Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Застосування алгоритму SAM до задачі детекції об’єктів

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” - 121

Керівник курсової роботи:

с.в., кандидат наук, Швай Н.О

_____(підпис)

“_____” _____ 2022р.

Виконала студентка

Старовойт А.В.

_____(підпис)

“_____” _____ 2022 р.

Київ 2022

Зміст

АННОТАЦІЯ

У процесі написання курсової роботи було натреновано нейронну мережу YOLOv3 використовуючи алгоритм оптимізації SAM на даних PascalVOC. Порівняно результати та якість знаходження об’єктів алгоритму. Досліджено тему детекції об’єктів та нейронних мереж. Отримано підтвердження дієвості алгоритму SAM для задачі детекції об’єктів.

ВСТУП

Штучний інтелект посідає важливе місце у сучасному світі. Майже не помічаючи цього, в усіх куточках світу, люди використовують здобутки цієї сфери в щоденному житті, і в виняткових важливих справах. Коли ми беремо кредит, отримуємо рекламу, здаємо аналізи в лікаря, вирішуємо чергову проблему за допомогою чат-бота – ми прямо чи опосередковано маємо справу зі штучним інтелектом. Він покращує та спрощує наші життя.

Поглянемо на одну з галузей штучного інтелекту – комп'ютерний зір. Коли ми підписуємо документ за допомогою «Дія. Підпис», чи знаходимо назву пташки за її фотографією в гугл, чи використовуємо автопілот в машині, або ж отримуємо штраф за перевищення швидкості з відеореєстратора – це все заслуга комп'ютерного зору. Аби навчити комп'ютери розпізнавати об'єкти, або «бачити», сучасні дослідники використовують нейронні мережі. Це одна з «найгарячіших» тем сучасних досліджень, як в академічних колах, так і в приватних компаніях (таких як Google, Facebook чи Amazon). Проте, незважаючи на шалену популярність, треба визнати, що драстичних змін чи відкриттів в цій сфері не планується. Як слушно зазначає Лі Кайфу, в своїй книзі «Супердержави: Китай, Силіконова долина та Новий світ», напрямок розвитку нейронних мереж та штучного інтелекту вже усталений, і від дослідників потребуються лише покращення існуючих алгоритмів та впровадження їх в наше повсякденне життя в усіх можливих сферах. «Лампочку» вже винайдено, залишається лише побудувати систему освітлення в містах та містечках!

Отже, фокус сучасних досліджень нейронних мереж направлений на покращення результатів існуючих алгоритмів та

застосування цих алгоритмів до більших, важливіших та різноманітніших задач. Тому, в цій курсовій роботі ми будемо досліджувати один з нових алгоритмів оптимізації, спробуємо застосувати його до нової задачі та перевірити його дієвість. Алгоритм SAM (Sharpness Aware Minimization) вже чудово показав себе в ряді експериментів та є одним з найперспективніших нових досліджень. Мета цієї курсової роботи – зрозуміти та побачити потенціал цього алгоритму. Це спроба спрогнозувати, чи буде він наступним великим трендом в галузі навчання нейронних мереж.

В ході дослідження буде натреновано нейронну мережу YOLOv3 для задачі детекції об’єктів на даних PascalVOC з використанням цього алгоритму оптимізації. Завдання дослідження – показати перевагу використання SAM. Для виконання дослідження будуть використані відкриті репозиторії нейронної мережі та алгоритму. Робота в основному базується на наукових публікаціях, які виходять разом з вихідним кодом алгоритмів та мереж, й описують пророблену роботу, експерименти й результати досліджень. В теоретичній частині буде короткий огляд сучасного стану індустрії, популярних нейронних мереж та алгоритмів.

РОЗДІЛ 1: АНАЛІЗ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА ПРЕДМЕНТОЇ ОБЛАСТІ

1.1 Аналіз завдання

Головна задача цієї курсової роботи – випробувати новий алгоритм оптимізації SAM на задачі з детекції об’єктів, для того, аби підтвердити чи спростувати дієвість та доцільність його застосування. Для здійснення цієї мети буде натреновано нейронну мережу YOLOv3 (you-only-look-once) на даних PascalVOC в двох варіантах – з використанням алгоритму SAM та без нього. Аналіз отриманих результатів дасть змогу робити висновки про доцільність використання цього алгоритму.

1.2 Аналіз предметної області

Як було зазначено в вступі, алгоритми детекції об’єктів є надзвичайно популярними та використовуються в найрізноманітніших місцях. Це дуже гаряча тема для досліджень, щоденно виходять нові алгоритми, дослідження та наукові публікації. Основний фокус досліджень – покращення наявних моделей, знаходження універсальних алгоритмів оптимізації, які б працювали на більш ніж одному виді нейронної мережі. Центральними проблемами в цій області є те, що здебільшого нові оптимізатори показують хороші результати на початкових етапах, але важко узагальнюються. Вони дієві лише в вузькому прошарку досліджень, але не можуть використовуватися в широкому спектрі задач.

Серед сотень нових публікацій важко віднайти нові прориви чи тренди машинного навчання. Проте, деякі алгоритми базуються на цікавих ідеях та демонструють дуже амбітні результати. Їх продовжують вивчати,

щоб перевірити життєздатність. Одним з таких цікавих алгоритмів є Sharpness Aware Minimization, або SAM, який буде розглянуто та досліджено в даній роботі. Є шанс, що саме він задаватиме тренди в цій сфері людського пізнання, і стане одним з ключових алгоритмів оптимізації в машинному навчанні.

РОЗДІЛ 2: ЗАДАЧА ВИЯВЛЕННЯ ОБ'ЄКТІВ ТА ПОПУЛЯРНІ НЕЙРОННІ МЕРЕЖІ

2.1 Що таке виявлення об'єктів?

Виявлення об'єктів (*object detection*) – це одна за задач машинного навчання, що полягає в:

1. знаходженні на зображені певних об'єктів та визначенні до якого класу вони належать (*image classification*)
2. визначенні місцезнаходження та координат виявлених об'єктів (*object localization*)

Важливо також відрізняти *object detection* від single-object detection. Різниця полягає в тому, що single-object detection визначає місцезнаходження на зображені лише одного об'єкту з кожного класу[1].

На рисунку 1 можна побачити різницю між описаними вище видами завдань. В першому стовпчику бачимо те, що вказано в початковій анотації для картинки (ground truth), а в інших стовпчиках результати роботи мереж разом з метриками, що оцінюють їх роботу.

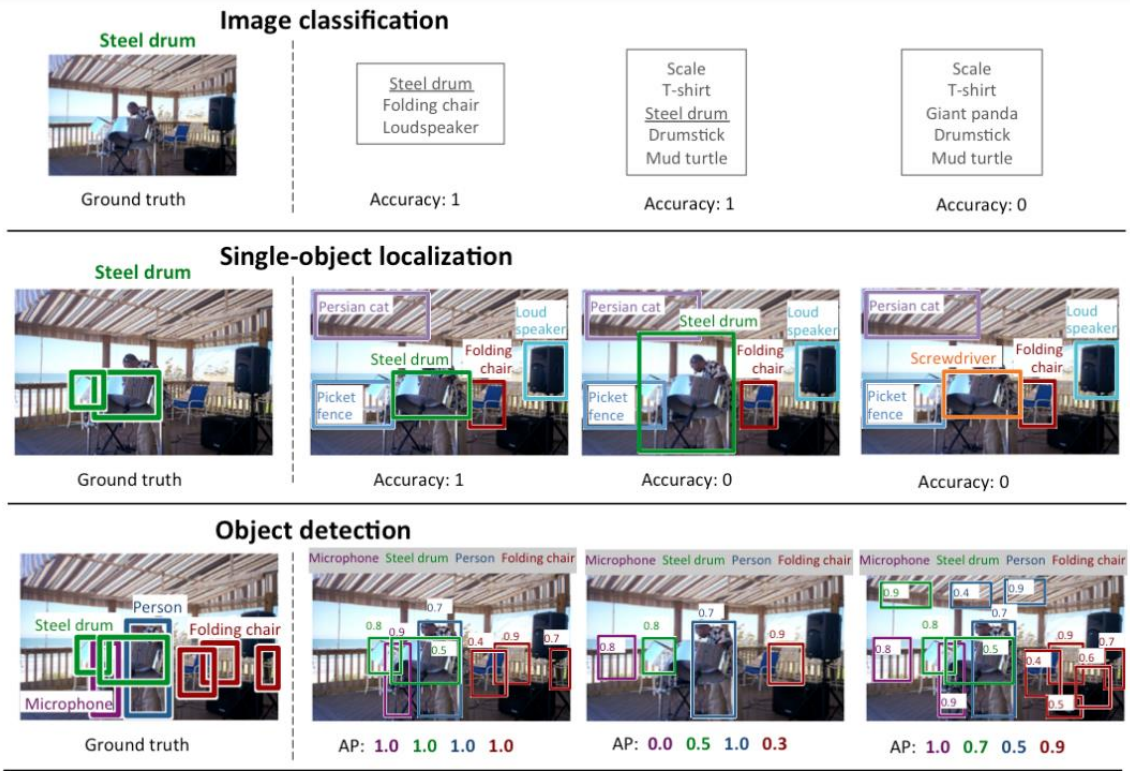

Рисунок 1: різниця між різними типами завдань комп'ютерного зору[1]

Комп’ютерне виявлення об’єктів широко застосовується в таких сферах: розпізнавання облич, самокеровані автомобілі (автопілот), розпізнавання пішоходів.

2.2 Моделі нейронних мереж для виявлення об’єктів

Серед безлічі моделей, які існують на даний момент для задачі розпізнавання об’єктів в цій науковій роботі буде розглянуто дві родини найпопулярніших. Це родина R-CNN (Region-Based Convolutional Neural Network) та YOLO (You-only-look-once). Обидві родини моделей широко застосовуються для задач виявлення об’єктів, є ефективними та результативними.

2.2.1 R-CNN (Region-Based Convolutional Neural Network)

Основна ідея даної родини нейронних мереж – виявлення регіонів на картинці, де може знаходитися об’єкт, та застосування нейронної мережі до кожного такого регіону, для виявлення специфічних ознак та встановлення класу об’єкту. В цій курсовій роботі буде розглянуто три мережі даної родини та коротко представлено їх результати.

2.2.1.1 Модель R-CNN

Модель R-CNN була вперше описана 2014 року в праці Роса Грішека з університету Берклі [2].

Модель спершу пропонує регіони, де може знаходитися об’єкт. Селективний пошук[4] спершу прогнозує координати (bounding boxes), а після вже Convolutional Neural Network визначає об’єкти в пропонованих боксах.

Недоліки даної мережі описали автори статті про YOLO[3]: *“Search generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time)”.* Серед них: велика кількість етапів, кожен з який окремо має бути оптимізований, і, відповідно, низька швидкість надання передбачення.

В нових моделях цієї родини – Fast R-CNN Faster R-CNN вдалося значно покращити час обробки, не погіршуючи при цьому якість передбачень, як видно на Рисунку 2.

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Рисунок 2: порівняння результатів та швидкості роботи різних мереж родини R-CNN

Отже, моделі родини R-CNN є досить складними, проте дуже дієвими в задачі детекції об’єктів.

2.2.2 YOLO (You-only-look-once)

Інша сім’я нейронних мереж, яку ми розглянемо – це YOLO (You-only-look-once). Ідея полягає в тому, що вона використовує єдину мережу для визначення класів та координат (bounding boxes) об’єктів. На відміну від R-CNN, як підкреслено в назві, ми пропускаємо картинку через мережу лише один раз, не шукаючи регіони окремо. Це дає змогу значно пришвидшити прогнозування, роблячи прогнози буквально в реальному часі. Як зазначають автори статті: *«Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second»*[3]

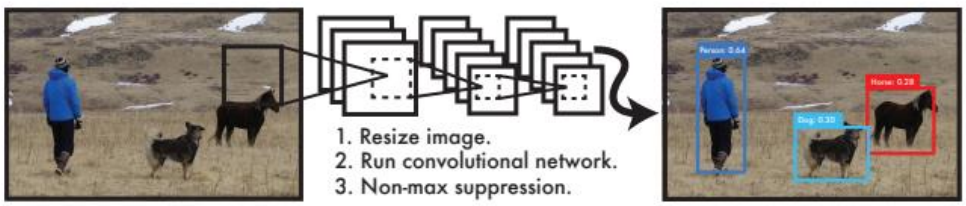

Рисунок 3: Демонстрація принципу роботи YOLO – одна нейронна мережа для визначення об’єктів та їх місцезнаходження[3]

Розглянемо детальніше обробку конкретного зображення. Алгоритм розбиває картинку сіткою (S*S) на регіони. В кожному регіоні мережа знаходить можливі центри об’єктів. В результаті, після останнього рівня, отримуємо вектор передбачень розміру S* S* (B* 5 +C), де S – розмір сітки, B – наперед визначена кількість передбачень для кожної комірки, C – кількість можливих класів. Таким чином, кожна комірка прогнозує «B» об’єктів. Кожен з яких характеризує 5+C параметрів: p – ймовірність, що передбачення вірне, (x,y) – координата центра об’єкту, w - ширина, h - висота, та C параметрів (для кожного можливого класу визначається його ймовірність).

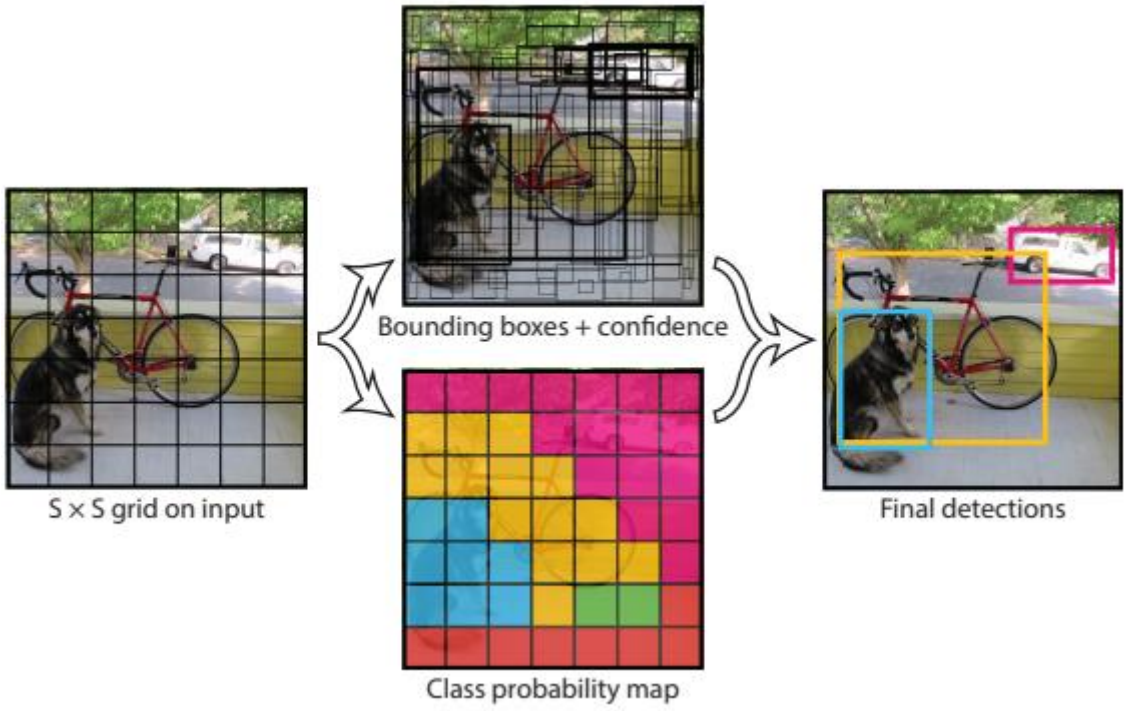

Рисунок 1: Демонстрація алгоритму YOLO. Розділення картинки на сітку, визначення координат об'єктів та належність до класу [3]

Для прикладу, на тренуванні даних PascalVOC застосовувалася сітка 7*7, з двома можливими об’єктами в кожній комірці. Датасет має 20 можливих класів, отже вихідний вектор має розмір 7*7*(2*20+5)= 49*45 [3].

Як бачимо з попередніх розрахунків, на одну картинку припадає 98 передбачених об’єктів. Звичайно, не всі з них насправді містять об’єкт, а деякі правдиві об’єкти можуть передбачити декілька комірок одразу. Тому важливо використати правильний алгоритм виключення малоймовірних передбачень. За допомогою виключення з результатів передбачень з низькою ймовірністю, та використання функції IOU (Intersection over union) вдається суттєво зменшити кількість передбачень та отримати правильні результати.

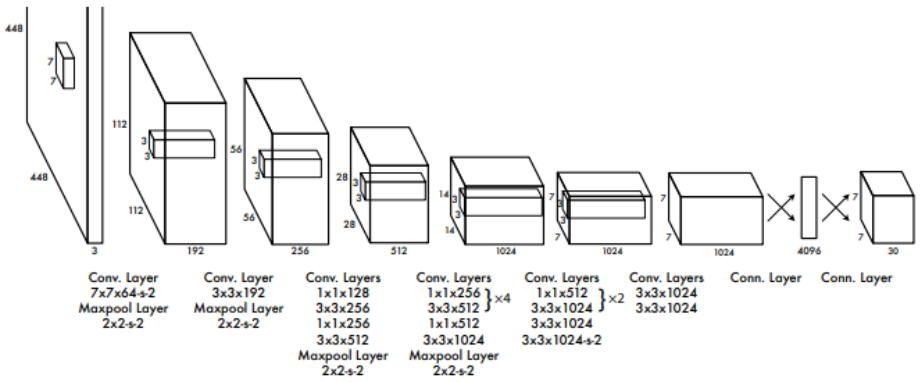

Рисунок 4 Будова нейронної мережі Yolo [3]

На рисунку 3 можна побачити схематичну будову мережі Yolo. Ця мережа складається з 24 Convolutional layers, за якими слідують 2 fully connected рівні. Convolutional рівні відповідальні за визначення специфічних характеристик зображення, в той час як fully connected визначають координати та ймовірності (дані, які ми отримуємо в результаті роботи мережі).

2.2.2.1 YOLOv2 (YOLO9000) & YOLOv3

Мережа YOLO показала себе дуже добре в різноманітних експериментах, в порівнянні до альтернативних методів. Алгоритм надзвичайно швидкий та простий, до того ж , він випереджує свої альтернативи в точності. Невдовзі, в грудні 2016, автори представили покращену мережу YOLO9000[5]. Своїй назві мережа завдячує тим, що вона була натренована на 9000 різних класів об’єктів, що на той час було величезною кількістю. Проблема задачі визначення об’єктів полягає в тому, що дуже складно знайти достатню кількість потрібних даних. Labeling об’єктів порівняно дорогий, адже треба визначати не лише клас об’єкту, а й координати й класи багатьох об’єктів, які присутні на зображенні. А для тренування необхідні мільйони зображень!

Отже, нова модель була натренована на значно більшому датасеті. Іншою проблемою мережі, яку автори намагалися розв’язати, були порівняно низькі показники з локалізації. Крім того, автори представили низку інших покращень:

- Batch normalization – додавання цієї нормалізації після кожного рівня дозволило покращити mAP на 2%
- High resolution classifier – зазвичай, нейронні мережі використовують розширення менше за 256x256. YOLO була натренована на

зображеннях з розширенням 224x224, а потрім дотренована на зображеннях 448x448, і використовує це розширення для класифікації. Таке покращення дозволило збільшити mAP на 4%.

- Anchor boxes – наперед визначені bounding boxes з корисними формами та розмірами, що використовуються при тренуванні. Це ідея була позичена з моделі Faster R-CNN та модифікована. Бокси визначаються за допомогою k-means analysis на тренувальному датасеті.

Важливим є те, що мережа не визначає координати зображення з нуля, а модифікує наперед визначені бокси, що дозволяє створити більш стабільну модель, яка не реагує на незначні зміни. [6]

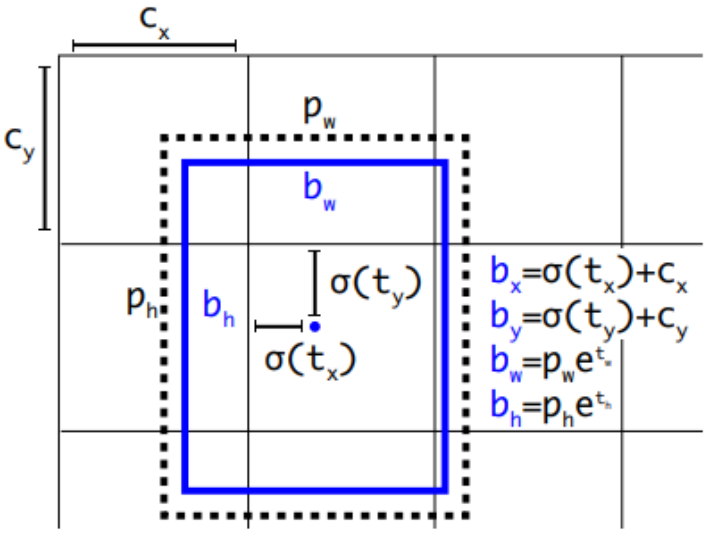

Рисунок 5: Обрахування координат об'єкта враховуючи визначені anchor boxes, YOLOv2. Центр прогнозується відносно комірки, а висота й довжина модифікують наперед визначені бокси [5]

В наступній моделі – Yolov3 – були додані деякі незначні покращення, такі як глибша нейронна мережа та інша репрезентація, що дало ще оптимістичніші результати, зберігши надзвичайну швидкість

детекції! Новий алгоритм використовує іншу нейронну мережу Darknet 53 для виявлення специфічних ознак картинки (feature extraction), що містить аж 53 Convolutional layers (на відміну від оригінальних 24!) [7]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рисунок 6: Мережа для feature detection. Darknet 53 [7]

Вже наявні нові моделі сім'ї, які досягли ще більших успіхів в визначені об'єктів (такі як YOLOv5). Проте, вони виходять за межі даного дослідження і не будуть розглянуті в цій курсовій роботі.

Підсумовуючи, в цьому розділі було розглянуто 2 родини моделей детекції об'єктів, з акцентом на родину YOLO, адже вона буде використовуватися в практичному експерименті. Обидві є важливими в сфері розпізнавання об'єктів і мають свої переваги й недоліки. R-CNN є зазвичай більш точною, але дуже складною й довгою моделлю. YOLO є

надзвичайно швидкою, простою та інтуїтивно зрозумілішою моделлю. На мою думку, зі збільшенням тренувальних даних вона має більші перспективи та буде демонструвати кращі результати. До того ж, швидкість передбачення часто є вирішальним критерієм в сучасних умовах та сферах застосування.

РОЗДІЛ 3: АЛГОРИТМИ ОПТИМІЗАЦІЇ. SAM

3.1 Огляд розділу

Алгоритми оптимізації відповідальні за мінімізацію втрат (losses) для отримання найбільш оптимальних результатів. На даний момент було запропоновано й вивчено безліч алгоритмів оптимізації. В даному розділі ми розглянемо найбільш вживані та продуктивні алгоритми, тренди, напрямки та проблеми оптимізації в сучасних нейронних мережах. У крайній частині розділу ми детально оглянемо алгоритм SAM. Розберемося в принципах роботи та теорією, яка за ним стоїть.

3.2.1 Gradient Descent

Gradient Descent – це алгоритм оптимізації, що знаходить локальний мінімум функції, яка диференційована (тобто функція, в якій можна обчислити похідну). Від використовується для знаходження значень параметрів функції (коефіцієнтів), які максимально мінімізують cost функцію[11]

Початково, параметри ініціалізуються випадковими значеннями, і після кожного проходження по мережі вираховується cost функція, яка визначає, наскільки прогнозовані результати відповідають дійсності. Далі обраховується градієнт цієї функції для визначення оновлення параметрів.

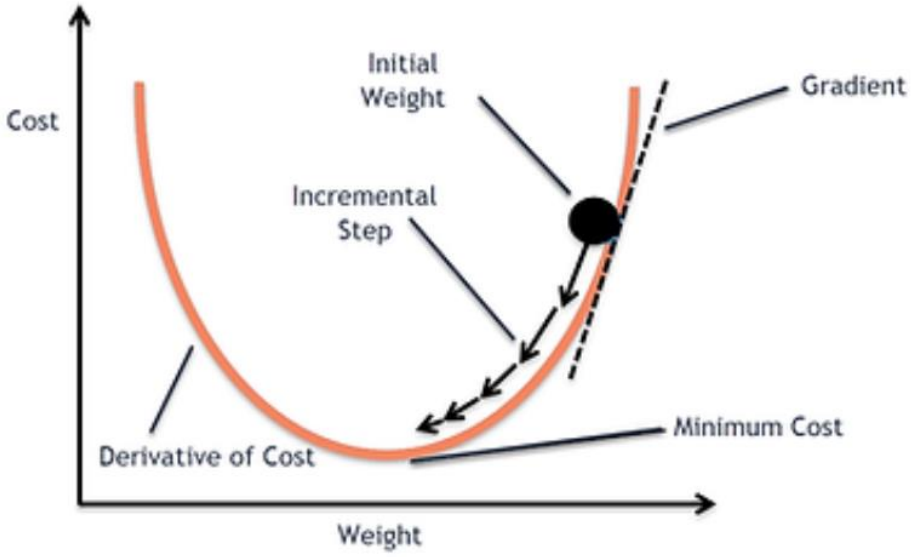

Рисунок 7: Схематичне зображення Gradient Descend

На Рисунку 7 зображено графік loss функції. Наше завдання – дійти до мінімуму цієї функції. Тому параметри оновлюються новими значеннями (w1 = w0- l* dw(cost))

То що ж таке градієнт? «Градієнт визначає наскільки результат функції зміниться, якщо ми трошки змінимо вхідні параметри» - Лекс Фрідман (MIT) .

Важливим параметром для алгоритмів оптимізації є learning rate (l). Цей параметр фактично визначає, наскільки великі кроки ми будемо робити прямуючи до мінімуму. Завелике значення завадить нам дістатися пункту призначення, бо ми будемо «перестрибувати» мінімум. Занадто маленький параметр значно сповільнить роботу мережі, тому необхідно

уважно підійти до цієї справи. В сучасних моделях використовують функції, які поступово зменшують learning rate, в міру того, як ми наближаємося до мінімуму.

З плюсів Gradient Descent: він простий, легко обчислюється та реалізується. Проте, він є дуже повільним та затратним, бо для однієї ітерації доведеться рахувати cost функцію на всіх даних (що часто налічують мільйони зображень). Розглянемо, як інший алгоритм вирішує цю проблему в наступній секції.

3.2.2 Stochastic Gradient Descent (SGD)

SGD – це різновид Gradient Descent, який вирішує його головну проблему – надзвичайно довге та затратне обчислення. Цей алгоритм виконує оновлення на кожному тренувальному прикладі (для кожної пари (x, y)). Це робить його не таким точним, трохи «розмазаним» (noisy), він не завжди буде рухатися в правильному напрямку.

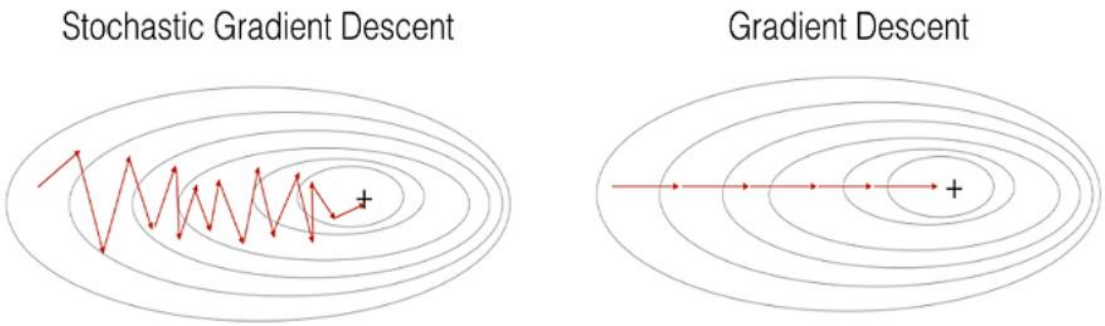

Рисунок 8: SGD vs Gradient Descend. "+" позначає локальний мінімум. Як бачимо, перший шлях видається трошки непевним, але обидва алгоритми доходять до кінцевої мети

Зазвичай, щоб отримати вигоду від обидвох алгоритмів (швидкість та точність), використовується mini-batch gradient descend. Суть в тому, що

ми згодовуємо алгоритму дані в маленьких порціях (батчах), і обраховуємо «cost» функцію на кожній порції. Це дає змогу запобігти великим коливанням SGD та не витратити на тренування моделі роки, як при Gradient Descent. До того ж, коли ми обраховуємо функцію на кожному окремому прикладі, ми втрачаємо пришвидшення за рахунок векторизації.

3.2.3 SGD with momentum

Попри вже наявні покращення, mini-batch gradient descend все ж показує себе як дуже шумний (noisy) алгоритм. Оновлення параметрів залежить від похідної, яка набуває різних значень і не завжди вказує на той самий напрямок. Саме тому було винайдено алгоритм, що згладжує криву оновлень - SGD with momentum. Ідея полягає у використанні експотенційно зваженого середнього для надання ваги попереднім оновленням параметрів.

Тепер формула оновлення коефіцієнтів виглядає так:

$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

В дію також вступає черговий гіперпараметр бетта, який визначає, наскільки важливими для нас є попередні зміни. Зазвичай він коливається біля значення 0.9.

Цей алгоритм має всі переваги SGD, а також доходить до локального мінімуму швидше за GD. З недоліків можна назвати необхідність використовувати та обраховувати нові змінні й параметри.

Отже, ми розглянули деякі з провідних алгоритмів оптимізації, які використовуються зараз, або були основою новіших алгоритмів. В основному, їх принцип роботи зрозумілий та подібний, змінюються лише

деталі, які, проте, дають змогу значно покращити чи пришвидшити результати моделі. Інші популярні алгоритми в тій чи іншій мірі модифікують описані вище. Вони не становлять суттєвого інтересу в рамках цієї роботи, тож в наступній секції ми перейдемо до опису власне предмету дослідження – алгоритму SAM.

3.3 Алгоритм SAM

Алгоритм SAM використовує абсолютно новий підхід до оптимізації. Якщо розглянуті вище алгоритми орієнтуються на мінімізацію loss функції, то SAM мінімізує ще й «різкість», або sharpness ландшафту функції loss. В сучасних нейронних мережах є неймовірна кількість параметрів, що дозволяє виявляти маленькі об’єкти та неочевидні особливості зображення. Проте, це має також і свої мінуси. Нейронні мережі мають тенденцію надмірно пристосовуватися до тренувальних даних (overfit), і, як наслідок, втрачають здатність робити правильні передбачення на нових зображеннях (generalize).

Нові дослідження показують, що ландшафт, і loss function sharpness впливають на здатність мережі до узагальнень [9]. Деякі підходи оптимізації мережі, правильний підбір параметрів, dropout рівні можуть змінити вигляд функції та її здатність узагальнюватися відповідно. SAM optimization полягає в пошуку «сусідів», які мають так само низький показник loss. Емпірично доведено, що алгоритм покращує здатність до узагальнення на таких датасетах як CIFAR{10–100} і ImageNet.[11]

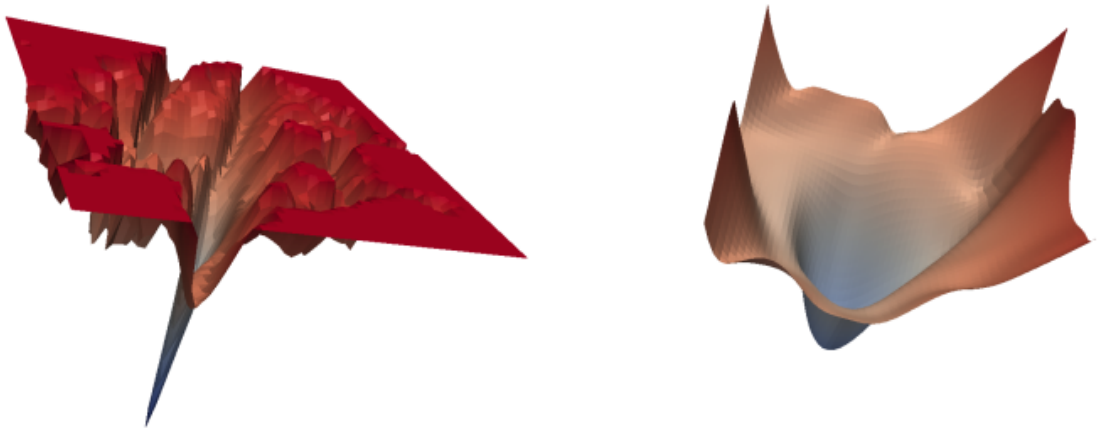

Рисунок 9 "гострий" мінімум отриманий в мережі ResNet з оптимізатором SGD(зліва), мінімум отриманий з використанням SAM алгоритму (справа) [10]

3.3.1 Умовні позначення

Надалі в роботі використовуються такі позначення:

- Тренувальний сет S= $\mathcal{S} \triangleq \cup_{i=1}^{n} \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ взяті з дистрибуції D
- $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^d$ параметри моделі
- Loss function: $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$,
- Training set loss $L_S(\boldsymbol{w}) \triangleq \frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$
- Distribution loss $L_{\mathscr{D}}(\boldsymbol{w}) \triangleq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D}[l(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})]$

Завдання моделі, тренуючись тільки на тренувальному сеті, досягти найменшого показника loss (підібрати параметри w) на сеті дистрибуції.

3.3.2 Теоретична основа

В основі алгоритму лежить наступна теорема: для будь якого p більше 0 з великою імовірністю:

$$L_{\mathscr{D}}(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + h(\|\boldsymbol{w}\|_2^2/\rho^2),$$

Де h : R+ → R+ - строго зростаюча функція.

Для більшої ясності можна переписати формулу в такому вигляді:

$$[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) - L_{\mathcal{S}}(\boldsymbol{w})] + L_{\mathcal{S}}(\boldsymbol{w}) + h(\|\boldsymbol{w}\|_2^2/\rho^2).$$

Те, що знаходиться в квадратних дужках і є «різкістю». Зауважимо, що чим більше loss зростає навколо w, тим більшою буде різкість. Різкість вимірює, як швидко може зрозстати функція loss при незначній зміні параметрів w.

Задля мінімізації функція h була зігнорована (дослідники ще вивчають її вплив на мережу та алгоритм). В результаті ми отримуємо стандартний алгоритм, який легко може бути розв'язаний сучасними машинами:

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w}+\boldsymbol{\epsilon}),$$

Де p це гіперпарметр від 1 до нескінченності, хоча найчастіше використовують значення 2.

3.3.3 Алгоритм

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
  Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
  Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
  Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
  Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
  $t = t + 1$;
end
return $\boldsymbol{w}_t$

Рівняння 1: алгоритм використання оптимізатора[10]

Вхідні дані: тренувальний сет зображень S, loss function, batch size, step size(learning rate), параметр розміру сусідства

Вихід: Модель, натренована з алгоритмом SAM

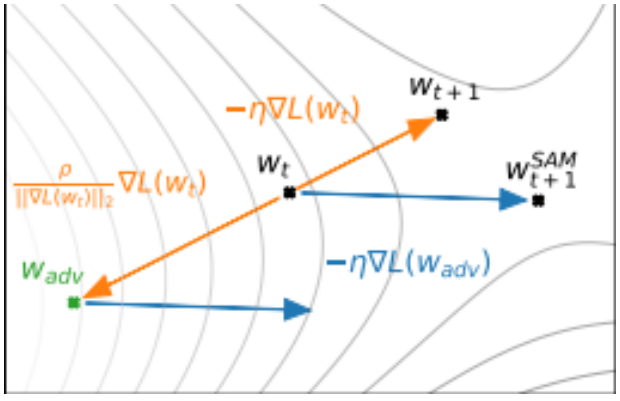

Рисунок 10: Схематичне зображення кроку алгоритму [10]

Як можна побачити на Рисунку 10, з стандартним алгоритмом ми б оновили параметри w -> w t+1, а з алгоритмом SAM ми прямуємо до w_{t+1}^{SAM}

В даному розділі ми оглянули алгоритми оптимізації, їх принцип роботи. Більшість алгоритмів базується на мінімізації функції loss, що допомагає досягти визначних результатів в детекції об’єктів. Щоправда, в останніх дослідженнях вказується на велику кореляцію між ландшафтом функції та її здатністю до узагальнення. Алгоритм SAM враховує цю кореляцію у своїй роботі, тому він є цікавим та перспективним предметом дослідження.

РОЗДІЛ 4: ПРАКТИЧНА ЧАСТИНА. АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

В даному розділі детально подивимося на практичну частину курсової роботи, оглянемо, що було зроблено та які інструменти використано. Також будуть показані результати дослідження та проаналізовано їх наслідки.

4.1 Методологія та використані ресурси

Для виконання дослідження вирішено використати нейронну мережу YOLOv3. ЇЇ код було взято з офіційного репозиторію[12] та модифіковано, для проведення експерименту.

Також було використано відкритий код з оптимізатором SAM [13]. Експеримент було проведено на датасеті PascalVOC[14], це одна з класичних колекцій для завдання детекції зображень. В ньому містяться 20 класів(категорій) об’єктів. Загальна кількість тренувального сету: 16 000 зображень, тестувального: 4 тисячі зображень. Тренувальний тест складається з: val + train sets 2012 та 2007 років, тест сет – test 2007.

Виконання навчання було виконано на платформі Google Colab, що дозволило ефективно використати ресурси та значно пришвидшити експерименти.

Для навчання було використано вже натреновані параметри моделі та стандартні гіперпараметри, запропоновані авторами досліджень. Розмір зображення: 417x417, розмір batch – 16. В якості базового оптимізатора для алгоритму SAM було обрано SGD. Задля отримання релевантних результатів обидві моделі (з SAM та без SAM) були натреновані на 35

епох. Візуалізації результатів ми завдячуємо платформі Weights& Biases[15]

4.2 Практична частина

Використання алгоритму SAM досить просте, і не вимагає великих змін. В цілому, ми маємо модифікувати код всього у 2 місцях:

1) Ініціалізація оптимізатора

```
base_optimizer = SGD
optimizer = SAM([
            {'params': g0},
            {'params': g1, 'weight_decay': hyp['weight_decay']},
```

2) Зміна в циклі train

```
def closure():
    loss, loss_items1 = compute_loss(model(imgs), targets.to(device))
    loss.backward()
    return loss

# Forward
with amp.autocast(enabled=cuda):
    pred = model(imgs)  # forward
    loss, loss_items = compute_loss(pred, targets.to(device))  # loss scaled by batch_size
    if RANK != -1:
        loss *= WORLD_SIZE  # gradient averaged between devices in DDP mode
    if opt.quad:
        loss *= 4.

# Backward
loss.mean().backward()

# Optimize
if ni - last_opt_step >= accumulate:
    optimizer.step(closure)
    optimizer.zero_grad()
```

Отже, спочатку ми ініціалізуємо оптимізатор, і передаємо йому як параметр базовий оптимізатор. В нашому випадку це SGD з бібліотеки

pytorch. В головному циклі ми оголошуємо функцію closure, де відбувається один «прохід» по мережі. Обрахунок loss функції та gradient descend (backward prop). Цю функцію ми передаємо аргументом в оптимізатор, і він виконує подвійний прохід через мережу та оновлює параметри враховуючи sharpness loss функції. Варто зауважити, що за таких умов навчання відбувається в 2 рази повільніше, адже на кожній епосі ми проходимо кроки forward-backward 2 рази.

4.3 Аналіз результатів

От ми й дійшли до найцікавішого етапу дослідження – аналізу результатів. Чи вдалося нашому алгоритму покращити результати роботи мережі? Чи вдалося покращити узагальнення та зменшити пристосування до тренувального сету? Давайте поглянемо.

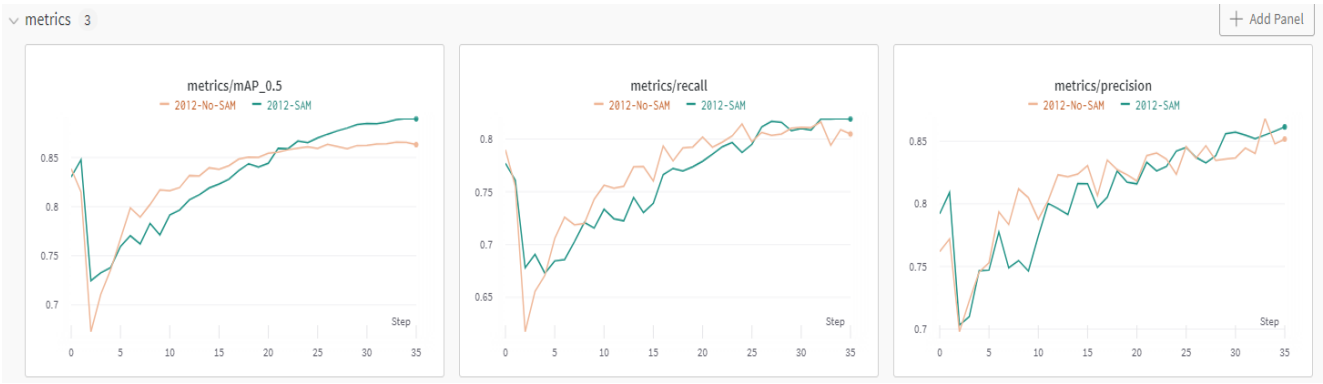

Рисунок 11: Метрики для оцінки роботи мережі. Зеленим кольором позначена модель з використанням SAM, жовтим – без

Як бачимо, ми маємо незначне покращення показника mAP .5 під кінець тренування. В інших метриках Recall та Precision обидва алгоритми демонструють схожі результати, і важно сказати який показав себе краще.

Але поглянемо на показники loss в тренувальному та валідаційному сеті – саме вони підкажуть нам, чи добре алгоритм справляється з пристосуванням та чи виправдовує наші очікування.

Рисунок 12: loss values on train and val datasets

З графіків на Рисунку 10 можемо бачити, що на тренувальних даних оригінальна модель показує трохи кращий результат. Але якщо ми звернемо увагу на дані валідації (нижній рядок), то очевидною стає тенденція алгоритму до зациклення на тренувальних даних (overfitting). Особливо це видно на графіку Obj_loss, де приблизно до 20 епохи результат поступово покращувався, але далі ми спостерігаємо збільшення кількості помилок. На противагу, модель що використовує SAM продовжує вдосконалюватися та знижувати кількість помилок. Схожу ситуацію маємо в метриках class loss (відповідає за визначення класу об'єкту) box loss (відповідає за визначення координат). Наявні дані дають підстави думати, що зі збільшенням епох результати будуть ще більш разючими, а модель SAM продовжить покращення в метриках mAP, Recall, Precision.

ВИСНОВКИ

Метою цієї курсової роботи є дослідження алгоритму оптимізації SAM для задачі детекції (виявлення) об’єктів. В теоретичній частині було оглянуто загальний стан розвитку машинного навчання та комп’ютерного зору, сучасні рішення задачі виявлення об’єктів, алгоритми оптимізації цих рішень. Детально описано алгоритм дії Sharpness Aware Minimization та його теоретична основа, модель нейронної мережі YOLOv3, що була використана пізніше в практичній частині.

В рамках практичної частини було натреновано модель YOLOv3 на даних PascalVOC в двох варіантах: з алгоритмом оптимізації SAM та без нього. Такий підхід дозволив порівняти отримані результати й зробити певні висновки що до дієвості алгоритму. В загальному, обидві моделі надали схожі результати, проте зі збільшенням кількості епох можемо спостерігати позитивну динаміку моделі з алгоритмом SAM. В той час коли звичайна модель почала занадто пристосовуватися до тренувальних даних та погіршувати результати на тестових зображеннях, модель з застосуванням SAM виявилася стійкою до цього. Можемо констатувати, що аналізований алгоритм допомагає моделі краще узагальнювати свої прогнози та видавати надійніші результати. Можна припустити, що при подальшому тренуванні модель дасть більш однозначні результати.

В подальшому пропонується розглянути алгоритм Adaptive Sharpness Aware Minimization, провести тренування на більше епох та проекспериментувати з різними гіперпараметрами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ImageNet Large Scale Visual Recognition Challenge/ Olga Russakovsky et al., 30 Jan 2015. URL: https://arxiv.org/abs/1409.0575 (date of access: 15.05.2022).
2. Rich feature hierarchies for accurate object detection and semantic segmentation/ Ross Girshick et al, UC Berkeley, 22 Oct 2014
3. You Only Look Once: Unified, Real-Time Object Detection/ Joseph Redmon et al., 9 May 2016. URL: https://arxiv.org/pdf/1506.02640.pdf (date of access: 16.05.2022)
4. J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. International journal of computer vision, 104(2):154–171, 2013
5. Joseph Redmon, Ali Farhadi. YOLO9000: Better, Faster, Stronger. 25 Dec 2016. URL: https://arxiv.org/pdf/1612.08242.pdf (date of access: 16.05.2022)
6. Jason Brownlee A Gentle Introduction to Object Recognition With Deep Learning (update of January 27, 2021) URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/ (date of access: 19.05.2022)
7. Joseph Redmon, Ali Farhadi. YOLOv3: An Incremental Improvement. URL: https://arxiv.org/pdf/1804.02767.pdf (date of access: 19.05.2022)
8. Sharpness-Aware Minimization for Efficiently Improving Generalization by P. Foret et al. (December 2020).
9. Hao Li el al. Visualizing the Loss Landscape of Neural Nets. URL: https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf (date of access: 20.05.2022)
10. Sharpness-aware Minimization for Efficiently Improving Generalization / Pierre Foret el al. 28 Sep 2020 URL:

https://openreview.net/forum?id=6Tm1mposlrM (date of access: 20.05.2022)

11. Nagesh Singh Chauhan: Optimization Algorithms in Neural Networks URL: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html (date of access: 20.05.2022)
12. GitHub repository: https://github.com/ultralytics/yolov3 (date of access: 20.04.2022)
13. GitHub repository: https://github.com/davda54/sam/ (date of access: 20.04.2022)
14. PascalVOC dataset: http://host.robots.ox.ac.uk/pascal/VOC/ (date of access: 20.04.2022)
15. Web-site: https://wandb.ai/home