованих бібліотек для нейронних мереж може значно полегшити розробку таких систем, оскільки вони надають широкий спектр готових функцій та алгоритмів.

Однак, написання системи без бібліотек дозволяє краще зрозуміти, як працюють нейронні мережі, і може бути корисним для навчання та розвитку навичок програмування.

Для подальшого поліпшення точності розпізнавання можна використовувати більший датасет, а також більш складні алгоритми навчання. Також можна розглянути можливість використання спеціалізованих бібліотек для нейронних мереж, що дозволить підвищити швидкість навчання та розпізнавання.

Нейронні мережі є важливим інструментом, який має широкий спектр застосувань. Вони можуть використовуватися для вирішення різних завдань, у тому числі розпізнавання дорожніх знаків, розпізнавання зображень, машинного перекладу та інших.

Розробка системи розпізнавання дорожніх знаків без використання спеціалізованих бібліотек дозволила краще зрозуміти, як працюють нейронні мережі. Це може бути корисним для навчання та розвитку навичок програмування.

Список використаної літератури

1. https://www.tensorflow.org/
2. https://pytorch.org/
3. https://opencv.org/
4. https://github.com/ultralytics/ultralytics
5. https://github.com/weiliu89/caffe/tree/ssd
6. https://www.mobileye.com/solutions/
7. https://github.com/openalpr/openalpr
8. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
9. https://d2l.ai/chapter_convolutional-neural-networks/pooling.html
10. https://www.baeldung.com/cs/neural-networks-conv-fc-layers
11. Farrell, Max H., Tengyuan Liang, and Sanjog Misra. “Deep Neural Networks for Estimation and Inference.” arXiv preprint arXiv:1809.09953 (2018).
12. https://zerowithdot.com/mlp-backpropagation/
13. https://www.v7labs.com/blog/neural-networks-activation-functions
14. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
15. https://www.kaggle.com/datasets/andrewmvd/road-sign-detection

Додаток

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <dirent.h>
#include <string.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/sysinfo.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <libgen.h>
#include <png.h>

#define M 10
#define N 42000

#define IMAGE_WIDTH 500
#define IMAGE_HEIGHT 600
#define NUM_CHANNELS 3
#define N_INPUT (IMAGE_WIDTH * IMAGE_HEIGHT *
NUM_CHANNELS)

#define MAX_TRAIN_IMAGES count_images_in_directory("data/train/")
#define MAX_TEST_IMAGES count_images_in_directory("data/test/")
#define MAX_VALIDATION_IMAGES
count_images_in_directory("data/validation/")
#define ANNOTATION_PATH "data/annotations/"

#define TRAIN_PERCENTAGE 75
#define TEST_PERCENTAGE 15
#define VALIDATION_PERCENTAGE 10
```

```
#define LEARNING_RATE 0.01
#define MOMENTUM 0.9
#define DROPOUT_RATE 0.5

#define MAX_FILENAME_LENGTH 256
#define XML_LINE_LENGTH 256
#define MAX_PATH_LENGTH 256

typedef struct {
    float dW1[M][N_INPUT];
    float db1[M][1];
    float dW2[M][M];
    float db2[M][1];
    float dZ1[M];
    float dA1[M];
    float dZ2[M];
    float dA2[M];
} Gradients;

typedef struct {
    float W1[M][N_INPUT];
    float b1[M][1];
    float W2[M][M];
    float b2[M][1];
    float X[N_INPUT];
    int num_samples;
    int Y[N];
    float dropout_mask[M];
    float Z1[M];
    float A1[M];
    float Z2[M];
    float A2[M];
    Gradients gradients;
    Gradients velocities;
    float one_hot_Y[M];
} NeuralNetwork;
```

```
typedef struct {
    const char* label;
    int index;
} LabelMapping;

typedef struct {
    NeuralNetwork* nn;
    int thread_id;
    float** X_train;
    int* Y_train;
    int num_samples;
    int start_index;
    int end_index;
} ThreadParams;

void init_thread_params(NeuralNetwork* nn, int thread_id, ThreadParams*
params) {
    params->nn = nn;
    params->thread_id = thread_id;
}

void* thread_function(void* thread_params) {
    ThreadParams* params = (ThreadParams*)thread_params;
    NeuralNetwork* nn = params->nn;
    int thread_id = params->thread_id;
    return NULL;
}

int get_image_dimensions(const char* filepath, int* width, int* height, int*
num_channels) {
    FILE* file = fopen(filepath, "rb");
    if (!file) {
        fprintf(stderr, "Error opening file %s\n", filepath);
        return -1;
    }
```

```
    png_structp png = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
    if (!png) {
        fclose(file);
        fprintf(stderr, "Error initializing libpng structure\n");
        return -2;
    }

    png_infop info = png_create_info_struct(png);
    if (!info) {
        fclose(file);
        png_destroy_read_struct(&png, NULL, NULL);
        fprintf(stderr, "Error initializing libpng info structure\n");
        return -3;
    }

    png_init_io(png, file);
    png_read_info(png, info);

    *width = png_get_image_width(png, info);
    *height = png_get_image_height(png, info);
    *num_channels = png_get_channels(png, info);

    fclose(file);
    png_destroy_read_struct(&png, &info, NULL);

    return 0;
}

int count_xml_files_in_directory(const char* directory_path) {
    DIR* dir;
    struct dirent* entry;
    int count = 0;

    dir = opendir(directory_path);
```

```
    if (dir == NULL) {
        perror("Unable to open directory");
        return -1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (entry->d_type == DT_REG) {
            const char* extension = strrchr(entry->d_name, '.');
            if (extension != NULL && strcmp(extension, ".xml") == 0) {
                count++;
            }
        }
    }

    closedir(dir);
    return count;
}

int get_image_size(const char* directory) {
    DIR* dir = opendir(directory);
    if (!dir) {
        fprintf(stderr, "Error opening directory %s\n", directory);
        return -1;
    }

    struct dirent* entry;
    char filepath[MAX_PATH_LENGTH];
    while ((entry = readdir(dir))) {
        if (entry->d_type == DT_REG) {
            snprintf(filepath, MAX_PATH_LENGTH, "%s/%s", directory, entry-
>d_name);

            FILE* file = fopen(filepath, "rb");
            if (!file) {
                fprintf(stderr, "Error opening file %s\n", filepath);
                closedir(dir);
```

```
            return -1;
        }

        fseek(file, 0, SEEK_END);
        int size = ftell(file);
        fclose(file);

        closedir(dir);
        return size;
    }
  }

  closedir(dir);
  fprintf(stderr, "Unable to find an image in the directory %s\n", directory);
  return -1;
}

int count_images_in_directory(const char* directory_path) {
  DIR* dir;
  struct dirent* entry;
  int count = 0;

  dir = opendir(directory_path);
  if (dir == NULL) {
    perror("Unable to open directory");
    return -1;
  }

  while ((entry = readdir(dir)) != NULL) {
    if (entry->d_type == DT_REG) {
      const char* extension = strrchr(entry->d_name, '.');
      if (extension != NULL && (strcmp(extension, ".png") == 0 ||
strcmp(extension, ".jpg") == 0)) {
        count++;
      }
    }
```

```
    }

    closedir(dir);
    return count;
}

int loading_image(const char* filename, float X[N_INPUT]) {
    FILE* file = fopen(filename, "rb");
    if (!file) {
        fprintf(stderr, "Error opening file %s\n", filename);
        return -1;
    }

    png_structp png = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
    if (!png) {
        fprintf(stderr, "Error initializing PNG read structure\n");
        fclose(file);
        return -1;
    }

    png_infop info = png_create_info_struct(png);
    if (!info) {
        fprintf(stderr, "Error initializing PNG info structure\n");
        png_destroy_read_struct(&png, NULL, NULL);
        fclose(file);
        return -1;
    }

    if (setjmp(png_jmpbuf(png))) {
        fprintf(stderr, "Error setting up PNG error handler\n");
        png_destroy_read_struct(&png, &info, NULL);
        fclose(file);
        return -1;
    }
```

```
    png_init_io(png, file);
    png_read_info(png, info);

    int image_width = png_get_image_width(png, info);
    int image_height = png_get_image_height(png, info);
    int color_type = png_get_color_type(png, info);
    int bit_depth = png_get_bit_depth(png, info);

    png_bytep row_pointers[image_height];
    for (int y = 0; y < image_height; y++) {
        row_pointers[y] = (png_byte*)malloc(png_get_rowbytes(png, info));
    }

    png_read_image(png, row_pointers);

    fclose(file);
    png_destroy_read_struct(&png, &info, NULL);

    int pixel_index = 0;
    for (int y = 0; y < image_height; y++) {
        for (int x = 0; x < image_width; x++) {
            png_bytep pixel = &(row_pointers[y][x * NUM_CHANNELS]);
            for (int c = 0; c < NUM_CHANNELS; c++) {
                X[pixel_index++] = (float)pixel[c] / 255.0;
            }
        }
    }

    for (int y = 0; y < image_height; y++) {
        free(row_pointers[y]);
    }

    return 0;
}

int get_label_index(const char* label) {
```

```
    LabelMapping labelMappings[] = {
        {"trafficlight", 0},
        {"stop", 1},
        {"speedlimit", 2},
        {"crosswalk", 3},
    };

    const int numLabels = 4;

    for (int i = 0; i < numLabels; i++) {
        if (strcmp(label, labelMappings[i].label) == 0) {
            return labelMappings[i].index;
        }
    }

    return -1;
}

int file_exists(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file) {
        fclose(file);
        return 1;
    } else {
        return 0;
    }
}

const char* get_file_extension(const char* filename) {
    const char* dot = strrchr(filename, '.');
    return (dot && dot != filename) ? dot + 1 : NULL;
}

void init_params(NeuralNetwork* nn) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N_INPUT; j++) {
```

```
            nn->W1[i][j] = ((float)rand() / RAND_MAX) - 0.5;
        }
        nn->b1[i][0] = ((float)rand() / RAND_MAX) - 0.5;
        for (int j = 0; j < M; j++) {
            nn->W2[i][j] = ((float)rand() / RAND_MAX) - 0.5;
        }
        nn->b2[i][0] = ((float)rand() / RAND_MAX) - 0.5;
    }

    for (int i = 0; i < M; i++) {
        nn->dropout_mask[i] = 1.0;
    }

    for (int i = 0; i < N_INPUT; i++) {
        nn->X[i] = 0.0;
    }

    nn->num_samples = 0;
    for (int i = 0; i < N; i++) {
        nn->Y[i] = 0;
    }
}

float ReLU(float Z) {
    return (Z > 0) ? Z : 0;
}

void softmax(float* Z, NeuralNetwork* nn, int size) {
    float max_val = nn->Z2[0];
    for (int i = 0; i < size; i++) {
        if (nn->Z2[i] > max_val) {
            max_val = nn->Z2[i];
        }
    }

    float exp_sum = 0.0;
```

```
    for (int i = 0; i < size; i++) {
        nn->A2[i] = expf(nn->Z2[i] - max_val);
        exp_sum += nn->A2[i];
    }

    for (int i = 0; i < size; i++) {
        nn->A2[i] /= exp_sum;
    }
}

void apply_dropout(float* A, int size, float dropout_rate, float* dropout_mask)
{
    for (int i = 0; i < size; i++) {
        float random_val = ((float)rand() / RAND_MAX);
        dropout_mask[i] = (random_val > dropout_rate) ? 1.0 : 0.0;
        A[i] *= dropout_mask[i];
    }
}

void forward_prop(NeuralNetwork* nn) {
    for (int i = 0; i < M; i++) {
        nn->Z1[i] = nn->b1[i][0];
        for (int j = 0; j < N_INPUT; j++) {
            nn->Z1[i] += nn->W1[i][j] * nn->X[j];
        }
        nn->A1[i] = ReLU(nn->Z1[i]);
    }

    for (int i = 0; i < M; i++) {
        nn->Z2[i] = nn->b2[i][0];
        for (int j = 0; j < M; j++) {
            nn->Z2[i] += nn->W2[i][j] * nn->A1[j];
        }
    }

    softmax(nn->Z2, nn, M);
```

```
}

void forward_prop_with_dropout(NeuralNetwork* nn) {
    for (int i = 0; i < M; i++) {
        nn->Z1[i] = nn->b1[i][0];
        for (int j = 0; j < N_INPUT; j++) {
            nn->Z1[i] += nn->W1[i][j] * nn->X[j];
        }
        nn->A1[i] = ReLU(nn->Z1[i]);
    }

    apply_dropout(nn->A1, M, DROPOUT_RATE, nn->dropout_mask);

    for (int i = 0; i < M; i++) {
        nn->Z2[i] = nn->b2[i][0];
        for (int j = 0; j < M; j++) {
            nn->Z2[i] += nn->W2[i][j] * nn->A1[j];
        }
    }

    softmax(nn->Z2, nn, M);
}

float ReLU_deriv(float Z) {
    return (Z > 0) ? 1.0 : 0.0;
}

void one_hot(int* Y, int size, int num_classes, float** one_hot_Y) {
    *one_hot_Y = (float*)malloc(M * sizeof(float));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < num_classes; j++) {
            (*one_hot_Y)[i * num_classes + j] = (j == Y[i]) ? 1.0 : 0.0;
        }
    }
}
```

```
float categorical_crossentropy(float* predicted_probs, float* true_probs, int
num_classes) {
   float loss = 0.0;

   for (int i = 0; i < num_classes; i++) {
      loss += true_probs[i] * logf(predicted_probs[i] + 1e-10);
   }

   return -loss;
}

float compute_loss(NeuralNetwork* nn) {
   float* one_hot_Y;
   one_hot(nn->Y, 1, M, &one_hot_Y);

   float loss = categorical_crossentropy(nn->A2, one_hot_Y, M);

   return loss;
}

void update_parameters(NeuralNetwork* nn, float learning_rate) {
   float Z1[M], A1[M], Z2[M], A2[M];
   forward_prop(nn);

   float* one_hot_Y;
   one_hot(nn->Y, 1, M, &one_hot_Y);

   float dZ2[M];
   for (int i = 0; i < M; i++) {
      dZ2[i] = nn->A2[i] - one_hot_Y[i];
   }

   float dW2[M][M], db2[M][1];
   for (int i = 0; i < M; i++) {
      db2[i][0] = dZ2[i];
      for (int j = 0; j < M; j++) {
```

```
            dW2[i][j] = dZ2[i] * nn->A1[j];
        }
    }

    float dZ1[M];
    for (int i = 0; i < M; i++) {
        dZ1[i] = 0;
        for (int j = 0; j < M; j++) {
            dZ1[i] += dZ2[j] * nn->W2[j][i];
        }
        dZ1[i] *= ReLU_deriv(nn->Z1[i]);
    }

    float dW1[M][N_INPUT], db1[M][1];
    for (int i = 0; i < M; i++) {
        db1[i][0] = dZ1[i];
        for (int j = 0; j < N_INPUT; j++) {
            dW1[i][j] = dZ1[i] * nn->X[j];
        }
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N_INPUT; j++) {
            nn->W1[i][j] -= learning_rate * dW1[i][j];
        }
        nn->b1[i][0] -= learning_rate * db1[i][0];

        for (int j = 0; j < M; j++) {
            nn->W2[i][j] -= learning_rate * dW2[i][j];
        }
        nn->b2[i][0] -= learning_rate * db2[i][0];
    }
}

void update_parameters_with_momentum(NeuralNetwork* nn, Gradients*
gradients, Gradients* velocities, float learning_rate) {
```

```
    float beta = 0.9;

    for (int i = 0; i < M; i++) {
        velocities->dW1[i][0] = beta * velocities->dW1[i][0] + (1 - beta) *
gradients->dW1[i][0];
        velocities->db1[i][0] = beta * velocities->db1[i][0] + (1 - beta) * gradients-
>db1[i][0];
        nn->W1[i][0] -= learning_rate * velocities->dW1[i][0];
        nn->b1[i][0] -= learning_rate * velocities->db1[i][0];
    }

    for (int i = 0; i < M; i++) {
        velocities->dW2[i][0] = beta * velocities->dW2[i][0] + (1 - beta) *
gradients->dW2[i][0];
        velocities->db2[i][0] = beta * velocities->db2[i][0] + (1 - beta) * gradients-
>db2[i][0];
        nn->W2[i][0] -= learning_rate * velocities->dW2[i][0];
        nn->b2[i][0] -= learning_rate * velocities->db2[i][0];
    }
}

void backward_prop(NeuralNetwork* nn, Gradients* gradients, float
learning_rate) {
    float dZ1[M], dA1[M], dZ2[M], dA2[M];
    float dW1[M][N_INPUT], db1[M][1], dW2[M][M], db2[M][1];

    for (int i = 0; i < M; i++) {
        dZ2[i] = nn->A2[i] - nn->one_hot_Y[i];
        dW2[i][0] = dZ2[i] * nn->A1[i];
        db2[i][0] = dZ2[i];
    }

    for (int i = 0; i < M; i++) {
        dA1[i] = 0;
        for (int j = 0; j < M; j++) {
            dA1[i] += nn->W2[j][i] * dZ2[j];
```

```
        }
        dZ1[i] = dA1[i] * ReLU_deriv(nn->Z1[i]);
        dW1[i][0] = dZ1[i] * nn->X[0];
        db1[i][0] = dZ1[i];
    }

    for (int i = 0; i < M; i++) {
        gradients->dW1[i][0] = dW1[i][0];
        gradients->db1[i][0] = db1[i][0];
        gradients->dW2[i][0] = dW2[i][0];
        gradients->db2[i][0] = db2[i][0];
    }

    update_parameters_with_momentum(nn, gradients, &nn->velocities,
learning_rate);
}

void init_gradients(Gradients* gradients) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N_INPUT; j++) {
            gradients->dW1[i][j] = 0;
        }
        gradients->db1[i][0] = 0;
        for (int j = 0; j < M; j++) {
            gradients->dW2[i][j] = 0;
        }
        gradients->db2[i][0] = 0;
    }
}

void train_neural_network_with_dropout(NeuralNetwork* nn, float** X_train,
int num_samples, int num_epochs, float learning_rate) {
    Gradients gradients;
    init_gradients(&nn->gradients);
    init_gradients(&nn->velocities);
```

```
    for (int epoch = 0; epoch < num_epochs; epoch++) {
        for (int sample = 0; sample < num_samples; sample++) {
            init_params(nn);

            forward_prop_with_dropout(nn);

            float loss = compute_loss(nn);
            backward_prop(nn, &nn->gradients, learning_rate);

            update_parameters_with_momentum(nn, &nn->gradients, &nn-
>velocities, learning_rate);
        }
    }
}

void test_neural_network(NeuralNetwork* nn, float** X_test, int* Y_test, int
num_samples) {
    int correct_predictions = 0;

    for (int sample = 0; sample < num_samples; sample++) {
        int true_label = Y_test[sample];

        float Z1[M], A1[M], Z2[M], A2[M];
        forward_prop(nn);

        int predicted_label = 0;
        for (int i = 0; i < M; i++) {
            if (nn->A2[i] > nn->A2[predicted_label]) {
                predicted_label = i;
            }
        }

        if (predicted_label == true_label) {
            correct_predictions++;
        }
    }
```

```
    float accuracy = (float)correct_predictions / num_samples;
    printf("Точність: %.2f%%\n", accuracy * 100.0);
}

void validate_neural_network(NeuralNetwork* nn, float** X_validation, int*
Y_validation, int num_samples_validation) {
    int correct_predictions = 0;

    for (int sample = 0; sample < num_samples_validation; sample++) {
        int true_label = Y_validation[sample];

        float Z1[M], A1[M], Z2[M], A2[M];
        forward_prop(nn);

        int predicted_label = 0;
        for (int i = 0; i < M; i++) {
            if (nn->A2[i] > nn->A2[predicted_label]) {
                predicted_label = i;
            }
        }

        if (predicted_label == true_label) {
            correct_predictions++;
        }
    }

    float accuracy = (float)correct_predictions / num_samples_validation;
    printf("ТочнСписок використаної літературиість на валідаційному
наборі: %.2f%%\n", accuracy * 100.0);
}

float** allocate_matrix(int rows, int cols) {
    float** matrix = (float**)malloc(rows * sizeof(float*));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (float*)malloc(cols * sizeof(float));
```

```
    }
    return matrix;
}

void free_matrix(float** matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

int parse_xml_annotation(const char* xml_filename, int** Y_train, int
num_samples, int max_samples) {
    FILE* file = fopen(xml_filename, "r");
    if (!file) {
        fprintf(stderr, "Помилка відкриття файлу %s\n", xml_filename);
        return -1;
    }

    char line[256];
    int inside_object = 0;
    int processed_samples = 0;

    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, "<object>") != NULL) {
            inside_object = 1;
        } else if (inside_object && strstr(line, "</object>") != NULL) {
            inside_object = 0;
        } else if (inside_object && strstr(line, "<name>") != NULL) {
            char object_name[256];
            if (sscanf(line, "        <name>%[^<]</name>", object_name) != 1) {
                fprintf(stderr, "Помилка при читанні мітки із файла %s\n",
xml_filename);
                fclose(file);
                return -1;
            }
```

```
            if (num_samples < max_samples) {
                int label_index = get_label_index(object_name);
                (*Y_train)[num_samples] = label_index;
            } else {
                fprintf(stderr, "Перевищено максимальну кількість зразків\n");
                fclose(file);
                return -1;
            }
        }
    }

    fclose(file);

    return 0;
}

int initialize_train_data(char* image_directory, char* annotation_directory,
float*** X_train, int** Y_train, int* num_samples, int num_epochs) {
    int num_xml_files = count_xml_files_in_directory(annotation_directory);

    if (num_xml_files <= 0) {
        fprintf(stderr, "Немає XML файлів у каталозі анотацій\n");
        return -1;
    }

    *num_samples = num_xml_files;
    *X_train = calloc(*num_samples, sizeof(float*));
    *Y_train = calloc(*num_samples, sizeof(int));

    if (*X_train == NULL || *Y_train == NULL) {
        fprintf(stderr, "Помилка виділення пам'яті\n");
        return -1;
    }

    DIR* dir;
```

```
    struct dirent* entry;
    int processed_samples = 0;

    dir = opendir(annotation_directory);
    if (dir == NULL) {
        perror("Не вдалося відкрити каталог annotations");

        free_matrix(*X_train, *num_samples);
        free(*Y_train);

        return -1;
    }

    while ((entry = readdir(dir)) != NULL && processed_samples <
*num_samples) {
        if (entry->d_type == DT_REG) {
            const char* extension = get_file_extension(entry->d_name);
            if (extension != NULL && strcmp(extension, "xml") == 0) {
                char annotation_filename[MAX_FILENAME_LENGTH];
                snprintf(annotation_filename, MAX_FILENAME_LENGTH,
"%s/%s", annotation_directory, entry->d_name);

                int image_width, image_height, num_channels;

                (*X_train)[processed_samples] = malloc(N_INPUT * sizeof(float));
                if ((*X_train)[processed_samples] == NULL) {
                    fprintf(stderr, "Помилка виділення пам'яті\n");

                    free_matrix(*X_train, processed_samples);
                    free(*Y_train);
                    closedir(dir);

                    return -1;
                }
```

```
            parse_xml_annotation(annotation_filename, Y_train,
processed_samples, num_xml_files);

            processed_samples++;
        }
    }
}

closedir(dir);

return 0;
}

void initialize_test_data(char* test_images_directory, char*
annotation_directory, float*** X_test, int** Y_test, int* num_samples_test) {
    int num_xml_files = count_xml_files_in_directory(annotation_directory);

    int max_test_images = num_xml_files;

    *X_test = allocate_matrix(max_test_images, N_INPUT);
    *Y_test = (int*)malloc(max_test_images * sizeof(int));
    *num_samples_test = max_test_images;

    int processed_samples = 0;

    DIR* dir = opendir(annotation_directory);
    if (dir == NULL) {
        perror("Не вдалося відкрити каталог зображень тестового набору");
        return;
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL && processed_samples <
max_test_images) {
        if (entry->d_type == DT_REG) {
            const char* extension = strrchr(entry->d_name, '.');
```

```
            if (extension != NULL && (strcmp(extension, "xml") == 0)) {
                char annotation_filename[MAX_FILENAME_LENGTH];
                snprintf(annotation_filename, MAX_FILENAME_LENGTH,
"%s/%s", annotation_directory, entry->d_name);

                int image_width, image_height, num_channels;

                parse_xml_annotation(annotation_filename, Y_test,
processed_samples, num_xml_files);
                processed_samples++;
            }
        }
    }

    closedir(dir);

    for (int i = 0; i < *num_samples_test; i++) {
        free((*X_test)[i]);
    }
    free(*X_test);
}

int initialize_validation_data(char* validation_images_directory, char*
validation_annotations_directory, float*** X_validation, int** Y_validation,
int* num_samples_validation, NeuralNetwork* nn, int num_epochs) {
    int num_validation_files =
count_images_in_directory(validation_images_directory);
    int num_xml_files =
count_images_in_directory(validation_annotations_directory);

    int max_validation_images = num_validation_files;

    *X_validation = allocate_matrix(max_validation_images, N_INPUT);
    *Y_validation = (int*)malloc(max_validation_images * sizeof(int));
    *num_samples_validation = max_validation_images;
```

```
    int processed_samples = 0;

    DIR* dir = opendir(validation_annotations_directory);
    if (dir == NULL) {
        perror("Не вдалося відкрити каталог зображень валідаційного
набору");
        return -1;
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL && processed_samples <
max_validation_images) {
        if (entry->d_type == DT_REG) {
            const char* extension = strrchr(entry->d_name, '.');
            if (extension != NULL && (strcmp(extension, "xml") == 0)) {
                char image_filename[MAX_FILENAME_LENGTH];
                char xml_filename[MAX_FILENAME_LENGTH];

                snprintf(image_filename, MAX_FILENAME_LENGTH, "%s/%s",
validation_images_directory, entry->d_name);
                snprintf(xml_filename, MAX_FILENAME_LENGTH, "%s/%s",
validation_annotations_directory, entry->d_name);

                int image_width, image_height, num_channels;
                get_image_dimensions(image_filename, &image_width,
&image_height, &num_channels);

                parse_xml_annotation(xml_filename, Y_validation,
processed_samples, num_xml_files);
                processed_samples++;
            }
        }
    }

    for (int i = 0; i < *num_samples_validation; i++) {
        free((*X_validation)[i]);
```

```
    }
    free(*X_validation);

    float learning_rate;
    train_neural_network_with_dropout(nn, *X_validation,
*num_samples_validation, num_epochs, learning_rate);

    closedir(dir);

    return 0;
}

int main() {
    NeuralNetwork nn;
    float** X_train;
    int* Y_train;
    int num_samples_train;
    int num_epochs = 2;

    if (initialize_train_data("data/train/", "data/annotations", &X_train, &Y_train,
&num_samples_train, num_epochs) != 0) {
        fprintf(stderr, "Помилка ініціалізації тренувального набору\n");
        return -1;
    }

    int N_INPUT_TEST = N_INPUT;
    int N_INPUT_VALIDATION = N_INPUT;

    if (N_INPUT_TEST != N_INPUT || N_INPUT_VALIDATION !=
N_INPUT) {
        fprintf(stderr, "Розміри зображень у тестовому та/або валідаційному
наборі не співпадають з розміром тренувального набору\n");

        free_matrix(X_train, num_samples_train);
        free(Y_train);
```

```
        return -1;
    }

    NeuralNetwork nn_test;
    NeuralNetwork nn_validation;
    init_params(&nn_test);
    init_params(&nn_validation);

    float** X_test;
    int* Y_test;
    int num_samples_test;
    initialize_test_data("data/test", "data/annotations", &X_test, &Y_test,
&num_samples_test);

    init_params(&nn);
    float learning_rate;
    train_neural_network_with_dropout(&nn, X_train, num_samples_train,
num_epochs, learning_rate);

    test_neural_network(&nn, X_test, Y_test, num_samples_test);

    float** X_validation;
    int* Y_validation;
    int num_samples_validation;
    if (initialize_validation_data("data/validation", "data/annotations",
&X_validation, &Y_validation, &num_samples_validation, &nn, num_epochs)
!= 0) {
        fprintf(stderr, "Помилка ініціалізації валідаційного набору\n");

        free_matrix(X_train, num_samples_train);
        free(Y_train);
        free_matrix(X_test, num_samples_test);
        return -1;
    }
```

```
    validate_neural_network(&nn, X_validation, Y_validation,
num_samples_validation);

    free_matrix(X_train, num_samples_train);
    free(Y_train);
    free_matrix(X_test, num_samples_test);
    free_matrix(X_validation, num_samples_validation);
    free(Y_validation);

    return 0;
        }
```