Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: **Програмно-алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA**

Виконав: студент VI курсу групи КН-61(м)
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Северенюк М.Р.
(прізвище та ініціали)

Керівник проф.Соколовський Я.І.
(прізвище та ініціали)

Рецензент проф.Кособуцький П.С.
(прізвище та ініціали)

Львів – 2022

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Крошний І. М.
"____" _____2022_ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Северенюк Михайло Русланович
(прізвище, ім'я, по батькові)

1. Тема роботи Програмно-алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA

керівник роботи *д.т.н., проф.Соколовський Я.І.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "_13_"грудня 2021 року № С-617
2. Термін подання студентом роботи 12 грудня 2022 року
3. Вихідні дані до роботи Формулювання задачі та її формалізація. Аналіз існуючих структур даних. Огляд алгоритмів та програмного забезпечення для реалізації технічного завдання.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Стан проблемної області
2. Інформаційне забезпечення
3.Математичне забезпечення
4.Програмне забезпечення
5.Розроблення стартап - проекту
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди доповіді, актуальність теми, постановка завдання, реалізація розробленої програмної системи, аналіз отриманих результатів, висновки

6. Дата видачі завдання 15 грудня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Системний аналіз стану проблемної області. Огляд літературних джерел згідно досліджувальної теми.	*15.01.2022 р.* *02.02.2022 р.*	виконано
2.	Постановка задачі і етапи проектування	*08.02.2022 р.* *16.02.2022 р.*	виконано
3.	Розроблення математичного та алгоритмічного забезпечення	*07.03.2022 р.* *07.04.2022 р*	виконано
4.	Програмна реалізація та аналіз результатів	*10.05.2022 р.* *30.05.2022 р.*	виконано
5.	Оформлення записки до дипломного проекту	*02.11.2022 р.* *02.12.2022 р.*	виконано
6.	Здача пояснювальної записки на рецензування	*10.12.2022 р.*	виконано
7.	Підготовка доповіді	*15.12.2022 р.* *16.12.2022 р.*	виконано

Студент _____ Северенюк М. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Соколовський Я.І.**
(підпис) (прізвище та ініціали)

ЗМІСТ

АНОТАЦІЇ 5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ 10
1.1 Огляд предметної області 10
1.2 Пошук тексту як задача сегментації зображення 12
1.3 Висновки до розділу 13
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 14
2.1 Синтетичні датасети 14
2.2 Природні датасети 15
2.3 Висновки до розділу 16

РОЗДІЛ 3. МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ 17
3.1 Модель пошуку тексту CRAFT 17
3.2 Алгоритмічне забезпечення навчання ШНМ 19
3.3 Функція втрат для моделі CRAFT 22
3.4 Висновки до розділу 23

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ 24
4.1 Використані технології. Загальні відомості про Chainer та його структуру 24
4.2 Загальні відомості про PyTorch та його структуру 29
4.3 Підготовка даних. 30
4.4 Використання Python фреймворку Chainer 33
4.5 Особливості програмної реалізації моделі CRAFT за допомогою фреймворку PyTorch 37
4.6 Особливості застосування хмарних технологій 43
4.7 Висновки до розділу 44
РОЗДІЛ 5. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ 46
5.1 Аналіз результатів тестового набору даних без тренування 46
5.2 Аналіз результатів даних з тренуванням ШНМ .Використання хмарних технологій 47
5.3 Висновки до розділу 48
РОЗДІЛ 6. Розроблення стартап – проекту 49
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТКИ 55
Додаток А. Формат json файлу з анотаціями до даних з датасету TextOCR 55
Додаток Б. Пакет optimizer 56
Додаток Г. Пакет training 59

АНОТАЦІЯ

Магістерська кваліфікаційна робота студента Северенюка Михайла Руслановича тему: “Програмно-алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA” містить 60 сторінок, 12 джерел за списком використаної літератури та розділ додатків.

У роботі розглянуто задачу пошуку тексту на природних зображеннях за допомогою штучних нейронних мереж. А саме модель CRAFT застосовується до датасету анотованих природних зображень з текстом — TextOCR. Навчання мережі відбувається за допомогою технологій CUDA.

Ключові слова: пошук тексту, розпізнавання тексту, штучна нейронна мережа, згорткова нейронна мережа, Python, PyTorch, CUDA.

The master's qualification work of student Severeniuk Mykhailo Ruslanovich on the topic: “Development of software and algorithms of parallel training of artificial neural networks for text detection and recognition using CUDA technologies” contains 60 pages, 12 sources from the list of references and a section of appendices.

In the work problem of scene text detection based on artificcial neural networks was investigated. Specifically, CRAFT model was applied to the dataset of annotated natural images with text – TextOCR. Network training was performed using CUDA technologies.

Keywords: text detection, text recognition, artificial neural network, convolution neural network, Python, PyTorch, CUDA.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Глибинне навчання	підгалузь машинного навчання, що спеціалізується на штучних нейронних мережах.
ЗНМ	згорткова нейронна мережа.
Машинне навчання	галузь штучного інтелекту, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності "навчатися" (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоб бути програмованими явно.
Карта ознак	від англійського feature map, результат застосування одного окремого фільтра згорткової нейронної мережі.
Розмір порції	від batch size, кількість даних що подаються на вхід нейронної мережі за один крок тренування чи валідації.
ШНМ	штучна нейронна мережа.
CPU	central processing unit (центральний процесор).
GPU	graphics processing unit (графічний процесор).
ICDAR	International Conference on Document Analysis and Recognition
OCR	optical character recognition (оптичне розпізнавання символів)

ВСТУП

Людина дуже добре справляється з розпізнаванням тексту. Якщо десь на зображенні є текст, ми можемо легко його знайти. Навіть якщо він написаний невідомою нам мовою, ми однаково можемо зрозуміти, що це текст. Але в цей же час для комп'ютерів ця задача є надзвичайно складною. Так сучасні алгоритми дозволяють ефективно аналізувати фотографії чи скани документів, де ми маємо справу з однорідним фоном та друкованим текстом. Ця задача є добре досліджена і вже зараз багато де використовується на практиці.

Однак, все стає набагато складніше, коли ми хочемо аналізувати текст на природних зображеннях. Наприклад, це можуть бути фотографії вивіски магазину, номерних знаків автомобіля, меню в ресторані тощо. В такому випадку текст може мати різний колір, стиль та фон, умови за яких зроблено фото також можуть відрізнятись, і взагалі сам текст може займати лише невелику частину кадру. Класичні алгоритми комп'ютерного зору дозволяють вирішувати цю задачу лише в часткових випадках з багатьма обмеженнями. Але хотілося б мати універсальну модель для розпізнавання тексту стійку до більшості з цих перешкод.

Досягти цього можуть допомогти штучні нейронні мережі (ШНМ). Пошук тексту в кадрі можна розглядати як задачу сегментації зображення, у вирішенні яких новітні архітектури ШНМ дуже добре себе продемонстрували. В попередній роботі [1] було розглянуто особливості програмної реалізації процедури навчання штучних нейронних мереж з використанням графічного процесору та хмарних технологій для вирішення задачі виявлення внутрішньочерепних кровотеч за зображеннями отриманими за допомогою комп'ютерної томографії.

Часто одні й ті самі архітектури ШНМ дозволяють вирішити задачі з абсолютно різних сфер життя, вимагаючи лише різних наборів даних. А сама підготовка даних для пошуку тексту є відносно простою, оскільки людське око легко справляється з цією задачею.

Проблемою є те, що навчання ШНМ вимагає підбору мільйонів параметрів та обробки гігабайт інформації, а отже займає надзвичайно багато часу. Технологія CUDA дозволяє, використовуючи GPU для обчислень, значно скоротити необхідний

для тренування час [2]. Однак потужностей GPU середнього цінового сегменту все ще недостатньо для вирішення сучасних задач. Тому використовуються хмарні сервіси, що за погодинну оплату надають доступ до передових графічних процесорів. Їх застосування також буде розглянуто в цій роботі.

Метою роботи є застосування сучасних методів пошуку тексту на зображенні та реалізація штучної нейронної мережі для вирішення цієї задачі,а також розроблення програмно-алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA.

Для досягнення поставленої мети у роботі виконано такі завдання:

1. Проаналізувати наявні розробки в сфері застосування штучних нейронних мереж (ШНМ) до задачі пошуку та розпізнавання тексту на природних зображеннях.
2. Підібрати набір зображень, що містять текст, для тренування ШНМ. Основною вимогою є те, щоб окрім самих фотографій датасет містив також правильно виділені області з текстом.
3. Серед архітектур ШНМ, розглянутих в першому пункті, обрати таку, яка найбільше підходить для вирішення задачі.
4. Реалізувати алгоритм навчання даної моделі на обраному датасеті з використанням технологій CUDA.
5. Модифікувати його, щоб максимізувати точність результатів. Проаналізувати результати

Об’єкт дослідження — текст на зображенні з різними завадами, який необхідно виділити в зрозумілу для комп’ютера форму, для подальшої автоматизації задачі.

Предмет дослідження — штучна нейронна мережа, яка буде займатись пошуком тексту на зображенні.

Наукова новизна полягає в адаптації та застосуванні алгоритму покращення якості пошуку тексту на зображенні на підставі ШНМ , а також в дослідженні розпаралелення навчання ШНМ з використанням технологій CUDA.

Практичне значення роботи полягає в тому, що розроблений програмно-алгоритмічний комплекс може бути застосований для покращення якості пошуку тексту на зображенні ,що дозволятиме ідентифікувати проблемні ділянки тексту.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ

1.1 Огляд предметної області

Задача розпізнавання тексту на природних зображеннях є одним з напрямків оптичного розпізнавання символів (OCR — optical character recognition). В цій сфері вже зараз існують досить надійні рішення для часткового випадку, коли ми працюємо зі сканованими документами. Так бібліотека Tesseract вже кілька років успішно застосовується у багатьох програмах. Але в загальному випадку ця задача досі залишається дуже складною.

Водночас, вона має багато практичних застосувань, таких як пошук за зображенням, автоматизація виробництва, робототехніка, автоматичний переклад тощо. Тому вона привернула увагу багатьох дослідників і зараз дуже активно розвивається. Регулярно проводяться змагання “ICDAR Robust Reading Competition” для яких публікується спеціальний датасет з фотографій, що містять текст, і кожен хто забажає може запропонувати свою модель. Це дозволило значно просунутись у вирішенні цієї задачі.

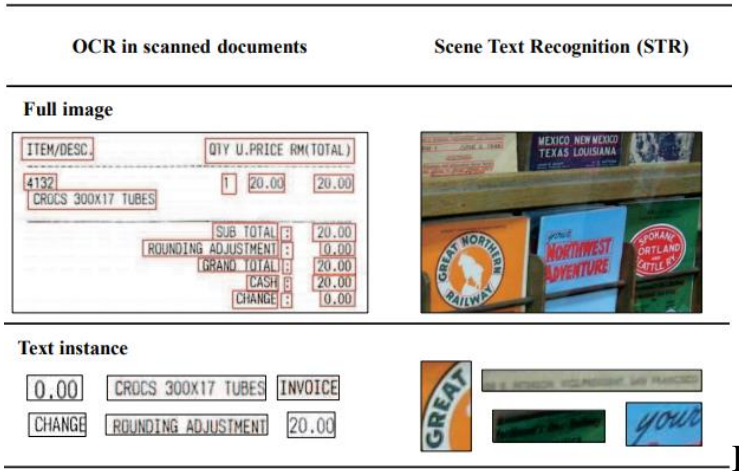


Рисунок 1.1. Пошук тексту для сканованих (ліворуч) та природних (праворуч) зображень [3]

Серед основних проблем, які виникають, коли ми хочемо перейти від роботи зі сканованими документами до аналізу тексту на довільних зображеннях є такі (рис 1.1):

- Складний фон
- Різноманітні кольори, шрифти, розміри та орієнтація тексту
- Нерівномірне освітлення, низька роздільна здатність, розмиття
- Окрім тексту зображення може містити довільні інші об’єкти

Задачу аналізу тексту на зображенні можна поділити на дві основні підзадачі — пошук тексту та його розпізнавання (рис. 1.2). Пошук тексту можна також поділити на два етапи — локалізація (пошук всіх ділянок які потенційно містять текст) та верифікація (виділення серед цих ділянок тих які дійсно містять текст). Задача розпізнавання тексту ж полягає в тому, щоб маючи на вході зображення рядка, слова чи літери отримати текстове представлення. Опційно між цими двома етапами також може застосовуватись сегментація тексту — виділення окремих слів чи літер з ділянки, що містить текст.

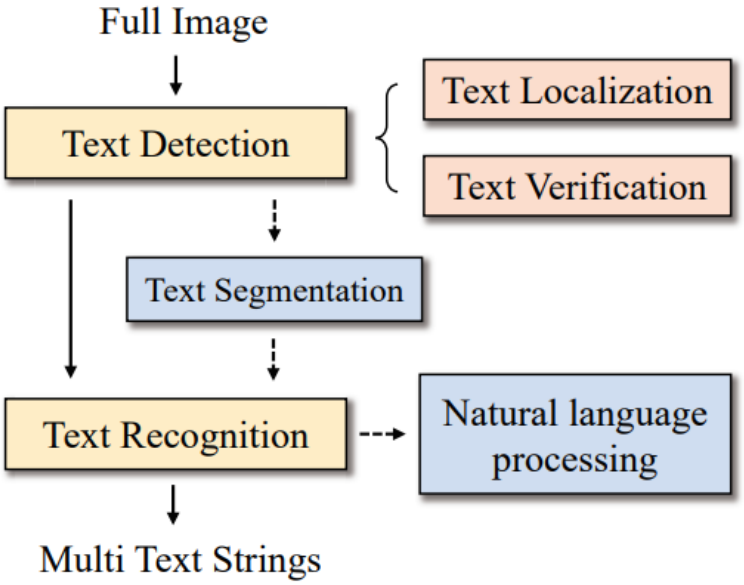


Рисунок 1.2. Основні етапи аналізу тексту на зображенні [3]

В цій роботі ми сфокусуємось саме на завданні пошуку тексту, оскільки для його ефективного розпізнавання критично важливо правильно виділити область з текстом на зображенні.

1.2 Пошук тексту як задача сегментації зображення

Пошук тексту по суті є задачею сегментації зображення. Сегментацією зображення називають процес локалізації об’єктів на зображенні. Це одна з ключових задач комп’ютерного зору. Існує безліч алгоритмів сегментації різного рівня складності. Більшість з них зводяться до того, що кожен піксель відносять до відповідного класу на основі певних спільних для цього класу характеристик, як от яскравість, колір чи текстура. Найпростішим алгоритмом є порогування, метод при якому обирається певне значення (поріг) і всі пікселі з інтенсивностями більшими за поріг відносять до одного класу, а з меншими до іншого. І дійсно такий простий метод можна застосувати за ідеальних умов, коли ми працюємо зі сканованим чорним текстом на білому фоні.

Однак часто, об’єкти які нас цікавлять на зображенні не так просто виділити, оскільки вони можуть бути дуже неоднорідними. Як вже згадувалось, виділити будь-який текст не вдасться просто за спільним кольором чи текстурою. Серед класичних алгоритмів комп’ютерного зору не існує загальних методів сегментації, які б однаково добре працювали у всіх випадках. Кожна задача вимагає індивідуального підходу.

Допомогти вирішити ці проблеми можуть ШНМ. По-перше ті самі архітектури ШНМ можуть використовуватись для вирішення дуже широкого кола задач. А по-друге, підготовка навчальних даних для пошуку тексту є відносно легкою, оскільки людина без особливих зусиль з нею справляється.

В [3] роботі вже було показано, як згорткова ШНМ може бути ефективно застосована до задачі сегментації медичних зображень. В цій роботі розглянемо застосування ШНМ до задачі пошуку тексту.

При дослідженні рішень такого типу дуже часто зустрічаються дві моделі повністю згорткових нейронних мереж:

- EAST (Efficiant and Accurate Scene Text Detector) — швидкий та точний детектор тексту, модель запропоновна в роботі Сі. Жоу та ін. 2017 року [4].
- CRAFT (Character Region Awareness for Text Detection) — модель, що при пошуку тексту звертає увагу на окремі символи, запропонована Ю. Баеком та ін. в 2019 році [5].

Як бачимо це доволі сучасні моделі ШНМ, однак, що цікаво, обидві вони мають U-Net-подібну архітектуру, як і мережа, що розглядалась у минулій роботі для зовсім іншої прикладної задачі [1]. При цьому ці моделі показали дуже добрі результати, та навіть використовуються у деяких бібліотеках як стандартні рішення для пошуку тексту (наприклад EAST детектор зустрічається в найпопулярнішій бібліотеці алгоритмів комп'ютерного зору — OpenCV [6]).

1.3 Висновки до розділу

- Здійснено огляд предметної області.
- Обгрунтовано застосування ШНМ для пошук тексту як задачу сегментації зображення.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Синтетичні датасети

Синтетичні датасети — це такі де на звичайні зображення без тексту алгоритмічно додаються слова різного кольору, шрифту та форми. Такі зображення зазвичай виглядають дуже не природно (рис. 2.1). Однак вони добрі тим, що без значних зусиль дозволяють отримати багато даних, необхідних для тренування складних ШНМ. І на практиці саме такі датасети дозволили досягти значних результатів.

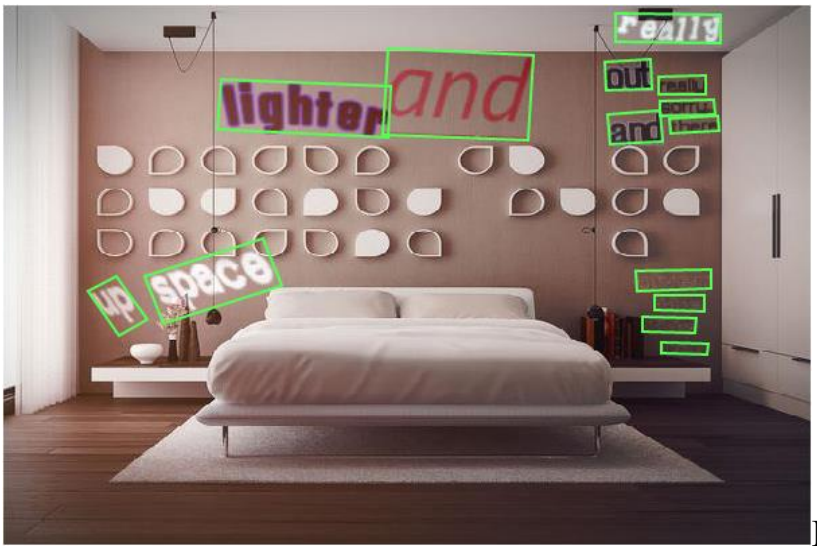


Рисунок 2.1. Приклад згенерованого зображення з датасету SynthText

Серед популярних синтетичних датасетів є такі:

Synth90k — містить 9 мільйонів синтетичних зображень з текстом з набору з 90 000 англійських слів. Слова накладені на природні зображення з випадковими трансформаціями та ефектами, такими як кольори, шрифти, розмиття та шуми. Кожне зображення анотоване словом, яке зображено на ньому для навчання.

SynthText — містить 800 000 зображень з 6 мільйонами синтетичними зразками тексту. Так само слова додаються з випадковим шрифтом і трансформовані відповідно то поверхні на якій знаходяться. І також слова містять анотації для навчання. Окрім цього алгоритм використаний для створення датасету має відкритий вихідний код, тому є можливість згенерувати власні зображення в такому ж стилі.

2.2 Природні датасети

Природні датасети — складаються зі звичайних зображень, що містять текст, попередньо анотований для навчання. По суті це є реальні зображення, з якими, ми очікуємо, буде працювати нейронна мережа на практиці (рис. 2.2). Тому було б добре використовувати їх для тренування. Але переважно вони містять значно менше зображень ніж синтетичні — декілька тисяч файлів. Цього може бути недостатньо, щоб натренувати ШНМ з нуля. Тому такі датасети переважно використовуються для додаткового навчання мережі, що вже була натренована на штучному датасеті, для покращення точності (цю процедуру називають transfer learning). А також для тестування алгоритмів.

Рисунок 2.2. Приклад природного зображення з текстом з датасету TextOCR

Серед популярних природних датасетів такі:

ICDAR 2013 (IC13) — містить 561 зображення: 420 для тренування та 141 для тестування. В цьому датасеті частина даних взята з датасету ICDAR 2003 та додано нові зображення. Загалом позначено 1015 слів.

ICDAR 2015 (IC15) — містить 1500 зображень: 1000 для навчання та 500 для тестування. Загалом позначено 2077 слів, включаючи понад 200 не стандартних (знятих за поганих умов чи таких що мають складну форму).

ICDAR 2017 (IC17) — складається з 7200 зображень для навчання, 1800 валідаційних та 9000 тестових. Вони містять текст 9-ма мовами.

MSRA-TD500 (TD500) — містить 500 зображень: 300 для навчання та 200 для тестування. Зображення містять слова англійською та китайською.

TotalText — презентований на конференції ICDAR 2017, містить 1255 навчальних та 300 тестових зображень, переважно це текст вигнутої форми.

CTW-1500 — складається з 1000 навчальних та 500 тестових зображення, так само переважно вигнутої форми.

TextOCR — великий датасет з природними зображенням, що нещодавно опублікував дослідницький відділ компанії Facebook [7]. Містить близько мільйона позначених слів на 28 134 зображеннях.

Цей останній датасет унікальний тим, що це один з небагатьох набір природних даних з позначеним на них текстом, такого об’єму. Його ми і будемо досліджувати в даній роботі. Він складається з набору зображень з унікальною назвою, та файлу у json форматі, що ставить кожному зображенню у відповідність список виділених ділянок з текстом. Цей файл складається з трьох частин:

- “imgs” — містить інформацію про кожне зображенням
- “anns” — містить інформацію про кожну ділянку з текстом
- “imgToAnns” — кожному зображенню ставить у відповідність список ділянок з текстом

  Детальніше структура цього документа наведена в Додатку А.

2.3 Висновки до розділу

- Проаналізовано структури синтетичних та природніх датасетів.
- Обгрунтовано вибір використання природніх датасетів .

РОЗДІЛ 3. МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Модель пошуку тексту CRAFT

Розглянемо детальніше раніше згадану модель CRAFT. Її схематичне зображення можна бачити на рисунку 3.1. Це повністю згорткова ШНМ подібна до мережі U-Net[1]. Тобто складається з двох симетричних частин — енкодера та декодера.

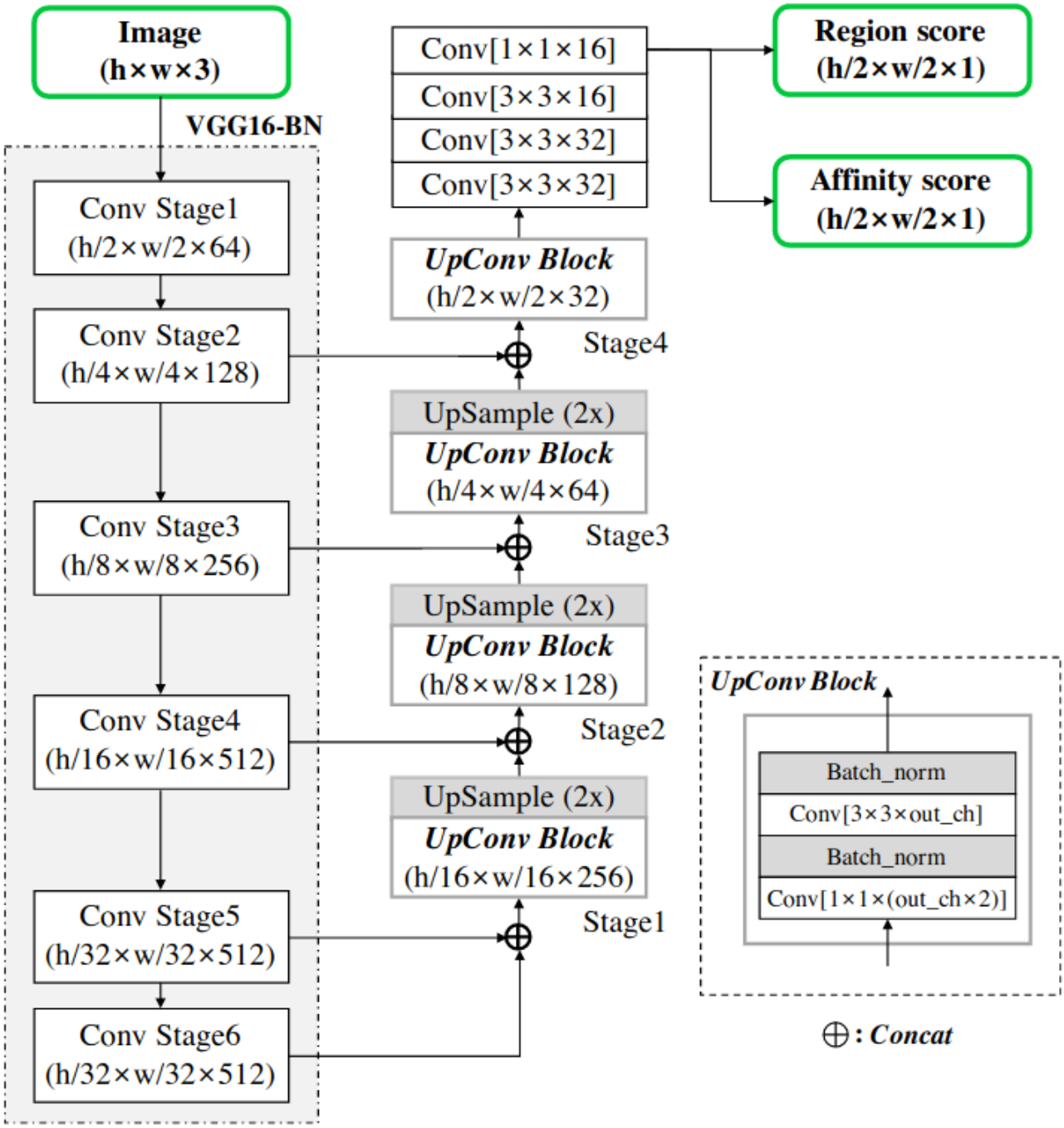


Рисунок 3.1. Схематичне зображення архітектури мережі CRAFT [5]

Енкодер працює як класична ЗНМ, де кожен шар зменшує роздільну здатність вхідного зображення та збільшує кількість карт ознак на виході. Завдяки ньому ми з заданого зображення отримуємо багато карт ознак невеликої роздільної здатності. Декодер ж навпаки, відновлює повнорозмірне зображення з отриманих на вході карт ознак.

В якості енкодера тут використана популярна ЗНМ VGG-16 [8]. Вона складається з шести згорткових блоків. Ці блоки складаються з 2-3 згорток з ядром 3x3, при чому перед згорткою до вхідної матриці додається рамка, через що на виході ми отримуємо матрицю такого ж розміру. Між такими блоками використовується максимізаційне агрегування (max pooling) 2x2 (рис. 3.2) для зменшення роздільної здатності.

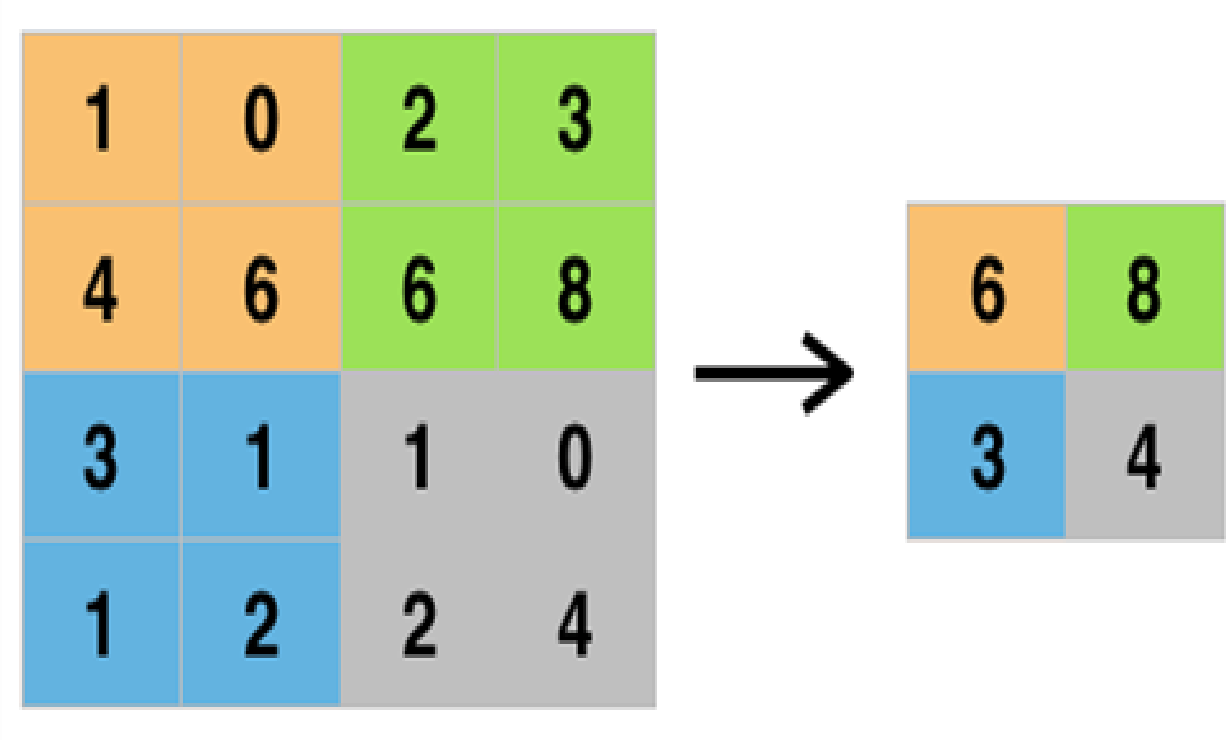


Рисунок 3.2. Приклад максимізаційного агрегування. Вхідне зображення розділяється на блоки та з кожного знаходиться максимум

Декодер ж складається з обернених згорткових блоків (UpConv Block), які зменшують вдвічі кількість вхідних карт ознак, але при цьому повертають зображення вдвічі більшої роздільної здатності на виході.

Окрім цього, як і в мережі U-Net, до вхідних карт ознак для кожного блоку декодера конкатенуються відповідні карти ознак з енкодера, це дозволяє передати додаткову інформацію про низькорівневі ознаки та покращити якість результату.

Останніми шарами декодера є кілька згорток 3x3 та остання згортка 1x1, в результаті чого ми отримуємо два зображення з роздільною здатністю вдвічі меншою ніж у вхідного. Одне з них, Region Score (Оцінка локації), вказує нам локацію окремих символів на зображенні. Інше, Affinity Score (Оцінка спорідненості), використовується для об'єднання окремих символів у слова.

Трактувати ці оцінки можна наступним чином. Оцінка локації $S_r(p)$ для кожного пікселя p вказує імовірність того, що цей піксель є центром літери. А оцінка спорідненості $S_a(p)$ — імовірність того, що даний піксель p є центром регіону між літерами одного слова.

Ці дві оцінки дозволяють легко знайти QuadBox (прямокутник мінімальної площі навколо об'єкту який нас цікавить) для кожного слова, чи для кожної літери окремо [5]. Частіше нас будуть цікавити слова і для їх пошуку автори даної моделі пропонують наступний простий метод:

1. Створити матрицю M заповнену нулями
2. $M(p) = 1$ якщо $S_r(p) > \tau_r$ або $S_a(p) > \tau_a$, де τ_r і τ_a певні порогові значення для локації та спорідненості відповідно
3. Застосувати алгоритм пошуку з'єднаних компонентів
4. Для кожного компонента знайти мінімальний описаний прямокутники

Маючи чотири точки знайденого прямокутника, ми можемо вирізати ділянку з текстом, що нас цікавить і передати його на вхід іншій мережі, яка поверне на виході слово, зображене на цій ділянці.

3.2 Алгоритмічне забезпечення навчання ШНМ

Алгоритмічне забезпечення навчання ШНМ розглянемо на прикладі однієї з найпростіших моделей ШНМ — багатошарового перцептрону (рис. 3.3).

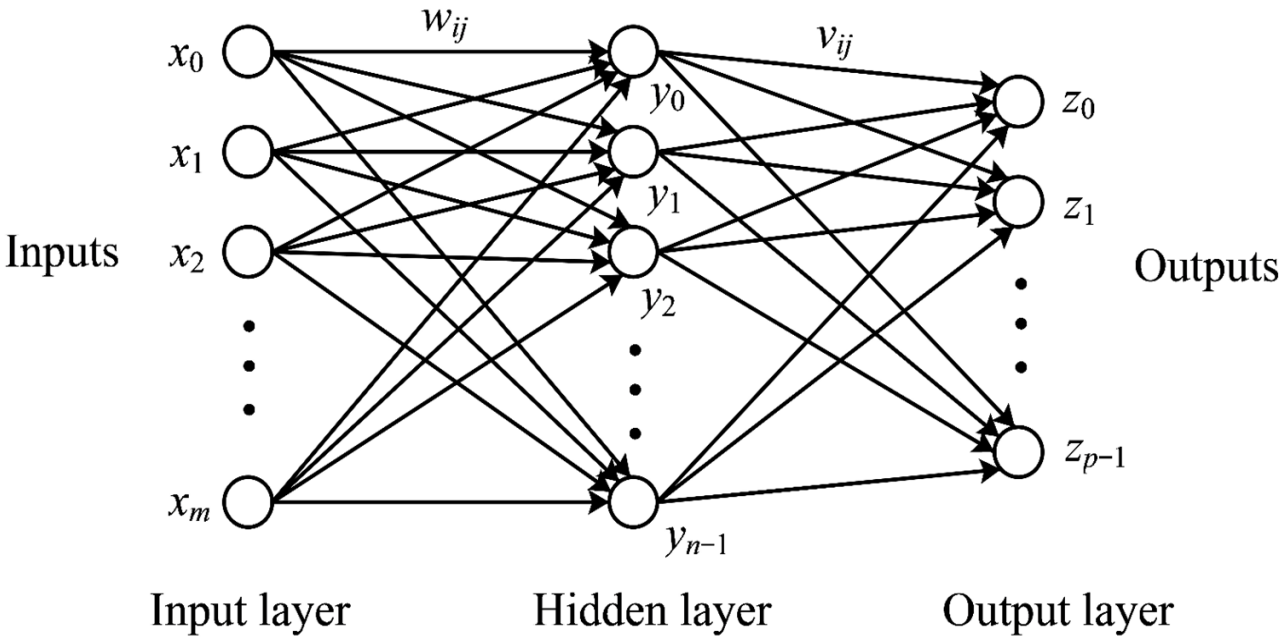


Рисунок 3.3. Багатошаровий перцептрон з одним прихованим шаром

Кожен штучний нейрон виглядає наступним чином (рис. 3.4):

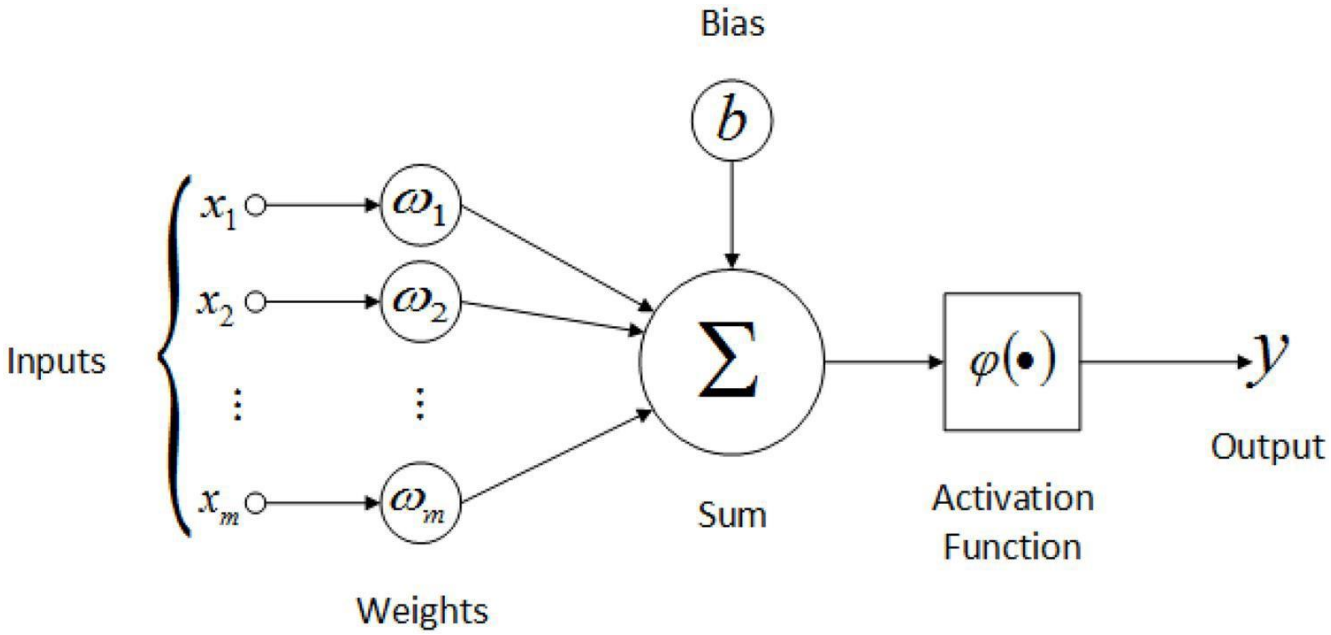


Рисунок 3.4. Штучний нейрон

Математично його можна описати такою функцією:

(1)

де — вектор з вхідними даними довжини m, — ваги цього нейрону, (i = 1,m), — деяке число, відоме також як зсув.

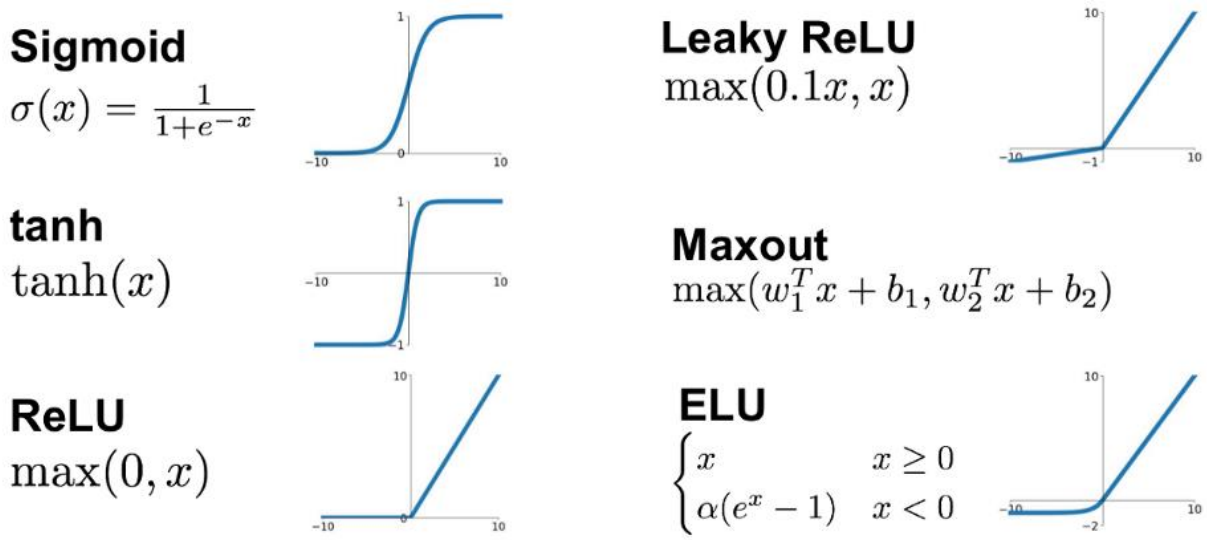
Так само можна записати шар ШНМ:

(2)

де W — матриця, стовпці якої — це вектори ваг кожного нейрону, а b — вектор зсувів кожного нейрону.

Тоді саму ШНМ можна представити як композицію функцій кожного нейрону. Наприклад двошаровий перцептрон з рисунку 4.4 може бути записаний як [illegible], де [illegible] та [illegible] — функції першого та другого шарів відповідно.

Активаційну функцію обирають залежно від задачі. В перших ШНМ це була порогова функція, що при перевищенні певного значення рівна 1, а інакше — 0. Пізніше почали використовувати нелінійні функції такі як сигмоїда чи тангенс гіперболічний, оскільки вони дозволили вирішувати складніші лінійно нероздільні задачі. Також дуже добре себе показала кусково лінійна функція ReLU. Приклади деяких активаційних функцій можна побачити нижче (рис. 3.5)



Ще одним важливим компонентом ШНМ є функція втрат. Вона дає змогу оцінити, наскільки результат, що дає і нейронна мережа відрізняється від очікуваного. Наприклад це може бути середнє квадратичне відхилення:

https://colab.research.google.com/drive/1wdoDtlR0N4G7bOyJ11x-

7lXJS1kF4Fg-?authuser=1#scrollTo=bECsDzbK55HV [illegible] (3)

де Θ — параметри ШНМ (ваги та зміщення), n — обсяг вибірки,  — очікуваний результат i-го елемента вибірки,  — результат, що дає ШНМ для i-го елемента вибірки.

Тренування мережі відбувається за допомогою мінімізації функції втрат. Для цього, методом градієнтного спуску, оновлюють параметри Θ:

 (4)

де  — коефіцієнт навчання,  — градієнт функції втрат.

Наведений вище опис базується на книзі Яна Ґудфелоу [9].

3.3 Функція втрат для моделі CRAFT

Розглянемо визначення функції втрат для моделі CRFAT. Нехай для слова w, $R(w)$ — прямокутник описаний навколо ділянки, що містить це слово, а $l(w)$ — довжина слова. При діленні слова на окремі символи, ми можемо обчислити довжину символів $l^c(w)$. Тоді оцінка впевненості $s_{conf}(w)$ для зразка w може бути обчислена як:

 (5)

Тоді для кожного пікселя оцінка впевненості S_C обчислюється так:

 (6)

Функція втрат J визначається так:

 (7)

де $S_r^*(p)$ та $S_a^*(p)$ позначають істинне значення оцінок локації та спорідненості відповідно, $S_r(p)$ та $S_a(p)$ — значення передбачені мережею [5].

Тренування мережі відбувається в декілька етапів. Спочатку на синтетичному датасеті SynthText, після чого вона, для більшої універсальності, дотреновується на деяких зображеннях з природних наборів даних з пункту 2.2. При чому не використовувався датасет TextOCR, оскільки його опублікували пізніше. Тому в цій роботі дослідимо його.

3.4 Висновки до розділу

- Наведено аналіз моделі пошуку тексту CRAFT. Це згорткова ШНМ подібна до мережі U-Net.
- Проаналізовано алгоритмічне забезпечення навчання ШНМ на прикладі моделей ШНМ — багатошарового перцептрону.
- Розглянуто визначення функції втрат для моделі CRFAT.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Використані технології. Загальні відомості про Chainer та його структуру.

Технологія CUDA дозволяє використовувати потужності графічного процесора для виконання математичних обчислень. Як було розглянуто в роботі [7], це дозволяє значно прискорити процес тренування. Однак, використання цієї технології для навчання ШНМ напряму з мов C/C++, було б дуже складним, оскільки вимагало б приділяти дуже багато часу низькорівневим деталям реалізації. Тому було розроблено чимало фреймворків для роботи з ШНМ більш високорівневими мовами, як от Python.

Chainer - це потужний, гнучкий та інтуїтивно зрозумілий фреймворк нейронних мереж, який підтримує майже усі архітектури нейронних мереж (рис.4.1). Він був розроблений Preferred Networks запуск якого відбувся в Японії. Ця структура дозволяє писати складні архітектури просто та інтуїтивно. Chainer надає набір функцій, необхідних для досліджень та розробок, використовуючи глибоке вивчення, такі як проектування нейронних мереж, навчання та оцінювання. Це допомагає "подолати розрив між алгоритмами та реалізаціями глибокого навчання".

Особливості Chainer:

- обчислення CUDA та підтримка декількох GPU через PyCUDA;
- різні мережеві архітектури (прямоточні мережі, коннективи, рекурентні мережі, рекурсивні мережі, кожна партійна архітектура);
- багатовимірний масив і реалізація шару;
- передові обчислення, керуючі поточні виступи, заздалегідь визначені функції;
- пряма передача для обробки вхідних даних;
- backprop для градієнтних обчислень;
- схема "Define-by-Run" як основна концепція.

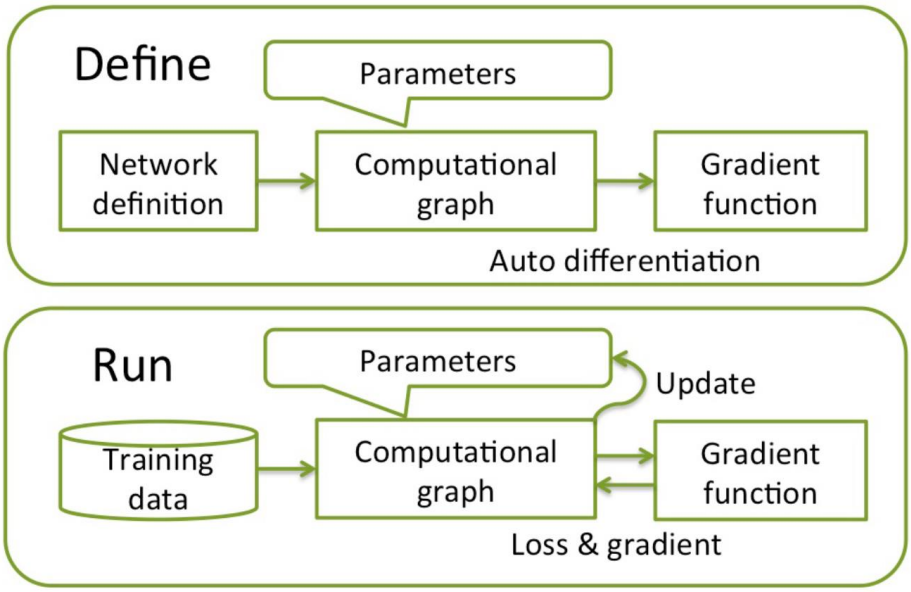


Рис 4.1. Схема Define by-Run

На відміну від інших платформ із інтерфейсом Python, такими як Theano та TensorFlow, Chainer надає імперативні способи декларування нейронних мереж, підтримуючи сумісні з Numpy операції між масивами. Chainer також включає цифрову бібліотеку обчислень на основі GPU під назвою CuPy. CuPy - це реалізація сумісного з NumPy багатомірного масиву на NVIDIA CUDA. Вона також використовує бібліотеки CUDA, включаючи cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT та NCCL, щоб повністю використовувати архітектуру GPU.

Інтерфейс CuPy сумісний з NumPy; в більшості випадків його можна використовувати як заміну. Скомпільовані бінарні файли кешуються та використовуються повторно в наступних викликах. Він також підтримує PyCUDA, як користувальницьке покоління ядра, що дозволяє нам писати швидкі реалізації для GPU. Chainer використовує пул пам'яті для розподілу пам'яті GPU. Як показано в попередніх розділах, Chainer створює та руйнує багато масивів під час навчання та оцінки ітерацій. Це не

дуже добре підходить для архітектури CUDA, оскільки розподіл пам'яті та випуск в CUDA (тобто функції cudaMalloc та cudaFree) синхронізують обчислення CPU і GPU, що завдає шкоди продуктивності. Щоб уникнути розподілу та вилучення пам'яті під час обчислень, Chainer використовує пул пам'яті CuPy як стандартний розподіл пам'яті. Chainer змінює типовий розподіл CuPy на пул пам'яті, тому користувач може використовувати функції CuPy безпосередньо, не маючи справу з розподілом пам'яті.

CuPy також включає в себе наступні функції для виконання:

- Визначені користувачем функції(kernels) CUDA
- Визначені користувачем скорочення ядер CUDA
- Очищення ядер CUDA для оптимізації обчислення
- Настроюваний розподільник пам'яті та пул пам'яті
- cuDNN утиліти

CuPy використовує динамічно визначений синтез функцій(kernel): коли потрібен виклик ядра, він збирає код ядра, оптимізований для форм і типів даних аргументів, відправляє його на пристрій GPU і виконує kernel. Скомпільований код кешується до каталогу $ (HOME) /. Cupy / kernel_cache (цей шлях кешу можна перезаписати, встановивши змінну середовища CUPY_CACHE_DIR). Це може сповільнити роботу першого виклику ядра, хоча це сповільнення буде вирішено під час другого виконання. CuPy також кешує код ядра, який надсилається на пристрій GPU в процесі, що зменшує час перенесення ядра на подальші виклики.

CuPy має концепцію поточного пристрою, який є пристроєм за замовчуванням, на якому відбувається розподіл, маніпулювання, розрахунок масивів тощо. Припустимо, що ідентифікатор поточного пристрою становить 0. Наступний код виділяє вміст масиву на GPU 0.

```
x_on_gpu0 = cp.array([1, 2, 3, 4, 5])
```

Поточний пристрій може бути змінений з використанням функционалу: cupy.cuda.Device.use()

```
x_on_gpu0 = cp.array([1, 2, 3, 4, 5])
cp.cuda.Device(1).use()
x_on_gpu1 = cp.array([1, 2, 3, 4, 5])
```

Передача даних з одного пристрою на інший відбувається з використанням наступних функцій:

```
x_cpu = np.array([1, 2, 3])
x_gpu = cp.asarray(x_cpu)  # move the data to the current device.

with cp.cuda.Device(0):
    x_gpu_0 = cp.ndarray([1, 2, 3])  # create an array in GPU 0
 with cp.cuda.Device(1):
     x_gpu_1 = cp.asarray(x_gpu_0)  # move the array to GPU 1
```

Клас Variable представляє собою одиницю обчислення, обгортаючи numpy.ndarray. Користувачі можуть визначати операції та функції (випадки функції) безпосередньо на Variable. Оскільки Variable пам'ятають, як вони створені, Variable y має операцію additive як її батьківський (.creator).

```
print(y.creator)
<chainer.functions.math.basic_math.AddConstant at 0x7f939XXXXX>
```

Цей механізм дає можливість зворотньому(backword) обчисленню, відстежуючи весь шлях від функції остаточної втрати до входу, який запам'ятовується шляхом виконання прямого(forward) обчислення, без попереднього визначення графа обчислень.

Багато чисельних операцій та функцій активації наведено в chainer.functions. Стандартні операції нейронної мережі, такі як повноприєднані лінійні та згорткові шари, реалізовані в Chainer як приклад Link. Link можна розглядати як функцію разом з відповідними навчальними параметрами (наприклад, параметрами ваги та зміщення). Також можна створити Link, який сам містить кілька інших Links. Такий контейнер посилань називається Chain. Це дозволяє Chainer підтримувати моделювання нейронної мережі як ієрархію зв'язків і ланцюгів. Chainer також підтримує найсучасніші методи оптимізації, серіалізацію та швидкіші обчислення з CUDA за допомогою CuPy.

Для того, щоб реалізувати нейронні мережі, необхідно поєднувати функції з параметрами та оптимізувати параметри. Можна використовувати клас Link для цього. Link - це об'єкт, який містить параметри (тобто цілі оптимізації). Найбільш фундаментальними є Links, які ведуть себе як звичайні функції, замінюючи аргументи за параметрами. Одним з найбільш часто використовуваних посилань є лінійне посилання(Linear Link). Він представляє математичну функцію

$$\mathbf{f(x) = Wx + b,}$$

де матриця W і вектор b є параметрами. Це посилання відповідає чистому аналогу linear(), який приймає x, W, b як аргументи. Параметри Link зберігаються як атрибути. Кожен параметр є екземпляром Variable. У випадку лінійного зв'язку, два параметри, W і b, зберігаються. За замовчуванням матриця W ініціалізується випадковим способом, тоді як вектор b ініціалізується нулями. Це найкращий спосіб ініціалізувати ці параметри.

Градієнти параметрів обчислюються методом backward(). Варто звернути увагу, що градієнти накопичуються методом, а не перезаписуються. Отже, спочатку потрібно очистити градієнти, щоб відновити обчислення. Це можна зробити, викликаючи метод cleargrads().

Щоб налаштувати значення параметрів для мінімізації втрат тощо, необхіднооптимізувати їх класом оптимізатора. Він запускає числовий алгоритм оптимізації на задану Link. Багато алгоритмів реалізовано в модулі оптимізаторів.

У Chainer реалізовані такі оптимізатори:

- AdaDelta - метод адаптивного навчального шляху
- Adam - метод стохастичної оптимізації
- MomentumSGD
- SGD - метод стохастичного зрізу градієнта
- та інші

Деякі маніпуляції з параметрами / градієнтами, наприклад затухання ваги та відсічення градієнта, можна зробити, встановивши функції гака(hook) для оптимізатора. Функції hook викликаються після градієнтного обчислення і безпосередньо перед фактичним оновленням параметрів.

Chainer забезпечує стандартну реалізацію навчальних циклів реалізованих модулем chainer.training та надає функцію, що називається Trainer, яка спрощує процедуру тренування моделі.

4.2. Загальні відомості про PyTorch та його структуру.

Як і Chainer, PyTorch базується на моделі Define-by-Run (рис. 4.2). Основною ідеєю якого є те, що граф обчислень будується динамічно прямо під час тренування. Під час проходу вперед, кожна функція, окрім обчислення значення, зберігає історію обчислень разом з посиланням на попередній вузол у графі. Оскільки структура графу залежить від шляху виконання програми, стало можливе використання синтаксичних конструкцій мови програмування, таких як умовні оператори та цикли. Окрім цього тепер з'явилась можливість використання зневоджувачів для детальнішого дослідження процесу тренування [10].

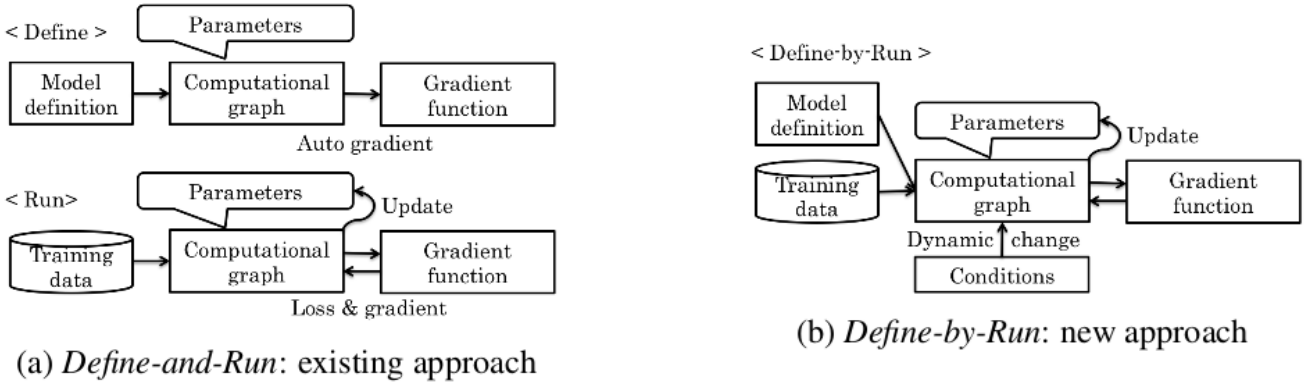


Рисунок 4.2. Схема процесу тренування для підходу
a)Define-and-Run b)Define-by-Run

Як середовище програмування було обрано Jupyter Notebook, оскільки інтерактивний режим виконання програми, що він пропонує є зручним для досліджень. Окрім цього він дозволяє зручно візуалізовувати дані, а ipynb-файли які створюються в ньому (також їх називають просто ноутбуками) підтримуються для виконання багатьма хмарними сервісами.

Також на деяких етапах обробки зображення застосовується бібліотека алгоритмів комп'ютерного зору OpenCV та бібліотека для роботи з матрицями NumPy.

4.3 Підготовка даних

Для задач пошуку тексту популярним форматом подання даних є такий, як використовується у датасетах ICDAR. Директорія з даними містить такі 4 піддиректорії:

- train_img — зображення використані для тренування

- train_gt — txt файли з координатами всіх ділянок тексту на відповідному зображенні використані для тренування
- test_img — зображення використані для тестування
- test_gt — txt файли з координатами всіх ділянок тексту на відповідному зображенні використані для тестування

Текстові файли повинні мат таки ж назву як і зображення якому вони відповідають, та містити стільки рядків з координатами, скільки на цьому зображенні виділено ділянок з текстом. Кожен рядок, складається з 8 цілих значень, записаних через кому, що позначають координати 4 кутів відповідної ділянки прямокутника.

```python
def get_ground_truth(file, json_annotation):
    filename = os.path.splitext(file)[0]
    assert filename in json_annotation["imgToAnns"]
    annotations = json_annotation["imgToAnns"][filename]

    gt = []
    for annotation in annotations:
        info = json_annotation["anns"][annotation]
        points = info["points"]
        points = np.reshape(points, (-1, 2)).astype(np.float32)
        rect = cv2.minAreaRect(points)
        points = np.int0(cv2.boxPoints(rect))
        points = np.ravel(points)
        assert len(points) == 8, f"{annotation} - {len(points)}"
        gt.append(points)

    return gt


def to_icdar_format(data_dir, json_annotation):
    for file in os.listdir(data_dir):
        if not file.endswith(".jpg"):
            print(file)
            continue

        try:
            gt = get_ground_truth(file, json_annotation)
        except Exception:
            print(file)
            continue

        gt_str = "\n".join([",".join([str(x) for x in line]) for line in gt])

        os.makedirs(os.path.join(data_dir, "img"), exist_ok=True)
        os.makedirs(os.path.join(data_dir, "gt"), exist_ok=True)

        os.rename(os.path.join(data_dir, file), os.path.join(data_dir, "img", file))
        with open(os.path.join(data_dir, "gt", os.path.splitext(file)[0] + ".txt"), "w") as fp:
            fp.write(gt_str)
```

Рисунок 4.2. Перетворення формату даних

Для перетворення даних до такого формату було реалізовано функцію to_icdar_format(data_dir, json_annotation) (рис. 4.2), що приймає папку з даними та json файл з анотаціями на вхід, та створює відповідні txt файли з анотаціями у потрібному форматі.

Варто згадати, що у json файлі з анотацією часом може бути більше ніж 4 точки, в такому випадку використовуючи функцію бібліотеки OpenCV — minAreaRect() шукаємо мінімальний описаний прямокутник навколо цих точок.

А функція train_test_split(data_dir, train_ratio) (рис. 4.3) — ділить файли у папці data_dir/ на навчальний та тестовий набори у заданій пропорції. В даному випадку дані ділились порівно, тобто train_ratio=0.5.

```python
def train_test_split(data_dir, train_ratio):
    files = [os.path.splitext(file)[0] for file in os.listdir(os.path.join(data_dir, "img"))]
    split_index = int(train_ratio * len(files))

    for file in files[split_index:]:
        os.makedirs(os.path.join(data_dir, "test_img"), exist_ok=True)
        os.rename(os.path.join(data_dir, "img", file + ".jpg"), os.path.join(data_dir, "test_img", file + ".jpg"))

        os.makedirs(os.path.join(data_dir, "test_gt"), exist_ok=True)
        os.rename(os.path.join(data_dir, "gt", file + ".txt"), os.path.join(data_dir, "test_gt", file + ".txt"))

    os.rename(os.path.join(data_dir, "img"), os.path.join(data_dir, "train_img"))
    os.rename(os.path.join(data_dir, "gt"), os.path.join(data_dir, "train_gt"))
```

Рисунок 4.3. Поділ на навчальний та тестовий набори

4.4 Використання Python фреймворку Chainer

Chainer забезпечує стандартну реалізацію навчальних циклів реалізованих модулем chainer.training. Він побудований на базі багатьох інших основних функцій Chainer, включаючи Variable та Function, Link / Chain / ChainList, Optimizer, Dataset і Reporter. У порівнянні з абстракцією навчального циклу інших наборів інструментів для машинного навчання, навчальна база Chainer спрямована на максимальну гнучкість, зберігаючи при цьому простоту для типових звичаїв. Більшість компонентів підключаються, і користувачі можуть перезаписати визначення.

Повна процедура навчання складається з наступних етапів:

- Підготовка набору даних
- Строрення ітератора набору даних
- Визначення мережі
- Вибір алгоритмів оптимізації
- Написання тренувального циклу

- Отримати набір прикладів (міні-пакет) з навчального набору даних.
- Подайте міні-пакет до потрібної мережі.

- Запустити перший прохід мережі та обчислити втрати(loss).
- Просто викликати метод backward () до змінної loss, щоб обчислити градієнти для всіх тренованих параметрів.
- Запустіть оптимізатор, щоб оновити ці параметри.

- Збережіть навчену модель
- Виконайте класифікацію за збереженою моделлю та перевірте продуктивність мережі на тестових наборах.

Ядром абстракції навчального циклу є Тренер(Trainer), який реалізує сам навчальний цикл. Навчальний цикл складається з двох частин: один - Updater, який фактично оновлює параметри для навчання, а інша - Extension для довільних функцій, відмінних від оновлення параметрів.
Типовий навчальний цикл складається з наступних процедур:

- Ітерації над навчальними наборами даних
- Первинна обробка витягнутих міні-партій
- forward / backword обчислення нейронних мереж
- Оновлення параметрів
- Оцінка поточних параметрів на наборах даних для перевірки
- Записування та друк проміжних результатів

Тренер - це клас, який містить всі необхідні компоненти, необхідні для тренувань. В принципі, все, що потрібно передати тренеру, - оновлювач. Однак Updater містить ітератор і оптимізатор. Оскільки Iterator може отримати доступ до набору даних, а Оптимізатор має посилання на модель, Updater може отримати доступ до моделі, щоб оновити її параметри.

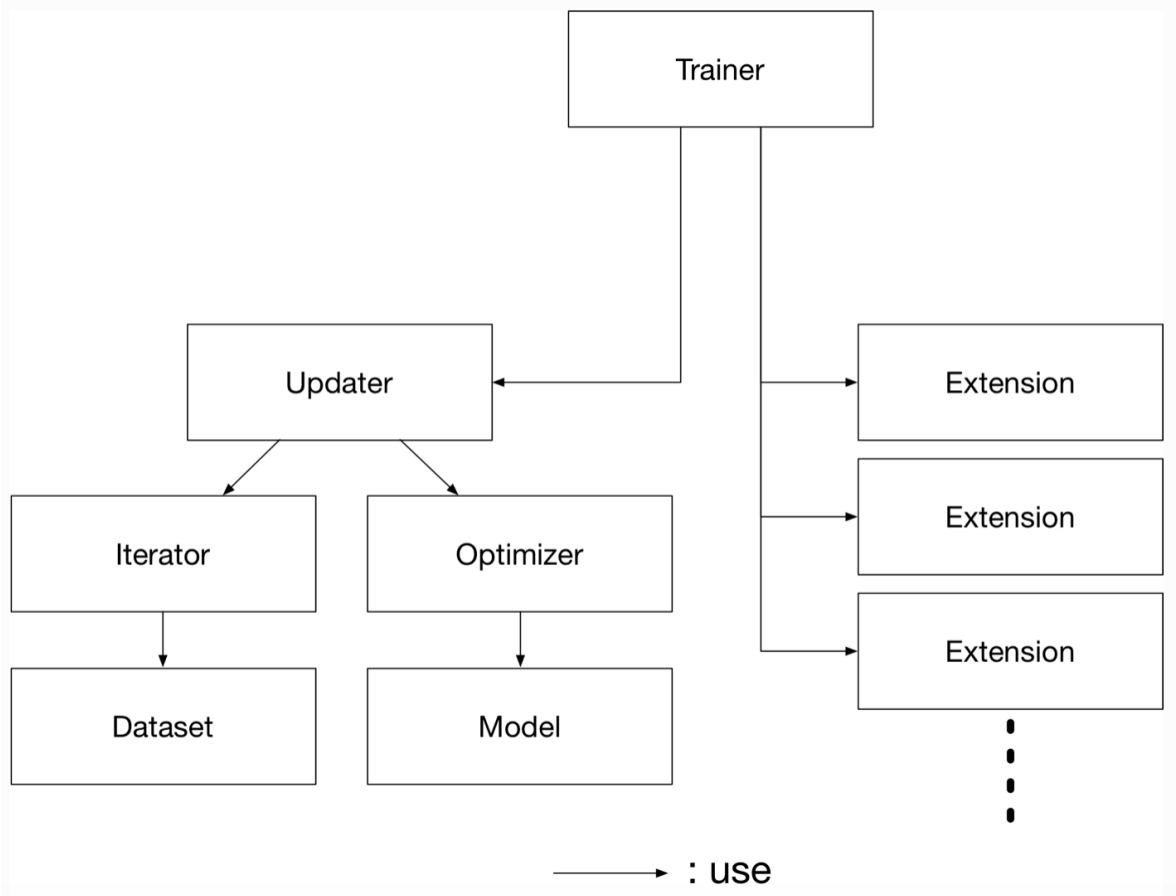


Рис.4.4 Фреймворк Chainer

Розширення Trainer надають наступні можливості:

- Зберегти файли журналів автоматично (LogReport)
- Періодично відображати навчальну інформацію терміналу (PrintReport)
- Візуалізувати прогрес у збитках періодично побудувати графік і зберегти його як файл зображення (PlotReport)
- Автоматично періодично серіалізувати стан (snapshot () / snapshot_object ())
- Показувати індикатор проміжної панелі на термінал, щоб показати прогрес навчання (ProgressBar).
- Збереження архітектури моделі як точковий файл Graphviz (dump_graph ())

Найчастіше використовують такі розширення:

- LogReport - це розширення збирає втрати та значення точності кожну епоху чи ітерацію і зберігає в лог-файлі. Файл журналу буде розташовуватися під вихідною директорією (вказаний зовнішнім аргументом об'єкта Trainer).
- snapshot - це розширення зберігає об'єкт Trainer у визначеному терміновому порядку (defaut: кожну епоху) у вихідному каталозі. Об'єкт Trainer, як згадувалося раніше, має програму оновлення, яка містить оптимізатор та модель усередині. Тому, поки у вас є файл знімків, ви можете використовувати його, щоб повернутися до тренінгу або робити висновки з використанням раніше підготовленої моделі пізніше.
- dump_graph() - це розширення зберігає структуру обчислювального графа моделі. Графік зберігається у вигляді формату графів Graphviz у вихідному каталозі тренера.
- Evaluator - ітератори, що використовують цільовий набір оцінок та об'єкт моделі, повинні використовувати розширення оцінювача. Він оцінює модель, використовуючи заданий набір даних (зазвичай це набір даних для перевірки) у вказаний інтервал часу.
- PrintReport - це розширення виводить сплюснуті значення до стандартного виводу.

Перед тим як встановлювати Chainer потрібно оновити pip репозиторій з використанням наступної команди у командному рядку:

```
pip install -U setuptools
```

Наступним етапом буде встановлення деяких додаткових бібліотек NumPy та CuPy.

NumPy - це фундаментальний пакет для наукових обчислень з Python. Щоб встановити NumPy потрібно виконати наступну команду:

```
pip install numpy
```

CuPy - це реалізація сумісного з NumPy багатомірного масиву на CUDA. CuPy використовує g++ компілятор C++, тому перед встановленням CuPy потрібно встановити g++:

```
apt-get install g++
```

Для встановлення CuPy потрібно виконати наступну команду:

```
pip install cupy
```

Якщо усі необхідні залежності були встановлені, можна перейти до встановлення chainer:

```
pip install chainer
```

Для того щоб перевірити чи всі залежності встановлені правильно, потрібно перейти у Python concole та перевірити наступні змінні:

`chainer.cuda.available` - якщо значення змінної True, це означає що chainer успішно імпортує пакет cupy

`chainer.cuda.cudnn_enabled` - якщо значення змінної True, це означає що підтримка cuDNN доступна

4.5 Особливості програмної реалізації моделі CRAFT за допомогою фреймворку PyTorch

Хоч основні ідеї, що використовуються в PyTorch подібні до тих, що були в Chainer, однак сам код програми досить сильно відрізняється. Для тренування ШНМ потрібні такі компоненти [12]:

- Model — сама ШНМ, представлена у вигляді об’єкта класу, що наслідує `torch.nn.Module` та реалізовує метод `forward`, який описує прямий прохід по нейронній мережі
- Dataset — об’єкт класу, що наслідує `torch.utils.data.Dataset` та реалізує методи `__len__` та `__getitem__` потрібні для отримання кількості елементів датасету та отримання довільного елементу за індексом відповідно. Оскільки набір даних зазвичай дуже великий, щоб повністю тримати його в пам’яті, Dataset використовується, щоб при доступі до елементу набору даних зчитати його і та за потреби застосувати певні перетворення
- DataLoader — об’єкт, що використовується для ефективного завантаження об’єктів датасету, при створенні приймає об’єкт датасету, та набір налаштувань, щодо того як завантажувати дані: розмір порції (`batch_size`), кількість потоків використаних для завантаження (`num_workers`) та чи потрібно перемішувати дані (`shuffle`)
- Optimizer — об’єкт, що при створені приймає набір параметрів моделі та додаткові параметри конкретного методу оптимізації, він використовується для самого навчання, тобто оновлення параметрів мережі для мінімізації функції втрат

Спочатку створимо клас для роботи з датасетом (рис. 4.5). Він приймає на вході шлях до директорії з вже підготованими даними та логічне значення, яке вказує на те, чи завантажити навчальну вибірку, чи тестову. Та за індексом видає всю потрібну інформацію про відповідні зразки з датасету.

```python
import dataclasses
import torchvision
from torch.utils.data import Dataset


@dataclasses.dataclass
class Entry:
    id: str
    img_path: str
    gt_path: str

class TextOCRDataset(Dataset):
    def __init__(self, root_dir, train):
        train_test_name = "train" if train else "test"
        self.img_dir = os.path.join(root_dir, f"{train_test_name}_img")
        self.gt_dir = os.path.join(root_dir, f"{train_test_name}_gt")
        self.entries = []
        for file in os.listdir(self.img_dir):
            id = os.path.splitext(file)[0]
            img_path = os.path.join(self.img_dir, f"{id}.jpg")
            gt_path = os.path.join(self.img_dir, f"{id}.txt")
            self.entries.append(Entry(id, img_path, gt_path))

    def __len__(self):
        return len(self.entries)

    def __getitem__(self, idx):
        entry = self.entries[idx]

        image, region_image, affinity_image, confidence_mask, confidences = prepare_craft_input(entry)

        return image, region_image, affinity_image, confidence_mask, confidences
```

Рисунок 4.5. Клас TextOCRDataset

Далі розглянемо реалізацію самої мережі. В якості енкодера використаємо стандартну реалізацію мережі VGG-16 без останніх повністю зв’язних шарів. А для декодера нам спочатку потрібно визначити операцію оберненої згортки, або ж upConv шар (рис. 4.6).

```
class UpConv(nn.Module):
    def __init__(self, in_ch, mid_ch, out_ch):
        super(double_conv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_ch + mid_ch, mid_ch, kernel_size=1),
            nn.BatchNorm2d(mid_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_ch, out_ch, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        x = self.conv(x)
        return x
```

Рисунок 4.6. Реалізація оберненої згортки

Тепер можемо використати ці складові для побудови самої ШНМ. Результат можна бачити на рисунку 4.6.

Далі можемо створити об’єкт датасету та ШНМ. Для цього вказуємо всі необхідні параметри (рис. 4.7). Було обрано розмір порції (batch size) = 16 та кількість 100 епох. Також створено оптимізатор Адам з коефіцієнтом навчання 0.0001.

Для моделі також використано метод `cuda()`, щоб перенести мережу у пам’ять GPU для апаратного прискорення тренування. Також потрібно пам’ятати, щоб дані, які передаємо мережі мають бути розташовані на цьому ж приладі, що і мережа. В більшості випадків це все що потрібно знати для використання графічного прискорювача за допомогою фреймворку PyTorch. Такий спосіб є дуже зручним, так як дозволяє розробнику фокусуватись над його основною роботою, а все розпаралелення відбувається автоматично. Якщо потрібно ефективно розпаралелити певну мало поширену операцію, реалізації якої у фреймворку нема, є можливість написати власне CUDA ядро [11]. Проте, в цьому дослідженні такої потреби не виникло.

```python
class CRAFT(nn.Module):
    def __init__(self, pretrained=False, freeze=False):
        super(CRAFT, self).__init__()

        """ Base network """
        self.basenet = vgg16_bn(pretrained, freeze)

        """ U network """
        self.upconv1 = UpConv(1024, 512, 256)
        self.upconv2 = UpConv(512, 256, 128)
        self.upconv3 = UpConv(256, 128, 64)
        self.upconv4 = UpConv(128, 64, 32)

        num_class = 2
        self.conv_cls = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.Conv2d(32, 16, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.Conv2d(16, 16, kernel_size=1), nn.ReLU(inplace=True),
            nn.Conv2d(16, num_class, kernel_size=1),
        )

        init_weights(self.upconv1.modules())
        init_weights(self.upconv2.modules())
        init_weights(self.upconv3.modules())
        init_weights(self.upconv4.modules())
        init_weights(self.conv_cls.modules())

    def forward(self, x):
        """ Base network """
        sources = self.basenet(x)

        """ U network """
        y = torch.cat([sources[0], sources[1]], dim=1)
        y = self.upconv1(y)

        y = F.interpolate(y, size=sources[2].size()[2:], mode='bilinear', align_corners=False)
        y = torch.cat([y, sources[2]], dim=1)
        y = self.upconv2(y)

        y = F.interpolate(y, size=sources[3].size()[2:], mode='bilinear', align_corners=False)
        y = torch.cat([y, sources[3]], dim=1)
        y = self.upconv3(y)

        y = F.interpolate(y, size=sources[4].size()[2:], mode='bilinear', align_corners=False)
        y = torch.cat([y, sources[4]], dim=1)
        feature = self.upconv4(y)

        y = self.conv_cls(feature)

        return y.permute(0,2,3,1), feature
```

Рисунок 4.7. Реалізація мережі CRAFT

```
target_size = 768
batch_size = 16
num_workers = 8
lr = 1e-4
training_lr = 1e-4
weight_decay = 5e-4
gamma = 0.8
max_epochs = 100


synthDataLoader = TextOCRDataset("/data/TextOCR", "train")
train_loader = torch.utils.data.DataLoader(synthDataLoader,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers,
                                           drop_last=True,
                                           pin_memory=True)

craft = CRAFT()
craft = torch.nn.DataParallel(craft).cuda()
craft.load_state_dict(torch.load("/data/CRAFT-Reimplementation/dataset/weights_7000.pth"))
optimizer = optim.Adam(craft.parameters(), lr=lr, weight_decay=weight_decay)
criterion = Maploss()
```

Рисунок 4.7. Створення моделі та ініціалізація параметрів

Однією з головних частин є також тренувальний цикл. Так називають функцію яка саме описує алгоритм тренування. На відміну від фреймворку Chainer де для тренування потрібно було лише створити об’єкт класу Trainer з відповідними параметрами, PyTorch вимагає ручного написання цієї функції. З однієї сторони це часто змушує писати багато однакового коду, але з іншої надає гнучкості, адже для різних задач, може знадобитись різний алгоритм навчання.

На рисунку 4.8 можна бачити використаний тренувальний цикл. Більшість коду тут відповідає за логування, візуалізацію проміжних результатів та збереження параметрів моделі. Основна частина з прямим проходом, зворотнім поширенням помилки та оптимізацією параметрів мережі займає доволі небагато рядків.

```
train_step = 0
epoch = 0
loss_value = 0
batch_time = 0
while epoch < max_epochs:
    for index, (image, region_image, affinity_image, confidence_mask, confidences) in enumerate(train_loader):
        start_time = time.time()
        craft.train()

        images = Variable(image).cuda()
        region_image_label = Variable(region_image).cuda()
        affinity_image_label = Variable(affinity_image).cuda()
        confidence_mask_label = Variable(confidence_mask).cuda()

        output, _ = craft(images)

        out1 = output[:, :, :, 0]
        out2 = output[:, :, :, 1]
        loss = criterion(region_image_label, affinity_image_label, out1, out2, confidence_mask_label)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        end_time = time.time()
        loss_value += loss.item()
        batch_time += (end_time - start_time)
        if train_step > 0 and train_step%5==0:
            mean_loss = loss_value / 5
            loss_value = 0
            display_batch_time = time.time()
            avg_batch_time = batch_time/5
            batch_time = 0
            print("{}, training_step: {}|{}, learning rate: {:.8f}, training_loss: {:.5f}, avg_batch_time: {:.5f}".format(
                time.strftime('%Y-%m-%d:%H:%M:%S',time.localtime(time.time())), train_step, whole_training_step, training_lr, mean_loss, avg_batch_time))

        train_step += 1

        if index % 1000 == 0 and index != 0:
            print('Saving state, index:', index)
            torch.save(craft.state_dict(),
                       '/data/CRAFT-Reimplementation/dataset/weights_' + repr(index) + '.pth')
```

Рисунок 4.8. Тренувальний цикл

4.6 Особливості застосування хмарних технологій

Як було продемонстровано у роботі [1], навчання моделі такого рівня складності за допомогою персонального комп’ютера середньої цінової категорії, навіть з використанням GPU, займає надто багато часу. Тому для досліджень було використано безкоштовний хмарний сервіс Google Colab.

Окрім безкоштовності, ще однією перевагою Colab є простота у користуванні. Дійсно для того, щоб розпочати роботу достатньо перейти за посиланням: https://colab.research.google.com/. В результаті буде відкрито середовище подібне до Jupyter Notebook, з яким одразу ж можна починати роботу. При цьому більшість популярних бібліотек вже буду встановлені на віртуальній машині, а більш вузькоспеціалізовані можна завантажити командою `!pip install`, в будь-якій вільній комірці. Також можна завантажити наявний ноутбук у хмарне сховище Google Drive, після чого просто відкрити його як звичайний файл, за замовчування його буде завантажено у Colab.

Google Drive можна використовувати і для зберігання даних для тренування ШНМ. Щоб використати їх в Colab, потрібно підключитись до свого сховища командою з рисунку 4.9. Після чого можна буде доступитись до даних диску за шляхом `/content/drive/MyDrive`.

```
from google.colab import drive
drive.mount('/content/drive')
```

Рисунок 4.9. Підключення Google Drive в Colab

За потреби в графічному процесорі для обчислень, потрібно перейти у Runtime -> Change runtime type та змінити поточне середовище виконання з CPU на GPU. На жаль, не можна обрати який саме графічний процесор буде наданий, оскільки його автоматично обирає балансувальник навантаження. Загалом наявні такі моделі GPU: Nvidia Tesla K80, Nvidia Tesla T4, Nvidia Tesla P4 та Nvidia Tesla P100 [12]. На момент досліджень, була використана Nvidia Tesla T4 з 16 Гб відеопам’яті.

4.7 Висновки до розділу

- Досліджено структуру фреймворку Chainer та його взаємодію з CUDA.
- Досліджено фреймворк PyTorch та його структуру. Як середовище програмування було обрано Jupyter Notebook, оскільки інтерактивний режим виконання програми, що він пропонує є зручним для досліджень. Також на деяких етапах обробки зображення застосовується бібліотека алгоритмів комп’ютерного зору OpenCV та бібліотека для роботи з матрицями NumPy.
- Для задач пошуку тексту популярним форматом подання даних є такий, як використовується у датасетах ICDAR. Для перетворення даних до такого формату було реалізовано метод to_icdar_format(data_dir, json_annotation).
- Здійснено програмну реалізацію на Python фреймворку Chainer.Наведено технологічну взаємодію з CUDA.
- Здійснено програмну реалізацію моделі CRAFT за допомогою фреймворку PyTorch. В якості енкодера використано стандартну реалізацію мережі VGG-16 без останніх повністю зв’язних шарів. А для декодера нам спочатку потрібно визначити операцію оберненої згортки, або ж upConv шар.

- Для моделі також використано метод `cuda()`, щоб перенести мережу у пам’ять GPU для апаратного прискорення тренування.
- Показано особливості застосування хмарних технологій.

РОЗДІЛ 5. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

5.1.Аналіз результатів тестового набору даних без тренування

Спочатку спробуємо проаналізувати тестовий набір даних без тренування. Тобто з тими вагами які, були встановлені в результаті тренування авторами цієї архітектури ШНМ, як оптимальні для наборів даних, що вони використовували.

Для оцінки моделі будемо використовувати такі метрики:

- Recall (R) — відношення кількості ділянок, що розпізнались як текст, до кількості ділянок з текстом в розглянутому датасеті
- Precision (P) — відношення кількості ділянок, що правильно розпізнались як текст до загальної кількості ділянок, що розпізнались як текст
- H-mean (або F-score) — обчислюється за формулою

IC13			IC15			IC17			MSRA-TD500			**TextOCR**		
R	P	H	R	P	H	R	P	H	R	P	H	**R**	**P**	**H**
93.1	97.4	95.2	84.3	89.8	86.9	68.2	80.6	73.9	78.2	88.2	82.9	**65.1**	**69.9**	**67.4**

Таблиця 5.1. Результати тестування мережі викладені авторами [5] та результат тестування на датасеті TextOCR, отриманий в даній роботі

Як бачимо з таблиці 5.1, результати є помітно гіршими. Але це пояснюється тим фактом, що під час тренування ШНМ в тренувальному наборі були зображення з тих самих датасетів, на яких мережа тестувалась. Звісно ці зображення є різними, але оскільки вони походять з одного датасету, між ними можуть бути певні закономірності. Тоді ж як датасет TextOCR є абсолютно новим для даної мережі. Більше того, цей датасет сам по собі є доволі складним, оскільки складається з великої кількості повністю природних зображень. Тому наступним етапом, варто підівчити ШНМ використовуючи частину даних з цього датасету.

5.2 Аналіз результатів даних з тренуванням ШНМ .Використання хмарних технологій.

Оскільки тренування ШНМ залежить від дуже багатьох параметрів, які ми не можемо контролювати, будемо розглядати результати як випадкові змінні. Через це всі експерименти повторювались 5 разів.

Спершу виміряємо час необхідний для завершення однієї епохи тренування.

Min	154.0
Max	540.6
STD	140.0
Average	374.6

Таблиця 5.2. Час навчання однієї епохи (в секундах)

Як вже було помічено у роботі [7], при використанні сервісу Google Colab, час виконання дуже мінливий. Це пов’язано з автоматичним балансуванням ресурсів, через що навіть з тою самою GPU однакові операції можуть займати різний час. В цьому випадку, як видно із таблиці 5.2, це також підтвердилось.

Після навчання повторимо експеримент з розпізнаванням тестового набору.

	Recall	**Precision**	**H-mean**
Min	75.4	77.1	76.2
Max	79.4	82.9	81.1
STD	1.5	1.9	1.6
Average	76.8	80.1	78.4

Таблиця 5.3. Результати тестування після додаткового тренування

Порівнюючи таблиці 5.1 та 5.3, видно що тепер результати значно покращились і тепер є співмірними з тими, які отримали автори цієї моделі на інших датасетах. Також можна помітити, що завжди показник Precision дещо вищий за Recall. Тобто модель частіше пропускає ділянки де дійсно є текст, ніж видає хибні спрацювання.

Але скоріш за все це не є характерною рисою саме цієї мережі, аналізуючи результати інших досліджень, можна помітити, що загалом це дуже поширена ситуація для задачі пошуку тексту і для інших моделей.

5.3 Висновки до розділу

- Проаналізовано тестовий набір даних без тренування. Тобто з тими вагами які, були встановлені в результаті тренування вибраної архітектури ШНМ.
- Проаналізовано результати даних з тренуванням ШНМ і використанням хмарних технологій. Оскільки тренування ШНМ залежить від дуже багатьох параметрів, які ми не можемо контролювати, то результати розглядалися як випадкові змінні.

РОЗДІЛ 6 . Розроблення стартап – проекту

Метою даного розділу є обґрунтування доцільності розробки програмно-алгоритмічного забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA. Для інформаційна система відвовідає всім необхідним вимогам пошуку та розпізнавання тексту :

- Швидкодія та легкість використання.
- Зручний інтерфейс.
- Зрозумілий підхід використання моделі прогнозування.

Розглянемо основні характеристики розробки програмно-алгоритмічного забезпечення розпаралелення навчання штучних нейронних мереж для пошуку та розпізнавання тексту з використанням технологій CUDA

Витрати на розробку і впровадження проектного рішення (К) визначаються як:

$$K_{заг} = K_1 + K_2 , \tag{6.1}$$

де K_1 – витрати на розробку проектного рішення, грн.;

K_2 – витрати на відлагодження і дослідну експлуатацію проектного рішення на ЕОМ, грн.

Витрати на розробку проектного рішення включають в себе:

1. витрати на оплату праці розробників ($B_{оп}$);
2. єдиний соціальний внесок ($B_{есв}$);
3. вартість додаткових виробів, що закуповуються ($B_{д}$);
4. накладні витрати ($B_{н}$);
5. інші витрати ($B_{ін}$).

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, які приймають участь в процесі проектування, їх чисельність, середньоденну заробітну плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загально-господарські витрати, невиробничі витрати.

Витрати на розробку проектного рішення обчислюємо за формулою:

$$K_1 = B_{оп} + B_{єсв} + B_д + B_н + B_{ін} + B_{со}, \qquad (6.2)$$

Витрати на відлагодження і дослідну експлуатацію підсистеми визначаємо з формули:

$$K_2 = S_{м.г.} \cdot t_{від}, \qquad (6.3)$$

де $S_{м.г.}$ – вартість однієї години роботи ПК, грн./год.

$t_{від}$ – кількість годин роботи ПК на відлагодження, год.

Комплексний показник якості (*Пя*) визначається шляхом порівняння показників якості проектованої системи і вибраного аналогу.

Для визначення $П_я$ використовується система показників технічного рівня і якості, яка містить в собі наступні групи, причому в кожній групі вказана в дужках мінімальна кількість показників:

1. Показники призначення (3-4);
2. Показники надійності (2-3);
3. Показники безпеки (1-2);
4. Патентно-правові показники (1-2);
5. Ергономічні показники (1-2) тощо.

Комплексний показник якості проектованої підсистеми визначаємо методом арифметичного середньозваженого з формули:

$$П_я = \sum_{i=1}^{m} Ci \times q_i, \qquad (6.4)$$

де m - кількість одиничних показників, прийнятих для оцінки якості проектованої системи;

q_i - коефіцієнт вагомості і-го одиничного показника якості, який визначає його відносну вагомість, % (коефіцієнти вагомості визначаються експертним методом, сума їх повинна дорівнювати 100 %);

C_i - відносні безрозмірні показники якості, визначені порівнянням числових значень одиничних показників проектованої підсистеми і аналога за формулами:

$$C_i = \frac{П_{npi}}{П_{ai}} \quad \text{або} \quad C_i = \frac{П_{ai}}{П_{npi}}, \tag{6.5}$$

де $П_{npi}$, $П_{ai}$ - кількісні значення і-го одиничного показника якості відповідно проектованої системи і аналога.

Критеріями поділу альтернативних стратегій розвитку є існуючий продукт (програмне забезпечення) та новий, а також супутні послуги.

При порівнянні програмних засобів в експлуатаційні витрати включають вартість підготовки даних (E_1) і вартість годин роботи ПК (E_2). Одноразові експлуатаційні витрати визначаються за формулою:

$$E_{П(A)} = E_{1П(A)} + E_{2П(A)} \tag{6.6}$$

де $E_{П(A)}$ - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

$E_{1П(A)}$ - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

$E_{2П(A)}$ - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Бюджетування є комплексно обґрунтованою системою розрахунку витрат, пов’язаних з виготовленням та реалізацією продукту, яка дає можливість здійснити аналіз витрат та розробити заходи щодо підвищення рентабельності виробництва. На даному етапі розробляється та економічно обґрунтуємо доцільність вибору однієї із стратегій.

Ціна споживання ($Ц_С$) – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$Ц_{С(П)} = Ц_П + В_{(Е)NPV} \quad (6.7)$$

де $Ц_П$ – ціна придбання проектного рішення, грн.;

$В_{(Е)NPV}$ – теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$Ц_П = К * \left(1 + \frac{П_Р}{100}\right) \times (1 + С_{ПДВ}) + К_О + К_к \quad (6.8)$$

де $П_р$ – норматив рентабельності (13%);

$К_о$ – витрати на прив'язку та освоєння проектного рішення на конкретному об’єкті, грн.;

$К_к$ – витрати на доукомплектування технічних засобів на об'єкті, грн.;

$С_{ПДВ}$ – ставка податку на додану вартість (20 %).

У цьому розділі була проведена економічна характеристика стартар -проекту .

ВИСНОВКИ

Під час виконання цієї магістерської роботи було проаналізовано сучасний стан справ в області пошуку тексту на природних зображеннях за допомогою ШНМ. Розглянуто основні набори даних та мережі.

Детальніше було досліджено новий датасет від Facebook AI Research — TextOCR. Було показано, що наявні моделі не достатньо добре справляються з пошуком тексту для цього набору даних. І запропоновано рішення, як це можна покращити.

Оскільки даний датасет містить так багато анотованих природних даних, він має значний потенціал до подальшого розвитку ШНМ для пошуку та розпізнавання тексту в майбутньому. Але навіть зараз навчання з таким датасетом дозволило досягти непоганої точності пошуку.

Звісно великі об’єми даних, хоч і дають можливість якісно тренувати ШНМ, але також вимагають значних обчислювальних потужностей, тому в цій роботі було застосовано технології CUDA та хмарний сервіс Google Colab.

В майбутньому планується поєднати цю модель з мережею для розпізнавання тексту. Після чого, в разі успіху, можна розглянути і інші різні практичні застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Sokolovskyy, Y., Manokhin, D., Kaplunsky, Y., Mokrytska, O. (2021). Development of software and algorithms of parallel learning of artificial neural networks using CUDA technologies. Technology Audit and Production Reserves, 5(2(61)), 21–25. doi: http://doi.org/10.15587/2706-5448.2021.239784

[2] [Електронний ресурс] http://www.mathros.net.ua/miniyizacija-funkcii-dekilkoh-zminnyh-vykorystovujuchy-metod-gradijentnogo-spusku.html

[3] Text Recognition in the Wild: A Survey / [Xiaoxue Chen, Lianwen Jin, Yuanzhi Zhu, Canjie Luo, and Tianwei Wang] // ACM. — 2020. — 34 pages. — Available from: https://arxiv.org/pdf/2005.03492.pdf

[4] EAST: An Efficient and Accurate Scene Text Detector / [Xinyu Zhou, Cong Yao, He Wen et al.] // Megvii Technology Inc. — 2017. — Available from: https://arxiv.org/pdf/1704.03155v2.pdf

[5] Character Region Awareness for Text Detection / [Youngmin Baek, Bado Lee, Dongyoon Han et al.] // Clova AI Research. — 2019. — Available from: https://arxiv.org/pdf/1904.01941.pdf

[6] OpenCV documentation: TextDetectionModel_EAST Class Reference [Electronic resource]. — 2022. — Available from: https://docs.opencv.org/4.x/d8/ddc/classcv_1_1dnn_1_1TextDetectionModel__EAST.html

[7] TextOCR: Towards Large-Scale End-to-End Reasoning for Arbitrary-Shaped Text / [Amanpreet Singh, Guan Pang, Mandy Toh, Jing Huang et al.]. — Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). — 2021. — pp. 8802-8812

[8] Karen Simonyan, Andrew Zisserman Very Deep Convolutional Netwotks for Large-Scale Image Recognition. — 2015.

[9] Ian Goodfellow Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — MIT Press, 2016. — 781pp.

[10] Chainer: a Next-Generation Open SourceFramework for Deep Learning / [Seiya Tokui, Kenta Oono, Shohei Hido, et al.]. — 2015.

[11] PyTorch Documentation [Electronic resource]. — 2021. — Available from: https://pytorch.org/docs/stable/index.html

[12] Colaboratory Frequently Asked Questions [Electronic resource]. — 2021. — Available from: https://research.google.com/colaboratory/faq.htmlw

Додаток А. Формат json файлу з анотаціями до даних з датасету TextOCR


```
{
  "imgs": {
    "OpenImages_ImageID_1": {
      "id": "OpenImages_ImageID_1",
      "width": "INT, Width of the image",
      "height": "INT, Height of the image",
      "set": "Split train|val|test",
      "filename": "train|test/OpenImages_ImageID_1.jpg"
    },
    "OpenImages_ImageID_2": {
      "...": "..."
    }
  },
  "anns": {
    "OpenImages_ImageID_1_1": {
      "id": "STR, OpenImages_ImageID_1_1, Specifies the nth annotation for an image",
      "image_id": "OpenImages_ImageID_1",
      "bbox": [
        "FLOAT x1",
        "FLOAT y1",
        "FLOAT x2",
        "FLOAT y2"
      ],
      "points": [
        "FLOAT x1",
        "FLOAT y1",
        "FLOAT x2",
        "FLOAT y2",
        "...",
        "FLOAT xN",
        "FLOAT yN"
      ],
      "utf8_string": "text for this annotation",
      "area": "FLOAT, area of this box"
    },
    "OpenImages_ImageID_1_2": {
      "...": "..."
    },
    "OpenImages_ImageID_2_1": {
      "...": "..."
    }
  },
  "img2Anns": {
    "OpenImages_ImageID_1": [
      "OpenImages_ImageID_1_1",
      "OpenImages_ImageID_1_2",
      "OpenImages_ImageID_1_2"
    ],
    "OpenImages_ImageID_N": [
      "..."
    ]
  }
}
```


Формат json файлу з анотаціями до даних з датасету TextOC

Додаток Б. Пакет optimizer

```
import collections
import copy
import warnings

import numpy
import six

from chainer import cuda
from chainer.optimizer import Optimizer
from chainer import link as link_module
from chainer import serializer as serializer_module
from chainer import variable
from chainer import cuda
from chainer import optimizer


class CompetitiveMethod(Optimizer):

    def __init__(self):
        super(CompetitiveMethod, self).__init__()

    def setup(self, link):
        super(CompetitiveMethod, self).setup(link)
        for param in link.params():
            param.update_rule = self.create_update_rule()

    def update(self, lossfun=None, *args, **kwargs):
        """Updates parameters based on a loss function or computed gradients.

        This method runs in two ways.

        - If ``lossfun`` is given, then it is used as a loss function to
          compute gradients.
        - Otherwise, this method assumes that the gradients are already
          computed.

        In both cases, the computed gradients are used to update parameters.
        The actual update routines are defined by the update rule of each
        parameter.

        """
        if lossfun is not None:
            use_cleargrads = getattr(self, '_use_cleargrads', True)
            delta_x, delta_w = lossfun(*args, **kwargs)
            loss = delta_x, delta_w
            if use_cleargrads:
                self.target.cleargrads()
            else:
                self.target.zerograds()
            # loss[0].backward()
            # loss[1].backward()
            del loss
```

```
        self.t += 1
        param = self.target.weight()
        param.update_rule.update(param, delta_x, delta_w)


    def create_update_rule(self):
        """Creates a new update rule object.

        This method creates an update rule object. It is called by
        :meth:`setup` to set up an update rule of each parameter.
        Each implementation of the gradient method should override this
method
        to provide the default update rule implementation.

        Return:
            UpdateRule: Update rule object.

        """
        raise NotImplementedError


class CompetitiveRule(optimizer.UpdateRule):

    """Update rule of vanilla stochastic gradient descent.

    See :class:`~chainer.optimizers.SGD` for the default values of the
    hyperparameters.

    Args:
        parent_hyperparam (~chainer.optimizer.Hyperparameter): Hyperparameter
            that provides the default values.
        lr (float): Learning rate.

    """

    def __init__(self, *args, **kwargs):
        super(CompetitiveRule, self).__init__()

    def update(self, param, *args, **kwargs):
        """Invokes hook functions and updates the parameter.

        Args:
            param (~chainer.Variable): Variable to be updated.

        """
        if not self.enabled:
            return

        self.t += 1
        if param.data is not None:
            self._prepare(param)
        for hook in six.itervalues(self._hooks):
            hook(self, param)
        self.update_core(param, *args, **kwargs)

    def update_core(self, param,  *args, **kwargs):
        """Updates the parameter.
```

```
        Implementation of UpdateRule should override this method or both of
        :meth:`_update_core_cpu` and :meth:`_update_core_gpu`.

        Args:
            param (~chainer.Variable): Variable to be updated.

        """
        with cuda.get_device_from_array(param.data) as dev:
            if int(dev) == -1:
                self.update_core_cpu(param, *args, **kwargs)
            else:
                self.update_core_gpu(param, *args, **kwargs)

    def update_core_cpu(self, param, delta_x, delta_w, reinforce=True, *args,
**kwargs):

        if param is None:
            return
        param.data += delta_x if reinforce else - delta_x
        param.data -= delta_w if reinforce else - delta_w

    def update_core_gpu(self, param, delta_x, delta_w, reinforce=True, *args,
**kwargs):

        if param is None:
            return
        cuda.elementwise('T delta_x, T delta_w, T reinforce', 'T param',
                         'param += reinforce ? delta_x : - delta_x; '
                         'param -= reinforce ? delta_w : - delta_w',
                         'sgd')(delta_x, delta_w, reinforce, param.data)


class Competitive(CompetitiveMethod):

    """Vanilla Stochastic Gradient Descent.

    Args:
        lr (float): Learning rate.

    """

    def __init__(self, *args, **kwargs):
        super(Competitive, self).__init__()

    def create_update_rule(self):
        return CompetitiveRule()
```

Додаток В. Пакет updater

```
from __future__ import division

from chainer import training


class CompetitiveUpdater(training.updater.StandardUpdater):

    def __init__(self, train_iter, optimizer, device):
        super(CompetitiveUpdater, self).__init__(
            train_iter, optimizer, device=device)
```

```
    def update_core(self):
        batch = self._iterators['main'].next()

        in_arrays = self.converter(batch, self.device)

        optimizer = self._optimizers['main']
        loss_func = self.loss_func or optimizer.target

        lr = 0.05 * (1.0 - float(self.iteration) /
len(self._iterators['main'].dataset))
        var = 2.0 * (1.0 - float(self.iteration) /
len(self._iterators['main'].dataset))

        if isinstance(in_arrays, tuple):
            optimizer.update(loss_func, *in_arrays, lr=lr, var=var)
        elif isinstance(in_arrays, dict):
            optimizer.update(loss_func, lr=lr, var=var, **in_arrays)
        else:
            optimizer.update(loss_func, in_arrays, lr=lr, var=var)
```

Дотаток Г. Пакет training

```
import uuid

from chainer import cuda
from chainer import iterators, optimizers, serializers
import chainer.links as L
from chainer import training
from chainer.training import extensions

import datasets
from models.mlff import MLFF

cuda.cudnn_enabled = False

train = datasets.get_bank()
test = datasets.get_bank()
val = datasets.get_bank()

batchsize = 1
out_file = 'results_data/mnist_results/mnists_result_cpu_{}'.for-
mat(uuid.uuid4())

train_iter = iterators.SerialIterator(train, batchsize)
test_iter = iterators.SerialIterator(test, batchsize, False, False)


gpu_id = -1  # Set to -1 if you use CPU

model = MLFF()
if gpu_id >= 0:
    model.to_gpu(gpu_id)


max_epoch = 10
```

```
# Wrap your model by Classifier and include the process of loss calculation
within your model.
# Since we do not specify a loss function here, the default 'soft-
max_cross_entropy' is used.
model = L.Classifier(model)

# selection of your optimizing method
optimizer = optimizers.SGD()

# Give the optimizer a reference to the model
optimizer.setup(model)

# Get an updater that uses the Iterator and Optimizer
updater = training.updater.StandardUpdater(train_iter, optimizer, de-
vice=gpu_id)

# Setup a Trainer
trainer = training.Trainer(updater, (max_epoch, 'epoch'), out=out_file)

trainer.extend(extensions.LogReport())
trainer.extend(extensions.snapshot(filename='snapshot_epoch-
{.updater.epoch}'))
trainer.extend(extensions.snapshot_object(model.predictor, file-
name='model_epoch-{.updater.epoch}'))
trainer.extend(extensions.Evaluator(test_iter, model, device=gpu_id))
trainer.extend(extensions.PrintReport(['epoch', 'main/loss', 'main/accuracy',
'validation/main/loss', 'validation/main/accuracy', 'elapsed_time']))
trainer.extend(extensions.PlotReport(['main/loss', 'validation/main/loss'],
x_key='epoch', file_name='loss.png'))
trainer.extend(extensions.PlotReport(['main/accuracy', 'validation/main/accu-
racy'], x_key='epoch', file_name='accuracy.png'))
trainer.extend(extensions.dump_graph('main/loss'))

trainer.run()
```