МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Модель і метод навчання резільєнтної до збурень системи розпізнавання медичних діагностичних зображень»
здобувача групи ІН.м-24 Кугука Василя Олександровича

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

Василь КУГУК
(підпис)

Керівник,
кандидат технічних наук, доцент В'ячеслав МОСКАЛЕНКО
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-24 Кугука Василя Олександровича

1. Тема роботи: «Модель і метод навчання резільєнтної до збурень системи розпізнавання медичних діагностичних зображень»
затверджую наказом по СумДУ від *«01» грудня 2023 року № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Опис технології оптимізації резільєнтності інтелектуальної діагностичної системи розпізнавання медичних зображень. 3) Розробка інформаційного та програмного забезпечення. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.11- 11.11.23	
2	*Опис технології оптимізації резільєнтності інтелектуальної діагностичної системи розпізнавання медичних зображень*	11.11-15.11.23	
3	*Розробка інформаційного та програмного забезпечення*	15.11-20.11.23	
4	*Аналіз отриманих результатів*	15.11-20.11.23	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	29.11-11.12.23	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 65 стр., 9 рис., 3 табл., 2 додатки, 44 використаних джерел.

Обґрунтування актуальності теми роботи – актуальність дослідження полягає у необхідності вирішення задачі підвищення стійкості систем штучного інтелекту до деструктивних впливів, що залишається важливою для критичних застосувань в сфері безпеки. У контексті оптимізації резільєнтності штучного інтелекту відзначається потреба в використанні ідей і методів мета-навчання.

Об'єкт дослідження — процес мета-навчання для оптимізації резільєнтності, а предметом є архітектурні розширення та методи, спрямовані на покращення стійкості до протиборчих атак, інжекцій несправностей та дрейфу концепцій.

Мета роботи — підвищення резільєнтності системи розпізнавання діагностичних зображень.

Методи дослідження — методи, які комбінують ідеї протиборчого навчання, навчання з ін'єкцією несправностей, та дрейфу концепцій в умовах обмеженої кількості зразків та стратегії градієнтної оптимізації.

Результати — розроблено інформаційне та програмне забезпечення, яке використовує незалежний від моделі метод мета-навчання, який показав підвищену ефективність на прикладі класифікації зображень. Експерименти, проведені на різних алгоритмах із використанням різних метрик, свідчать про покращення стійкості системи штучного інтелекту за допомогою запропонованого методу.

РЕЗІЛЬЄНТНІСТЬ, ДЕСТРУКТИВНІ ЗБУРЕННЯ, МЕТА-НАВЧАННЯ, ІН'ЄКЦІЯ НЕСПРАВНОСТЕЙ, ПРОТИБОРЧІ АТАКИ, ДРЕЙФ КОНЦЕПЦІЙ .

ЗМІСТ

ВСТУП 5
1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ 7
1.1 Сучасний стан та тенденції розвитку систем розпізнавання медичних діагностичних зображень 7
1.2 Деструктивні збурення на інтелектуальні системи 14
1.3 Сутність резільєнтності інтелектуальних систем 22
1.4 Формалізована постановка задачі 25
2 ТЕХНОЛОГІЯ ОПТИМІЗАЦІЇ РЕЗІЛЬЄНТНОСТІ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИЧНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ МЕДИЧНИХ ЗОБРАЖЕНЬ 27
2.1 Модель адаптера 27
2.2 Алгоритм мета-навчання для оптимізації резільєнтності 28
2.3 Критерій ефективності нейромережевого класифікатора 30
3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМИ ОПТИМІЗАЦІЇ РЕЗІЛЬЄНТНОСТІ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИЧНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ МЕДИЧНИХ ЗОБРАЖЕНЬ 33
3.1 Формування навчальних і тестових даних 33
3.2 Короткий опис програмного забезпечення 34
3.3 Аналіз результатів експериментів 36
ВИСНОВКИ 39
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40
ДОДАТОК А 47

ВСТУП

Технології штучного інтелекту набувають широкого використання в системах медичної діагностики стану здоров’я людини. В переважній більшості випадків алгоритми штучного інтелекту здійснюють аналіз візуальної інформації, що збирається діагностичним обладнанням. Одначасно з цим розвиваються методи формування деструктивних збурюючих впливів на системи штучного інтелекту. Протиборчі атаки (Adversarial attacks) та інжекція пошкоджень вагових коефіцієнтів нейромереж (Fault injections) можуть викривити результати діагностики, що призведе до неправильного лікування пацієнта. Тому підвищення резільєнтності систем розпізнавання медичних діагностичних зображень стає особливо актуальною. Медицина вимагає максимальної точності при діагностуванні, тому здатність системи поглинати збурення або швидко адаптуватися до них є критичною важливою.

Дослідники часто розглядають робастність і швидкість адаптації окремо. Так само існуючі методи забезпечення робастності розроблені, як правило, для поглинання окремих видів збурень. Сумісність таких методів при комбінованому використанні є малодослідженим питанням. Проте в реальному світі збурюючі фактори можуть поєднуватися і взаємопідсилювати деструктивний вплив на інтелектуальну систему діагностики. Надійність та резільєнтність систем розпізнавання діагностичних даних не тільки підвищують довіру медичних фахівців, але і гарантують безпеку пацієнтів. Комплексний підхід до дослідження резільєнтності може допомогти у розробленні більш універсальних та ефективних систем діагностики.

Тому задача розроблення моделі та методу навчання системи розпізнавання медичних діагностичних зображень, які забезпечують резільєнтність до деструктвиних збурюючих факторів, є актуальною.

Мета роботи – підвищення резільєнтності системи розпізнавання діагностичних зображень.

Об’єкт дослідження — процес оптимізації резільєнтності системи

розпізнавання діагностичних зображень.

Предмет дослідження — моделі і методи навчання резільєнтності систем розпізнавання діагностичних зображень з резільєнтністю до різнотипних збурень.

Методи дослідження — методи технології нейронних мереж, мета-навчання, методи іжекції несправностей до пам’яті, методи формування протиборчих атак.

Для досягнення мети дослідної роботи здійснюється формування вхідних навчаних та тестових вибірок з наборів даних MedMNIST v2 (https://medmnist.com/), розроблено модель та метод мета-навчання. Здійснюється оцінювання резільєнтності запропонованих рішень на вибірці реалізацій збурень, що діють на дані та вагові коефіцієнти моделі.

Робота має теоретичний та прикладний характер, заппропоновано методи формування збурень для мета-оптимізації резільєнтності системи розпізнавання медичних діагностичних зображень та порівнюється результати отримані в рамках тредиційного і запропонованого підходу. Результати, отримані на вибірках з відкритих наборів даних MedMNIST v2 підтверджують придатність моделей і алгоритмів мета-навчання для практичного використання. Запропонований метод забезпечує в середньому на 5,73% кращий інтегральний показник резільєнтності до ін'єкцій помилок порівняно зі звичайним навчанням в умовах інжекції помилок. Крім того, запропонований метод забезпечує кращий інтегральний показник резільєнтності до протиборчих атак ухилення порівняно з протиборчим навчанням в середньому на 6,40%. Він також продемонстрував середнє покращення на 5,3% інтегрального показника резільєнтності до зміни завдань порівняно зі звичайним тонким налаштуванням адаптерних блоків.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасний стан та тенденції розвитку систем розпізнавання медичних діагностичних зображень

Сучасний етап розвитку медичної галузі характеризується зростанням складності методів обстеження й діагностування, лікування й прогнозування перебігу та наслідків патологічних процесів. Функціональна ефективність процесів діагностування, лікування та реабілітації пацієнтів суттєво залежить від професійного рівня та досвіду лікаря. Тому охорона здоров'я є однією із галузей соціально-економічної сфери суспільства, де інтенсивно впроваджуються інтелектуальні технології. При цьому найбільш інформативним і найбільш поширеним способом подання діагностичної інформації є зображення або відеозобрання [1].

На рис. 1.1 [2] представлено концептуальну структуру інтелектуальної системи медичної діагностики з аналізом діагностичних зображень. Термін "сегментація" вказує на виявлення однотипних ділянок у візуальних данних. Первинне оброблення включає отримання зображення з вхідної послідовності, визначення об'єкта для ідентифікації та його пересилання до наступної стадії. Процес фільтрування полягає в видаленні фонових елементів, потенційних помилок та шумів, що могли виникнути під час створення зображення. Фаза розпізнавання представляє собою кінцевий крок оброблення візуальних даних.

Визначення оптимального методу оброблення медичних діагностичних зображень є важливим питанням, на яке немає однозначної відповіді, оскільки немає універсальних методів для зображень різного типу. Основна ціль оброблення таких зображень – подання даних про них у новому форматі [3]. При цьому особливе значення мають методи екстракції ознакового опису. Кожна ознака має бути специфічною для конкретного класу розпізнавання, що в реальних задачах буває не так часто.

Методи екстракції ознак можна умовно поділити на такі категорії [4]:

1) методи екстракції незмінюваних структурних ознак (моментів, потужності тощо);

2) методи екстракції змінюваних структурних ознак (частот, спектрів яскравості тощо);

3) методи екстракції ознак з описом структури.

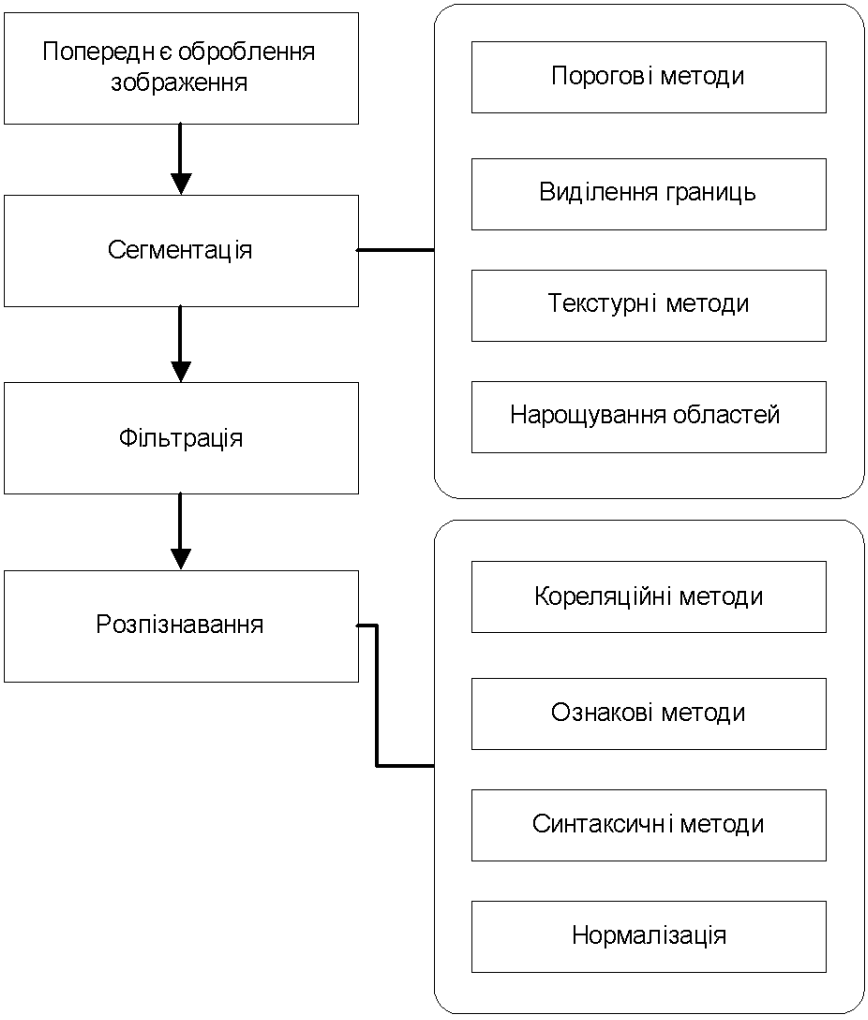


Рисунок 1.1 – Узагальнена структура інтелектуальної діагностичної ситеми

Велика група методів оброблення медичних зображень поєднують комплекс мультиканальних алгоритмів. В цьому контексті оригінальне зображення декомпонується на кілька частин за допомогою різних наборів фільтрів [3, 4]. Один з прикладів застосування даного методу показано у праці

[5], де розглядається використання Габорового фільтру та вейвлет-перетворень. У праці [6] розглядається застосування мультифрактального аналізу в контексті оброблення медичних діагностичних зображень. Оброблені таким чином зображення використовувалися при діагностуванні онкологічних захворювань у пацієнтів. Однак отримана точність становила 76%, що не досить прийнятно для практичного використання.

Вручну спроектовані ознаки, що використовуються у класичних методах машинного навчання, не є гнучкими і адаптованими до новизни, викидів і довільних умов формування діагностичних зображень. Тому у сучасних діагностичних системах широко використовуються здатні до навчання екстрактори ознак для автоматичного формування ознаки з сирих вхідних даних. Такі екстрактори ознак, як правило, основані на методах глибокого машинного навчання [7]. Саме навчання ієрархічного ознакового подання з великого обсягу нерозмічених даних є найбільш потужним інструментом аналізу інструментом аналізу діагностичної інформації (рис. 1.2).

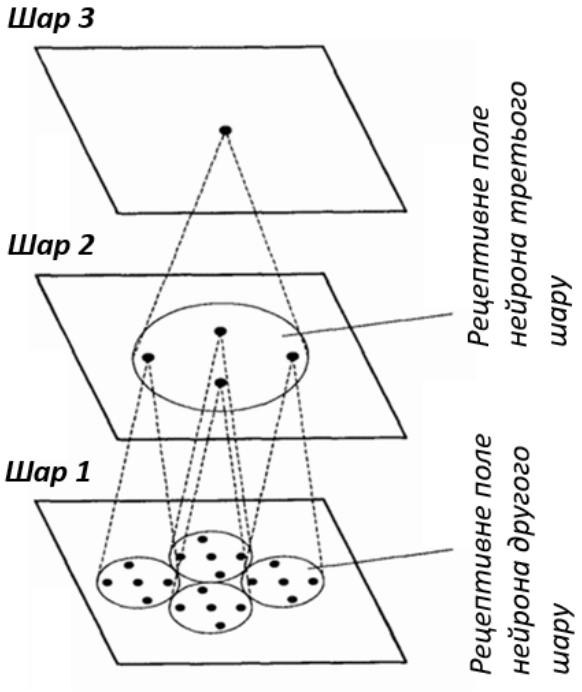


Рисунок 1.2 – Схема ієрархічного ознакового подання даних

Високорівневі ознаки формуються шляхом композиції низькорівневих. Вище по ієрархії знаходиться менше ознак, проте вони є складнішими, інформативнішими і менш чутливими до варіації спостережень одного і того ж класу розпізнавання. Як неієрархічні, так і ієрархічні моделі здатні апроксимувати будь-яку функцію з однаковою точністю, проте ієрархічні моделі потребують меншу кількість параметрів для цього [8].

Глибокі мережі можна поділити на повнозвязні, рекурентні, згорткові та основані на трансформаторах [4, 7, 8]. У повнозвязних мережах кожен нейрон пов’язаний з кожною просторово-часовою ознакою, що приводить як до високої обчислювальної складності моделі так і до ефекту перенавчання, коли модень втрачає властивість узагальнення і просто запам’ятовує навчальну вибірки. Рекурентні мережі призначені для аналізу послідовностей, де кожен нейрон спостерігає лише за обмеженим часовим проміжком, меншим ніж вікно пошуку темпорального шаблону. В той час як у повнозвязній мережі кожен нейрон повинен приймати на вхід дані з усіх моментів часу у вікні пошуку темпорального шаблону. Тобто рекурентні мережі здійснюють сканування даних в часовому вікні. Аналогічно у згортковій нейронній мережі кожен нейрон приймає на вхід лише частину просторового образу, тобто сканує вхідний образ фільтрами з ядром обемеженого розміру. Згорткові нейронні мережі є обчислювально ефективні і більш універсальні, оскільки можуть обробляти також і послідовності (послідовні кадри) для пошуку короткострокових темпоральних шаблонів [9].

Згорткові нейронні мережі з декількома шарами допомагають створити високорівневе представлення ознак спостережень з 1D, 2D та 3D топологією [9]. Вони вже проявили себе в ролі ефективних інструментів для машинного зору та аналітики часових послідовностей. З 1989 року до нині відбулися численні покращення в архітектурі згорткових мереж. Ці покращення включали опрацювання параметрів, введення регуляризаційних методів, перебудову структури та інше. Але сучасні напрямки удосконалення згорткових мереж

передбачають перегляд та переробку вже існуючих обробних компонентів, а також розробку нових структурних блоків і модулів. Залежно від виду архітектурних змін, згорткові мережі можуть бути розділені на сім ключових категорій (див. рис. 1.3) [10]:

– використання можливостей простору (spatial exploitation) за рахунок застосування різномасштабних згорткових фільтрів;

– використання можливостей глибини (depth exploitation) за рахунок збільшення кількості шарів і замикань для зниження ефекту затухання градієнту на нижніх шарах;

– використання можливостей мульти-шляхів (multi-path exploitation) за рахунок застосування міжшарових з’єднань та навчання підмереж на основі dropout і drop-connection;

– використання можливостей ширини (width exploitation) за рахунок збільшення ширини нейронної мережі на основі мульти-з’єднань і збільшення кількості фільтрів;

– використання можливостей карти ознак (feature map exploitation) за рахунок підвищення інформативності ознак основі селекції/зважування та стиснення-збудження (Squeeze-Excitation);

– використання можливостей підсилення каналів (channel boosting exploitation), за рахунок додавання нових каналів до оригінальних каналів, що сформовані за на основі переносу знань (transfer learning) або генеративних моделей;

– використання модулів уваги (attention based CNNs) у вигляді м'яких масок для карти ознак з метою фокусування моделі на важливих ділянках карти ознак.

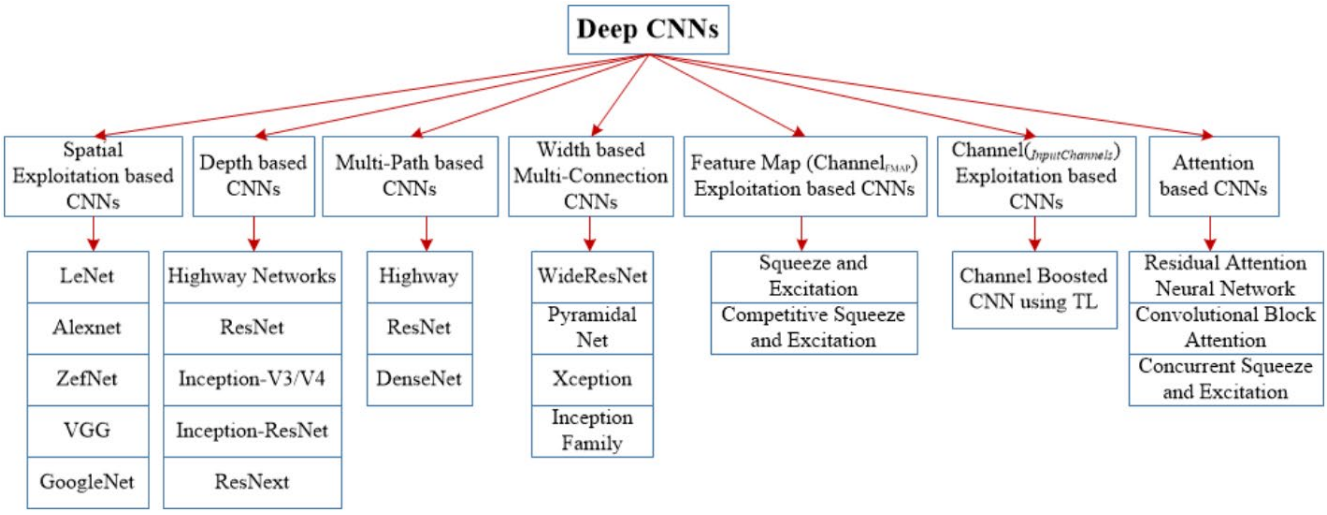


Рисунок 1.3 – Таксономія багатошарових (глибоких) згорткових нейронних мереж

Окрім архітектурних вдосконалень активно розробляються нові функції втрат та протоколи машинного навчання. При цьому для вирішення задач ефективного навчання класифікаційних моделей за умов незбалансованого та обмеженого обсягу розмічених навчальних зразків набули широкого використання два основні підходи до екстракції компактного ознакового подання. В рамках першоного підходу використовується автоекодери та різні методо самонавчання (self-learning), що навчаються без вчителя на великому обсязі нерозмічених навчальних даних, що дозволяє виявити структуру вхідних даних [11]. В рамках другого підходу реалізують навчання метрики подібності (similarity metric learning) на вибірці розмічених зразків, що дозволяє виявити відношень схожості між зразками різних класів.

Для оптимізації роботи нейронних мереж велике застосування знайшли системи уваги (Attention) та самоуваги (self-attention) [12], які допомагають ураховувати зв'язки між віддаленими елементами вхідної інформації та її проміжними обробками. Така модернізація привела до створення нових структурних елементів – трансформерів. Трансформер є послідовністю модулів уваги, що дозволяють перетворити вхідний ряд векторів на інший ряд, що містить контекст кожної одиниці [13]. Вхідні дані подаються у векторній формі

(embedding) із додаванням позиційного коду (positional encoding) для зберігання послідовності елементів. Для аналізу різних граней вхідних даних застосовуються паралельні модулі самоуваги (Multi-Head Attention), які створюють набір відображення підпросторів (representation subspaces).

Елементи на вході трансформера називають токени. У контексті машинного зору, ці токени представлені як частинки зображення чи фрагменти карт ознак високого рівня, які конвертуються в вектори перед тим, як вони надходять до модулю самоуваги. Зазвичай у трансформерах розмір токену залишається стабільним на вході та виході. Але в області машинного зору потрібно аналізувати детальний контекст у нижніх рівнях і загальний контекст у верхніх рівнях структури. Тому у Swin-трансформерах початковий енкодер працює з фрагментами розміром 4x4 пікселі, після чого менші фрагменти об'єднуються у більші блоки (Patch Merging) [14]. Оскільки модуль уваги є найбільш обчислювально вимогливою частиною трансформера, Swin-трансформери обмежують область уваги для кожного токена, звертаючи увагу лише на обмежену кількість сусідніх токенів. Для забезпечення високої якості представлення після кожного блоку, який включає віконні модулі уваги (Window Multi-Head Attention), використовуються шари із зміщеннями в рамках цих вікон уваги.

Здатність трансформерів моделювати довготривалі залежності в даних дозволяє здійснювати ефективне виділення глобальних ознак і зменшити індуктивне упередження моделі. Завдяки цьому за наявності великого обсягу навчальних даних трансформерам вдається досягти точності згорткових мереж, а інколи і перевершити їх точнісні характеристики. Проте попри всі існуючі удосконалення візуальних трансформерів в архітектурах BIT, PiT і SWIN трансформери досі поступаються згортковим нейронним мережам за обчислювальною складністю та швидкістю навчання на вибірках обмеженого обсягу [15].

1.2 Деструктивні збурення на інтелектуальні системи

Досить часто системи штучного інтелекту працюють в далеко від ідеальних умовах і можуть бути піддані взаємодії з деструктивними факторами. Ці деструктивні фактори можуть цілити як на середовище, де система розгорнена, так і на інформацію під час навчання або тестування (екзамену). Такі впливи можуть мати зловмисний характер або ж виникати природним чином. Джерелами деструктивного впливу на системи штучного інтелекту можуть бути наступні чинники:

– апаратні несправності;

– шум та змагальні атаки.

– дрейф концепцій;

– новизна в тестових даних;

– пропуски і помилки в даних.

Апаратні несправності (відмови) в обчислювальному обладнанні викликають помилки. Термін "помилка" вказує на такі прояви відмов у системі, коли реальний стан компонента системи не збігається з очікуваним [16]. Якщо відмова не призводить до помилки, її називають пасивною. Помилки, у свою чергу, можуть спричинити збої, коли система не може реалізувати свої задумані функції або дії. На рис. 1.4 зображено послідовність від апаратної відмови до збою, в результаті якої порушується нормальна діяльність нейронної мережі, що діє у комп'ютерному контексті.

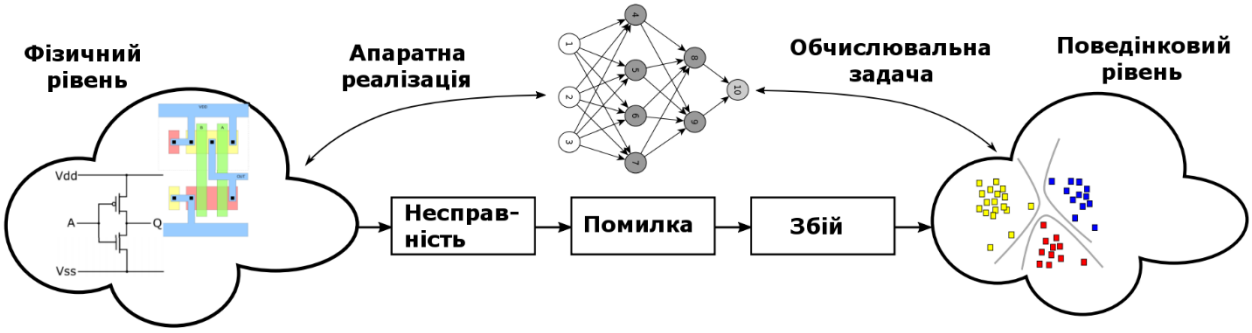


Рисунок 1.4 – Поширення несправності з фізичного рівня обчислювального середовища до поведінкового рівня нейромережевого додатку

Відмови (несправності) можна класифікувати за часовими характеристиками появи [16]:

– постійна відмова, що, як правило, є результатом незворотного фічичного пошкодження, є стабільна в часі;

– непостійна відмова, що часто є результатом зовнішніх збоїв, зберігається лише протягом короткого періоду часу.

Існують декілька фізичних способів введення відмов з метою нанесення шкоди. У реальних умовах відмови часто запроваджуються через збої в системних лічильниках, а саме в синхронізації схем [17], падіння напруги до певного значення [18], електромагнітне взаємодія з напівпровідниками [19], бомбардування великими іонами, лазерне опромінення пам'яті [20], а також за допомогою програмних атак типу rowhammer на біти пам'яті [21].

Лазерний випромінювач може викликати невірну роботу в SRAM. Під час його дії на кремнієві структури утворюється короткочасний електропровідний шлях, що веде до перемикання транзистора визначеним і керованим методом [20]. Адаптуючи характеристики лазеру, такі як діаметр, потужність випромінювання та місце взаємодії, атакуюча особа може маніпулювати окремими бітами SRAM [18]. В минулому лазери часто комбінувалися з

диференціальними методами аналізу несправностей, щоб отримати доступ до секретних ключів криптографічних мікросхем.

Атаки типу Rowhammer можуть викликати неправильну роботу в DRAM-пам`яті. Такий метод атаки базується на електричній взаємодії між ближніми комірками зберігання [21]. Часті запити до конкретного сегмента фізичного зберігання можуть призвести до зміни значення біту в сусідньому секторі пам’яті. Аналізуючи манери зміни бітів у DRAM-модулі та використовуючи функції управління пам’яттю, row hammer може стабільно змінювати біт за визначеною адресою в стеку програмного обладнання. Атаки Rowhammer часто застосовувались для порушення захисту пам'яті у віртуалізованих системах та отримання повних прав в Android-системах.

Лазерне випромінювання та атаки типу Rowhammer можуть з великою точністю здійснювати ін’єкцію відмов у пам'ять. Проте, для введення множини відмов необхідно регулювати лазер та переміщати цільові дані в пам'яті у випадку з Rowhammer. Ці процеси вимагають додаткових витрат, тому при розробці алгоритмів для нейронних мереж необхідно забезпечити стійкість до певної порції інвертованих бітів, аби ці атаки стали неприйнятними з практичної точки зору.

Вчені у сфері штучного інтелекту встановили, що алгоритми на базі нейромереж вразливі до так званих протиборчих атак (Adversarial attacks). Ці атаки полягають в додаванні спеціальних спотворень до навчальних або тестових даних з метою введення інтелектуальної системи в оману [22].

Стратегії змагальних атак можна розподілити на три типи: ухильні атаки (evasion), атаки отруєння (poisoning) та атаки "оракула" [23]. Ухильні атаки спрямовані на збивання з пантелику інтелектуальних систем під час їхньої роботи з прийняттям рішень, використовуючи процес оптимізації для знаходження незначних відхилень, які призведуть до невірних висновків. Залежно від частоти оновлення та оптимізації, ухильні атаки класифікуються як одноразові та ітераційні. Ітераційні атаки формують більш стійкі змагальні

зразки, але є обчислювально складними. Атаки отруєння мають на меті пошкодити дані чи логіку моделі для погіршення навчальних результатів [24]. Атаки "оракула" використовують доступ до програмного інтерфейсу для створення замінних, або сурогатних, моделей, які відтворюють більшу частину можливостей оригінальної моделі. Це допомагає у детальнішому розробленні ухильних атак для сурогатних моделей, які потім можна застосовувати до оригінальної моделі. Атаки "оракула" можна поділити на екстракцію, інверсію та виведення [23]. Метою екстракційних атак є отримання архітектурних особливостей моделі через аналіз вихідних прогнозів та ймовірностей класифікації. Інверсійні атаки спрямовані на відтворення даних, які використовувалися для навчання. Виведення дозволяє атакуючому розпізнавати специфічні елементи даних із загального набору навчальних даних.

Залежно від рівня знанань стосовно моделі аналізу даних, на які опираються протиборчі атаки, їх можна класифікувати на такі типи:

– атаки "білої шухляди" (white-box attacks), де зловмисник має повну інформацію про дані, модель і алгоритм навчання. Цей метод може використовуватися, наприклад, розробниками систем для підвищення якості моделі та перевірки її стійкості;

– атаки "сірої шухляди" (gray-box attacks), де зловмисник має часткову інформацію, достатню для атаки на систему, але не повну;

– атаки "чорної шухляди" (black-box attacks), де зловмисник має доступ лише до інтерфейсу для відправки даних і отримання відповіді, яка може бути в формі класифікації або імовірностей класів. Цей вид атаки становить найбільшу загрозу на практиці.

Основною складовою протиборчої атаки ухилення є протиборчий зразок даних x', що утворюється шляхом додавання шумової складової $x' = x + \varepsilon$, яка призводить до значної зміни рішення на виході нейромережевої моделі, що описується функцією $f(x)$, тобто $f(x') \neq f(x)$.

У випадку протиборчих атак “білого ящика” вагові коєфіцієнти θ нейромережі вважаються незмінними і оптимізація відбувається лише щодо вхідного тестового зразка x шляхом додавання оптимізованої шумової компоненти $x' = x + \varepsilon$. Серед методів вирішення даної оптимізаційної задачі найбільшої популярності набули метод швидкого знаку градієнта (Fast gradient sign, FGSM), алгоритм проекційного градієнтного спуску (projected gradient descent, PGD), атака Карліна і Вагнера (Carlini and Wagner, C&W), ітераційний алгоритм Бройдена-Флетчера-Гольдфарба-Шанно (L-BFGS-алгоритм) та їх модифікації.

Для створення атак "чорної шухляди" використовують різні підходи, такі як створення альтернативної моделі, оцінювання градієнта та різноманітні евристичні методи.

Для створення сурогатної моделі, зловмисник використовує введені дані для атаки на цільову модель та аналізує вихідні результати, які ця модель надає для розмітки даних [23]. Отримані дані з розміткою можуть бути використані для подальшої настройки сурогатної моделі або ж для дистиляції знань у сурогатну модель. Після навчання сурогатної моделі можна використовувати атаки “білої шухляди” для створення змагальних прикладів. Однак важливо враховувати, що успішність подібних атак суттєво залежить від якості навчання сурогатної моделі. Перенесення змагальних атак, розроблених для сурогатної моделі, на оригінальну модель може мати обмежений успіх у випадках, коли маємо справу з великими обсягами даних (наприклад, у задачі ImageNet), оскільки отримати високоякісний сурогат є важким завданням.

Останніми роками, на додачу до вже зазначених методів формування змагальних атак, було розглянуто чимало евристичних методик. Одним із найдоступніших є метод атаки, що базується на межі рішень (Decision Boundary Attack) [24]. Ця техніка розпочинає процес знаходження змагального спотворення з високих амплітуд, які викликають помилки у виводах нейромережі. Далі відбувається зниження амплітуди спотворення, здійснюючи

процес рандомного блукання між коректним та некоректним виводом, але зберігаючи його змагальний характер. Схожим чином у дослідженні [25] була представлена ідея пошуку універсального спотворення, що є незалежним від вхідних даних, для атак типу "чорний ящик". А в роботах [26] пропонується створювати універсальні змагальні атаки "чорного ящика" з використанням процедурних шумів, таких як шум Ворлі, шум Габора або шум Перліна.

Сучасні методи створення протиборчих атак неперервно вдосконалюються і поєднуються, щоб виходити за рамки захисних заходів. Існує безліч підходів до оцінки стійкості моделей у відношенні до змагальних атак. Проте ці оцінки зазвичай стосують певний рівнів збурень, які виражаються у метриках L_0, L_1, L_2 або L_∞ норми, що робить порівняння різних методів та оцінювання поточний стан проблеми дещо ускладненими. Також сам метод оцінювання має власні обмеження, винятки і прийняті компроміси, що ускладнює завдання об'єктивної оцінки стійкості моделей. Важливо відзначити, що більшість протиборчих атак розроблені для сценарію "білого ящика", і це обмеження ускладнює аналіз гібридних та нестандартних моделей.

Оскільки реальні середовища, які аналізує інтелектуальна модель, як правило, є нестаціонарними, у спостереженнях можуть виникати непередбачувані зміни. Якщо модель не встигає адаптуватися, то її знання не відповідатимуть закономірностям спостережуваної реальності. Ефективність такої моделі знизиться, оскільки відбудуться зміни у ймовірнісному розподілі даних. Саме такі зміни називають дрейфом концепцій. Прийнято виділяти такі типи дрейфу концепцій [27]:

– реальний дрейф концепцій (Real Concept Drift);
– віртуальний дрейф концепцій (Virtual Concept Drift);
– зміна апріорних ймовірностей концепцій (Class Prior Concept Drift).

Реальний дрейф концепцій означає зміну умовної ймовірності $P(y|x)$ без зміни чи з мінімальною зміною розподілу даних в просторі $P(x)$. Іншими словами реальний дрейф концепцій полягає у зміні залежності між вхідними

даними і цільовою змінною. Тобто $P_t(y \mid x) \neq P_{t+1}(y \mid x)$ за умови $P_t(x) = P_{t+1}(x)$, де $P_t(y \mid x)$ є ймовірністю $P(Y = y \mid X = x)$ в момент часу t (рис. 1.5,*б*). У випадку реального дрейфу концепцій реальні межі рішень змінюються, а модель може відтворювати застарілі межі рішень, що знижує ефективність інтелектуальної системи. Прикладом можуть бути дані з мітками, які залежать від настрою людини чи суспільної думки. Тоді зміна настрою чи суспільної думки призведе до зміни залежностей, які буде відображено в нових мітках на ті самі дані.

Віртуальний дрейф пов'язаний зі зміною розподілу даних в межах класу розпізнавання (рис. 1.5, *в*) при незмінній межі рішень. Віртуальний дрейф характеризується зміною умовної ймовірності $P(X \mid y)$ без впливу на апостеріорну ймовірність $P(y \mid X)$.

Зміна в апріорних ймовірностях класів $P(y)$ пов'язаний з незбалансованістю класів (рис. 1.5, *г*), появою зразків нового класу (рис. 1.5, ґ), або злиттям класів (рис. 1.5, *д*).

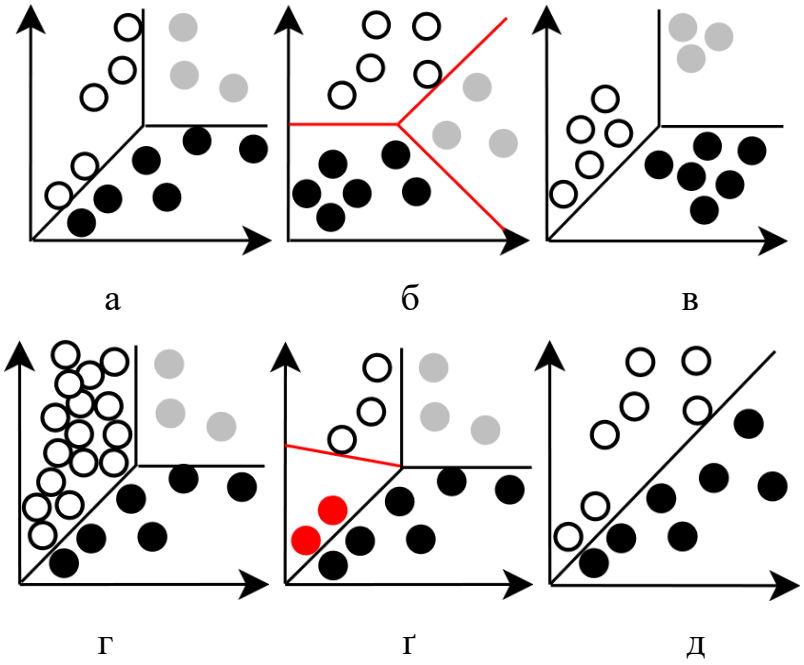


Рисунок 1.5 – Візуальна ілюстрація різних типів дрейфу на прикладі класифікатора : а – оригінальний розподіл даних і роздільна межа класифікаційних рішень;
б – реальний дрейф концепцій; в – віртуальний дрейф концепцій; г – незбалансованість класів; ґ – новий клас; д – злиття зразків класів

З огляду на темпи дрейфу концепцій можлива їх класифікація на раптові, етапні, періодичні та миттєві [28]. Раптовий дрейф передбачає раптову зміну однієї концепції на іншу. Внаслідок цього, ефективність моделі драматично падає, викликаючи потребу в оперативній адаптації до нової концепції. Етапний дрейф характеризується тривалішим переходом від однієї концепції до іншої порівняно з раптовим. Існують дві форми такого дрейфу: ледь помітний і стандартний. Наприклад, етапний дрейф може бути спричинений економічними змінами, зумовленими інфляцією або рецесією. З плином часу, нова концепція стабілізується. При періодичному дрейфі певна концепція повторно з’являється через значний час. Такий дрейф може бути циклічним або ациклічним. Циклічність часто пов'язана з сезонними змінами, як от підвищення попиту на охолоджуючі товари влітку. Ациклічний сценарій спостерігається, коли, скажімо, ціни на електрику ростуть через подорожчання нафти, але згодом

повертаються до звичного рівня. Миттєвий дрейф асоціюється із дуже стрімкими змінами або рідкісними подіями, які часто розглядаються як аномалії в стаціонарних даних. Зазвичай миттєвий дрейф не розцінюється як справжній дрейф концепцій.

Таким чином, до основних деструктивних факторів інтелектуальних систем належать інжекція апаратних несправностей (наприклад помилок в пам’яті), змагальні атаки, дрейф концепцій у вигляді новизни чи зміни задачі.

1.3 Сутність резільєнтності інтелектуальних систем

Термін "резільєнтність" походить з латини, від "resiliere", що має значення "відскакувати". Його первісно вживали в області фізики для характеристики властивостей матеріалів повертатися до оригінальної форми після деформації. В психології цей термін перейняли для опису спроможності індивідів ефективно справлятися зі стресовими ситуаціями та невигідними життєвими обставинами [29]. В контексті систем, резільєнтність означає внутрішній потенціал системи до адаптації, стабілізації та відновлення під час чи після переривань або змін, забезпечуючи при цьому неперервність критичних операцій у широкому спектрі умов [30]. У новітніх дослідженнях [31] підкреслюється, що резільєнтні системи мають властивості самостійно виявляти, реагувати на неблагоприятні умови, опиратися неполадкам та відновлюватися після непередбачуваних подій. Інші дослідники визначають резільєнтність як спроможність системи опиратися стрес-факторам [32]. У певних наукових роботах [33] акцентується на адаптивних якостях резільєнтності, включаючи гнучкість до змін, витримку під час дисфункцій і відновлення після них. Згідно з звітом Національної академії наук США 2012 року щодо стійкості до природних катастроф, резільєнтність системи включає чотири ключові фази реагування на кризові ситуації та загрози (рис. 1.6): 1) стратегічне планування та готовність; 2) поглинання шоку; 3) процес відновлення; 4) адаптаційні зміни [33].

Етап планування і підготовки до збурень у резільєнтній системі може бути реалізоване за рахунок таких дій:

– оцінювання ризиків шляхом здійснення аналізу системи та симуляції деструктивних збурень;

– впровадження методів детектування деструктивних збурень;

– усунення відомих вразливостей та впровадження множини заходів захисту системи від деструктивних збурень;

– забезпечення відповідних стратегій резервування та відновлення.

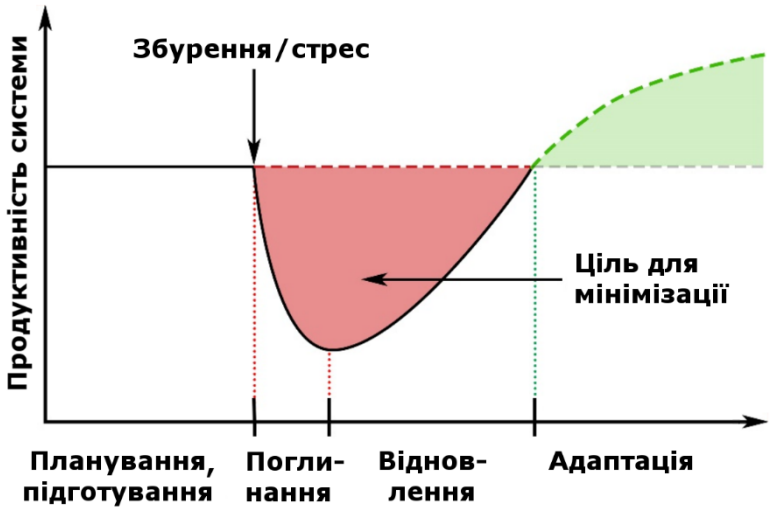


Рисунок 1.6 – Етапи резільєнтного функціонування

Етап поглинання (абсорбції) в системному дизайні відіграє ключову роль у забезпеченні стійкості та реакції системи на непередбачувані впливи. Під час цього етапу відбуваються зміни в базовій архітектурі системи, в залежності від того, які загрози їй загрожують. Механізми поглинання можуть мати складну багатошарову структуру, яка реалізує захист в глибину. Система здатна визначити, який саме механізм захисту варто використовувати, якщо попередні заходи не були успішними на попередньому рівні. У випадках, коли неможливо повністю уникнути деградації системи, може бути використаний механізм керованої деградації (graceful degradation). Цей підхід передбачає, що основні функції системи мають вищий пріоритет і продовжують працювати, навіть якщо

деякі несуттєві функції тимчасово припиняють свою роботу. Така стратегія дозволяє системі функціонувати якнайдовше, надаючи перевагу важливим операціям. Більше того, система може бути заздалегідь налаштована з набором передбачених станів, які представляють різні компроміси між збереженням функціональності, продуктивністю та економічною ефективністю. Ці стани дозволяють системі адаптуватися до змінних умов і зберігати прийнятний рівень продуктивності, навіть під час негативних впливів або обмежень.

Етап відновлення спрямований на швидке та ефективне відновлення функціональності та продуктивності системи після виникнення негативних подій. Основна мета - відновити стан системи якнайшвидше і якнайефективніше, мінімізуючи витрати. На цьому етапі вживаються заходи для відновлення всіх втрачених функцій і можливостей. Етап адаптації, натомість, фокусується на здатності системи змінюватися та адаптуватися до змінюючихся умов і майбутніх загроз. Цей етап спрямований на покращення системи, щоб вона була більш гнучкою та здатною адекватно реагувати на небезпеки, які можуть виникнути в майбутньому. Важливим є розвиток та вдосконалення системи, щоб забезпечити її довгострокову стійкість та ефективність в змінному середовищі.

Покращення робастності інтелектуальної систем до протиборчих атак базується на застосуванні різних методів і стратегій. Один з ключових методів полягає у використанні маскування градієнту, методів оптимізації стійкості та виявлення протиборчих атак [34]. Відмовостійкість до атак досягається за допомогою методів маскування помилок, впровадження явної надмірності та методів виявлення помилок [35]. Один з найпопулярніших підходів – це протиборче навчання, яке спрямоване на оптимізацію робастності системи. Під час навчання генеруються різні види збурень у навчальних даних, що допомагає системі бути більш робастною до негативних впливів. Додаткове підвищення робастності до незначних змін у домені або завданні досягається за допомогою методів міждоменного узагальнення [36]. Однак залишається актуальним

завдання розробки методів, які забезпечать оптимальну робастність до комплексного впливу різних видів збурень.

Одним із способів підвищення здатності мережі до узагальнення та зменшення вимог до навчальних даних при адаптації до змін є мета-навчання. У [37] розглянуто різні незалежні від моделі методи мета-навчання для реалізації Few-Shot Learning. В [38] зроблено спробу інтегрувати різні методи підвищення робастності нейронної мережі з мета-навчанням для Few-Shot Learning. В результаті було продемонстровано, що включення регуляризації та введення збурень може бути ефективно виконано як у внутрішньому, так і в зовнішньому циклі метанавчання. В іншому дослідженні [39] показано, що зовнішній цикл метанавчання може бути реалізований за допомогою еволюційної стратегії, що дозволяє використовувати навіть недиференційовані, негладкі мета-цілі. Однак бракує досліджень, що вивчають такі мета-цілі, як надійність, швидкість адаптації або інтегральні показники резільєнтності.

Таким чином, концепція резільєнтних систем полягає в реалізації механізмів підготовки, поглинання, відновлення і адаптації задля забезпечення стабільного надання послуг в надійний спосіб за умов внутрішніх та зовнішніх змін і впливів [43].

1.4 Формалізована постановка задачі

Нехай $\{\tau_i | i = \overline{1, N}\}$ є заданою множиною реалізації збурень для системи розпізнавання діагностичних медичних зображень. Як збурення τ_i можна розглядати протиборчі атаки, інжекцію несправностей, або зміну задачі. Нехай $\{D_{base} = \{D_{base}^{tr}; D_{base}^{val}\}$ є набір даних, на якому була натренована модель для виконання основної задачі у відомих умовах. Також дано набір даних $D = \{D_k^{tr}; D_k^{val} | k = \overline{1, K}\}$ для K задач навчання за невеликою кількістю зразків (few-shot learning tasks), де D_k^{tr} є вибірковими даними, що використовуються на етапі тонкого настроювання, та валідаційна підвибірка D_k^{val}, що використовується на

стадії мета-оновлення. Також дано множину параметрмів θ, ϕ, ω та W, де θ є параметрами претренованої і замороженої базової моделі розпізнавання діагностичних зображень, ϕ та ω є додатковими параметрами для адаптації базозової моделі, W є специфічними для задачі параметрами. Параметри вихідношо шару W_{base} для основної задачі є претринованими на наборі даних D_{base}.

Необхідно знайти такі значення параметрів ω^*, ϕ^*, які забезпечують максимальне очікуване значення критерію резільєнтності системи розпізнавання діагностичних зображень до впливу різнотипних збурюючих впливів:

$$\max_{\omega,\phi} \underset{\tau_i \sim p(\tau)}{E}[R_{\tau_i}(U_{\tau_i}(\theta, \phi, \omega, W, D))]. \quad (1)$$

Оператор U поєднує збурюючий вплив та адаптацію протягом T кроків, що здійснює відображення поточного стану параметрів ϕ до нового стану. Протиборчі атаки, інжекція несправностей, а також перемикання до нових задач можуть розглядатися як i-та реалізація збурюючого впливу τ_i. Функція R_{τ_i} обчислює значення інтегрального критерію резільєнтності до певних реалізацій збурюючих впливів протягом адаптації шляхом тюнінгу параметрів ω на тестовій вибірці $D_{\tau_i}^{val}$. R_{τ_i} є функцією метрики ефективності P_{τ_i}:

$$R_{\tau_i} = \frac{1}{P_0 T} \sum_{t=1}^{T} P_{\tau_i}(\theta, \omega, \phi_t, W_t, D_{\tau_i}^{val}), \quad (2)$$

де P_0 є метрикою ефективності моделі до впливу збурюючих чинників.

Градієнтне метанавчання вимагає знаходження таких значень параметрів ω^*,ϕ^*, які забезпечать мінімальну очікувану функцію втрат L на множині реалізацій різних типів збурень τ_i

$$\min_{\omega,\phi} \underset{\tau_i \sim p(\tau)}{E}[L_{\tau_i}(U_{\tau_i}(\theta, \phi, \omega, W, D))]. \quad (3)$$

2 ТЕХНОЛОГІЯ ОПТИМІЗАЦІЇ РЕЗІЛЬЄНТНОСТІ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИЧНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ МЕДИЧНИХ ЗОБРАЖЕНЬ

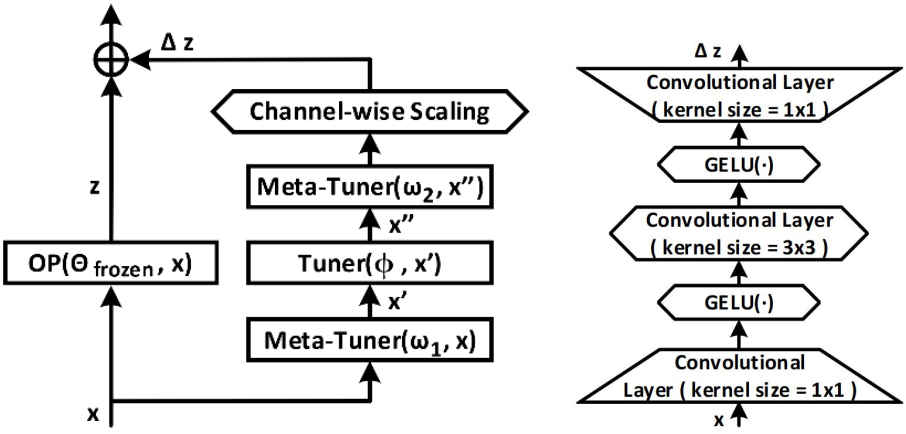
2.1 Модель адаптера

Пропонується приєднувати до замороженої претренованої моделі адаптери, які можуть бути ефективними в обчислювальному плані при тонкому налаштуванні [40]. При цьому вагові коефіцієнти моделі залишаються замороженими. Вихідна модель зазвичай складається з певних блоків або модулів, наприклад, Convolutional Residual Block. Для узагальнення будемо називати ці блоки замороженими операціями і позначати їх як OP(x). Паралельний метод підключення адаптера до заморожених блоків моделі є найбільш зручним та універсальним підходом (рис. 2.1). При цьому для забезпечення властивостей резільєнтності пропонується використовувати одразу три послідовні блоки адаптерів, два з яких налаштовуються під час мета-навчання [41]. Для балансування між різними модулями вводиться канальний коефіцієнт масштабування (channel-wise scaling factor).

Архітектура адаптерів або мета-адаптерів, зображена на рис. 2.1б, базується на двох згорткових шарах та вузького горла у вигляді зниження розмірності каналів. Згортковий адаптер має гіперпараметр γ, який регулює стиснення каналу в 1, 2, 4 або 8 разів. Якщо не деталізувати спеціалізацію вагових коефіцієнтів моделі розпізнавання зображень і позначити множину всіх параметрів $\Xi =< \theta, \phi, \omega, W >$, то процес мета-навчання для прямої максимізації математичного очікування інтегрального критерію резільєнтності може бути описано формулою

$$\Xi^* = \underset{\Xi}{argmax} \underset{\tau_i - p(\tau)}{E} \left[L_{\tau_i} \left(U_{\tau_i}(\Xi, D_{\tau_i}) \right) \right]. \quad (2.1)$$

де Ξ^*– оптимальні значення вагових коефіцієнтів результуючої моделі.



а б

Рисунок 2.1 – Схема паралельної корекції модулі базової моделі та архітектура адаптера: а – схема паралельної корекції блоків з замороженими вагами; б – архітектура згорткового адаптера

2.2 Алгоритм мета-навчання для оптимізації резільєнтності

Якщо використовувати алгоритм стохастичного градієнтного спуску SGD з T кроками в операторі U і використовувати мета-оновлення градієнта в зовнішньому циклі, то отримаємо алгоритм, показаний на рис. 2.2. Крім того, щоб спростити обчислення та підвищити стабільність, настроювані та метанастроювані параметри можна оновити за допомогою алгоритму REPTIL.

Аналіз рис. 2.2 показує, що тип деструктивного впливу не змінюється в межах одного кроку мета-адаптації. Однак кожен крок мета-адаптації починається з вибору типу збурення, після чого відбувається генерація n реалізацій збурення з подальшим вкладеним циклом адаптації до кожної з них. Одночасне комбінування збурень може бути неефективним. Наприклад, додавши до ваг інжекції збоїв, ми отримаємо застарілу модель, і застосування до неї протиборчих атак може бути неактуальним.

Вимоги: Розподіл збурень $p(\tau)$; Розміри кроку гіперпараметрів α, β;
Кількість кроків адаптації T.

Претренуємо ϕ, ω на оригінальних даних D_{base}
Поки не виконано робимо:
    Обрати тип збурення з набору {ін'єкції несправностей, протиборчих атак, дрейфу концепцій}
    Імплементація простого збурення $\tau_i \sim p(\tau),\ i = \overline{1,n}$
    Для i=1,2,...,n робимо:
        Клонуємо поточні параметри: $\hat{\theta}_{\tau_i}, \hat{\omega}_{\tau_i}, \hat{\phi}_{\tau_i}, \hat{W}_{\tau_i} \leftarrow copy(\theta, \omega, \phi, W_{base})$
        Якщо тип збурення дрейф концепцій:
            Вибираємо навчальні та валідаційні дані $D_{\tau_i}^{tr}, D_{\tau_i}^{val}$ з нового завдання
        Інакше:
            Вибираємо навчальні та валідаційні дані $D_{\tau_i}^{tr}, D_{\tau_i}^{val}$ з D_{base}
        Якщо тип збурення ін'єкції несправностей:
            $\hat{\theta}_{\tau_i}, \hat{\omega}_{\tau_i}, \hat{\phi}_{\tau_i} \leftarrow Fault\_injection(\hat{\theta}_{\tau_i}, \hat{\omega}_{\tau_i}, \hat{\phi}_{\tau_i}, \hat{W}_{\tau_i})$
        Якщо тип збурення протиборчі атаки:
            $D_{\tau_i}^{tr}, D_{\tau_i}^{val} \leftarrow Adversarial\_perturbation(D_{\tau_i}^{tr}, D_{\tau_i}^{val})$
        $\phi_{\tau_i} \leftarrow SGD_{\phi,W}(L_{\tau_i}(\hat{\theta}_{\tau_i}, \hat{\omega}_{\tau_i}, \hat{\phi}_{\tau_i}, \hat{W}_{\tau_i}, D_{\tau_i}^{tr}),\ T,\ \alpha)$
    $\omega \leftarrow \omega - \beta \nabla_{\omega} \sum_{\tau_i \sim p(\tau)} L_{\tau_i}(\theta, \omega, \phi_{\tau_i}, D_{\tau_i}^{val})$
    $\phi \leftarrow \phi + \beta \frac{1}{n} \sum_{i=1}^{n} (\phi_{\tau_i} - \phi)$

Рисунок 2.2 – Псевдокод алгоритму мета-навчання для оптимізації резільєнтності системи розпізнавання

На основі функції $Adv\_perturbation()$ формуються протиборчі зразки. Для диференційованих моделей можуть бути використані FGSM-атаки або PGD-атаки [22]. Для недиференційованих моделей пропонується використовувати атаки на основі алгоритму пошуку стратегії еволюції адаптації коваріаційної матриці [22, 23]. Рівень збурення обмежується L_∞-нормою або L_0-нормою. При цьому, якщо зображення нормалізовано шляхом ділення яскравості пікселів на 255, то заданий рівень збурення також ділиться на 255.

Формування ін'єкцій несправностей виконується за методикою [43]. Пропонується обирати найскладніший для поглинання тип несправності, що передбачає генерування інверсії випадково обраного біта (bit-flip ін'єкція) у вазі моделі. Для диференційованих моделей пропонується пропускати тестовий набір даних через мережу та обчислювати градієнти, які потім можна відсортувати за

абсолютними значеннями. У top-k вагах з найбільшим градієнтом один біт інвертується у випадковій позиції. Частка ваг, для яких один випадковий біт інвертується, може бути позначена як частота помилок.

Зміна завдань необхідна для імітації дрейфу концепції та новизни. Формування вибірки інших завдань можна здійснити шляхом рандомізації домену того самого завдання або шляхом вибірки завдань з відповідних доменів, але з дійсно різних завдань. Ці два підходи також можна комбінувати.

2.3 Критерій ефективності нейромережевого класифікатора

Критерії ефективності нейромережевого класифікатора визначаються за допомогою різних метрик, таких як True Positive (TP), False Positive (FP), True Negative (TN), та False Negative (FN). Ці метрики використовуються для оцінки результатів класифікації на вибірці даних. На основі цих чотирьох метрик можна визначити ряд інших метрик для оцінки ефективності класифікатора: Accuracy, Micro Averaging та Macro Averaging.

True Positive (TP) - кількість об'єктів, які належать до цільового, позитивного класу і були правильно ідентифіковані та класифіковані моделлю саме як представники цього позитивного класу. Іншими словами, це випадки, коли нейромережа змогла коректно розпізнати об'єкт, що належить до класу "позитивних" (наприклад, зображення собаки розпізнано як зображення саме собаки). Чим вища кількість таких випадків і, відповідно, значення метрики True Positive, тим краще навчена нейромережа може класифікувати дані позитивного класу, до якого належать цільові об'єкти в заданій предметній галузі.

False Positive (FP) - кількість об'єктів, які насправді належать до негативного, допоміжного класу (тобто не є цільовими об'єктами розпізнавання для поточної задачі), але були помилково ідентифіковані та класифіковані моделлю як представники позитивного класу. Іншими словами, це випадки, коли нейромережа на фото кішки відповіла, що це - собака. Висока кількість таких невірних спрацьовувань (high false positive rate) вказує на те, що навчена модель

часто плутає об'єкти негативного та позитивного класів між собою і, відповідно, потребує доопрацювання та покращення її дискримінативної здатності.

True Negative (TN) - кількість об'єктів, які справді належать до негативного, допоміжного класу (не є цільовими об'єктами) і були коректно віднесені моделлю саме до цього негативного класу. Тобто, коли на фото була зображена кішка і нейромережа правильно класифікувала це фото як "не собака". Високе значення цієї метрики (high true negative rate) показує, що навчена нейромережа добре розпізнає сторонні об'єкти і рідко плутає їх з цільовими об'єктами позитивного класу при класифікації.

False Negative (FN) - кількість об'єктів, які насправді належать до позитивного, цільового класу (є саме тими об'єктами, які повинна розпізнавати поточна модель), але були невірно класифіковані моделлю як представники негативного, допоміжного класу. Наприклад, коли на фото собаки нейромережа відповіла, що це зображення кішки. Висока кількість таких помилок (high false negative rate) означає, що модель часто не розпізнає саме цільові об'єкти позитивного класу, що є її основним призначенням в поточній предметній галузі.

Accuracy - це загальна частка або відсоток всіх правильних класифікацій (true positives і true negatives) з усієї вибірки даних. Іншими словами, це загальна кількість правильних відповідей поділена на загальну кількість об'єктів у тестовій вибірці. Значення Accuracy близьке до 100% свідчить, що модель загалом добре і точно класифікує дані, розділяючи об'єкти на цільові та нецільові категорії з невеликою кількістю помилок як False Positive, так і False Negative.

Micro averaging - підхід до оцінки метрик класифікації, за якого спочатку підраховують загальну кількість True Positives, True Negatives, False Positives та False Negatives по всьому наборі тестових даних без розподілу їх окремо за класами. А вже після цього обчислюють загальні для усього датасету метрики Accuracy, Precision, Recall та інші. Такий підхід дає більшу вагу більш частим у даних класам, оскільки їх представників просто більше в загальній вибірці.

Macro averaging - протилежний до попереднього підхід, за якого спочатку

окремо для кожного класу підраховують його метрики (Precision, Recall та інші), а потім роблять усереднення отриманих значень. Таким чином, всі класи впливають на кінцевий результат рівнозначно і рівномірно, незалежно від того наскільки даний клас представлено у вибірці. Такий підхід корисний, коли потрібно оптимізувати якість класифікації саме для менш чисельних категорій.

Отже, аналіз та оптимізація всіх цих метрик допомагає покращити загальну якість роботи нейромережі при класифікації складних даних на практиці в різних предметних областях.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМИ ОПТИМІЗАЦІЇ РЕЗІЛЬЄНТНОСТІ ІНТЕЛЕКТУАЛЬНОЇ ДІАГНОСТИЧНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ МЕДИЧНИХ ЗОБРАЖЕНЬ

3.1 Формування навчальних і тестових даних

Для простоти експериментів як базову модель використаємо Resnet-18, попередньо навчену на DermaMNIST [44]. Вибірка DermaMNIST містить дермоскопічні, що класифіковані на 7 класів пігментних уражень шкіри. На рис. 3.1 у першому ряді показані оригінальні зображення, а у другому ряді збурена версія зображень. Таким чином можна візуально порівняти вплив збурень на зображення.

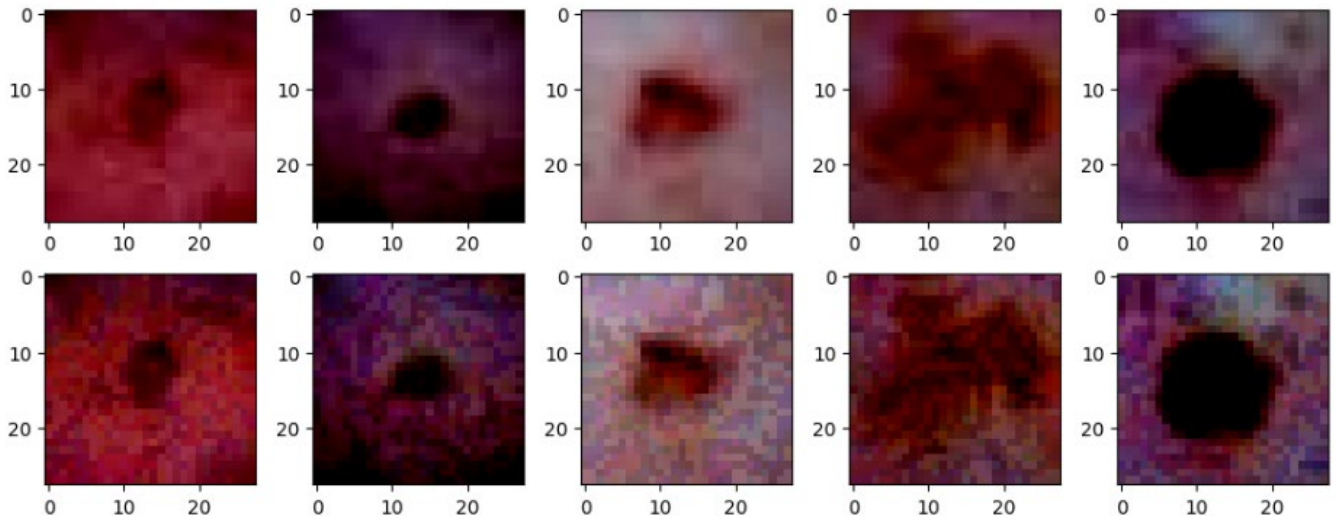


Рисунок 3.1 – Оригінальні та збурені зображення з датасету DermaMNIST.

Для навчання базової моделі використано лише 3 перші класи. Архітектуру адаптера та мета-адаптера обрано однаковою у вигляді двошарової згорткової мережі зі зменшенням розмірності каналу до вузького місця дискретизації ($\gamma = 2$). Для генерації нових завдань використовується решта класів набору даних DermaMNIST, з якого вибираються дані для випадково обраних 3 класів ($n_{way} =$

3). Пропонується використовувати 16 зображень на клас (k_shot=16), які під час адаптації надсилаються міні-пакетами по 4 зображення (mini_batch_size=4). Таким чином, кількість кроків адаптації становить T=(k_shot*n_way)/mini_batch_size=12 ітерацій. Базове завдання використовується під час метанавчання разом з новими завданнями. Швидкість навчання внутрішнього та зовнішнього циклу метанавчання дорівнює $\alpha = 0{,}001$ та $\beta = 0{,}0001$ відповідно. Максимальна кількість мета-ітерацій – 300. Однак для зупинки метанавчання використовується алгоритм ранньої зупинки, який припиняє виконання, якщо критерій не змінюється протягом більш ніж 10 послідовних ітерацій більш ніж на 0.001.

3.2 Короткий опис програмного забезпечення

Програмне забезпечення дозволяє створити стійкий класифікатор зображень на основі ResNet18 з використанням адаптерів, мета-адаптерів і мета-навчання.

Опис роботи програмного забезпечення:

1. Імпортуються необхідні бібліотеки:
   - medmnist - для роботи з медичними зображеннями
   - tqdm - для відображення прогресу під час навчання
   - numpy - для роботи з багатовимірними масивами
   - torch - основна бібліотека для машинного навчання
   - torch.nn - містить нейронні мережі
   - torch.optim - оптимізатори для навчання нейронних мереж
   - torch.utils.data - робота з даними
   - torchvision.transforms - перетворення зображень
   - torch.autograd - автоматичне обчислення градієнтів
   - copy - для глибокого копіювання об'єктів
   - art - Adversarial Robustness Toolbox для генерації протиборчих прикладів

2. Завантажуються дані DermMNIST, які містять зображення шкіри з різними захворюваннями. Виділяється підмножина класів. Дані розділяються на навчальні та тестові.

3. Описуються допоміжні функції:

- train_one_epoch - навчання протягом однієї епохи
- validate - оцінка моделі на валідаційних даних

4. Створюється базова модель класифікатора на основі ResNet18. Вона доначується на виділеній підмножині класів.

5. Реалізуються класи Adapter та MetaAdapter - адаптер та мета-адаптер відповідно. Вони використовують невеликі перетворення входу для адаптації основної моделі.

6. Клас ModifiedBasicBlock використовує адаптер та мета-адаптер всередині BasicBlock моделі ResNet18. Це дозволяє легко інтегрувати їх в архітектуру.

7. Клас ModifiedResNet18 заміняє BasicBlock-и на ModifiedBasicBlock для додавання можливості адаптації.

8. Створюється модифікована модель на основі ResNet18. Вона доначується, використовуючи лише адаптер та мета-адаптер.

9. Реалізується функція inject_fault для імітації збоїв в роботі моделі шляхом додавання шуму до ваг.

10. Створюються функції для генерації протиборчих прикладів та зміни підмножини класів.

11. Реалізується мета-навчання моделі в режимі few-shot learning для адаптації до збоїв, протиборчих прикладів та зміни задачі. Використовуються алгоритми MAML та Reptile.

Отже, програма дозволяє побудувати глибоку нейронну мережу, стійку до різних типів збурень за допомогою методів адаптації та мета-навчання. Код реалізації надано у Додатку Б

3.3 Аналіз результатів експериментів

Результати тестування впливу інжекції несправностей на продуктивність моделі наведено в табл. 3.1. При додаванні адаптерів та мета-адаптерів перед тестуванням вони попередньо навчаються за градієнтними алгоритмами зі збуреннями та без них до досягнення точності базової моделі. Середня точність $\overline{Acc}$ та інтегральний показник резільєнтності $\overline{R}$ моделі оцінюються на 100 реалізаціях інжекції несправностей із заданою частотою відмов. Під час тестування мета-адаптери фіксуються, і адаптери можуть бути використані для адаптації до збурень та розрахунку показника стійкості.

Таблиця 3.1 – Експериментальні дані тестування моделі на резільєнтність до ін'єкції несправностей

Fault rate	Лише претренована базова модель	Претренована модель з адаптерами та мета-адаптерами, що тренуються під час ін'єкції несправностей		Претренована модель з адаптерами і мета-адаптерами, що мета-треновані для оптимізації резільєнтності	
	$\overline{Acc}$	$\overline{Acc}$	$\overline{R}$	$\overline{Acc}$	$\overline{R}$
0	91,5%	91,8%	-	92,5%	-
1	89,2%	90,1%	0,973	91,3%	0,984
3	84,1%	86,6%	0,941	90,1%	0,971
5	82,0%	84,5%	0,882	86,4%	0,953
10	74,4%	79,1%	0,832	84,1%	0,920

Аналіз табл. 3.1 показує, що адаптери з мета-адаптерами можуть підвищити резільєнтність попередньо навченої моделі до несправностей, поглинаючи їхній вплив. Більше того, запропонований метод мета-навчання з урахуванням резільєнтності забезпечує кращий інтегральний показник

резільєнтності порівняно з простим тренуванням в умовах ін'єкції несправностей, в середньому на 5,73%. Це означає, що протягом 12 ітерацій налаштування відновлення продуктивності відбувається в середньому швидше, якщо модель була підготовлена на основі метанавчання з урахуванням резільєнтності. Вимірювання резільєнтності в експерименті мають стандартне відхилення, що не перевищує 1.0.

Результати тестування впливу протиборчих атак на продуктивність моделі наведено в табл. 3.2. Адаптери та мета-адаптери навчаються без збурень та зі збуреннями до тих пір, доки результуюча модель не досягне точності базової моделі для наступного тесту на резільєнтність. Середнє значення точності та інтегрального показника резільєнтності моделі оцінюється на 100 реалізаціях протиборчих зразків із заданим рівнем збурень. Після заморожування параметрів мета-адаптерів, адаптери можна використовувати для адаптації до збурення та обчислення показника резільєнтності.

Таблиця 3.2 – Експериментальні дані тестування моделі на резільєнтність до протиборчих атак

Рівень збурення	Лише претренована базова модель	Попередньо претренована модель з адаптерами і мета-адаптерами, що тренуються з використанням протиборчих зразків		Претренована модель з адаптерами і мета-адаптерами, що мета-треновані для оптимізації резільєнтності	
	$\overline{\boldsymbol{Acc}}$	$\overline{\boldsymbol{Acc}}$	$\overline{\boldsymbol{R}}$	$\overline{\boldsymbol{Acc}}$	$\overline{\boldsymbol{R}}$
0	91,5%	91,8%	-	92,5%	-
1	90,6%	90,1	0,961	91,0%	0,982
3	87,1%	87,9	0,932	90,1%	0,981
5	81,5%	81,7	0,861	84,7%	0,920
10	73,8%	74,9	0,822	77,6%	0,915

Аналіз табл. 3.2 показує, що мета-адаптери можуть підвищити резільєнтність навченої моделі до протиборчих атак, поглинаючи частину збурень. Більше того, запропонований метод мета-навчання з урахуванням резільєнтності забезпечує кращий інтегральний показник резільєнтності порівняно з навчанням з використаннями протиборчих зразків в середньому на 6,40%. Це означає, що за 12 ітерацій налаштування продуктивність відновлюється швидше після метанавчання з урахуванням відмовостійкості. Вимірювання стійкості в експерименті мають стандартне відхилення, що не перевищує 1,1.

Перевага використання мета-навчання замість звичайного тонкого налаштування адаптерів була оцінена на основі результатів експерименту, які наведені в табл. 3.3.

Таблиця 3.3 – Експериментальні дані тестування моделі на резільєнтність до дрейфу концепцій

Метод попереднього навчання адпатерів і мета-адаптерів	$\overline{\boldsymbol{R}}$
Попереднє навчання на базовому наборі даних доки точність моделі не досягне точності базової моделі	0,925
Мета-навчання для оптимізації резільєнтності	0,978

Аналіз табл. 3.3 показує, що при адаптації до нових завдань мета-навчені адаптери та мета-адаптери забезпечують в середньому на 5,3% вище значення інтегрального показника стійкості за T=12 ітерацій адаптації, ніж просте попереднє навчання на базовому завданні.

Таким чином, запропонований алгоритм мета-навчання для оптимізації резільєнтності системи розпізнавання медичних зображень забезпечує підвищення її резільєнтності до збурень порівняно з традиційними підходами, такими як тонке налаштування на в умовах дії збурення.

ВИСНОВКИ

Наукова новизна одержаних результатів полягає в тому, що вперше запропоновано застосувати метод мета-навчання, що оптимізує резільєнтність системи, [42] до моделі розпізнавання медичних зображень, що функціонує в умовах ін'єкцій помилок, протиборчих атак та зміни завдань. Розроблено інформаційне та програмне забезпечення, що передбачає використання адаптерів та мета-адаптерів, які виконують паралельну корекцію структурних блоків моделі розпізнавання образів. Запропонований метод мета-навчання полягає в тому, що на кожній ітерації мета-оптимізації генерується n реалізацій певного типу збурення та використовуються результати адаптації для мета-оновлення адаптерів та мета-адаптерів.

Експериментально доведено, що запропоноване мета-навчання з урахуванням резільєнтності покращує здатність базової моделі поглинати збурення та збільшує швидкість адаптації порівняно з традиційними підходами. Розроблене інформаційне та програмне забезпечення демонструє кращий показник резільєнтності до ін'єкцій помилок порівняно з простим навчанням в умовах дії ін’єктора помилок в середньому на 5,73%. Крім того, метод забезпечує кращу резільєнтність до протиборчих атак ухилення порівняно з протиборчим навчанням в середньому на 6,4%. Він також продемонстрував середнє покращення на 5,3% резільєнтності до зміни завдань порівняно зі звичайним тонким налаштуванням адаптерних блоків.

Результати традиційного мета-навчання та запропонованого мета-навчання з урахуванням стійкості порівнюються з точки зору впливу на резільєнтності до збурень. Підтверджено перевагу представленого методу.

Практичне значення отриманих результатів полягає у формуванні нової методологічної бази для розробки алгоритмів оптимізації резільєнтності інтелектуальних системи розпізнавання медичних діагностичних зображень, що є важливим для галузі охорони здоров’я.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kaur S., Singla J., Nkenyereye L., Jha S., Prashar D., Joshi G. P., El-Sappagh S., Islam Md. S. and Islam S. M. R. (2020). Medical Diagnostic Systems Using Artificial Intelligence (AI) Algorithms: Principles and Perspectives. In IEEE Access (Vol. 8, pp. 228049–228069). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/access.2020.3042273

2. Kumar Y., Koul A., Singla R. and Ijaz M. F. (2022). Artificial intelligence in disease diagnosis: a systematic literature review, synthesizing framework and future research agenda. In Journal of Ambient Intelligence and Humanized Computing (Vol. 14, Issue 7, pp. 8459–8486). Springer Science and Business Media LLC. https://doi.org/10.1007/s12652-021-03612-z

3. Latif J., Xiao C., Imran A. and Tu S. (2019). Medical Imaging using Machine Learning and Deep Learning Algorithms: A Review. In 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET). 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET). IEEE. https://doi.org/10.1109/icomet.2019.8673502

4. Oren O., Gersh B. J. and Bhatt D. L. (2020). Artificial intelligence in medical imaging: switching from radiographic pathological data to clinically meaningful endpoints. In The Lancet Digital Health (Vol. 2, Issue 9, pp. e486–e488). Elsevier BV. https://doi.org/10.1016/s2589-7500(20)30160-6

5. Samantaray A. K. and Rahulkar A. D. (2020). New design of adaptive Gabor wavelet filter bank for medical image retrieval. In IET Image Processing (Vol. 14, Issue 4, pp. 679–687). Institution of Engineering and Technology (IET). https://doi.org/10.1049/iet-ipr.2019.1024

6. Rong Q., Thangaraj C., Easwaramoorthy D. and He S. (2021). Multifractal based image processing for estimating the complexity of COVID-19 dynamics. In The European Physical Journal Special Topics (Vol. 230, Issues 21–22, pp. 3947–3954). Springer Science and Business Media LLC. https://doi.org/10.1140/epjs/s11734-021-00336-1

7. Chan H.-P., Samala R. K., Hadjiiski L. M. and Zhou C. (2020). Deep Learning in Medical Image Analysis. In Advances in Experimental Medicine and Biology (pp. 3–21). Springer International Publishing. https://doi.org/10.1007/978-3-030-33128-3_1

8. Kim D. E. and Gofman M. (2018). Comparison of shallow and deep neural networks for network intrusion detection. In 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC). 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC). IEEE. https://doi.org/10.1109/ccwc.2018.8301755

9. Yadav S. S. and Jadhav S. M. (2019). Deep convolutional neural network based medical image classification for disease diagnosis. In Journal of Big Data (Vol. 6, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1186/s40537-019-0276-2

10. Khan A., Sohail A., Zahoora U. and Qureshi A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. In Artificial Intelligence Review (Vol. 53, Issue 8, pp. 5455–5516). Springer Science and Business Media LLC. https://doi.org/10.1007/s10462-020-09825-6

11. Li X., Lu P., Hu L., Wang X. and Lu L. (2021). A novel self-learning semi-supervised deep learning network to detect fake news on social media. In Multimedia Tools and Applications (Vol. 81, Issue 14, pp. 19341–19349). Springer Science and Business Media LLC. https://doi.org/10.1007/s11042-021-11065-x

12. Chen C., Wu T., Guo Z. and Cheng J. (2021). Combination of deep neural network with attention mechanism enhances the explainability of protein contact prediction. In Proteins: Structure, Function, and Bioinformatics (Vol. 89, Issue 6, pp. 697–707). Wiley. https://doi.org/10.1002/prot.26052

13. Zhang Y., Wang J., Gorriz J. M. and Wang S. (2023). Deep Learning and Vision Transformer for Medical Image Analysis. In Journal of Imaging (Vol. 9, Issue 7, p. 147). MDPI AG. https://doi.org/10.3390/jimaging9070147

14. Springenberg M., Frommholz A., Wenzel M., Weicken E., Ma J. and Strodthoff N. (2023). From modern CNNs to vision transformers: Assessing the performance, robustness, and classification strategies of deep learning models in histopathology. In Medical Image Analysis (Vol. 87, p. 102809). Elsevier BV. https://doi.org/10.1016/j.media.2023.102809

15. Zhang T. and Meng J. (2023). STFGSM: Intelligent Image Classification Model Based on Swin Transformer and Fast Gradient Sign Method. In Proceedings of the 2023 7th International Conference on Deep Learning Technologies. ICDLT 2023: 2023 7th International Conference on Deep Learning Technologies. ACM. https://doi.org/10.1145/3613330.3613339

16. Dunn I., Pouget H., Kroening D. and Melham T. (2021). Exposing previously undetectable faults in deep neural networks. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA ’21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM. https://doi.org/10.1145/3460319.3464801

17. Liu Y., Wei L., Luo B. and Xu Q. (2017). Fault injection attack on deep neural network. In 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE. https://doi.org/10.1109/iccad.2017.8203770

18. Breier J., Hou X., Jap D., Ma L., Bhasin S. and Liu Y. (2018). Practical Fault Attack on Deep Neural Networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS ’18: 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.1145/3243734.3278519

19. Santos F. F. dos, Kritikakou A., Condia J. E. R., Balaguera J. D. G., Reorda M. S., Sentieys O. and Rech P. (2022). Characterizing a Neutron-Induced Fault Model for Deep Neural Networks (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2211.13094

20. Hou X., Breier J., Jap D., Ma L., Bhasin S. and Liu Y. (2020). Security Evaluation of Deep Neural Network Resistance Against Laser Fault Injection. In 2020 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA). 2020 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA). IEEE. https://doi.org/10.1109/ipfa49335.2020.9261013

21. Qian C., Zhang M., Nie Y., Lu S. and Cao H. (2023). A Survey of Bit-Flip Attacks on Deep Neural Network and Corresponding Defense Methods. In Electronics (Vol. 12, Issue 4, p. 853). MDPI AG. https://doi.org/10.3390/electronics12040853

22. Ismail Fawaz H., Forestier G., Weber J., Idoumghar L. and Muller P.-A. (2019). Adversarial Attacks on Deep Neural Networks for Time Series Classification. In 2019 International Joint Conference on Neural Networks (IJCNN). 2019 International Joint Conference on Neural Networks (IJCNN). IEEE. https://doi.org/10.1109/ijcnn.2019.8851936

23. Bai Y., Wang Y., Zeng Y., Jiang Y. and Xia S.-T. (2023). Query efficient black-box adversarial attack on deep neural networks. In Pattern Recognition (Vol. 133, p. 109037). Elsevier BV. https://doi.org/10.1016/j.patcog.2022.109037

24. Altoub M., Al Qurashi F., Yigitcanlar T., Corchado J. M. and Mehmood R. (2022). An Ontological Knowledge Base of Poisoning Attacks on Deep Neural Networks. In Applied Sciences (Vol. 12, Issue 21, p. 11053). MDPI AG. https://doi.org/10.3390/app122111053

25. Zhang Y., Ruan W., Wang F. and Huang X. (2023). Generalizing universal adversarial perturbations for deep neural networks. In Machine Learning (Vol. 112, Issue 5, pp. 1597–1626). Springer Science and Business Media LLC. https://doi.org/10.1007/s10994-023-06306-z

26. Co K. T., Muñoz-González L., de Maupeou S. and Lupu E. C. (2019). Procedural Noise Adversarial Examples for Black-Box Attacks on Deep Convolutional Networks. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and

Communications Security. CCS ’19: 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.1145/3319535.3345660

27. Xiang Q., Zi L., Cong X. and Wang Y. (2023). Concept Drift Adaptation Methods under the Deep Learning Framework: A Literature Review. In Applied Sciences (Vol. 13, Issue 11, p. 6515). MDPI AG. https://doi.org/10.3390/app13116515

28. Priya S. and Uthra R. A. (2021). Deep learning framework for handling concept drift and class imbalanced complex decision-making on streaming data. In Complex & Intelligent Systems (Vol. 9, Issue 4, pp. 3499–3515). Springer Science and Business Media LLC. https://doi.org/10.1007/s40747-021-00456-0

29. Drozd O., Kharchenko V., Rucinski A., Kochanski T., Garbos R. and Maevsky D. (2019). Development of Models in Resilient Computing. In 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT). 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT). IEEE. https://doi.org/10.1109/dessert.2019.8770035

30. Allenby B. and Fink J. (2005). Toward Inherently Secure and Resilient Societies. In Science (Vol. 309, Issue 5737, pp. 1034–1036). American Association for the Advancement of Science (AAAS). https://doi.org/10.1126/science.1111534

31. Yodo N. and Wang P. (2016). Engineering Resilience Quantification and System Design Implications: A Literature Survey. In Journal of Mechanical Design (Vol. 138, Issue 11). ASME International. https://doi.org/10.1115/1.4034223

32. Brtis J. S., McEvilley M. A. and Pennock M. J. (2021). Resilience Requirements Patterns. In INCOSE International Symposium (Vol. 31, Issue 1, pp. 570–584). Wiley. https://doi.org/10.1002/j.2334-5837.2021.00855.x

33. Barker K., Lambert J. H., Zobel C. W., Tapia A. H., Ramirez-Marquez J. E., Albert L., Nicholson C. D. and Caragea C. (2017). Defining resilience analytics for interdependent cyber-physical-social networks. In Sustainable and Resilient Infrastructure (Vol. 2, Issue 2, pp. 59–67). Informa UK Limited. https://doi.org/10.1080/23789689.2017.1294859

34. Xie C., Wang J., Zhang Z., Ren Z. and Yuille A., "Mitigating Adversarial Effects Through Randomization," Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017. pp. 1–16. DOI: 10.48550/arXiv.1711.01991

35. Li W., Ning X., Ge G., Chen X., Wang Y. and Yang H., "FTT-NAS: Discovering Fault-Tolerant Neural Architecture", 2020 25th Asia South Pacific Des. Automat. Conf. (ASP-DAC), Beijing, China, 13–16 Jan. 2020. IEEE, 2020, pp. 2011-2016. DOI: 10.1109/asp-dac47756.2020.9045324

36. Volpi R., et al., "Generalizing to unseen domains via adversarial data augmentation," Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, QC, Canada, 2–8 December 2018, pp. 1–11. DOI: 10.5555/3327345.3327439

37. Yang X. and Xu J., "Few-shot Classification with First-order Task Agnostic Meta-learning," 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL &ICCEA), Changchun, China, 20–22 May 2022, pp. 2017–2020. – DOI:10.1109/cvidliccea56201.2022.9824307

38. Wang R., Xu K., Liu S., Chen P.-Y., Weng T.-W., Gan C. and Wang M. (2021). On Fast Adversarial Robustness Adaptation in Model-Agnostic Meta-Learning (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2102.10454

39. Song X. et al., "Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020 – 24 January 2021, pp. 3769–3776. DOI:10.1109/iros45743.2020.9341571

40. Bansal T., Alzubi S., Wang T., Lee Jay-Yoon and McCallum A., "Meta-Adapters: Parameter Efficient Few-shot Fine-tuning through Meta-Learning" First Conference on Automated Machine Learning, 2022, 18 p. Available at: https://openreview.net/pdf?id=bQt8dWKsfso

41. Wu N., Hou H., Jia X., Chang X. and Li H. (2021). Low-Resource Neural Machine Translation Based on Improved Reptile Meta-learning Method. In Communications in Computer and Information Science (pp. 39–50). Springer Singapore. https://doi.org/10.1007/978-981-16-7512-6_4

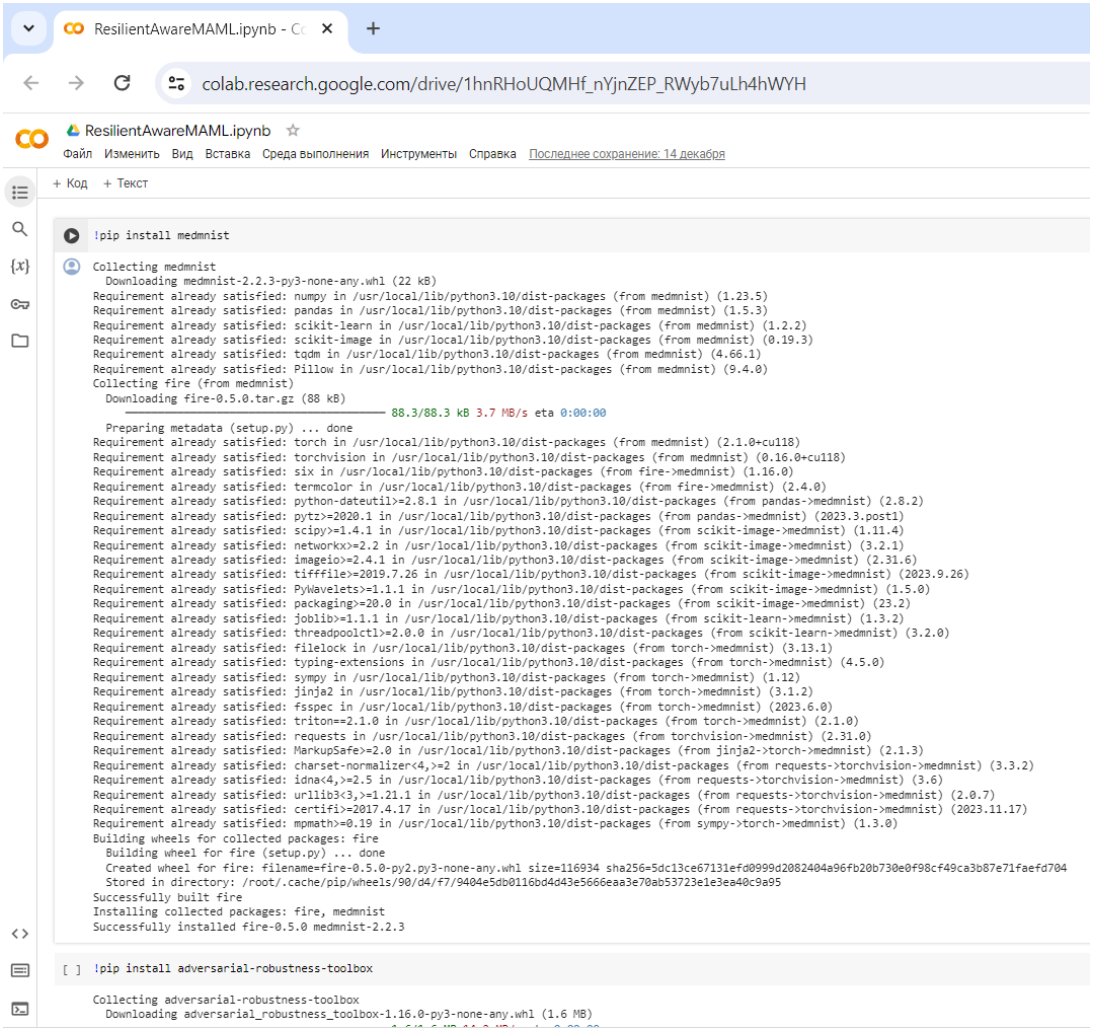
42. Moskalenko V. V. (2023). Model-agnostic meta-learning for resilience optimization of artificial intelligence system In Radio Electronics, Computer Science, Control (Issue 2, p. 79). National University “Zaporizhzhia Polytechnic”; https://doi.org/10.15588/1607-3274-2023-2-9

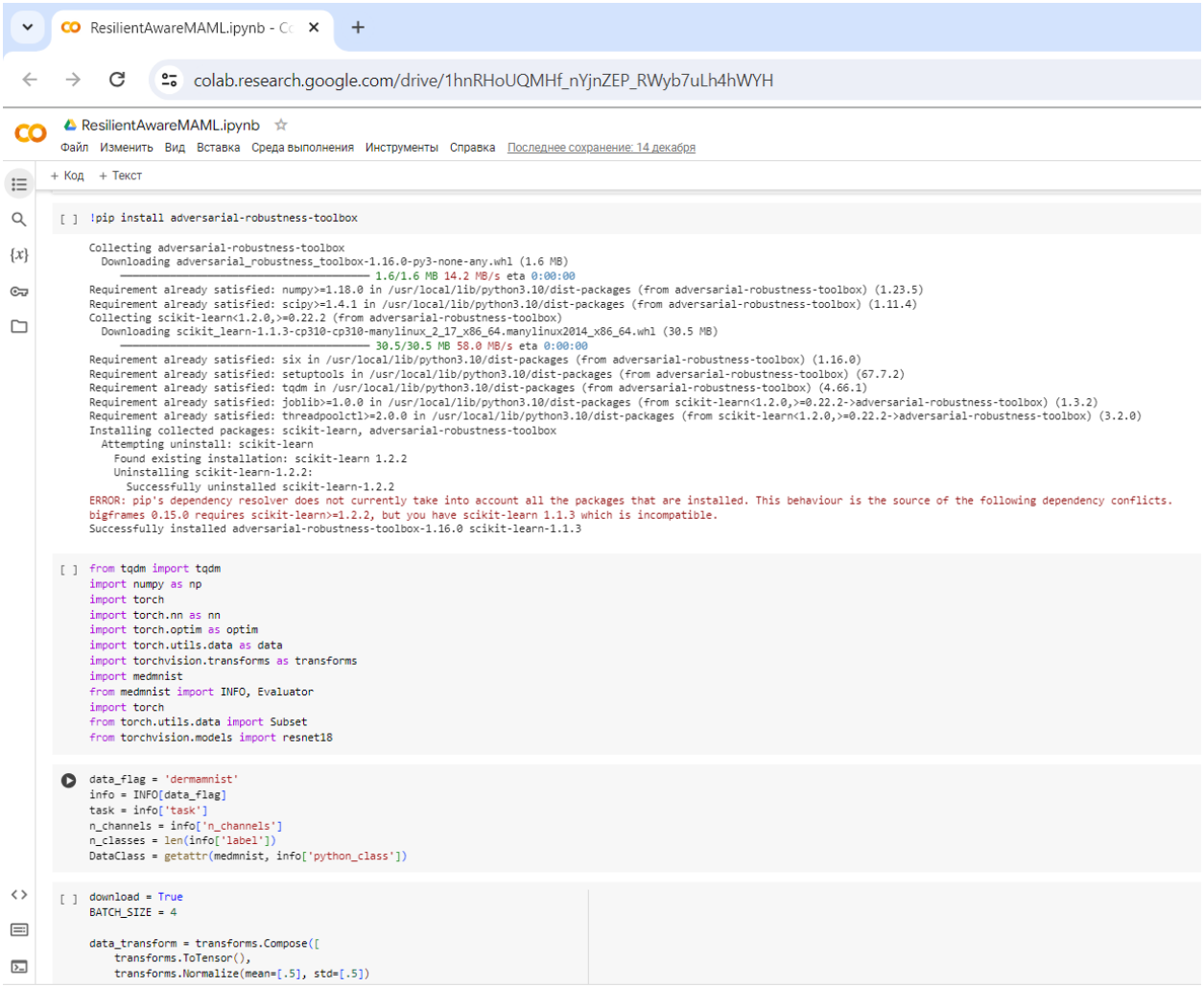
43. Moskalenko V., Kharchenko V., Moskalenko A. and Petrov S. (2022). Model and Training Method of the Resilient Image Classifier Considering Faults, Concept Drift, and Adversarial Attacks. In Algorithms (Vol. 15, Issue 10, p. 384). MDPI AG. https://doi.org/10.3390/a15100384

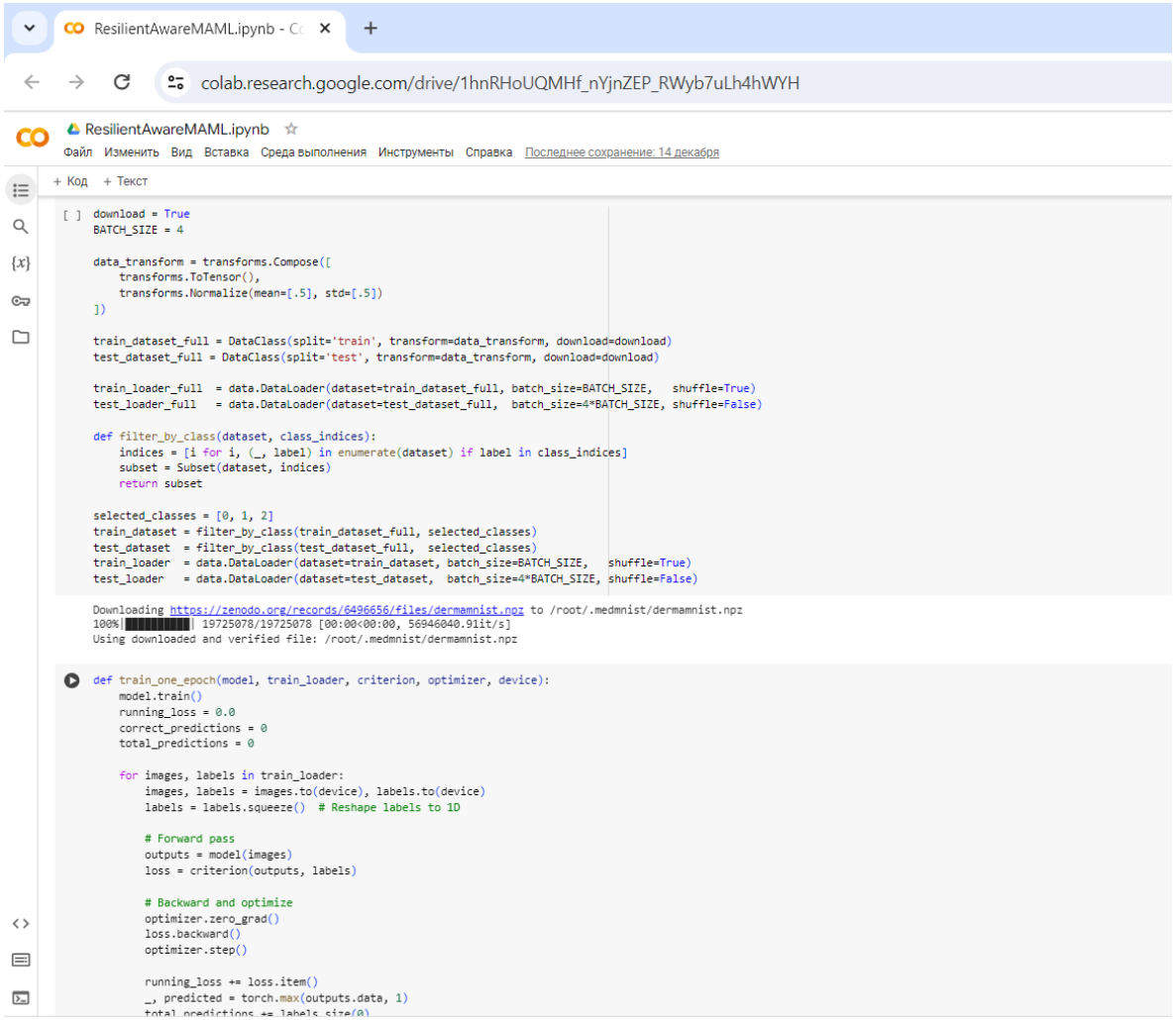
44. Yang J., Shi R., Wei D., Liu Z., Zhao L., Ke B., Pfister H. and Ni B. (2023). MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification. In Scientific Data (Vol. 10, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1038/s41597-022-01721-8

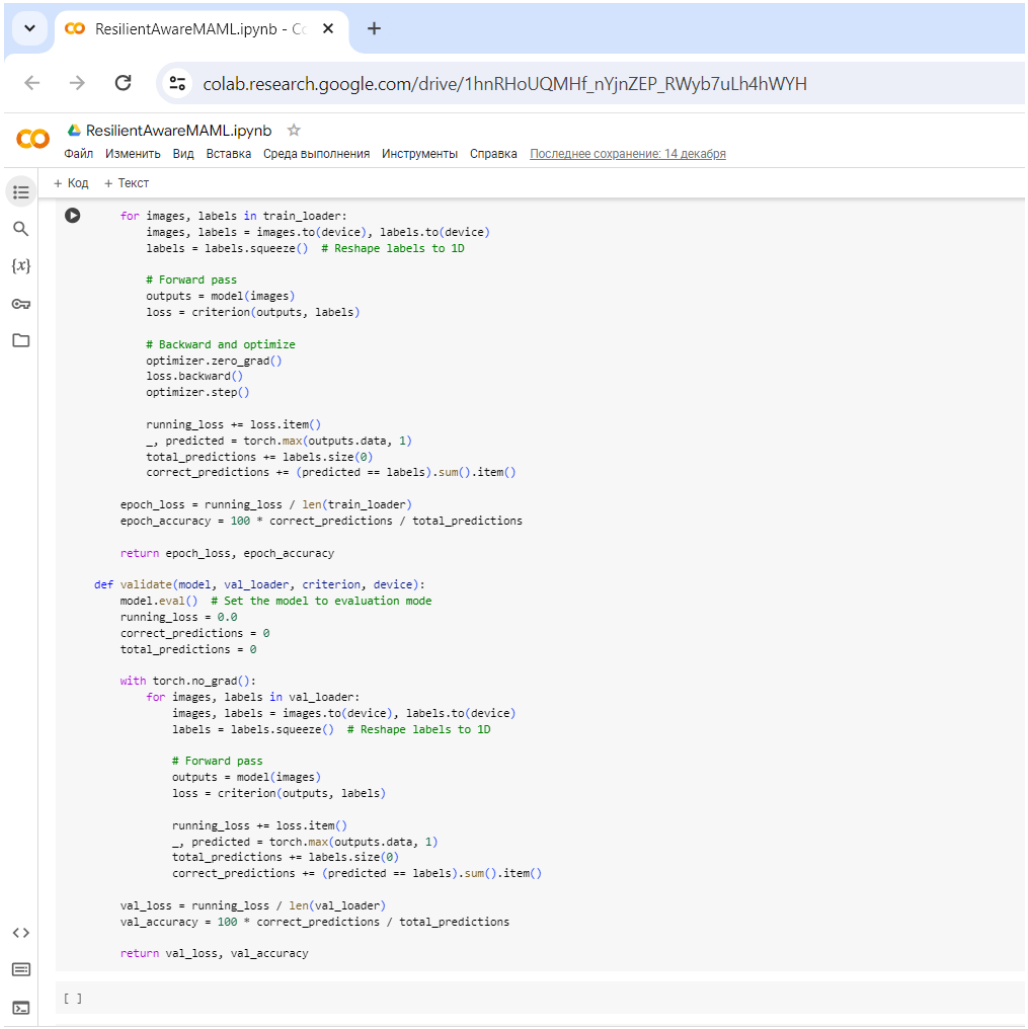
ДОДАТОК А

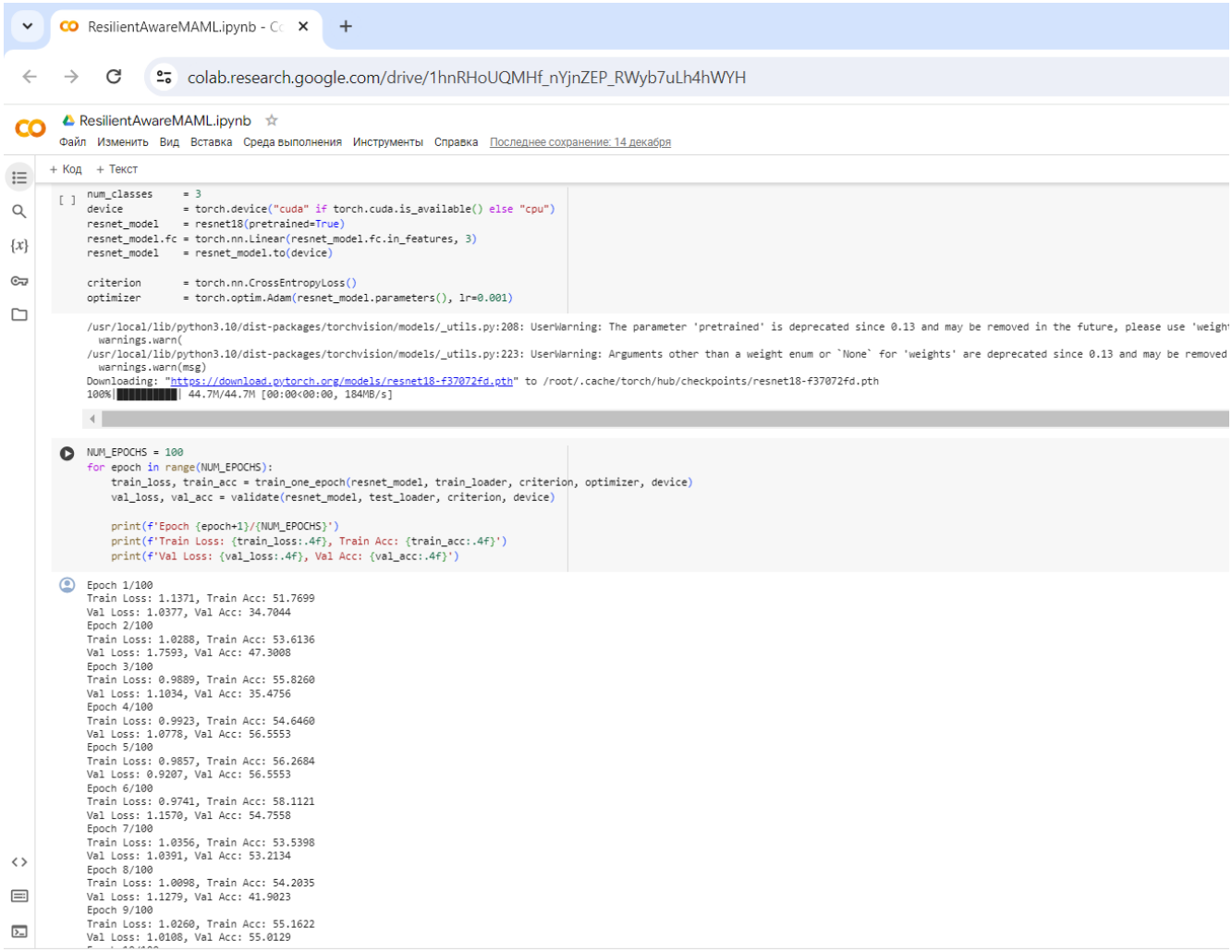
РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

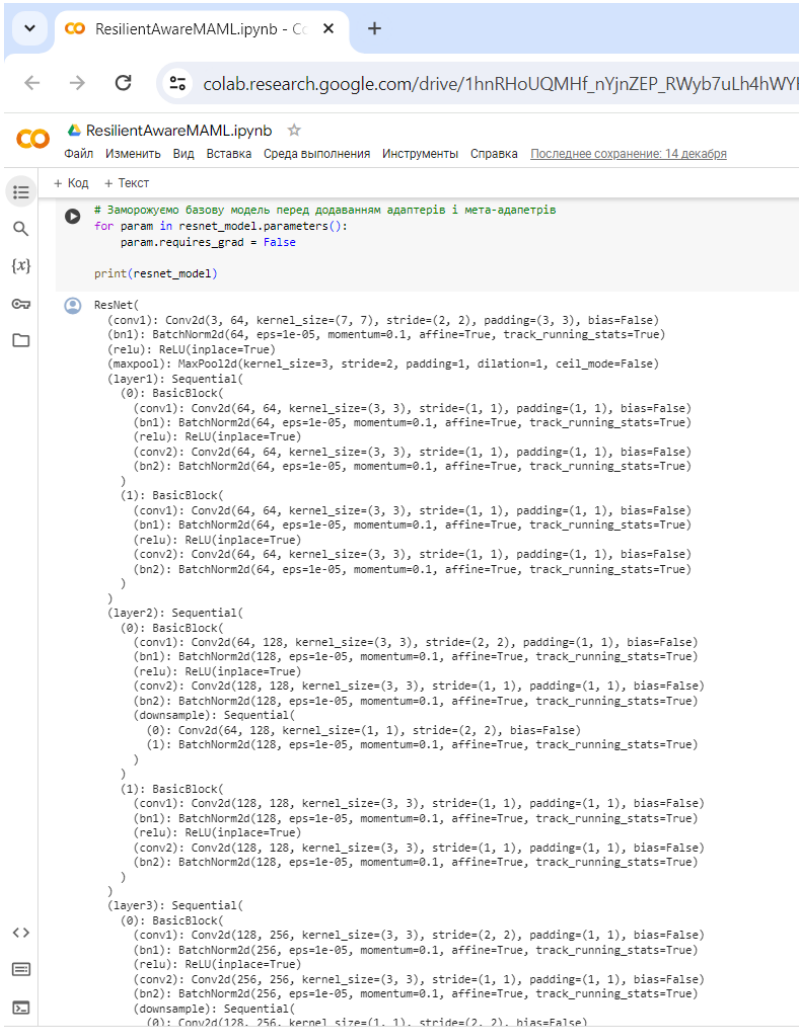


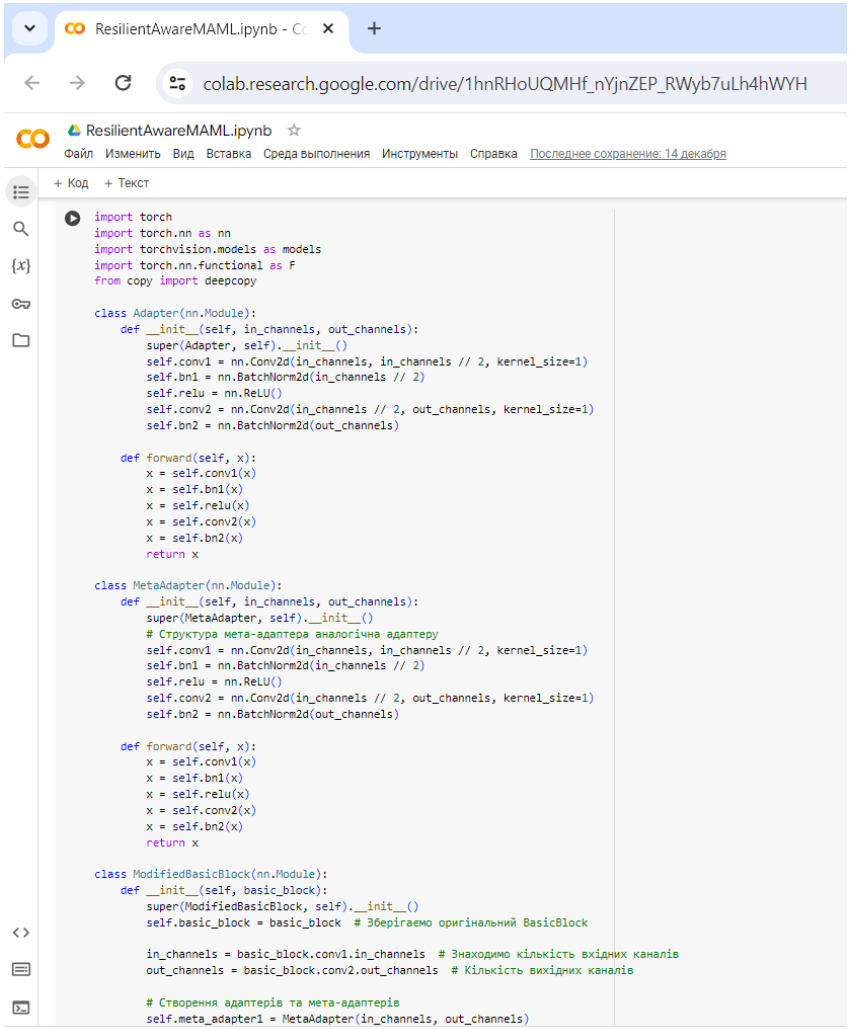

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ] !pip install adversarial-robustness-toolbox
Collecting adversarial-robustness-toolbox
Downloading adversarial_robustness_toolbox-1.16.0-py3-none-any.whl (1.6 MB)
1.6/1.6 MB 14.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.23.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.11.4)
Collecting scikit-learn<1.2.0,>=0.22.2 (from adversarial-robustness-toolbox)
Downloading scikit_learn-1.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (30.5 MB)
30.5/30.5 MB 58.0 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (4.66.1)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (3.2.0)
Installing collected packages: scikit-learn, adversarial-robustness-toolbox
Attempting uninstall: scikit-learn
Found existing installation: scikit-learn 1.2.2
Uninstalling scikit-learn-1.2.2:
Successfully uninstalled scikit-learn-1.2.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
bigframes 0.15.0 requires scikit-learn>=1.2.2, but you have scikit-learn 1.1.3 which is incompatible.
Successfully installed adversarial-robustness-toolbox-1.16.0 scikit-learn-1.1.3
[ ] from tqdm import tqdm
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import torchvision.transforms as transforms
import medmnist
from medmnist import INFO, Evaluator
import torch
from torch.utils.data import Subset
from torchvision.models import resnet18
data_flag = 'dermamnist'
info = INFO[data_flag]
task = info['task']
n_channels = info['n_channels']
n_classes = len(info['label'])
DataClass = getattr(medmnist, info['python_class'])
[ ] download = True
BATCH_SIZE = 4
data_transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize(mean=[.5], std=[.5])

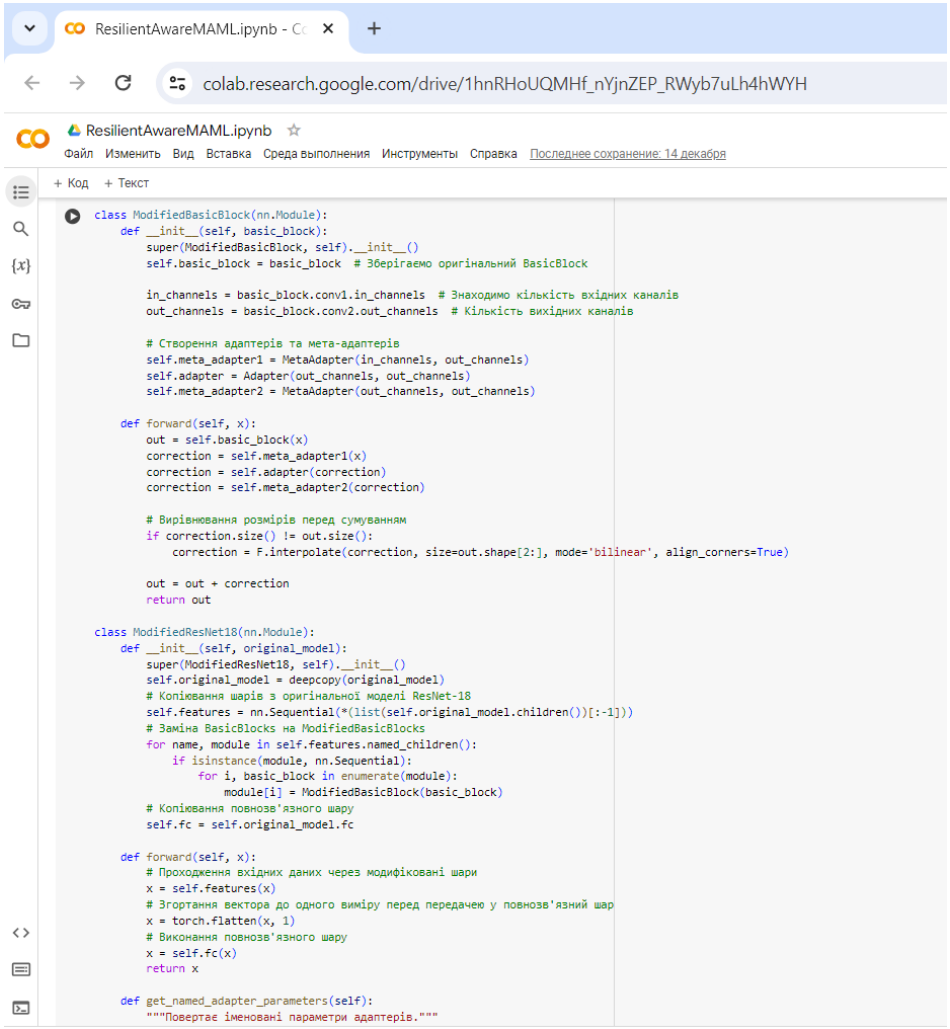

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ] download = True
BATCH_SIZE = 4
data_transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize(mean=[.5], std=[.5])
])
train_dataset_full = DataClass(split='train', transform=data_transform, download=download)
test_dataset_full = DataClass(split='test', transform=data_transform, download=download)
train_loader_full = data.DataLoader(dataset=train_dataset_full, batch_size=BATCH_SIZE, shuffle=True)
test_loader_full = data.DataLoader(dataset=test_dataset_full, batch_size=4*BATCH_SIZE, shuffle=False)
def filter_by_class(dataset, class_indices):
indices = [i for i, (_, label) in enumerate(dataset) if label in class_indices]
subset = Subset(dataset, indices)
return subset
selected_classes = [0, 1, 2]
train_dataset = filter_by_class(train_dataset_full, selected_classes)
test_dataset = filter_by_class(test_dataset_full, selected_classes)
train_loader = data.DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = data.DataLoader(dataset=test_dataset, batch_size=4*BATCH_SIZE, shuffle=False)
Downloading https://zenodo.org/records/6496656/files/dermamnist.npz to /root/.medmnist/dermamnist.npz
100%| 19725078/19725078 [00:00<00:00, 56946040.91it/s]
Using downloaded and verified file: /root/.medmnist/dermamnist.npz
def train_one_epoch(model, train_loader, criterion, optimizer, device):
model.train()
running_loss = 0.0
correct_predictions = 0
total_predictions = 0
for images, labels in train_loader:
images, labels = images.to(device), labels.to(device)
labels = labels.squeeze() # Reshape labels to 1D
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
running_loss += loss.item()
_, predicted = torch.max(outputs.data, 1)

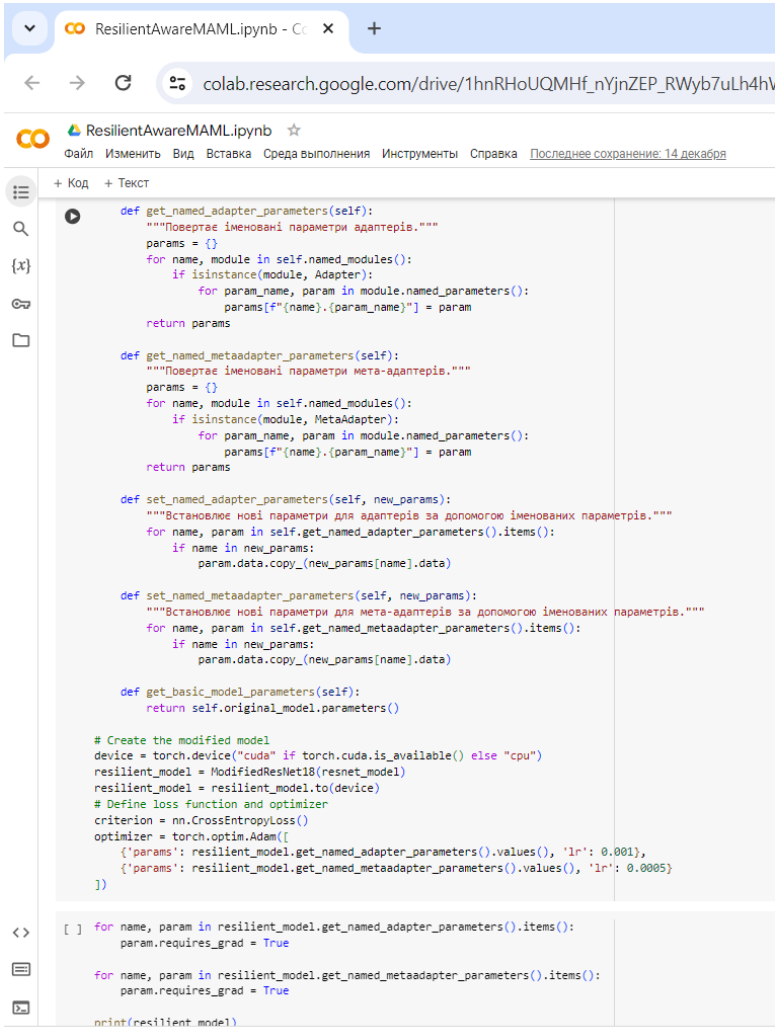

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
for images, labels in train_loader:
images, labels = images.to(device), labels.to(device)
labels = labels.squeeze() # Reshape labels to 1D
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
running_loss += loss.item()
_, predicted = torch.max(outputs.data, 1)
total_predictions += labels.size(0)
correct_predictions += (predicted == labels).sum().item()
epoch_loss = running_loss / len(train_loader)
epoch_accuracy = 100 * correct_predictions / total_predictions
return epoch_loss, epoch_accuracy
def validate(model, val_loader, criterion, device):
model.eval() # Set the model to evaluation mode
running_loss = 0.0
correct_predictions = 0
total_predictions = 0
with torch.no_grad():
for images, labels in val_loader:
images, labels = images.to(device), labels.to(device)
labels = labels.squeeze() # Reshape labels to 1D
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)
running_loss += loss.item()
_, predicted = torch.max(outputs.data, 1)
total_predictions += labels.size(0)
correct_predictions += (predicted == labels).sum().item()
val_loss = running_loss / len(val_loader)
val_accuracy = 100 * correct_predictions / total_predictions
return val_loss, val_accuracy
[ ]

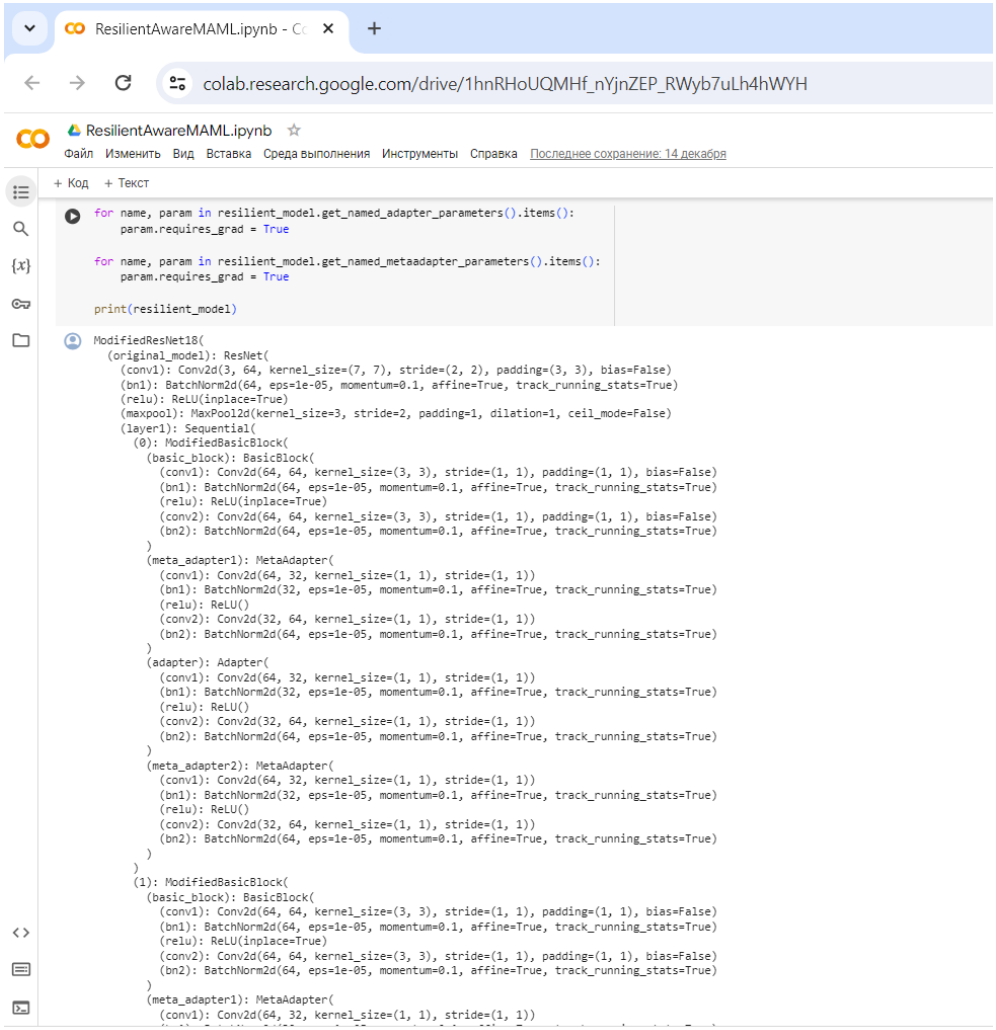

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
num_classes = 3
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
resnet_model = resnet18(pretrained=True)
resnet_model.fc = torch.nn.Linear(resnet_model.fc.in_features, 3)
resnet_model = resnet_model.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet_model.parameters(), lr=0.001)
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weigh
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%| 44.7M/44.7M [00:00<00:00, 184MB/s]
NUM_EPOCHS = 100
for epoch in range(NUM_EPOCHS):
train_loss, train_acc = train_one_epoch(resnet_model, train_loader, criterion, optimizer, device)
val_loss, val_acc = validate(resnet_model, test_loader, criterion, device)
print(f'Epoch {epoch+1}/{NUM_EPOCHS}')
print(f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}')
print(f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}')
Epoch 1/100
Train Loss: 1.1371, Train Acc: 51.7699
Val Loss: 1.0377, Val Acc: 34.7044
Epoch 2/100
Train Loss: 1.0288, Train Acc: 53.6136
Val Loss: 1.7593, Val Acc: 47.3008
Epoch 3/100
Train Loss: 0.9889, Train Acc: 55.8260
Val Loss: 1.1034, Val Acc: 35.4756
Epoch 4/100
Train Loss: 0.9923, Train Acc: 54.6460
Val Loss: 1.0778, Val Acc: 56.5553
Epoch 5/100
Train Loss: 0.9857, Train Acc: 56.2684
Val Loss: 0.9207, Val Acc: 56.5553
Epoch 6/100
Train Loss: 0.9741, Train Acc: 58.1121
Val Loss: 1.1570, Val Acc: 54.7558
Epoch 7/100
Train Loss: 1.0356, Train Acc: 53.5398
Val Loss: 1.0391, Val Acc: 53.2134
Epoch 8/100
Train Loss: 1.0098, Train Acc: 54.2035
Val Loss: 1.1279, Val Acc: 41.9023
Epoch 9/100
Train Loss: 1.0260, Train Acc: 55.1622
Val Loss: 1.0108, Val Acc: 55.0129

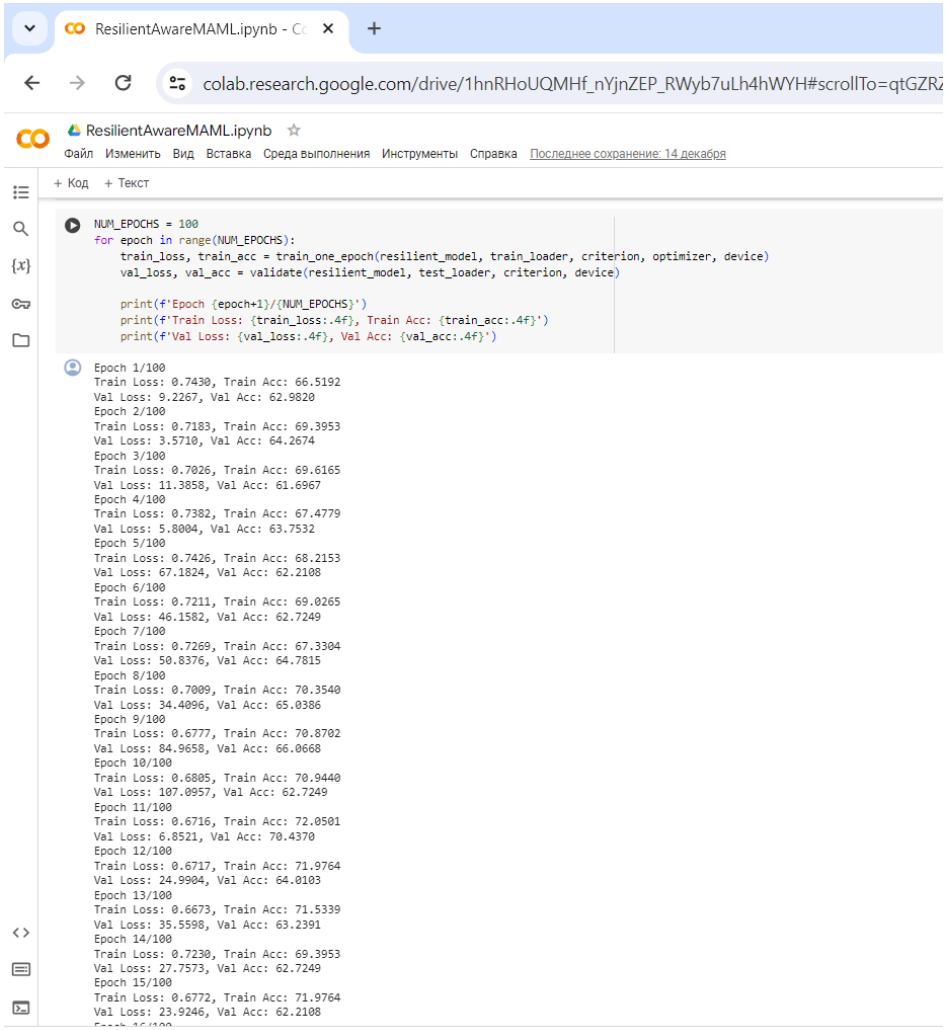

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYI
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
# Заморожуємо базову модель перед додаванням адаптерів і мета-адаптрів
for param in resnet_model.parameters():
param.requires_grad = False
print(resnet_model)
ResNet(
(conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
(layer1): Sequential(
(0): BasicBlock(
(conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(1): BasicBlock(
(conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
)
(layer2): Sequential(
(0): BasicBlock(
(conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(downsample): Sequential(
(0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
)
(1): BasicBlock(
(conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
)
(layer3): Sequential(
(0): BasicBlock(
(conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(downsample): Sequential(

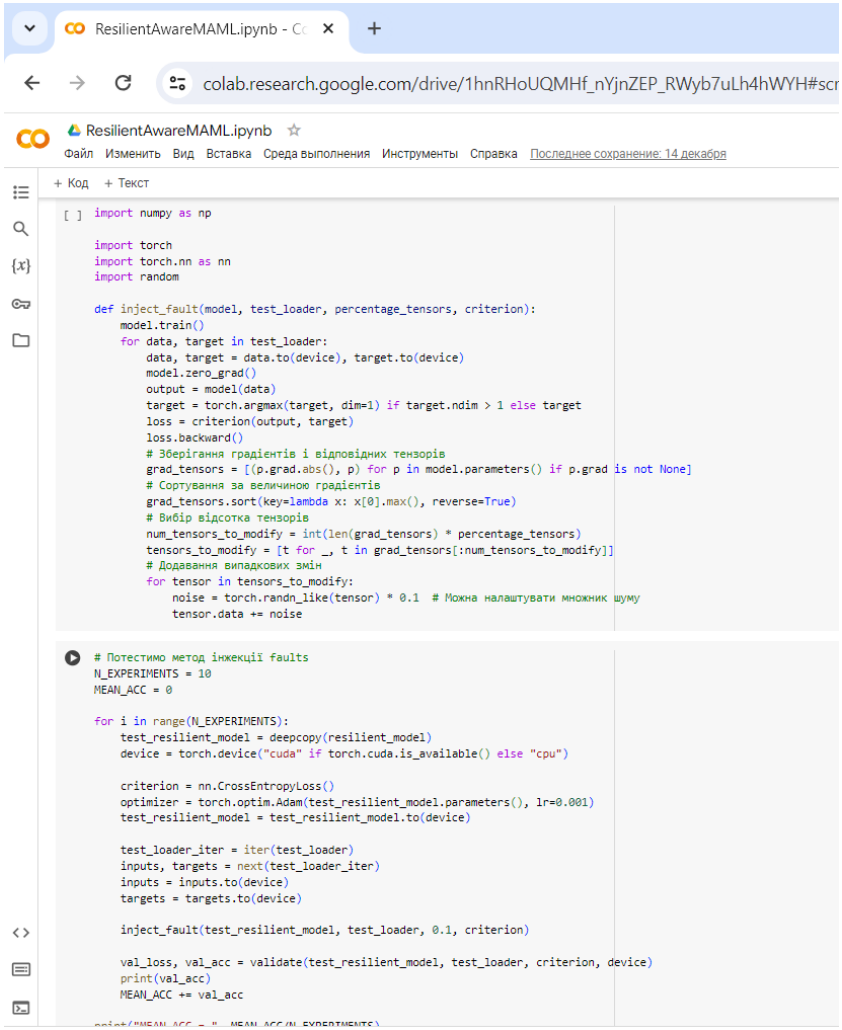

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
import torch
import torch.nn as nn
import torchvision.models as models
import torch.nn.functional as F
from copy import deepcopy
class Adapter(nn.Module):
def __init__(self, in_channels, out_channels):
super(Adapter, self).__init__()
self.conv1 = nn.Conv2d(in_channels, in_channels // 2, kernel_size=1)
self.bn1 = nn.BatchNorm2d(in_channels // 2)
self.relu = nn.ReLU()
self.conv2 = nn.Conv2d(in_channels // 2, out_channels, kernel_size=1)
self.bn2 = nn.BatchNorm2d(out_channels)
def forward(self, x):
x = self.conv1(x)
x = self.bn1(x)
x = self.relu(x)
x = self.conv2(x)
x = self.bn2(x)
return x
class MetaAdapter(nn.Module):
def __init__(self, in_channels, out_channels):
super(MetaAdapter, self).__init__()
# Структура мета-адаптера аналогічна адаптеру
self.conv1 = nn.Conv2d(in_channels, in_channels // 2, kernel_size=1)
self.bn1 = nn.BatchNorm2d(in_channels // 2)
self.relu = nn.ReLU()
self.conv2 = nn.Conv2d(in_channels // 2, out_channels, kernel_size=1)
self.bn2 = nn.BatchNorm2d(out_channels)
def forward(self, x):
x = self.conv1(x)
x = self.bn1(x)
x = self.relu(x)
x = self.conv2(x)
x = self.bn2(x)
return x
class ModifiedBasicBlock(nn.Module):
def __init__(self, basic_block):
super(ModifiedBasicBlock, self).__init__()
self.basic_block = basic_block # Зберігаємо оригінальний BasicBlock
in_channels = basic_block.conv1.in_channels # Знаходимо кількість вхідних каналів
out_channels = basic_block.conv2.out_channels # Кількість вихідних каналів
# Створення адаптерів та мета-адаптерів
self.meta_adapter1 = MetaAdapter(in_channels, out_channels)

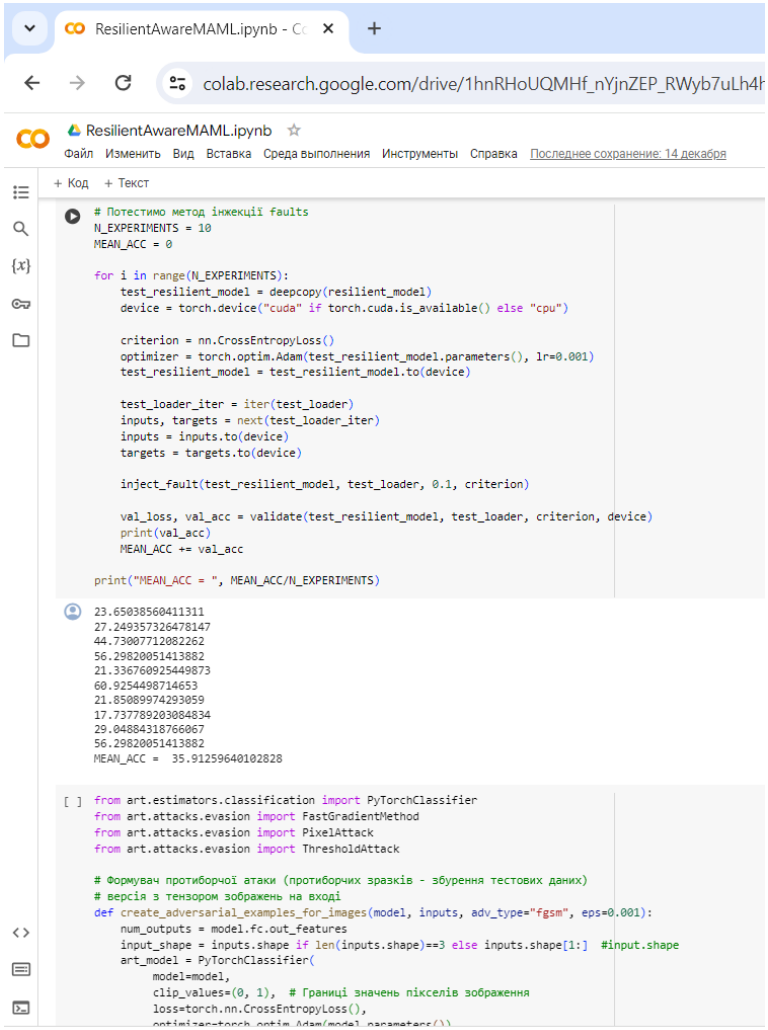

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
class ModifiedBasicBlock(nn.Module):
def __init__(self, basic_block):
super(ModifiedBasicBlock, self).__init__()
self.basic_block = basic_block # Зберігаємо оригінальний BasicBlock
in_channels = basic_block.conv1.in_channels # Знаходимо кількість вхідних каналів
out_channels = basic_block.conv2.out_channels # Кількість вихідних каналів
# Створення адаптерів та мета-адаптерів
self.meta_adapter1 = MetaAdapter(in_channels, out_channels)
self.adapter = Adapter(out_channels, out_channels)
self.meta_adapter2 = MetaAdapter(out_channels, out_channels)
def forward(self, x):
out = self.basic_block(x)
correction = self.meta_adapter1(x)
correction = self.adapter(correction)
correction = self.meta_adapter2(correction)
# Вирівнювання розмірів перед сумуванням
if correction.size() != out.size():
correction = F.interpolate(correction, size=out.shape[2:], mode='bilinear', align_corners=True)
out = out + correction
return out
class ModifiedResNet18(nn.Module):
def __init__(self, original_model):
super(ModifiedResNet18, self).__init__()
self.original_model = deepcopy(original_model)
# Копіювання шарів з оригінальної моделі ResNet-18
self.features = nn.Sequential(*(list(self.original_model.children())[:-1]))
# Заміна BasicBlocks на ModifiedBasicBlocks
for name, module in self.features.named_children():
if isinstance(module, nn.Sequential):
for i, basic_block in enumerate(module):
module[i] = ModifiedBasicBlock(basic_block)
# Копіювання повнозв'язного шару
self.fc = self.original_model.fc
def forward(self, x):
# Проходження вхідних даних через модифіковані шари
x = self.features(x)
# Згортання вектора до одного виміру перед передачею у повнозв'язний шар
x = torch.flatten(x, 1)
# Виконання повнозв'язного шару
x = self.fc(x)
return x
def get_named_adapter_parameters(self):
"""Повертає іменовані параметри адаптерів."""

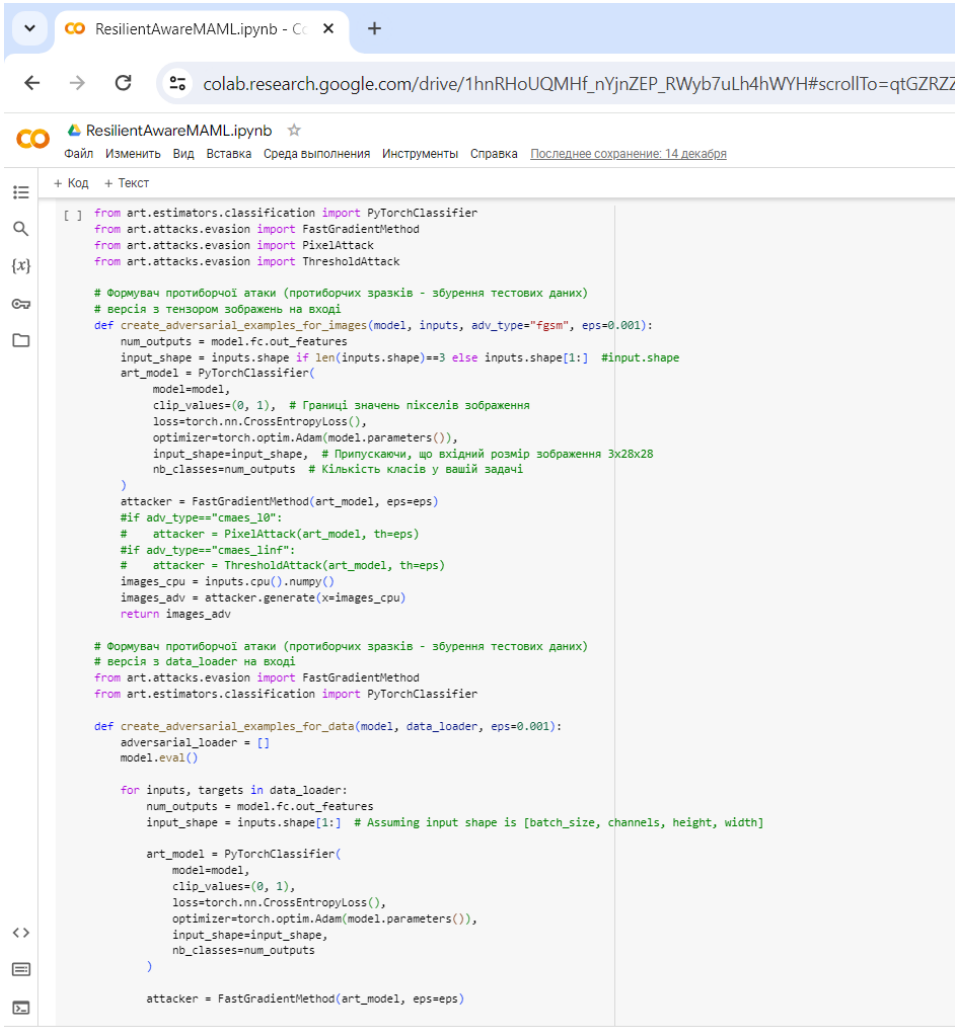

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4h
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
def get_named_adapter_parameters(self):
"""Повертає іменовані параметри адаптерів."""
params = {}
for name, module in self.named_modules():
if isinstance(module, Adapter):
for param_name, param in module.named_parameters():
params[f"{name}.{param_name}"] = param
return params
def get_named_metaadapter_parameters(self):
"""Повертає іменовані параметри мета-адаптерів."""
params = {}
for name, module in self.named_modules():
if isinstance(module, MetaAdapter):
for param_name, param in module.named_parameters():
params[f"{name}.{param_name}"] = param
return params
def set_named_adapter_parameters(self, new_params):
"""Встановлює нові параметри для адаптерів за допомогою іменованих параметрів."""
for name, param in self.get_named_adapter_parameters().items():
if name in new_params:
param.data.copy_(new_params[name].data)
def set_named_metaadapter_parameters(self, new_params):
"""Встановлює нові параметри для мета-адаптерів за допомогою іменованих параметрів."""
for name, param in self.get_named_metaadapter_parameters().items():
if name in new_params:
param.data.copy_(new_params[name].data)
def get_basic_model_parameters(self):
return self.original_model.parameters()
# Create the modified model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
resilient_model = ModifiedResNet18(resnet_model)
resilient_model = resilient_model.to(device)
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam([
{'params': resilient_model.get_named_adapter_parameters().values(), 'lr': 0.001},
{'params': resilient_model.get_named_metaadapter_parameters().values(), 'lr': 0.0005}
])
[ ]
for name, param in resilient_model.get_named_adapter_parameters().items():
param.requires_grad = True
for name, param in resilient_model.get_named_metaadapter_parameters().items():
param.requires_grad = True
print(resilient_model)

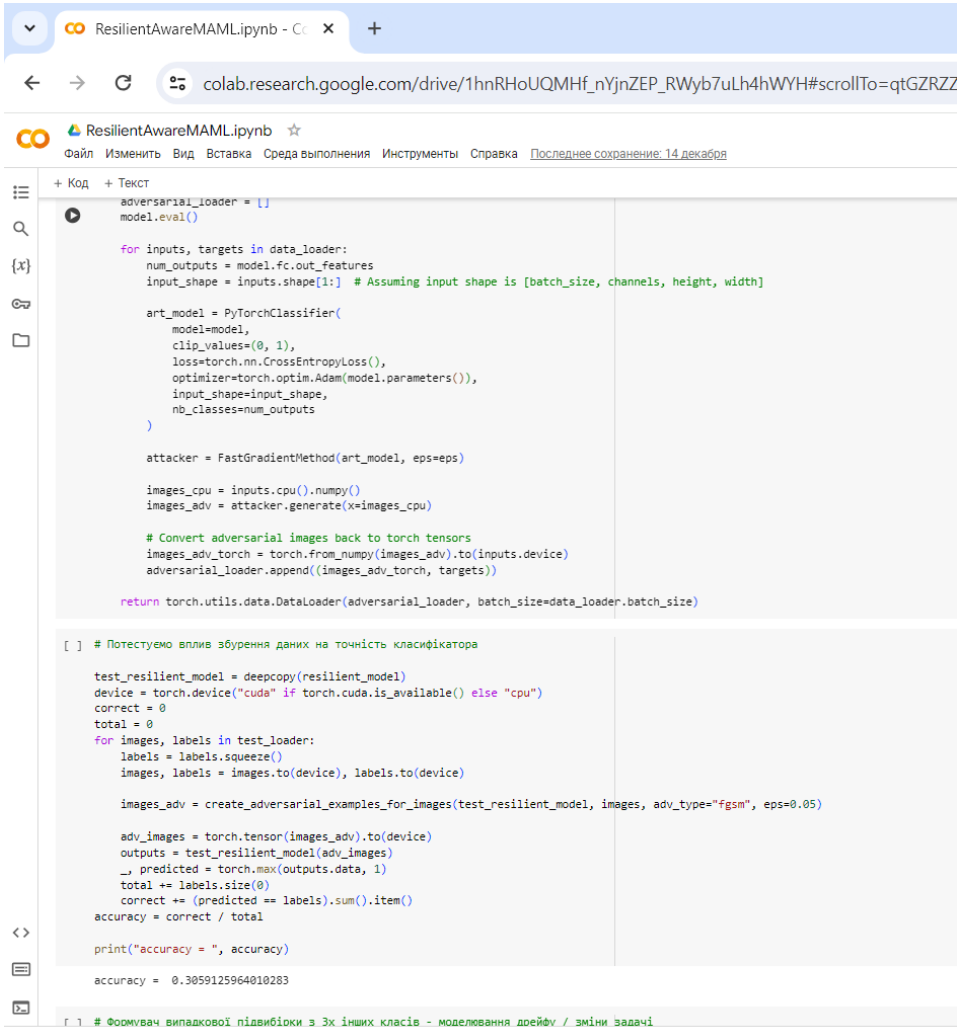

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
for name, param in resilient_model.get_named_adapter_parameters().items():
param.requires_grad = True
for name, param in resilient_model.get_named_metaadapter_parameters().items():
param.requires_grad = True
print(resilient_model)
ModifiedResNet18(
(original_model): ResNet(
(conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
(layer1): Sequential(
(0): ModifiedBasicBlock(
(basic_block): BasicBlock(
(conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(meta_adapter1): MetaAdapter(
(conv1): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
(bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU()
(conv2): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1))
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(adapter): Adapter(
(conv1): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
(bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU()
(conv2): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1))
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(meta_adapter2): MetaAdapter(
(conv1): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
(bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU()
(conv2): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1))
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
)
(1): ModifiedBasicBlock(
(basic_block): BasicBlock(
(conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(meta_adapter1): MetaAdapter(
(conv1): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))

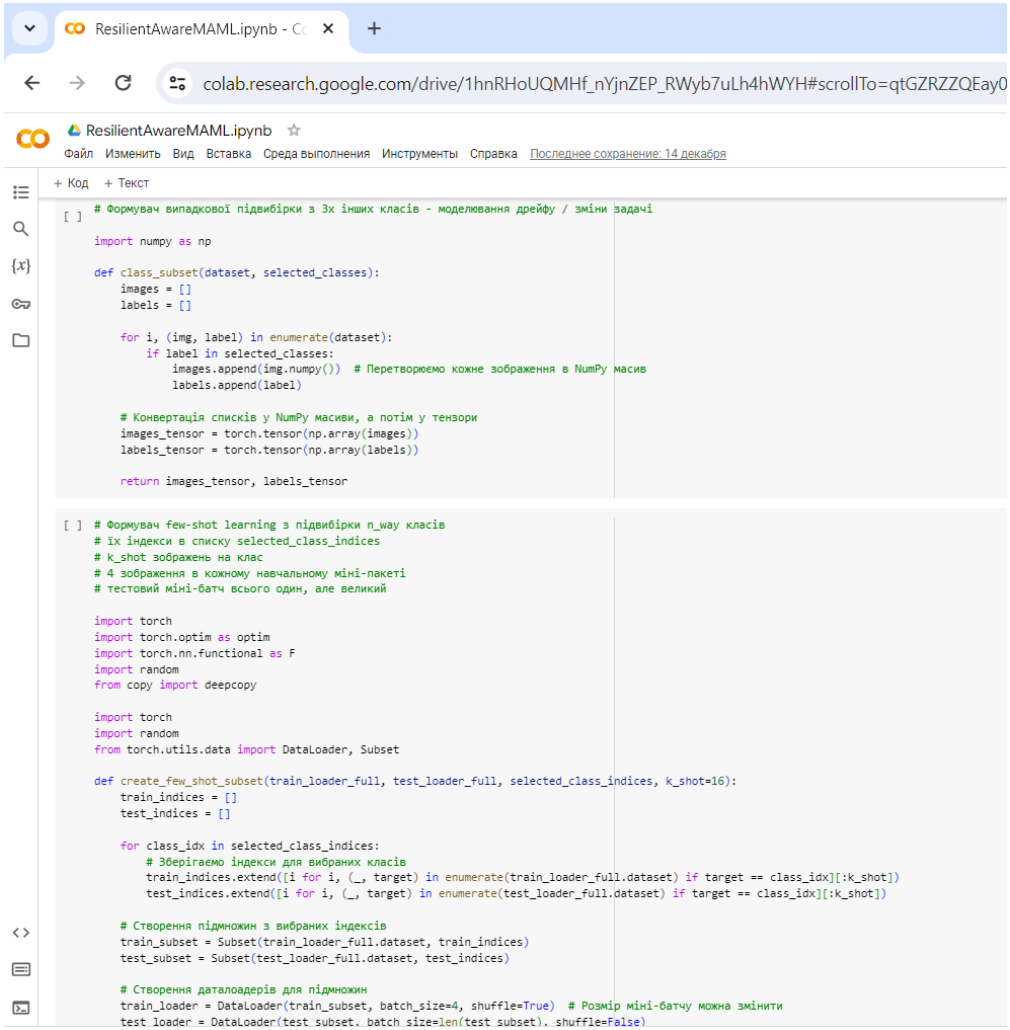

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZ
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
NUM_EPOCHS = 100
for epoch in range(NUM_EPOCHS):
train_loss, train_acc = train_one_epoch(resilient_model, train_loader, criterion, optimizer, device)
val_loss, val_acc = validate(resilient_model, test_loader, criterion, device)
print(f'Epoch {epoch+1}/{NUM_EPOCHS}')
print(f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}')
print(f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}')
Epoch 1/100
Train Loss: 0.7430, Train Acc: 66.5192
Val Loss: 9.2267, Val Acc: 62.9820
Epoch 2/100
Train Loss: 0.7183, Train Acc: 69.3953
Val Loss: 3.5710, Val Acc: 64.2674
Epoch 3/100
Train Loss: 0.7026, Train Acc: 69.6165
Val Loss: 11.3858, Val Acc: 61.6967
Epoch 4/100
Train Loss: 0.7382, Train Acc: 67.4779
Val Loss: 5.8004, Val Acc: 63.7532
Epoch 5/100
Train Loss: 0.7426, Train Acc: 68.2153
Val Loss: 67.1824, Val Acc: 62.2108
Epoch 6/100
Train Loss: 0.7211, Train Acc: 69.0265
Val Loss: 46.1582, Val Acc: 62.7249
Epoch 7/100
Train Loss: 0.7269, Train Acc: 67.3304
Val Loss: 50.8376, Val Acc: 64.7815
Epoch 8/100
Train Loss: 0.7009, Train Acc: 70.3540
Val Loss: 34.4096, Val Acc: 65.0386
Epoch 9/100
Train Loss: 0.6777, Train Acc: 70.8702
Val Loss: 84.9658, Val Acc: 66.0668
Epoch 10/100
Train Loss: 0.6805, Train Acc: 70.9440
Val Loss: 107.0957, Val Acc: 62.7249
Epoch 11/100
Train Loss: 0.6716, Train Acc: 72.0501
Val Loss: 6.8521, Val Acc: 70.4370
Epoch 12/100
Train Loss: 0.6717, Train Acc: 71.9764
Val Loss: 24.9904, Val Acc: 64.0103
Epoch 13/100
Train Loss: 0.6673, Train Acc: 71.5339
Val Loss: 35.5598, Val Acc: 63.2391
Epoch 14/100
Train Loss: 0.7230, Train Acc: 69.3953
Val Loss: 27.7573, Val Acc: 62.7249
Epoch 15/100
Train Loss: 0.6772, Train Acc: 71.9764
Val Loss: 23.9246, Val Acc: 62.2108

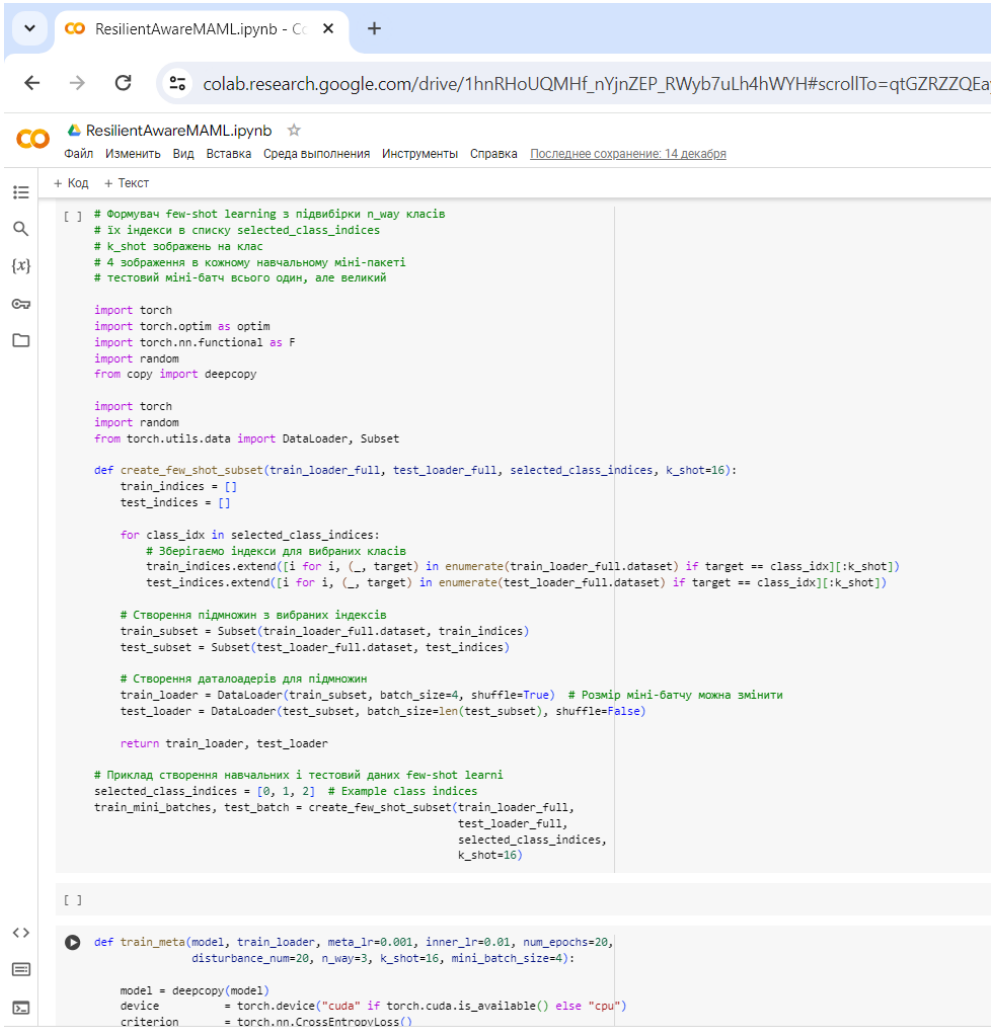

ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scr
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ] import numpy as np
import torch
import torch.nn as nn
import random
def inject_fault(model, test_loader, percentage_tensors, criterion):
model.train()
for data, target in test_loader:
data, target = data.to(device), target.to(device)
model.zero_grad()
output = model(data)
target = torch.argmax(target, dim=1) if target.ndim > 1 else target
loss = criterion(output, target)
loss.backward()
# Зберігання градієнтів і відповідних тензорів
grad_tensors = [(p.grad.abs(), p) for p in model.parameters() if p.grad is not None]
# Сортування за величиною градієнтів
grad_tensors.sort(key=lambda x: x[0].max(), reverse=True)
# Вибір відсотка тензорів
num_tensors_to_modify = int(len(grad_tensors) * percentage_tensors)
tensors_to_modify = [t for _, t in grad_tensors[:num_tensors_to_modify]]
# Додавання випадкових змін
for tensor in tensors_to_modify:
noise = torch.randn_like(tensor) * 0.1 # Можна налаштувати множник шуму
tensor.data += noise
# Потестимо метод інжекції faults
N_EXPERIMENTS = 10
MEAN_ACC = 0
for i in range(N_EXPERIMENTS):
test_resilient_model = deepcopy(resilient_model)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(test_resilient_model.parameters(), lr=0.001)
test_resilient_model = test_resilient_model.to(device)
test_loader_iter = iter(test_loader)
inputs, targets = next(test_loader_iter)
inputs = inputs.to(device)
targets = targets.to(device)
inject_fault(test_resilient_model, test_loader, 0.1, criterion)
val_loss, val_acc = validate(test_resilient_model, test_loader, criterion, device)
print(val_acc)
MEAN_ACC += val_acc


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
# Потестимо метод інжекції faults
N_EXPERIMENTS = 10
MEAN_ACC = 0
for i in range(N_EXPERIMENTS):
test_resilient_model = deepcopy(resilient_model)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(test_resilient_model.parameters(), lr=0.001)
test_resilient_model = test_resilient_model.to(device)
test_loader_iter = iter(test_loader)
inputs, targets = next(test_loader_iter)
inputs = inputs.to(device)
targets = targets.to(device)
inject_fault(test_resilient_model, test_loader, 0.1, criterion)
val_loss, val_acc = validate(test_resilient_model, test_loader, criterion, device)
print(val_acc)
MEAN_ACC += val_acc
print("MEAN_ACC = ", MEAN_ACC/N_EXPERIMENTS)
23.65038560411311
27.249357326478147
44.73007712082262
56.29820051413882
21.336760925449873
60.9254498714653
21.85089974293059
17.737789203084834
29.04884318766067
56.29820051413882
MEAN_ACC = 35.9125964010282
[ ]
from art.estimators.classification import PyTorchClassifier
from art.attacks.evasion import FastGradientMethod
from art.attacks.evasion import PixelAttack
from art.attacks.evasion import ThresholdAttack
# Формувач протиборчої атаки (протиборчих зразків - збурення тестових даних)
# версія з тензором зображень на вході
def create_adversarial_examples_for_images(model, inputs, adv_type="fgsm", eps=0.001):
num_outputs = model.fc.out_features
input_shape = inputs.shape if len(inputs.shape)==3 else inputs.shape[1:] #input.shape
art_model = PyTorchClassifier(
model=model,
clip_values=(0, 1), # Границі значень пікселів зображення
loss=torch.nn.CrossEntropyLoss(),


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZ
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ]
from art.estimators.classification import PyTorchClassifier
from art.attacks.evasion import FastGradientMethod
from art.attacks.evasion import PixelAttack
from art.attacks.evasion import ThresholdAttack
# Формувач протиборчої атаки (протиборчих зразків - збурення тестових даних)
# версія з тензором зображень на вході
def create_adversarial_examples_for_images(model, inputs, adv_type="fgsm", eps=0.001):
num_outputs = model.fc.out_features
input_shape = inputs.shape if len(inputs.shape)==3 else inputs.shape[1:] #input.shape
art_model = PyTorchClassifier(
model=model,
clip_values=(0, 1), # Границі значень пікселів зображення
loss=torch.nn.CrossEntropyLoss(),
optimizer=torch.optim.Adam(model.parameters()),
input_shape=input_shape, # Припускаючи, що вхідний розмір зображення 3x28x28
nb_classes=num_outputs # Кількість класів у вашій задачі
)
attacker = FastGradientMethod(art_model, eps=eps)
#if adv_type=="cmaes_l0":
attacker = PixelAttack(art_model, th=eps)
#if adv_type=="cmaes_linf":
# attacker = ThresholdAttack(art_model, th=eps)
images_cpu = inputs.cpu().numpy()
images_adv = attacker.generate(x=images_cpu)
return images_adv
# Формувач протиборчої атаки (протиборчих зразків - збурення тестових даних)
# версія з data_loader на вході
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import PyTorchClassifier
def create_adversarial_examples_for_data(model, data_loader, eps=0.001):
adversarial_loader = []
model.eval()
for inputs, targets in data_loader:
num_outputs = model.fc.out_features
input_shape = inputs.shape[1:] # Assuming input shape is [batch_size, channels, height, width]
art_model = PyTorchClassifier(
model=model,
clip_values=(0, 1),
loss=torch.nn.CrossEntropyLoss(),
optimizer=torch.optim.Adam(model.parameters()),
input_shape=input_shape,
nb_classes=num_outputs
)
attacker = FastGradientMethod(art_model, eps=eps)


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZZ
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
adversarial_loader = []
model.eval()
for inputs, targets in data_loader:
num_outputs = model.fc.out_features
input_shape = inputs.shape[1:] # Assuming input shape is [batch_size, channels, height, width]
art_model = PyTorchClassifier(
model=model,
clip_values=(0, 1),
loss=torch.nn.CrossEntropyLoss(),
optimizer=torch.optim.Adam(model.parameters()),
input_shape=input_shape,
nb_classes=num_outputs
)
attacker = FastGradientMethod(art_model, eps=eps)
images_cpu = inputs.cpu().numpy()
images_adv = attacker.generate(x=images_cpu)
# Convert adversarial images back to torch tensors
images_adv_torch = torch.from_numpy(images_adv).to(inputs.device)
adversarial_loader.append((images_adv_torch, targets))
return torch.utils.data.DataLoader(adversarial_loader, batch_size=data_loader.batch_size)
[ ] # Потестуємо вплив збурення даних на точність класифікатора
test_resilient_model = deepcopy(resilient_model)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
correct = 0
total = 0
for images, labels in test_loader:
labels = labels.squeeze()
images, labels = images.to(device), labels.to(device)
images_adv = create_adversarial_examples_for_images(test_resilient_model, images, adv_type="fgsm", eps=0.05)
adv_images = torch.tensor(images_adv).to(device)
outputs = test_resilient_model(adv_images)
_, predicted = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()
accuracy = correct / total
print("accuracy = ", accuracy)
accuracy = 0.3059125964010283
[ ] # Формувач випадкової підвибірки з 3х інших класів - моделювання дрейфу / зміни задачі


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZZQEay0
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ]
# Формувач випадкової підвибірки з 3х інших класів - моделювання дрейфу / зміни задачі
import numpy as np
def class_subset(dataset, selected_classes):
images = []
labels = []
for i, (img, label) in enumerate(dataset):
if label in selected_classes:
images.append(img.numpy()) # Перетворюємо кожне зображення в NumPy масив
labels.append(label)
# Конвертація списків у NumPy масиви, а потім у тензори
images_tensor = torch.tensor(np.array(images))
labels_tensor = torch.tensor(np.array(labels))
return images_tensor, labels_tensor
[ ]
# Формувач few-shot learning з підвибірки n_way класів
# їх індекси в списку selected_class_indices
# k_shot зображень на клас
# 4 зображення в кожному навчальному міні-пакеті
# тестовий міні-батч всього один, але великий
import torch
import torch.optim as optim
import torch.nn.functional as F
import random
from copy import deepcopy
import torch
import random
from torch.utils.data import DataLoader, Subset
def create_few_shot_subset(train_loader_full, test_loader_full, selected_class_indices, k_shot=16):
train_indices = []
test_indices = []
for class_idx in selected_class_indices:
# Зберігаємо індекси для вибраних класів
train_indices.extend([i for i, (_, target) in enumerate(train_loader_full.dataset) if target == class_idx][:k_shot])
test_indices.extend([i for i, (_, target) in enumerate(test_loader_full.dataset) if target == class_idx][:k_shot])
# Створення підмножин з вибраних індексів
train_subset = Subset(train_loader_full.dataset, train_indices)
test_subset = Subset(test_loader_full.dataset, test_indices)
# Створення даталоадерів для підмножин
train_loader = DataLoader(train_subset, batch_size=4, shuffle=True) # Розмір міні-батчу можна змінити
test_loader = DataLoader(test_subset, batch_size=len(test_subset), shuffle=False)



ResilientAwareMAML.ipynb - C

colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZZQEa

ResilientAwareMAML.ipynb

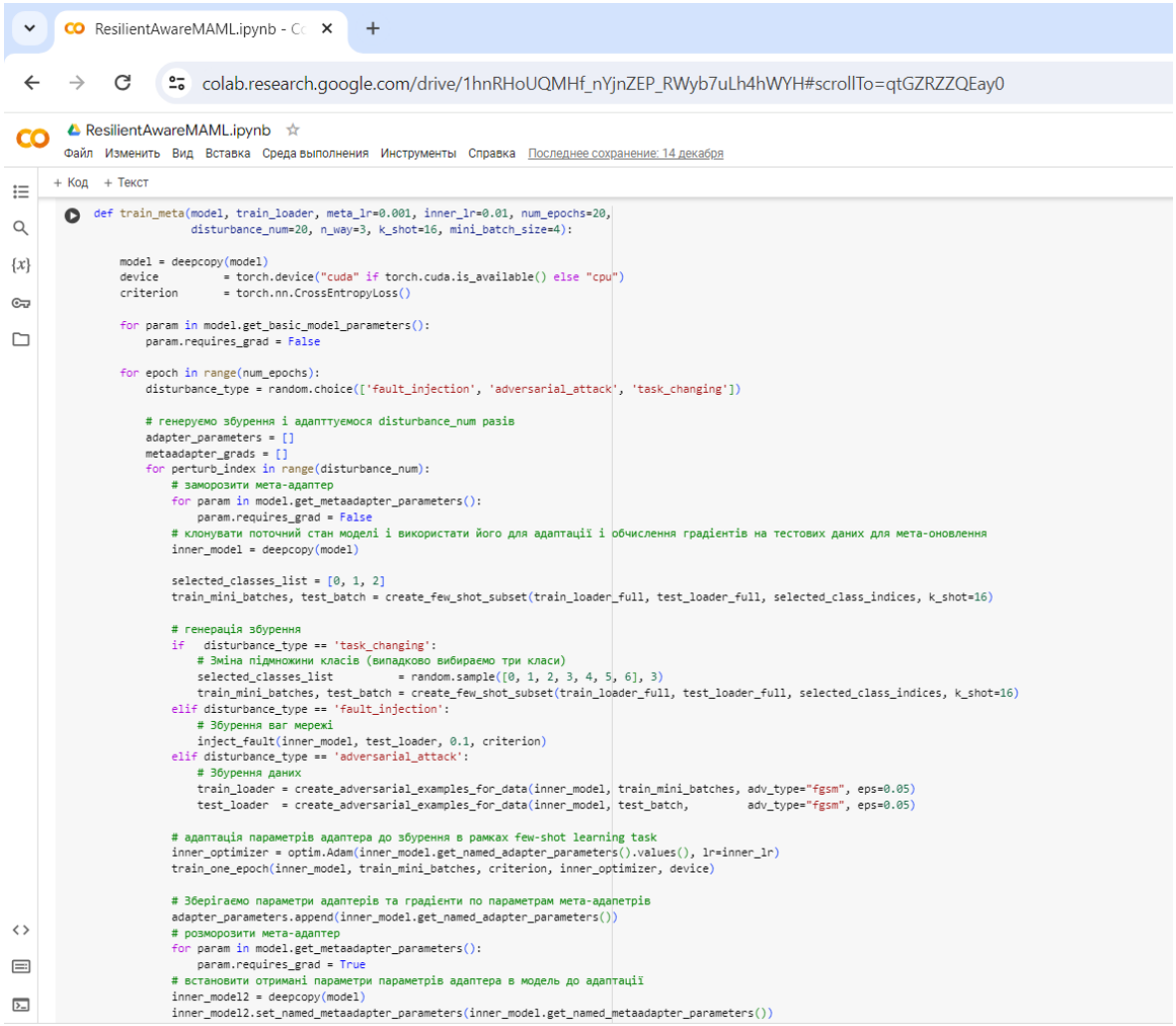
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря

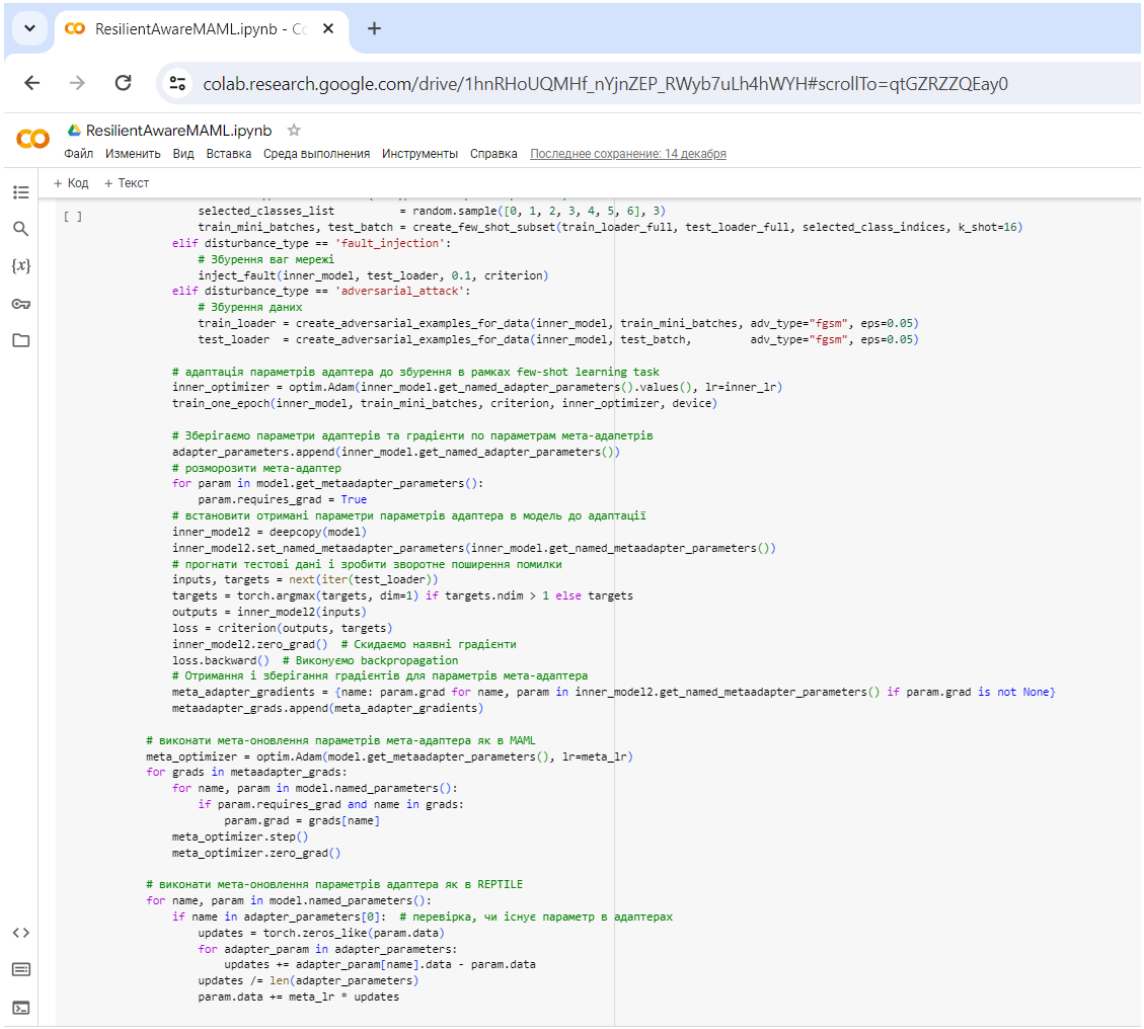
+ Код + Текст

```
[ ] # Формувач few-shot learning з підвибірки n_way класів
    # їх індекси в списку selected_class_indices
    # k_shot зображень на клас
    # 4 зображення в кожному навчальному міні-пакеті
    # тестовий міні-батч всього один, але великий

    import torch
    import torch.optim as optim
    import torch.nn.functional as F
    import random
    from copy import deepcopy

    import torch
    import random
    from torch.utils.data import DataLoader, Subset

    def create_few_shot_subset(train_loader_full, test_loader_full, selected_class_indices, k_shot=16):
        train_indices = []
        test_indices = []

        for class_idx in selected_class_indices:
            # Зберігаємо індекси для вибраних класів
            train_indices.extend([i for i, (_, target) in enumerate(train_loader_full.dataset) if target == class_idx][:k_shot])
            test_indices.extend([i for i, (_, target) in enumerate(test_loader_full.dataset) if target == class_idx][:k_shot])

        # Створення підмножин з вибраних індексів
        train_subset = Subset(train_loader_full.dataset, train_indices)
        test_subset = Subset(test_loader_full.dataset, test_indices)

        # Створення даталоадерів для підмножин
        train_loader = DataLoader(train_subset, batch_size=4, shuffle=True)  # Розмір міні-батчу можна змінити
        test_loader = DataLoader(test_subset, batch_size=len(test_subset), shuffle=False)

        return train_loader, test_loader

    # Приклад створення навчальних і тестовий даних few-shot learni
    selected_class_indices = [0, 1, 2]  # Example class indices
    train_mini_batches, test_batch = create_few_shot_subset(train_loader_full,
                                                            test_loader_full,
                                                            selected_class_indices,
                                                            k_shot=16)
```

```
[ ]
```

```
def train_meta(model, train_loader, meta_lr=0.001, inner_lr=0.01, num_epochs=20,
               disturbance_num=20, n_way=3, k_shot=16, mini_batch_size=4):

    model = deepcopy(model)
    device          = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    criterion       = torch.nn.CrossEntropyLoss()
```


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZZQEay0
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
def train_meta(model, train_loader, meta_lr=0.001, inner_lr=0.01, num_epochs=20,
disturbance_num=20, n_way=3, k_shot=16, mini_batch_size=4):
model = deepcopy(model)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = torch.nn.CrossEntropyLoss()
for param in model.get_basic_model_parameters():
param.requires_grad = False
for epoch in range(num_epochs):
disturbance_type = random.choice(['fault_injection', 'adversarial_attack', 'task_changing'])
# генеруємо збурення і адапттуємося disturbance_num разів
adapter_parameters = []
metaadapter_grads = []
for perturb_index in range(disturbance_num):
# заморозити мета-адаптер
for param in model.get_metaadapter_parameters():
param.requires_grad = False
# клонувати поточний стан моделі і використати його для адаптації і обчислення градієнтів на тестових даних для мета-оновлення
inner_model = deepcopy(model)
selected_classes_list = [0, 1, 2]
train_mini_batches, test_batch = create_few_shot_subset(train_loader_full, test_loader_full, selected_class_indices, k_shot=16)
# генерація збурення
if disturbance_type == 'task_changing':
# Зміна підмножини класів (випадково вибираємо три класи)
selected_classes_list = random.sample([0, 1, 2, 3, 4, 5, 6], 3)
train_mini_batches, test_batch = create_few_shot_subset(train_loader_full, test_loader_full, selected_class_indices, k_shot=16)
elif disturbance_type == 'fault_injection':
# Збурення ваг мережі
inject_fault(inner_model, test_loader, 0.1, criterion)
elif disturbance_type == 'adversarial_attack':
# Збурення даних
train_loader = create_adversarial_examples_for_data(inner_model, train_mini_batches, adv_type="fgsm", eps=0.05)
test_loader = create_adversarial_examples_for_data(inner_model, test_batch, adv_type="fgsm", eps=0.05)
# адаптація параметрів адаптера до збурення в рамках few-shot learning task
inner_optimizer = optim.Adam(inner_model.get_named_adapter_parameters().values(), lr=inner_lr)
train_one_epoch(inner_model, train_mini_batches, criterion, inner_optimizer, device)
# Зберігаємо параметри адаптерів та градієнти по параметрам мета-адапетрів
adapter_parameters.append(inner_model.get_named_adapter_parameters())
# розморозити мета-адаптер
for param in model.get_metaadapter_parameters():
param.requires_grad = True
# встановити отримані параметри параметрів адаптера в модель до адаптації
inner_model2 = deepcopy(model)
inner_model2.set_named_metaadapter_parameters(inner_model.get_named_metaadapter_parameters())


ResilientAwareMAML.ipynb - C
colab.research.google.com/drive/1hnRHoUQMHf_nYjnZEP_RWyb7uLh4hWYH#scrollTo=qtGZRZZQEay0
ResilientAwareMAML.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее сохранение: 14 декабря
+ Код + Текст
[ ]
selected_classes_list = random.sample([0, 1, 2, 3, 4, 5, 6], 3)
train_mini_batches, test_batch = create_few_shot_subset(train_loader_full, test_loader_full, selected_class_indices, k_shot=16)
elif disturbance_type == 'fault_injection':
# Збурення ваг мережі
inject_fault(inner_model, test_loader, 0.1, criterion)
elif disturbance_type == 'adversarial_attack':
# Збурення даних
train_loader = create_adversarial_examples_for_data(inner_model, train_mini_batches, adv_type="fgsm", eps=0.05)
test_loader = create_adversarial_examples_for_data(inner_model, test_batch, adv_type="fgsm", eps=0.05)
# адаптація параметрів адаптера до збурення в рамках few-shot learning task
inner_optimizer = optim.Adam(inner_model.get_named_adapter_parameters().values(), lr=inner_lr)
train_one_epoch(inner_model, train_mini_batches, criterion, inner_optimizer, device)
# Зберігаємо параметри адаптерів та градієнти по параметрам мета-адапетрів
adapter_parameters.append(inner_model.get_named_adapter_parameters())
# розморозити мета-адаптер
for param in model.get_metaadapter_parameters():
param.requires_grad = True
# встановити отримані параметри параметрів адаптера в модель до адаптації
inner_model2 = deepcopy(model)
inner_model2.set_named_metaadapter_parameters(inner_model.get_named_metaadapter_parameters())
# прогнати тестові дані і зробити зворотне поширення помилки
inputs, targets = next(iter(test_loader))
targets = torch.argmax(targets, dim=1) if targets.ndim > 1 else targets
outputs = inner_model2(inputs)
loss = criterion(outputs, targets)
inner_model2.zero_grad() # Скидаємо наявні градієнти
loss.backward() # Виконуємо backpropagation
# Отримання і зберігання градієнтів для параметрів мета-адаптера
meta_adapter_gradients = {name: param.grad for name, param in inner_model2.get_named_metaadapter_parameters() if param.grad is not None}
metaadapter_grads.append(meta_adapter_gradients)
# виконати мета-оновлення параметрів мета-адаптера як в MAML
meta_optimizer = optim.Adam(model.get_metaadapter_parameters(), lr=meta_lr)
for grads in metaadapter_grads:
for name, param in model.named_parameters():
if param.requires_grad and name in grads:
param.grad = grads[name]
meta_optimizer.step()
meta_optimizer.zero_grad()
# виконати мета-оновлення параметрів адаптера як в REPTILE
for name, param in model.named_parameters():
if name in adapter_parameters[0]: # перевірка, чи існує параметр в адаптерах
updates = torch.zeros_like(param.data)
for adapter_param in adapter_parameters:
updates += adapter_param[name].data - param.data
updates /= len(adapter_parameters)
param.data += meta_lr * updates