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)
_____ грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія підвищення стійкості системи розпізнавання зображень до програмно-апаратних помилок»
здобувачки групи ІН.м - 22 Шелест Аліни Віталіївни

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

_____ Аліна ШЕЛЕСТ
(підпис)

Керівник, старша викладачка
кафедри комп'ютерних наук,
кандидат технічних наук Альона МОСКАЛЕНКО _____
(підпис)

Суми – 2023

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувачки групи ІН.м-22 Шелест Аліни Віталіївни

1. Тема роботи: «Інформаційна технологія підвищення стійкості системи розпізнавання зображень до програмно-апаратних помилок»
затверджую наказом по СумДУ *від 06 грудня 2023 р. №1412-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 проблематики та актуальності теми. Постановка задачі. 2)Огляд програмно-апаратних помилок, методу навчання нейронних мереж та способу ін'єкції. 3) Розробка інформаційної системи підвищення стійкості системи розпізнавання зображень до програмно-апаратних помилок. 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 _____ (підпис)

Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблематики та актуальності теми. Постановка задачі.*	06.11.2023 – 13.11.2023	
2	*Огляд програмно-апаратних помилок, методу навчання нейронних мереж та способу ін'єкції.*	14.11.2023 – 21.11.2023	
3	*Розробка інформаційної системи підвищення стійкості системи розпізнавання зображень до програмно-апаратних помилок.*	22.11.2023 – 06.12.2023	

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
4	*Аналіз отриманих результатів*	07.12.2023 – 08.12.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	08.12.2023 – 17.12.2023	

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

АНОТАЦІЯ

Записка: 74 стр., 51 рис., 1 додаток, 14 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, так як розвиток нейронних мереж у критичних сферах обумовлений дотриманням їх стійкості та надійності.

Об’єкт дослідження — аналіз зміни точності, які спричиняють ін’єкції помилок у багатоверсійні моделі з різноархітектурними моделями.

Мета роботи — розробка інформаційної технології підвищення стійкості системи розпізнавання зображень до програмно-апаратних помилок.

Методи дослідження — методи ін’єкції помилок, голосування за прогнозами.

Результати — розроблено інформаційну систему, яка створює багатоверсійні моделі з однакових та різних типів архітектур, ін’єктує помилки, розраховує показник стійкості до збоїв, голосує за прогнозами.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ГОЛОСУВАННЯ ЗА ПРОГНОЗАМИ, ІН’ЄКЦІЯ ПОМИЛОК, PYTHON, PYTORCH, NVDNN

ЗМІСТ

ВСТУП

Створення інформаційної технології для підвищення стійкості систем розпізнавання зображень до програмно-апаратних помилок є вкрай актуальним завданням сучасності. Важливість надійності для критичних систем полягає в тому, що глибокі нейронні мережі (DNN) забезпечують прорив у багатьох сферах, таких як комп'ютерний зір і обробка природної мови, але надійність цих моделей є важливим питанням. Ненадійність моделей DNN, особливо їхню вразливість до збурень даних, ставить під загрозу їхнє використання у системах, де критично важливі безпека та точність. Також, забезпечення теоретичної надійності окремої моделі залишається складним завданням. Тому зосередження уваги на підвищенні стійкості систем глибокого навчання за допомогою надлишкових моделей вважається перспективним напрямком. Класичне N-версійне програмування (NVP), яке є ефективним для забезпечення надійності традиційного програмного забезпечення, не може бути безпосередньо застосоване до моделей глибокого навчання через їх автоматичне вивчення з даних, а не з кодування розробниками. Важливим аспектом є розробка незалежних моделей з використанням різних факторів незалежності, таких як незалежне навчання, незалежна мережа та незалежні дані. Експерименти на датасетах, таких як MNIST та CIFAR-10, показали, що всі ці фактори ефективні для підвищення стійкості систем глибокого навчання. Зокрема, незалежні дані для тренування мають найбільше значення у створенні кількох моделей, які мінімізують спільні помилки [1].

Об'єктом дослідження в цій роботі є системи розпізнавання зображень, що включають різноманітні компоненти, які можуть бути піддані програмно-апаратним помилкам. Буде проведено експерименти з використанням реальних наборів CIFAR-10 для оцінки ефективності запропонованих підходів та методів у реальних умовах.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасний стан та тенденції розвитку моделей штучного інтелекту

Останні кілька десятиліть спостерігається революційний прогрес в області штучного інтелекту (далі - ШІ), який вніс зміни в більшість аспектів нашого життя, від повсякденних речей, таких, як наприклад, спілкування, до складних, промислових процесів. Розвиток у цій галузі, зумовлений швидким прогресом технологічних нововведень та впровадженням інноваційних рішень, яке теж зростає, в різних галузях. Це, у свою чергу, призвело до виникнення нових трендів та викликів, на чому формується сучасний ШІ.

Особливе місце серед ключових аспектів займає використання цієї технології в кібербезпеці. Це включає в себе залучення інтелектуальних алгоритмів для покращення здібностей кіберзахисту, зокрема у виявленні відхилень і в аналізі ризиків. ШІ має здібність навчатися на основі певних даних та може бути використаним у розробці фінальних систем ШІ, які істотно поліпшують захист інформації [1].

Особливо висвітлюється розвиток рішень, які складаються з детального аналізу намірів та методів, які кіберзлочинці застосовують для проведення атак, які в свою чергу, мають свої особливості, які залежать від мети, задля чого і відбувається атака. Прикладом може бути те, що стратегія атаки на банківську систему, буде відрізнятись від атак на соціальні мережі.

Інші технології, які використовуються в ШІ для кіберзахисту, включають аналіз соціальних мереж для раннього виявлення ознак кібератак, використання автоматичних систем для прийняття рішень і, також, створення складних систем, які дозволяють своєчасно реагувати на загрози в кіберпросторі [1].

Отже, ШІ відіграє ключову роль у посиленні кіберзахисту, забезпечуючи ефективнішу реакцію на кіберзагрози, збільшуючи швидкість і точність виявлення та реакцію на атаки.

Також, не менш важливим аспектом у цій галузі є створення надійних, зрозумілих і етичних систем ШІ [1]. Сучасні дослідження акцентують увагу на тому, що ШІ має бути надійним, зосереджуватись на обґрунтованості, перевірці та валідації розроблених можливостей і методів забезпечення для боротьби з атаками зловмисників, такими як “отруєння”, обхід та інверсія моделі. Також важливо забезпечити безпеку систем ШІ від цих загроз, а це потребує постійної роботи над покращенням та оптимізації алгоритмів і підходів до їх розробки.

Особлива увага приділяється підвищенню зрозумілості та пояснюваності рішень, що приймаються системами ШІ. Важливо, щоб користувачі могли довіряти цим системам і правильно розуміти їхні висновки. З цією метою дослідники розробляють нові підходи та інструменти, які дозволяють нам краще розуміти та тримати контроль над процесами, що відбуваються всередині нейронних мереж [1].

Оскільки існує тенденція більшої та інтенсивнішої інтеграції технології в наше суспільство, так само зростає потреба в розробці етичних принципів і правил їх використання. Це означає не лише створення технологій, які поважають права та свободи людини, а й розвиток механізмів контролю та відповідальності за дії, що здійснюються за допомогою ШІ.

У сфері штучного інтелекту стратегічне інвестиційне планування є ключовим фактором, який визначає напрямок розвитку та впровадження новітніх технологій. Існує три основні інвестиційні горизонти ШІ, кожен з яких відкриває унікальні можливості та ставить перед дослідниками та розробниками нові виклики [1]. Ці горизонти допомагають краще зрозуміти, як інвестиції в наукові дослідження та технологічні інновації можуть сформувати майбутнє ШІ та його застосування в різних сферах. Перший горизонт спрямований на досягнення надійного змістовного аналізу, який включає застосування ШІ для виявлення цінної інформації у великих обсягах даних, різноманітного формату. Другий зосереджений на розвитку співпраці

між людьми та машинами, використовуючи ШІ для покращення когнітивних можливостей людини. Третій пропонує ввести контекстне розуміння в системи штучного інтелекту, дозволяючи машинам краще пристосуватись до проблемних кейсів і забезпечувати високу надійність та впевненість у своїх рекомендаціях.

Сучасний стан розвитку моделей ШІ характеризується динамічним зростанням та інноваційними проривами. Останнім часом у сфері ШІ спостерігається тенденція до збільшення розмірів нейронних мереж, що пов’язано з бажанням підвищити точність і функціональність систем. Цей розвиток супроводжується великою різноманітністю архітектур, від глибоких згорткових нейронних мереж (CNN), які стали стандартом у візуальному розпізнаванні, до складних рекурентних нейронних мереж (RNN) і трансформаційних моделей (таких як GPT-3), які показують незрівнянні результати обробки природної мови [2].

Актуальність цієї проблеми посилюється стрімким розвитком глибокого навчання та його застосуванням у різних сферах, таких як комп’ютерний зір і обробка природної мови. Однак ключовим обмеженням для широкого впровадження цих технологій у критично важливих для безпеки системах є їхня надійність. Однією з основних проблем при розробці нейронних мереж є їхня висока чутливість до збурень у вхідних даних. Це ставить під сумнів здатність цих систем забезпечувати стабільну та надійну роботу в реальних умовах. Другою важливою проблемою є складність виправлення помилок у моделях без ризику введення нових. Це підкреслює необхідність розробки більш надійних і відмовостійких систем [2].

Розміри мереж не тільки зростають, але й середовища розгортання стають більш різноманітними та спеціалізованими. Хоча моделі AI традиційно розгорталися в основному на центральних процесорах (CPU), тепер вони поступово переходять на графічні процесори (GPU), які забезпечують значно кращу продуктивність завдяки паралельній обробці даних. Графічні процесори

забезпечують прискорення обчислень завдяки великій кількості ядер, які можуть виконувати операції з плаваючою комою паралельно, що ідеально підходить для навчання та використання нейронних мереж. Графічні процесори використовують архітектуру SIMD (одна інструкція, кілька даних), яка дозволяє одній інструкції виконувати ту саму операцію над багатьма елементами даних одночасно. ЦП, з іншого боку, використовують більш універсальну архітектуру, яка не так ефективно працює з обчисленнями, типовими для навчальних моделей. Графічні процесори мають високу пропускну здатність пам’яті, що дозволяє швидко передавати великі обсяги даних між пам’яттю та обчислювальними ядрами. Це важливо для ефективного навчання нейронних мереж, де потрібно часто оновлювати велику кількість параметрів [3].

Тенденція до спеціалізації також призвела до розробки блоків обробки тензорів (TPU), які оптимізовані для високошвидкісного виконання операцій, характерних для машинного навчання, зокрема тензорних обчислень. TPU забезпечують значне прискорення висновків і навчання моделей штучного інтелекту порівняно з традиційними процесорами та графічними процесорами. Для графічних процесорів розроблено оптимізовані бібліотеки, наприклад CUDA для NVIDIA, які забезпечують додаткове прискорення навчання шляхом оптимізації низькорівневих операцій. ТПУ розроблені для максимальної ефективності роботи. Вони споживають менше енергії на операцію порівняно з процесорами та графічними процесорами, що робить їх економічно ефективними в масштабі. Оскільки TPU розроблено компанією Google, вони тісно інтегровані з сервісами Google Cloud, забезпечуючи легкий доступ до обчислювальної потужності для організацій, які можуть не мати власної потужної інфраструктури. TPU мають вбудовані інструменти для автоматизації операцій оптимізації, таких як квантування та упаковка даних, що спрощує розробку та реалізацію моделі [3].

Крім того, програмовані логічні інтегральні схеми (FPGA) набувають популярності завдяки своїй гнучкості та енергоефективності. Вони дозволяють налаштовувати апаратне забезпечення спеціально для завдань ШІ, що може призвести до підвищення продуктивності в певних застосунках [4].

У сукупності ці розробки відкривають нові можливості для створення розумніших і ефективніших систем ШІ. Однак вони також ставлять перед дослідниками та інженерами нові виклики, пов’язані з підвищенням стійкості цих систем до збоїв і помилок, які можуть виникнути на будь-якому етапі роботи системи, від процесу навчання до розгортання та експлуатації.

1.2 Типи збоїв і методи їх інжекції в середовище розгортання нейронних мереж

Моделі штучних нейронних мереж привернули значний інтерес дослідників і відновили зростання додатків, пов’язаних зі штучним інтелектом, таких як глибоке навчання. Однак, окрім оптимізації архітектури для високої загальної продуктивності, важливим аспектом дослідження нейронних мереж є їх стійкість до неточностей, невизначеностей і збоїв.

Людський мозок, згідно з нейробіологічними дослідженнями, здатний терпіти невелику кількість помилок у синапсах або нейронах і навіть використовувати шум як джерело обчислень. З цих спостережень випливає, що більшість моделей нейронних мереж, абстрагованих від біологічних, мають вбудовані або внутрішні властивості відмовостійкості. Однак таку еквівалентну відмовостійкість не можна стверджувати лише на підставі грубої архітектурної схожості, особливо для невеликих нейронних мереж [5].

Нейронні мережі виявляють стійкість до зашумлених вхідних даних і витончену деградацію завдяки своїй стійкості до неточних обчислень, коли вони реалізовані на фізичному субстраті. Стійкість до близькості може бути використана для значного підвищення продуктивності та енергоефективності шляхом розробки спеціалізованих нейронних прискорювачів низької точності, які працюють на потоках даних датчиків.

Однак на практиці нейронна мережа має дуже обмежену відмовостійкість і не може вважатися внутрішньо відмовостійкою без відповідного дизайну. Крім того, оскільки обчислення та інформація природним чином розподіляються в нейронних мережах, традиційні методи обмеження помилок і реплікації, ключові до звичайних рішень з відмовостійкістю, не можуть бути безпосередньо застосовані для обмеження розповсюдження помилок при реалізації на потенційно несправних субстратах.

У відмовостійких системах є три фундаментальні поняття: несправність, помилка та збій. (див. Рис. 1.1) Між ними існує причинно-наслідковий зв'язок, який переходить від фізичного рівня до рівня поведінки, як показано на прикладі нейронної мережі, яка виконує обчислювальні завдання та реалізована на цифровій підкладці.

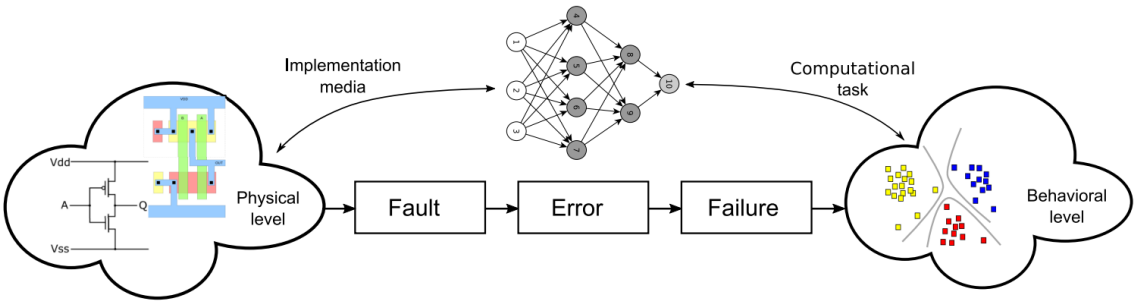

Рисунок 1.1 - Причинно-наслідковий зв'язок між несправністю, помилкою та невдачею та його поширення від рівня фізичної реалізації до поведінкового прикладний рівень моделі нейронної мережі [5]

Несправність — це ненормальний фізичний стан у системі, що призводить до помилок. Помилка - це прояв збою в системі, відхилення від очікуваного результату, коли логічний стан елемента відрізняється від його прогнозованого значення. Збій означає неспроможність системи виконувати

заплановані функції або поведінку через помилки в її елементах або порушення в оточенні [5]

Розповсюдження помилки на системний рівень спричиняє збій системи, однак не завжди призводить до помилки, оскільки вона може залишатися неактивною. Активною несправністю вважається та, яка викликає помилку; інакше її називають сплячим.

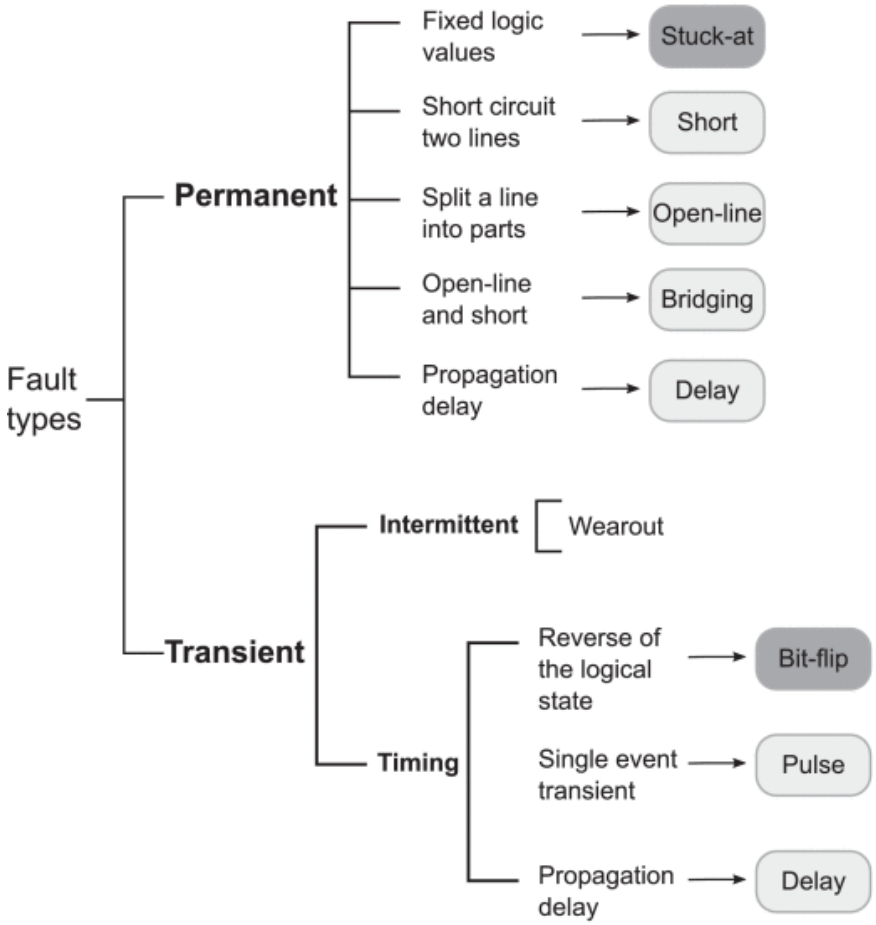

Рисунок 1.2 – Типи несправностей із деякими типовими причинами та механізмами постійних і тимчасових несправностей, а також відповідні моделі несправностей в сірих округлих рамках [5]

За часовими характеристиками несправності можна класифікувати наступним чином, як показано на Рис. 1.2:

- постійна несправність є стійкою та стабільною протягом тривалого часу; це в основному результат незворотної фізичної шкоди,

- тимчасова несправність може існувати лише протягом короткого періоду часу і часто є результатом зовнішніх збурень [5]

Тимчасові збої, що повторюються з певною частотою, називають періодичними. Зазвичай такий збій виникає через нестабільну роботу пристрою, і його важче виявити, ніж постійні. Короткочасні та періодичні збої становлять більшу частину збоїв, які виникають у цифрових обчислювальних системах, побудованих із застосуванням сучасних напівпровідникових технологій.

Можливо і майбутні технології впровадження страждатимуть від проблем, які є результатом тимчасових збоїв через зниження якості пристрою, високий рівень мінливості процесу, обставини та оточення, а також істотне зниження продуктивності обумовлене високим навантаженням. Часові збої змінюють часову поведінку, а не структуру схем; впливаючи на параметри схем, які визначають таймінгові характеристики пристрою, такі як затримка поширення, час утримання та налаштування [6].

Розуміння цих помилок є ключовим для розробки ефективних стратегій їх управління та запобігання, то ж розглянемо методи введення цих помилок в середовище розгортання нейромереж.

Методи введення помилок у середовище розгортання нейронних мереж, зокрема для підвищення відмовостійкості, можна розділити на дві основні категорії: явне збільшення надлишковості та модифікація навчання/тренування [6].

Явне збільшення надлишковості: цей підхід передбачає додавання надмірності до вже натренованої нейронної мережі, приділяючи увагу прихованим нейронам та їх з'єднанням у мережах. Процес починається з базової мережі, яка навчається, щоб виконати певне завдання, а потім додається надлишковість після того, як навчання буде завершеним.

Використовувані методи включають доповнення мережі: це включає розмноження критичних нейронів або рівномірний розподіл синаптичної ваги разом з відокремленням нейронів і видаленням зайвих ваг.

Одна з репрезентативних робіт належить Чу і Ва: вони використали гібридну схему надлишковості, що включає в себе просторову, тимчасову та інформаційну надмірності, спрямовуючи увагу на критичні нейрони, особливо на виходах мережі. Такий підхід спрямований на усунення тимчасових, періодичних і постійних несправностей [6].

Також, Еммерсон і Демпер досліджували відмовостійкість у MLP (багатошарових сприйняттях) для задач ідентифікації образів, запропонувавши механізм розмноження для кожного прихованого нейрона та його з’єднань (посилення) [6].

Чіу та ін. виміряли чутливість каналів і вузлів у вихідній мережі таким чином: видалили неважливі вузли та ввели надлишкові, щоб розділити навантаження на критичні вузли.

Пхатак і Корен: їхній метод передбачав розмноження прихованих одиниць у нейронних мережах прямого зв’язку з одним прихованим шаром для усунення постійних несправностей типу «зависання» [6].

Діас і Антунес запропонували техніку підвищення відмовостійкості шляхом зміни архітектури мережі після навчання, дублюючи чутливі елементи: входи, зміщення, ваги або нейрони [6].

Модифікація навчання/тренування містить в собі зміну традиційних схем навчання для забезпечення відмовостійкості. Це можна зробити шляхом додавання шуму чи ін’єкції помилок під час навчання або шляхом включення термінів регуляризації/штрафів у функцію витрат навчання (основна мета цієї функції – виміряти ефективність, а тобто визначити, наскільки модель відповідає навчальним даним і які витрати, тобто помилки, пов'язані з поточними налаштуваннями) [6].

Загалом методи підвищення надлишковості можуть бути ефективними, але нерідко призводять до створення великих мереж із багатьма прихованими вузлами та параметрами. Таким чином, обрізка має ключове значення для ідентифікації та усунення надлишкових одиниць. На відміну від звичайних підходів до надлишковості (таких як потрійна модульна надмірність), ці методи не використовують голосування більшістю для маскування недоліків. Замість цього вони використовують внутрішні характеристики нейронних мереж: зважене підсумовування та робота вузлів прихованого шару поблизу їх точок насичення.

1.3 Формалізована постановка задачі

Дано навчальну і тестову вибірку CIFAR-10, також дано набір архітектур нейронних мереж Resnet-18, Resnet-50, MobileNet. Дано спосіб формування синтетичних збоїв в нейронну мережу для симуляції атак на обчислювальне середовище розготання нейронної мережі. Необхідно вибрати з яких моделей будувати ансамбль, який метод навчання використовувати для забезпечення максимальної стійкості до інжекції несправностей різного рівня. Нехай R показник стійкості до апаратно-програмних збоїв/несправностей заданого рівня, який потрібно максимізувати R=1 - (ACC-ACC_inject)/ACC.

2 МОДЕЛІ І МЕТОДИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДВИЩЕННЯ СТІЙКОСТІ ДО ПРОГРАМНО-АПАРАТНИХ ПОМИЛОК

2.1 Модель класифікатора зображень з підвищеною стійкістю до збоїв

Модель класифікатора зображень – багатоверсійна глибока нейронна мережа NV-DNN (N-Version Deep Neural Network) (див. Рис.2.1) розроблена для підвищення відмовостійкості в системах глибокого навчання.

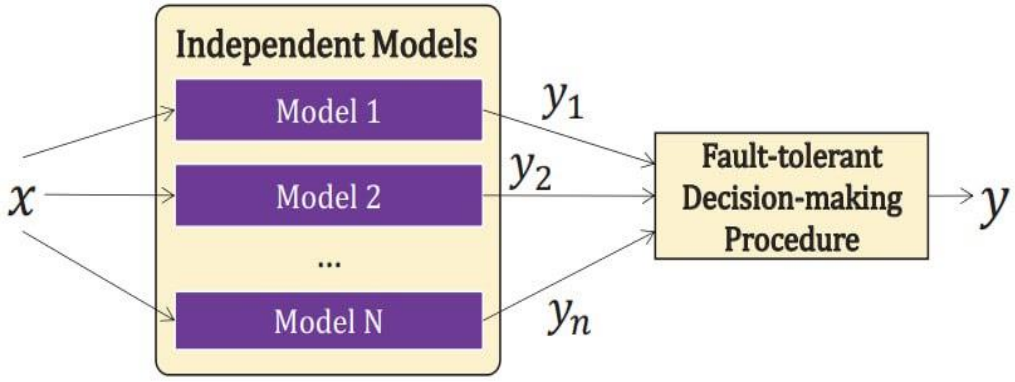

Рисунок 2.1 – Концептуальний фреймворк NV-DNN [7]

Модель NV-DNN використовує принцип надмірності моделі, натхненний класичною парадигмою N-версійного програмування (NVP). Він полягає у створенні кількох незалежних версій моделей глибокого навчання, кожна з яких навчається незалежно, що знижує ймовірність виникнення однакових помилок у різних моделях за однакових умов введення. У процесі розробки NV-DNN використовуються різні фактори незалежності, такі як незалежне навчання, незалежна архітектура мережі та незалежні дані. Незалежне навчання передбачає введення випадковості в процес навчання, а незалежна архітектура мережі дозволяє різним моделям вивчати різні функції. Незалежні дані передбачають використання різних наборів даних для навчання кожної

моделі, що забезпечує різноманітність завдань оптимізації. Модель NV-DNN використовує процедуру прийняття рішень, яка обчислює кінцевий результат на основі виходів кожної з моделей. Простий варіант такої процедури може бути заснований на голосуванні: підсумовуються ймовірності кожної мітки, заданої моделями, і в якості кінцевого результату вибирається мітка з максимальною ймовірністю. Також NV-DNN як відмовостійке рішення може використовувати більш складні процедури прийняття рішень для відмовостійкості, наприклад, за допомогою механізму відмовостійкості. Таким чином, модель класифікатора зображень NV-DNN представляє інноваційний підхід до створення надійних систем глибокого навчання, що забезпечує високу надійність і точність розпізнавання зображень навіть за наявності потенційних апаратних і програмних збоїв [7].

NV-DNN можна розглядати як ансамбль моделей. Концепція ансамблю в машинному навчанні полягає у використанні кількох навчальних моделей для досягнення кращих результатів на відміну від результатів з моделями окремо. NV-DNN включає декілька моделей нейронних мереж, кожна з яких вчиться окремо, але спільно використовується для підвищення загальної точності та надійності системи. Кожна модель може мати власну архітектуру, тренувальні дані та ініціалізацію параметрів, що робить їх незалежними. Результати кожної моделі об'єднуються, наприклад, за допомогою процедури голосування, для отримання кінцевого рішення класифікації - це підвищує загальну стійкість до збоїв, які можуть бути у будь-якій окремій модел [7]і.

Голосування більшістю (MajVot або majority voting) за прогнозами є одним із найпопулярніших методів підвищення ефективності класифікації. Він полягає в тому, що кожна модель в ансамблі «голосує» за певний клас. Клас, який має найбільшу кількість голосів, вважається кінцевим прогнозом ансамблю.

Наукові дослідження показують, що використання голосування більшістю може суттєво збільшити точність моделей. Наприклад, у

дослідженні класифікації ожиріння використання гібридної моделі, де було застосовано голосування більшістю, досягло точності 97,16%, що є кращим результатом, ніж у окремих моделях [8].

Також, було проведено дослідження використання методу голосування більшості для оптимізації загальної продуктивності класифікації пухлин головного мозку. Результати показали, що застосування даного алгоритму на основі глибокого навчання призвело до підвищення середньої точності чотирьох наборів даних на 2,02%, 1,11%, 1,04%, 2,67% і 1,65% відповідно для AlexNet, VGG16, моделі ResNet18, GoogleNet і ResNet50. Зокрема, запропонований алгоритм MajVot на основі машинного навчання, провалідовано на імітованих даних зображення обличчя, що збільшило точність класифікації зображень обличь по гендерним ознакам на 2,88%, 0,71%, 1,90%, 2,24% і 0,35% порівняно з AlexNet, VGG16, ResNet18 , GoogleNet і ResNet50. відповідно. Підсумовуючи, дослідники зауважують, що даний алгоритм показує результати, які показують надійність для класифікації, демонструючи здатність використовувати комбінований потенціал кількох моделей [9].

Дослідження демонструють, що даний спосіб можна використовувати для збільшення ефективності ансамблів та їхнього відсотку точності для таких важливих завдань у медичній сфері, де точність є суттєвим фактором.

2.2 Метод навчання нейронних мереж

Метод навчання нейронних мереж є ключовим компонентом у розробці ефективних систем штучного інтелекту. Основним методом навчання більшості нейронних мереж є зворотне поширення. Окрім цього, існують також інноваційні підходи, такі як додавання шуму до градієнта, які можуть покращити стійкість мережі до збоїв. Зворотне поширення помилок є стандартним методом для навчання нейронних мереж [10].

Метод включає в себе два основні кроки: пряме та зворотнє поширення. У першому випадку, вхідні дані проходять через мережу, потім формується

прогноз. Далі в алгоритмі зворотного поширення, вираховується градієнт функції втрати, і цей градієнт використовується для оновлення ваг в мережі для того, щоб зменшити цю втрату. Зворотнє поширення враховує ефект кожного параметра на помилку і залежності між параметрами, що дає змогу налаштовувати ваги для оптимального навчання.

Додавання шуму до градієнтів під час навчання може збільшити стійкість нейронних мереж до збоїв. Цей підхід базується на ідеї, що шум може допомогти мережі навчитися ігнорувати невеликі варіації або перешкоди, підвищуючи її загальну відмовостійкість. Процес виглядає наступним чином: під час кроку зворотного поширення помилки випадковий шум додається до градієнтів, розрахованих для кожної ваги. Цей шум може бути створений з різних розподілів, наприклад, нормальних або однорідних. Шум дозволяє мережі «навчатися» в умовах нестабільності, що збільшує її здатність адаптуватися до помилок або збоїв у реальних умовах [10].

2.3 Критерії оцінювання ефективності нейронних мереж та їх стійкості до збоїв

Точність (accuracy) і втрати (loss) - основні критерії для оцінки ефективності роботи нейронних мереж у прогнозуванні результатів на основі вхідних даних. Точність відображає відсоток правильних передбачень, тоді як втрати показують рівень відхилення передбачень від фактичних значень. Ці показники можуть бути використані для зіставлення різних архітектур нейронних мереж, налаштувань гіперпараметрів і методів навчання на конкретних даних [11].

Оцінюючи нейронні мережі за цими критеріями, можна отримати комплексне розуміння їх ефективності та надійності, що є ключовим для їх використання в критично важливих та високонавантажених областях застосування.

Набори для валідації та тестування є різними частинами даних, які використовуються не для тренування нейронної мережі, а для перевірки її

здатності до узагальнення. Набори для валідації використовуються для налаштування параметрів та вибору оптимальної моделі, тоді як набори для тестування використовуються для того, щоб оцінити остаточну ефективність та уникнути перенавчання. Слід завжди розділяти дані на тренувальні, перевірочні та тестові набори перед експлуатацією нейронної мережі [11].

Один з ключових критеріїв для оцінювання ефективності нейронних мереж, особливо в контексті їх стійкості до апаратно-програмних збоїв/несправностей, є показник R.

Показник стійкості R розраховується за формулою (2.1)

$$R = 1 - \frac{\mathrm{ACC} - \mathrm{ACC}_{\mathrm{inject}}}{\mathrm{ACC}}, \qquad (2.1)$$

де ACC − це точність моделі без внесення збоїв,

$\mathrm{ACC}_{\mathrm{inject}}$ − точність моделі після імітації збоїв.

R означає, що вплив збоїв на точність моделі мінімальний. Таким чином, максимізація R є ключовою метою при розробці стійких нейронних мереж.R дозволяє оцінити, наскільки сильно збої впливають на ефективність моделі. Чим менший падіння у точності моделі після введення збоїв тим вищий показник R і, відповідно, краща стійкість моделі.

Низьке значення R вказує на те, що модель значно вразлива до збоїв, тобто її продуктивність сильно падає при наявності збоїв. Це може бути важливим індикатором для подальшого аналізу та оптимізації моделі.

R може бути застосований для оцінки різних аспектів стійкості нейронних мереж, таких як:

- вплив апаратних збоїв, наприклад, помилок у пам'яті або процесорах,
- вплив програмних збоїв, таких як помилки у програмному забезпеченні або внесення збурень у дані. У експериментальних дослідженнях показник R може бути використаний для оцінки та порівняння різних підходів до підвищення стійкості, дозволяючи визначити, які методи є

найефективнішими для зниження впливу збоїв на продуктивність нейронних мереж.

Стійкість (Robustness) нейронних мереж означає здатність моделі ефективно функціонувати та підтримувати високу точність прогнозування навіть за наявності шуму, змінних вхідних умов або спроб ввести в оману. Це ключовий аспект при оцінці надійності та практичності використання нейронних мереж у реальних умовах. Стійкість до шуму має відношення до здатності моделі правильно обробляти вхідні дані, які можуть бути з шумом або з випадковими відхиленнями. Для зорових системах це правильне визначення об’єктів на зображеннях із низькою якістю або з різним степенем освітлення [11].

Стійкість до мінливих умов означає змогу нейронної системи відповідним чином реагувати на зміни в умовах даних, наприклад, на нові варіанти набору даних. Стійкість до атак обумовлена використанням нейронних мереж в кібербезпеці. Надійність також відноситься до здатності моделі протистояти спробам фальсифікації, спричиненій шкідливими змінами у вхідних даних для помилкових передбачень.

Розробка надійних нейронних мереж має на меті використання різних методів, а саме скрупульозну підготовку даних, регуляризацію моделі, використання стратегії доповнення даних і імплементація механізмів детекції відхилень. Не менш важливим є ретельне тестування та валідація моделі за різних умов, для засвідчення її стійкості до потенційних труднощів.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДВИЩЕННЯ СТІЙКОСТІ ДО ПРОГРАМНО-АПАРАТНИХ ПОМИЛОК

3.1 Формування навчальних та тестових даних

Формування навчальних і тестових даних є ключовим етапом у використанні набору даних CIFAR-10 (див. Рис. 3.1), який є одним із найпопулярніших датасетів у сфері машинного навчання та комп'ютерного зору. CIFAR-10 — це набір даних, який містить 60 000 кольорових зображень розміром 32x32 пікселя, розділених на 10 класів по 6 000 зображень на клас.

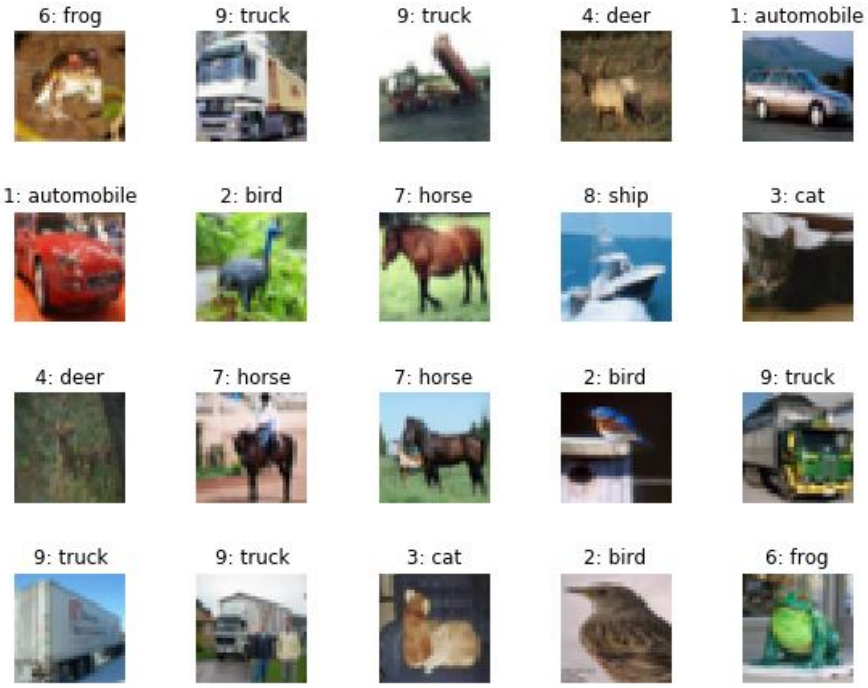

Рисунок 3.1 – Приклад зображень датасету CIFAR-10 [12]

Навчальний набір складається з 50 000 зображень. Ці зображення використовуються для навчання моделі. Кожне зображення містить ярлик однієї з 10 категорій, до яких належать автомобілі, птахи, коти, собаки, коні, літаки, вантажівки, олені, жаби та кораблі [12].

Тестовий набір містить 10 000 зображень, які використовуються для оцінки ефективності навченої моделі. Ці зображення мають бути відокремлені

від навчального набору для забезпечення об’єктивності оцінювання. Тестові дані дають можливість визначити, наскільки добре модель узагальнює нові дані, які не використовувалися під час навчання [12].

Доповнення або аугментація даних — це техніка, яка використовується для створення додаткових навчальних даних із наявного набору шляхом внесення змін до зображень, таких як обертання, зміна масштабу, обрізка, тощо. Цей підхід допомагає запобігти перетренуванню та вдосконалює здатність моделі до узагальнення.

CIFAR-10 часто використовується в наукових дослідженнях і освітніх проєктах як стандартний набір даних для демонстрації та тестування алгоритмів глибокого навчання.

Формування та використання навчальних і тестових даних CIFAR-10 є фундаментальними для успішної розробки та перевірки моделей глибокого навчання, гарантуючи точність та надійність у розпізнаванні зображень.

3.2 Короткий опис програмного забезпечення

Написання програмного коду здійснювалось в Jupyter Notebook та Google colab мовою програмування python3. Дане програмне забезпечення потребує використання сторонніх бібліотек, таких як:

- ядро Torch базується на тензорних операціях, що дозволяє ефективно обробляти великі обсяги даних. Тензори в Torch мають схожості з масивами NumPy, проте додатково підтримують виконання операцій на графічних процесорах (GPU).
- Torchvision є доповненням до Torch, спеціалізованим на обробці зображень та комп'ютерному зорі.
- Torchvision.models. Однією з ключових опцій torchvision.models є наявність багатьох попередньо натренованих моделей. Ці моделі були навчені на великих та різноманітних датасетах, таких як ImageNet, і можуть бути застосовані безпосередньо, та в якості основи для навчання у конкретних задачах.

- NumPy (Numerical Python) – основна бібліотека для наукових обчислень у Python. Вона надає змогу використовувати масивні багатовимірні об'єкти масивів та інструменти для роботи з цими масивами. NumPy оптимізований для високої продуктивності операцій для обробки великих масивів даних.

- Scipy.stats.mode в Python використовується для використання функції mode з модуля scipy.stats, який є частиною бібліотеки SciPy. Ця функція виконує визначення моди в даних. Дана функція використовується для голосування за прогнозами.

- Transforms є модулем у бібліотеці torchvision, який містить в собі інструменти для перетворень зображень, які застосовуються при обробки даних та підготовки до тренування моделей глибокого навчання. Цей модуль є суттєвим в процесі попередньої обробки зображень.

- Torch.nn – ключовий модуль у бібліотеці PyTorch, який забезпечує базу для розробки нейронних мереж. Для тренування моделей, torch.nn включає функції втрат, такі як Mean Squared Error, Cross-Entropy Loss, які використовують для оцінки розбіжності між передбаченнями моделі та фактичними даними.

- Torch.optim є важливим модулем у бібліотеці PyTorch, що дозволяє використовувати алгоритми оптимізації для тренування нейронних мереж., а саме для оновлення ваг моделі на основі виведених градієнтів в процесі навчання. Оптимізатори в torch.optim дозволяють налаштування гіперпараметрів, таких як швидкість навчання (learning rate), момент (momentum), і вага регуляризації, що дає можливість гнучко підлаштувати процес навчання під певні задачі. Ефективних оптимізаторів покращують швидкість збіжності тренувального процесу та підвищити загальну ефективність нейронної мережі.

- PyTorch — одна з ведучих бібліотек машинного та глибокого навчання. Це програмне забезпечення з відкритим кодом, яке використовується

для створення, навчання та розгортання моделей ШІ. PyTorch має різноманітні інструменти і бібліотеки, такі як TorchText, TorchVision і TorchAudio, які зводять до мінімуму складність завдань, які мають зв’язок з обробкою тексту, зображень і аудіо. Зокерма, він підтримує багато плагінів та розширень, які дозволяють впроваджувати PyTorch з різними інструментами та платформами. Через свою продуктивність та гнучкість PyTorch знаходить широке застосування у наукових дослідженнях і промислової галузі. Це популярний засіб серед дослідників для експериментів із ШІ в силу своєї простоти конфігурації та застосування, а також спроможності легко впроваджуватись з іншими API та бібліотеками [13].

Ін’єкції були виконані за допомогою функцій фреймворка PyTorchFI, який є відкритим програмним фреймворком, розробленим для введення помилок (fault injection) у нейронні мережі, які створені за допомогою популярної бібліотеки PyTorch для машинного навчання та глибокого навчання.

PyTorchFI розроблений з метою дозволити дослідникам та розробникам аналізувати та визначати вразливість нейронних мереж до різних типів помилок, таких як помилки у пам'яті, процесорах або даних. Фреймворк дозволяє вводити помилки безпосередньо під час виконання нейронної мережі. Це може включати зміну ваг, активацій або виходів шарів мережі, що дозволяє досліджувати вплив таких змін на загальну продуктивність мережі. PyTorchFI надає користувачам можливість налаштувати типи та характеристики помилок, які потрібно ввести, забезпечуючи велику гнучкість для експериментів. Це включає можливість вибору конкретних шарів або нейронів для введення помилок [14].

Фреймворк може бути використаний з різноманітними типами архітектур нейронних мереж, що робить його корисним для широкого спектру застосувань у галузі машинного та глибокого навчання.

PyTorchFI є важливим інструментом для дослідження стійкості нейронних мереж до помилок, дозволяючи розробникам визначити, які частини мережі є найбільш вразливими та як ці збої можуть вплинути на загальну роботу системи. Використання PyTorchFI може допомогти у покращенні розуміння та підвищенні надійності нейронних мереж, особливо у критичних застосуваннях, де помилки можуть мати серйозні наслідки.

Експерименти проводились за допомогою трьох моделей pytorch: resnet18, resnet50, mobilenet. Усі ці моделі були натреновані та зібрані в ансамбль задля підвищення стійкості системи до програмно-апаратних помилок.

Були створені та використані в роботі допоміжні функції (див. Таб. 3.1).

Таблиця 3.2 – Допоміжні функції та їх призначення

Назва функції	Призначення
evaluate_accuracy	Вираховує точність моделі
train_model	Тренує модель
get_predictions	Повертає прогноз моделі
evaluate_ensemble_acKrcuracy	Вираховує точність ансамблю
vote	Голосування по прогнозах

Спочатку було передоброблено та завантажено дані CIFAR-10. Після формування тестових та навчальних даних було визначено моделі resnet18, resnet50, mobilenet. Обов’язковим етапом є визначення оптимізатору Adam та функції втрат CrossEntropyLoss для тренування моделей, яке виконано методом train_model. Після тренування моделей відбувається ін’єкція помилок в певний шар з певними параметрами. Далі формується два ансамблі з кожної архітектури моделей до ін’єкції та після для вирахування точності та показника стійкості. Також виконано прогнозування кожної моделі методом get_predictions після ін’єкцій для голосування по прогнозам.

3.3 Постановка та аналіз результатів експериментів

У ході виконання роботи було створені по три екземпляри кожної з моделей resnet18, resnet50, mobilenet. Тренування моделей виконувалось методом на аугментованих вхідних даних датасету CIFAR-10, що максимізує точність моделей (див. Рис. 3.2) та зменшує час тренування.

```
Epoch 1/10, Accuracy: 69.82%
Epoch 2/10, Accuracy: 79.17%
Epoch 3/10, Accuracy: 82.14%
Epoch 4/10, Accuracy: 84.85%
Epoch 5/10, Accuracy: 86.55%
Epoch 6/10, Accuracy: 88.39%
Epoch 7/10, Accuracy: 89.58%
Epoch 8/10, Accuracy: 90.86%
Epoch 9/10, Accuracy: 91.04%
Epoch 10/10, Accuracy: 92.15%
Finished Training
Epoch 1/10, Accuracy: 69.73%
Epoch 2/10, Accuracy: 79.17%
Epoch 3/10, Accuracy: 82.84%
Epoch 4/10, Accuracy: 84.89%
Epoch 5/10, Accuracy: 86.91%
Epoch 6/10, Accuracy: 88.41%
Epoch 7/10, Accuracy: 89.59%
Epoch 8/10, Accuracy: 90.52%
Epoch 9/10, Accuracy: 91.91%
Epoch 10/10, Accuracy: 92.55%
Finished Training
Epoch 1/10, Accuracy: 69.82%
Epoch 2/10, Accuracy: 79.24%
Epoch 3/10, Accuracy: 82.78%
Epoch 4/10, Accuracy: 84.88%
Epoch 5/10, Accuracy: 86.55%
Epoch 6/10, Accuracy: 88.47%
Epoch 7/10, Accuracy: 89.62%
Epoch 8/10, Accuracy: 90.86%
Epoch 9/10, Accuracy: 91.66%
Epoch 10/10, Accuracy: 92.42%
Finished Training
```

Рисунок 3.2 – Результати тренування моделей з відповідними точностями

Експерименти будуть проводитись ін’єктуванням різних помилок, з подальшим голосування за прогнозами зіпсованих моделей та обчислення показника стійкості до помилок.

3.3.1 Експеримент 1.

У якості тестових даних використовується 3 клас. Дано 3 натреновані моделі resnet18 (див. Рис. 3.3)

```
Epoch 1/10, Accuracy: 69.82%
Epoch 2/10, Accuracy: 79.17%
Epoch 3/10, Accuracy: 82.14%
Epoch 4/10, Accuracy: 84.85%
Epoch 5/10, Accuracy: 86.55%
Epoch 6/10, Accuracy: 88.39%
Epoch 7/10, Accuracy: 89.58%
Epoch 8/10, Accuracy: 90.86%
Epoch 9/10, Accuracy: 91.04%
Epoch 10/10, Accuracy: 92.15%
Finished Training
Epoch 1/10, Accuracy: 69.73%
Epoch 2/10, Accuracy: 79.17%
Epoch 3/10, Accuracy: 82.84%
Epoch 4/10, Accuracy: 84.89%
Epoch 5/10, Accuracy: 86.91%
Epoch 6/10, Accuracy: 88.41%
Epoch 7/10, Accuracy: 89.59%
Epoch 8/10, Accuracy: 90.52%
Epoch 9/10, Accuracy: 91.91%
Epoch 10/10, Accuracy: 92.55%
Finished Training
Epoch 1/10, Accuracy: 69.82%
Epoch 2/10, Accuracy: 79.24%
Epoch 3/10, Accuracy: 82.78%
Epoch 4/10, Accuracy: 84.88%
Epoch 5/10, Accuracy: 86.55%
Epoch 6/10, Accuracy: 88.47%
Epoch 7/10, Accuracy: 89.62%
Epoch 8/10, Accuracy: 90.86%
Epoch 9/10, Accuracy: 91.66%
Epoch 10/10, Accuracy: 92.42%
Finished Training
```

Рисунок 3.3 – Результати тренування моделей resnet18 з відповідними точностями

Ці моделі мають таку точність на тестових даних до ін’єкцій помилок, як зображено на Рисунку 3.4.

```
[13] accuracy1 = evaluate_accuracy(resnet18_1, test_loader, device)
     accuracy2 = evaluate_accuracy(resnet18_2, test_loader, device)
     accuracy3 = evaluate_accuracy(resnet18_3, test_loader, device)
```

13 сек.

```
print(f'Test Accuracy: {accuracy1}%')
print(f'Test Accuracy: {accuracy2}%')
print(f'Test Accuracy: {accuracy3}%')
```

0 сек.

```
Test Accuracy: 83.57%
Test Accuracy: 83.31%
Test Accuracy: 83.14%
```

Рисунок 3.4 – Обчислення точностей моделей resnet18

Виконуємо ін’єкцію функцією random_weight_inj, яка ін’єктує помилки у ваги (вибирає випадкове місце в вагах) в 3 шар та параметрами min_val=20, max_val=30 в кожну з моделей. Отримуємо прогнози кожної моделі та голосуємо по них (див. Рис. 3.5).

```
fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3, corrupt_layer=3, min_val=20, max_val=30
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2, predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.5 - Ін’єкція функцією random_weight_inj в 3 шар resnet18 та параметрами min_val=20, max_val=30

Отримуємо у відповіді 3 клас та створюємо ансамблі. На Рисунку 3.6 зображено, з яких моделей створено ансамблі.

```
ensemble_resnet18_corrupt = [corrupt_resnet18_1, corrupt_resnet18_2,corrupt_resnet18_3]

ensemble_resnet18 = [resnet18_1, resnet18_2,resnet18_3]
```

Рисунок 3.6 – Створення ансамблів з моделей після ін’єкції і до ін’єкції

Обчислюємо точність до і після ін’єкцій та показник стійкості R (див. Рис. 3.7). З результатів видно, що точність ансамблю вища за точність кожної з моделей, що доводить ефективність використання ансамблів.

```
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 86.95%
Ensemble Accuracy: 80.36%
0.9242093156986774
```

Рисунок 3.7 – Фрагмент коду Обчислення точностей та показника стійкості моделі resnet18

З результатів видно, що точність ансамблю вища за точність кожної з моделей, що доводить перевагу у використання ансамблів замість одиночних моделей. R = 92% є досить високим показником стійкості до даного типу помилок.

Перейдемо до ансамбля з моделей однакової архітектури resnet50. Дано 3 натреновані моделі resnet50 з певними точностями (див. Рис. 3.8)

```
Epoch 1/10, Accuracy: 71.09%
Epoch 2/10, Accuracy: 78.28%
Epoch 3/10, Accuracy: 83.10%
Epoch 4/10, Accuracy: 85.80%
Epoch 5/10, Accuracy: 87.40%
Epoch 6/10, Accuracy: 82.30%
Epoch 7/10, Accuracy: 86.58%
Epoch 8/10, Accuracy: 90.16%
Epoch 9/10, Accuracy: 91.70%
Epoch 10/10, Accuracy: 92.56%
Finished Training
Epoch 1/10, Accuracy: 69.87%
Epoch 2/10, Accuracy: 79.46%
Epoch 3/10, Accuracy: 82.94%
Epoch 4/10, Accuracy: 85.84%
Epoch 5/10, Accuracy: 87.25%
Epoch 6/10, Accuracy: 83.84%
Epoch 7/10, Accuracy: 88.80%
Epoch 8/10, Accuracy: 88.27%
Epoch 9/10, Accuracy: 91.37%
Epoch 10/10, Accuracy: 92.30%
Finished Training
Epoch 1/10, Accuracy: 68.18%
Epoch 2/10, Accuracy: 80.13%
Epoch 3/10, Accuracy: 83.35%
Epoch 4/10, Accuracy: 84.05%
Epoch 5/10, Accuracy: 86.25%
Epoch 6/10, Accuracy: 88.57%
Epoch 7/10, Accuracy: 89.71%
Epoch 8/10, Accuracy: 90.69%
Epoch 9/10, Accuracy: 91.79%
Epoch 10/10, Accuracy: 91.88%
Finished Training
```

Рисунок 3.8 – Результати тренування моделей resnet50 з відповідними точностями

Ці моделі мають такі точності на тестових даних перед виконанням ін’єкцій:

```
[21] accuracy1 = evaluate_accuracy(resnet50_1, test_loader, device)
     accuracy2 = evaluate_accuracy(resnet50_2, test_loader, device)
     accuracy3 = evaluate_accuracy(resnet50_3, test_loader, device)
```

```
print(f'Test Accuracy: {accuracy1}%')
print(f'Test Accuracy: {accuracy2}%')
print(f'Test Accuracy: {accuracy3}%')
```

```
Test Accuracy: 84.57%
Test Accuracy: 84.16%
Test Accuracy: 80.58%
```

Рисунок 3.9 - Обчислення точностей моделей resnet50

Виконуємо ін’єкцію функцією random_weight_inj в 3 шар з параметрами min_val=20, max_val=30 в кожну з моделей. Отримуємо прогнози кожної моделі та голосуємо по них (див. Рис. 3.10).

```
fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, corrupt_layer=3, min_val=20, max_val=30
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2, predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.10 - Ін’єкція функцією random_weight_inj в 3 шар resnet50 з параметрами min_val=20, max_val=30

Отримали у результаті голосування 3 клас. Вираховуємо показник стійкості R на Рисунку 3.11:

```
ensemble_resnet50 = [resnet50_1,resnet50_2,resnet50_3]
ensemble_resnet50_corrupt = [corrupt_resnet50_1,corrupt_resnet50_2,corrupt_resnet50_3]
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 87.17%
Ensemble Accuracy: 64.87%
0.7441780429046692
```

Рисунок 3.11 – Обчислення показника стійкості R

R = 74%, що є меншим показником стійкості до даного типу помилок, ніж у моделей Resnet18.

Ансамбль з моделей архітектури mobilenet. Дано 3 натреновані моделі mobilenet (див. Рис. 3.12).

```
Epoch 1/10, Accuracy: 69.22%
Epoch 2/10, Accuracy: 79.16%
Epoch 3/10, Accuracy: 82.06%
Epoch 4/10, Accuracy: 83.77%
Epoch 5/10, Accuracy: 85.19%
Epoch 6/10, Accuracy: 82.20%
Epoch 7/10, Accuracy: 77.77%
Epoch 8/10, Accuracy: 82.61%
Epoch 9/10, Accuracy: 85.90%
Epoch 10/10, Accuracy: 85.54%
Finished Training
Epoch 1/10, Accuracy: 69.29%
Epoch 2/10, Accuracy: 78.81%
Epoch 3/10, Accuracy: 81.92%
Epoch 4/10, Accuracy: 83.67%
Epoch 5/10, Accuracy: 84.68%
Epoch 6/10, Accuracy: 85.60%
Epoch 7/10, Accuracy: 86.63%
Epoch 8/10, Accuracy: 87.71%
Epoch 9/10, Accuracy: 88.09%
Epoch 10/10, Accuracy: 89.05%
Finished Training
Epoch 1/10, Accuracy: 69.09%
Epoch 2/10, Accuracy: 79.08%
Epoch 3/10, Accuracy: 81.37%
Epoch 4/10, Accuracy: 83.48%
Epoch 5/10, Accuracy: 84.76%
Epoch 6/10, Accuracy: 85.25%
Epoch 7/10, Accuracy: 86.17%
Epoch 8/10, Accuracy: 87.22%
Epoch 9/10, Accuracy: 87.32%
Epoch 10/10, Accuracy: 88.12%
Finished Training
```

Рисунок 3.12 – Результат тренування моделей mobilenet

Виконуємо ін’єкцію функцією random_weight_inj в 3 шар з параметрами min_val=20, max_val=30 в кожну з моделей (див. Рис. 3.13). Отримуємо прогнози кожної моделі та голосуємо по них.

```
fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, corrupt_layer=3, min_val=20, max_val=30
        )
# Тепер mobilenet_2 готова до використання з новими вагами і налаштованим класифікатором
predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2, predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.13 - Ін’єкція функцією random_weight_inj в 3 шар моделі mobilenet та параметрами min_val=20, max_val=30

Отримано у результаті голосування 3 клас, вважаємо голосування успішним.

Знаходимо точність ансамблю до і після ін’єкцій і обчислюємо показник стійкості до даного типу збою, як показано на Рисунку 3.14:

```
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt = [corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 81.01%
Ensemble Accuracy: 74.23%
0.9163066288112579
```

Рисунок 3.14 – Обчислення показника стійкості R ансамблю з моделей архітектури mobilenet

У результаті голосування отримано 3 клас та R=92% (див. Рис. 3.14), що є достатньо високим показником стійкості моделі до збоїв.

Ансамбль з моделей різних архітектур: resnet18, resnet50, mobilenet. Сформуємо ансамбль з трьох різних моделей resnet18, resnet50, mobilenet, як показано на Рисунку 3.15. Обчислимо показник стійкості та проведемо голосування за прогнозами.

```
ensemble_diff = [resnet50_1,resnet18_1,mobilenet_1]
ensemble_diff_corrupt = [corrupt_resnet50_1,corrupt_resnet18_1,corrupt_mobilenet_1]
accuracy = evaluate_ensemble_accuracy(ensemble_diff, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_diff_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 87.43%
Ensemble Accuracy: 85.04%
0.9726638453620039
```

```
predictions_diff_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_diff_2 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_diff_3 = get_predictions(corrupt_mobilenet_1, test_loader, device)

predictions4 = [predictions_diff_1, predictions_diff_2, predictions_diff_3]
final_predictions4 = vote(predictions4)
print(final_predictions4)
```

```
3
```

Рисунок 3.15 – Обчислення показника стійкості та голосування за прогнозами

Отримано R = 97% , що є найкращим результатом з усіх отриманих. З цього виходить, що ансамбль з різних моделей краще справляється з ін'єкціями даного типу.

3.3.2 Експеримент 2.

Виконаємо ті ж самі дії, але ін'єктуємо іншу помилку цією ж функцією, але з параметрами min_val=10, max_val=30 без вказування шару.

Ансамбль з моделей Resnet18. Ін'єктуємо помилку, як показано на Рисунку 3.16 та голосуємо за прогнозами цих моделей.

```
fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1,  min_val=10, max_val=30
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2,  min_val=10, max_val=30
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3,  min_val=10, max_val=30
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2, predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.16 – Ін'єкція помилок у моделі resnet18 та голосування за прогнозами

У результаті голосування отримано 3 клас. Обчислимо показник стійкості (див. Рис. 3.17).

```
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 85.94%
Ensemble Accuracy: 75.65%
0.880265301373051
```

Рисунок 3.17 – Обчислення показника стійкості

Отримано в результаті R = 88%, що свідчить про те, що даний ансамбль достатньо добре справляється з помилками такого типу.

Ансамбль з моделей Resnet50. Ін’єктуємо помилки в кожну з моделей ансамблю та голосуємо за прогнозами (див. Рис. 3.18)

```
fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, min_val=10, max_val=30
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, min_val=10, max_val=30
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, min_val=10, max_val=30
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2, predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.18 – Голосування за прогнозами

Отримано 3 клас як результат голосування за прогнозами. Обчислимо тепер показник стійкості до даного типу помилок (див. Рис. 3.19).

```
ensemble_resnet50 = [resnet50_1,resnet50_2,resnet50_3]
ensemble_resnet50_corrupt = [corrupt_resnet50_1,corrupt_resnet50_2,corrupt_resnet50_3]
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 87.26%
Ensemble Accuracy: 86.81%
0.9948429979371991
```

Рисунок 3.19 – Обчислення показника стійкості моделей до даного типу збою

Як результат маємо R = 99%, який показує, що модель майже не змінила точність після даної ін’єкції.

Ансамбль з моделей Mobilenet . Ін’єктуємо помилку, як показано на Рисунку 3.20 та голосуємо за прогнозами цих моделей :

```
fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, min_val=10, max_val=30
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, min_val=10, max_val=30
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, min_val=10, max_val=30
        )
# Тепер mobilenet_2 готова до використання з новими вагами і налаштованим класифікатором
predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2, predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
0
```

Рисунок 3.20 – Ін’єкція помилки та голосування за прогнозами

Отримано клас 0. Це хибний результат, робимо припущення, що модель не стійка до даної помилки. Обчислимо показник стійкості даного ансамблю до ін'єкцій, сформувавши спочатку ансамблі з моделей до ін'єкцій та зіпсованих моделей для того, щоб знайти їхні точності (див. Рис. 3.21)

```
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt = [corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 81.04%
Ensemble Accuracy: 49.83%
0.6148815399802565
```

Рисунок 3.21 – Обчислення показника стійкості

Отримано R = 61%, що свідчить про те, що модель має низьку стійкість до даного типу помилок.

Ансамбль з моделей різної архітектури resnet18, resnet50, mobilenet. Як показано на Рисунку 3.22, голосуємо за прогнозами цих моделей:

```
#1, min_val=10, max_val=30
predictions3 = [predictions_resnet50_1, predictions_resnet18_1, predictions_mobilenet_1]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.22 – Результат голосування за прогнозами в різноархітектурному ансамблі

Отримано успішний результат внаслідок голосування – 3 клас. Обчислимо точність до та після ін'єкції та показник стійкості (див. Рис. 3.23).

```
ensemble_diff = [resnet50_1,resnet18_1,mobilenet_1]
ensemble_diff_corrupt = [corrupt_resnet50_1,corrupt_resnet18_1,corrupt_mobilenet_1]
accuracy = evaluate_ensemble_accuracy(ensemble_diff, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_diff_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 87.78%
Ensemble Accuracy: 83.51%
0.9513556618819777
```

Рисунок 3.23 – Обчислення показника стійкості

Отримано R = 95%, що свідчить про високу стійкість моделі до даної ін’єкції.

3.3.3 Експеримент 3.

Виконаємо ін’єкції функцією zero_func_rand_weight, яка замінює вагу на нуль в випадково вибраному місці.

Ансамбль з моделей resnet18. Ін’єктуємо помилки в кожну з моделей ансамблю та голосуємо за прогнозами (див. Рис. 3.24)

```
corrupt_resnet18_1 =  zero_func_rand_weight(
            fi_resnet18_1
        )

corrupt_resnet18_2 =  zero_func_rand_weight(
            fi_resnet18_2
        )

corrupt_resnet18_3 =  zero_func_rand_weight(
            fi_resnet18_3
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2, predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.24 – Ін’єкція та голосування за прогнозами

У результаті отримано 3 клас, що є правильним варіантом. То ж, перевіримо стійкість даного ансамблю (див. Рис. 3.25).

```
#ZEROS
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 85.65%
Ensemble Accuracy: 85.76%
1.0012842965557502
```

Рисунок 3.25 – Обчислення показника стійкості

Отримано показник стійкості R = 100, а отже система стійка до даної ін'єкції.

Ансамбль з моделей resnet50. Ін'єктуємо помилки в кожну з моделей ансамблю та голосуємо за прогнозами (див. Рис. 3.26).

```
fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  zero_func_rand_weight(
            fi_resnet50_1
        )

corrupt_resnet50_2 =  zero_func_rand_weight(
            fi_resnet50_2
        )

corrupt_resnet50_3 =  zero_func_rand_weight(
            fi_resnet50_3
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2, predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.26 – Ін'єкція та голосування за прогнозами

Результатом є 3 клас і це успішне голосування. Виконаємо розрахунок показника стійкості, як показано на Рисунку 3.27.

```
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 79.19%
Ensemble Accuracy: 79.44%
1.0031569642631646
```

Рисунок 3.27 – Обчислення показника стійкості

Отримано показник стійкості R = 100, а отже система стійка до даної ін'єкції.

Ансамбль з моделей mobilenet. Ін'єктуємо помилки в кожну з моделей ансамблю та голосуємо за прогнозами (див. Рис. 3.28).

```
fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  # Ваш розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  zero_func_rand_weight(
            fi_mobilenet_1
        )

corrupt_mobilenet_2 =  zero_func_rand_weight(
            fi_mobilenet_2
        )

corrupt_mobilenet_3 =  zero_func_rand_weight(
            fi_mobilenet_3
        )
# Тепер mobilenet_2 готова до використання з новими вагами і налаштованим класифікатором
predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2, predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.28 – Введення ін'єкцій та голосування за прогнозами

Отримано в результаті голосування 3 клас, що є успішним результатом.

```
#zeros
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt = [corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 82.41%
Ensemble Accuracy: 82.3%
0.9986652105327023
```

Рисунок 3.29 – Показник стійкості до збоїв

Отримано показник стійкості R = 99%, що показує стійкість системи до даної ін’єкції.

Ансамбль з моделей різної архітектури resnet18, resnet50, mobilenet. Знаходимо передбачення моделей та голосуємо за прогнозами (див. Рис. 3.30).

```
#zeros
predictions3 = [predictions_resnet50_1, predictions_resnet18_1, predictions_mobilenet_1]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.30 – Голосування за прогнозами в різноархітектурному ансамблі

У результаті отримано 3 клас, що є коректним. Обчислимо точність до і після ін’єкції та показник стійкості даного ансамблю до ін’єкцій (див. Рис. 3.31).

```
Ensemble Accuracy: 85.99%
Ensemble Accuracy: 85.94%
0.9994185370391906
```

Рисунок 3.31 – Розрахунок точностей та показника стійкості

Отримано R = 99%, що визначає стійкість системи до даних помилок.

3.3.4 Експеримент 4.

Використаємо ін’єкцію з 2-го експерименту, але збільшимо min_val до 50 та max_val до 55, щоб зменшити різницю між ними до 5 одиниць.

Ансамбль з моделей Resnet-18. Знаходимо передбачення моделей та голосуємо за прогнозами (див. Рис. 3.32).

```
predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2, predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.32 – Результат голосування по прогнозам корумпованих моделей

Отримано успішний результат внаслідок голосування – 3 клас. Обчислимо точність до та після ін’єкції та Обчислимо показник стійкості (див. Рис. 3.33).

```
#random weight ing 50 55
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 79.52%
Ensemble Accuracy: 65.23%
0.8202967806841047
```

Рисунок 3.33 – Обчислення показника стійкості

Отримано показник R = 82%, що є гарним результатом.

Ансамбль з моделей Resnet50. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.34).

```
predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2, predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.34 – Голосування на прогнозах корумпованих моделей

Результат голосування – успішний. Знайдемо показник стійкості до збоїв (див. Рис. 3.35)

```
#random_weight_inj 50 55
ensemble_resnet50 = [resnet50_1,resnet50_2,resnet50_3]
ensemble_resnet50_corrupt = [corrupt_resnet50_1,corrupt_resnet50_2,corrupt_resnet50_3]
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 81.23%
Ensemble Accuracy: 50.09%
0.6166440970084944
```

Рисунок 3.35 – Показник стійкості

Показник стійкості R = 62%, що свідчить про низьку стійкість до збоїв, незважаючи на коректний результат голосування.

Ансамбль з моделей Mobilenet. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.36).

```
# Тепер mobilenet_2 готова до використання з новими вагами і налаштованим класифікатором
predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2, predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.36 – Голосування за прогнозами ін’єктованих моделей

Визначимо показник стійкость до даного типу помилок (див. Рис. 3.37).

```
#random_weight_inj 50 55
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt = [corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 70.1%
Ensemble Accuracy: 53.18%
0.7586305278174037
```

Рисунок 3.37 – Показник стійкості до помилок

Результатом є R = 75%, що є посередньою стійкістю до помилок.

Ансамбль з моделей різної архітектури resnet18, resnet50, mobilenet. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.38).

```
#min_val=50, max_val=55
predictions_diff_1 = get_predictions(corrupt_mobilenet_3, test_loader, device)
predictions_diff_2 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_diff_3 = get_predictions(corrupt_mobilenet_2, test_loader, device)

predictions4 = [predictions_diff_1, predictions_diff_2, predictions_diff_3]
final_predictions4 = vote(predictions4)
print(final_predictions4)
```

```
3
```

Рисунок 3.38 – Голосування за прогнозами пошкоджених помилками моделей.

Результат голосування успішний, то ж Обчислимо показник стійкості до помилок (див. Рис. 3.39).

```
accuracy = evaluate_ensemble_accuracy(ensemble_diff, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_diff_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 79.81%
Ensemble Accuracy: 79.54%
0.9966169652925699
```

Рисунок 3.39 – Обчислення показника стійкості

Результат: R = 99%, що є свідченням того, що ансамбль з різними архітектурами стійкий до даного типу помилок.

3.3.5 Експеримент 5.

У попередніх експериментах були достатньо високі показники стійкості, можливо були малі вхідні дані для ін’єкції. Збільшимо min_val до 100, max_val до 105.

Розглянемо ансамбль з моделей архітектури resnet18. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.40).

```
predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2, predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
0
```

Рисунок 3.40 – Ін’єкція та голосування за прогнозами ін’єктованих моделей

Отримано клас 0, що є хибним результатом. Перевіримо показник стійкості (див. Рис. 3.41).

```
#random weight ing 100 105
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 79.63%
Ensemble Accuracy: 65.0%
0.8162752731382645
```

Рисунок 3.41 – Показник стійкості

Отримано показник R = 81%, що є помірним показником стійкості, адже голосування показало невтішний результат.

Ансамбль з моделей різної архітектури resnet50. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.42).

```
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, corrupt_layer=1,min_val=100, max_val=105
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, corrupt_layer=1,min_val=100, max_val=105
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2, predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)
```

```
3
```

Рисунок 3.42 – Ін’єкція помилок та голосування за прогнозами пошкоджених моделей

Результат є успішним – клас 3. Обчислимо показник стійкості до помилок даної системи розпізнавання (див. Рис. 3.43).

```
#random_weight_inj 50 55
ensemble_resnet50 = [resnet50_1,resnet50_2,resnet50_3]
ensemble_resnet50_corrupt = [corrupt_resnet50_1,corrupt_resnet50_2,corrupt_resnet50_3]
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 80.96%
Ensemble Accuracy: 66.81%
0.8252223320158103
```

Рисунок 3.43 – Показник стійкості системи до помилок

Отримано покказник R=82%, що є кращим результатом, ніж в попередній моделі, адже голосування за прогнозами було успішним і кінцева точність вище.

Ансамбль з моделей mobilenet. Знайдемо передбачення моделей та проголосуємо за прогнозами (див. Рис. 3.44).

```
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, corrupt_layer=1,min_val=100, max_val=105
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, corrupt_layer=1,min_val=100, max_val=105
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3,corrupt_layer=1,min_val=100, max_val=105
        )
# Тепер mobilenet_2 готова до використання з новими вагами і налаштованим класифікатором
predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2, predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
```

```
3
```

Рисунок 3.44 – Ін’єкція та голосування за прогнозами моделей з помилками

Результатом голосування є 3 клас і це успішно. Розрахуємо показник стійкость з точностей до і після атак (див. Рис. 3.45).

```
#random_weight_inj 100 105
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt = [corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 70.06%
Ensemble Accuracy: 45.15%
0.6444476163288609
```

Рисунок 3.45 – Розрахунок показника стійкості

Показник стійкості дорівнює 64%, що є низьким результатом, не дивлячись на успішне голосування.

Ансамбль з моделей різної архітектури Resnet50, Resnet18, MobileNet. Виконаємо розрахунок показника стійкості (див. Рис. 3.46)

Resnet50 + Resnet18 + MobileNet

```
#100 106=5
ensemble_diff = [resnet50_1,resnet18_1,mobilenet_1]
ensemble_diff_corrupt = [corrupt_resnet50_1,corrupt_resnet18_1,corrupt_mobilenet_1]
accuracy = evaluate_ensemble_accuracy(ensemble_diff, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_diff_corrupt, test_loader, device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```

```
Ensemble Accuracy: 79.66%
Ensemble Accuracy: 62.84%
0.7888526236505148
```

Рисунок 3.46 – Розрахунок показника стійкості ансамблю до помилок

Результатом є показник 79%, що є посереднім результатом. Виконаємо голосування по прогнозам (див. Рис. 3.47)

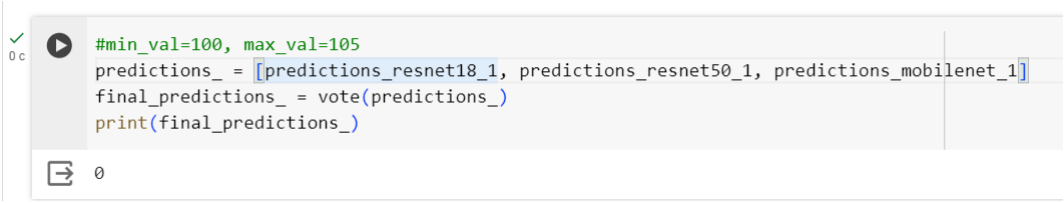

Рисунок 3.47 – Результат голосування за прогнозами

Результатом є клас 0, що є хибним результатом.

Порівняємо та проаналізуємо результати з експериментів, наведені в таблиці 3.2.

Таблиця 3.3 – Аналіз результатів експериментів

Тип помилки	Архітектури ансамблю	Результат голосування	R
random_weight_inj : corrupt_layer = 3 , min_val=20, max_val=30	resnet18	3	92%
	resnet50	3	74%
	mobilenet	3	92%
	r18+r50+mn	3	97%
random_weight_inj: min_val=10, max_val=30	resnet18	3	88%
	resnet50	3	99%
	mobilenet	0	62%
	r18+r50+mn	3	95%
zero_func_rand_weight	resnet18	3	100%
	resnet50	3	100%
	mobilenet	3	99%
	r18+r50+mn	3	99%
random_weight_inj: min_val=50, max_val=55	resnet18	3	82%
	resnet50	3	62%
	mobilenet	3	75%
	r18+r50+mn	3	99%
random_weight_inj : corrupt_layer = 1 , min_val=100, max_val=105	resnet18	0	81%
	resnet50	3	82%
	mobilenet	3	64%
	r18+r50+mn	0	78%

З таблиці можна зробити висновок, що 40% від найбільшого показника R належать ансамблю з різними архітектурами, 60% належить з однаковими архітектурами.

Ансамблі із трьох моделей ResNet18 мають високу стійкість із R = 92–100%, залежно від типу введення помилки.

Ансамблі із трьох моделей ResNet50 демонструють різну надійність із R у діапазоні від 74 до 100%.

Ансамбль із трьох моделей MobileNet має достатньо низьку стійкість, порівнюючи з попередніми архітектурами, особливо це помітно при ін’єкції помилок із високими значеннями: R коливається від 62 до 92%.

Ансамблі, що містять комбінації трьох архітектур ResNet18, ResNet50, MobileNet, мають показник R від 78 до 99%, що вказує на перевагу поєднання різних архітектур для підвищення загальної надійності у більшості випадків.

ВИСНОВКИ

У ході виконання роботи було проведено аналіз сучасного стану та тенденцій розвитку моделей штучного інтелекту у кібербезпеці, створенні надійних, зрозумілих і етичних систем. Виявлено три фундаментальні поняття: несправність, помилка та збій, та їх типи по часовій ознаці. Оглянуто методи їх інжекції в середовище розгортання нейронних мереж.

Описано модель класифікатора зображень з підвищеною стійкістю до збоїв та методів навчання нейронних мереж. Представлено критерії оцінювання ефективності та стійкості моделей.

Викладено практичний аспект роботи, включаючи формування навчальних та тестових даних та опис програмного забезпечення. Аналіз результатів проведених експериментів, демонструючи вплив запропонованих рішень на стійкість систем.

Досліджено експерименти з різними типами помилками, ін’єктованих в ансамблі з однаковими архітектурами моделей та різними. Проаналізовано, що ансамбль із трьох моделей ResNet18 має високу стійкість із R = 92–100%, залежно від типу введення помилки, ансамбль із трьох моделей ResNet50 демонструють різну надійність із R у діапазоні від 74 до 100%, ансамбль із трьох моделей MobileNet має достатньо низьку стійкість, порівнюючи з попередніми архітектурами, особливо при ін’єкції помилок із високими значеннями, R коливається від 62 до 92%, ансамблі, що містять комбінації трьох архітектур ResNet18, ResNet50, MobileNet, мають показник R від 78 до 99%, що вказує на перевагу поєднання різних архітектур для підвищення загальної надійності у більшості випадків. Отже, проведено аналіз стійкості різних ансамблів нейронних мереж до програмно-апаратних помилок, що є внеском у сферу штучного інтелекту для підвищення можливості моделей нейронних мереж протидіяти помилкам.

Для подальшого вдосконалення даної інформаційної технології пропонується розширення набору моделей, набору даних та типів помилок, на яких можна проводити дослідження для покращення технології.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] M. Haenlein and A. Kaplan, “A brief history of artificial intelligence: On the past, present, and future of artificial intelligence,” *Calif Manage Rev*, vol. 61, no. 4, 2019, doi: 10.1177/0008125619864925.

[2] H. Kimm, I. Paik, and H. Kimm, “Performance Comparision of TPU, GPU, CPU on Google Colaboratory over Distributed Deep Learning,” in *Proceedings - 2021 IEEE 14th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2021*, 2021. doi: 10.1109/MCSoC51149.2021.00053.

[3] E. Buber and B. Diri, “Performance analysis and CPU vs GPU comparison for deep learning,” in *2018 6th International Conference on Control Engineering and Information Technology, CEIT 2018*, 2018. doi: 10.1109/CEIT.2018.8751930.

[4] S. Mittal, “A survey of FPGA-based accelerators for convolutional neural networks,” *Neural Computing and Applications*, vol. 32, no. 4. 2020. doi: 10.1007/s00521-018-3761-1.

[5] F. Khalid, M. A. Hanif, and M. Shafique, “Exploiting Vulnerabilities in Deep Neural Networks: Adversarial and Fault-Injection Attacks,” May 2021.

[6] C. Torres-Huitzil and B. Girau, “Fault and Error Tolerance in Neural Networks: A Review,” *IEEE Access*, vol. 5. 2017. doi: 10.1109/ACCESS.2017.2742698.

[7] H. Xu, Z. Chen, W. Wu, Z. Jin, S. Y. Kuo, and M. Lyu, “NV-DNN: Towards fault-tolerant DNN systems with N-version programming,” in *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, DSN-W 2019*, 2019. doi: 10.1109/DSN-W.2019.00016.

[8] D. D. Solomon *et al.*, “Hybrid Majority Voting: Prediction and Classification Model for Obesity,” *Diagnostics*, vol. 13, no. 15, 2023, doi: 10.3390/diagnostics13152610.

[9] G. S. Tandel, A. Tiwari, and O. G. Kakde, “Performance optimisation of deep learning models using majority voting algorithm for brain tumour classification,” *Comput Biol Med*, vol. 135, 2021, doi: 10.1016/j.compbiomed.2021.104564.

[10] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Front Neurosci*, vol. 10, no. NOV, 2016, doi: 10.3389/fnins.2016.00508.

[11] S. R. Ahmed and E. Sonuç, “Evaluating the effectiveness of rationale-augmented convolutional neural networks for deepfake detection,” *Soft comput*, Oct. 2023, doi: 10.1007/s00500-023-09245-y.

[12] S. Aslam and A. B. Nassif, “Deep learning based CIFAR-10 classification,” in *2023 Advances in Science and Engineering Technology International Conferences, ASET 2023*, 2023. doi: 10.1109/ASET56582.2023.10180767.

[13] P. Mishra, “Introduction to PyTorch, Tensors, and Tensor Operations,” in *PyTorch Recipes*, Berkeley, CA: Apress, 2019, pp. 1–27. doi: 10.1007/978-1-4842-4258-2_1.

[14] A. Mahmoud *et al.*, “PyTorchFI: A Runtime Perturbation Tool for DNNs,” in *Proceedings - 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2020*, 2020. doi: 10.1109/DSN-W50199.2020.00014.

ДОДАТОК А

```
import pytorchfi
import torch
import torchvision.models as models
import pytorchfi.core as pfi_core
from pytorchfi.weight_error_models import (
    random_weight_inj,
    random_weight_location,
    zero_func_rand_weight
)
def evaluate_ensemble_accuracy(ensemble_models, test_loader, device):
    for model in ensemble_models:
        model.eval()
        model.to(device)

    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # Агрегація передбачень від усіх моделей
            ensemble_outputs = None
            for model in ensemble_models:
                outputs = model(inputs)
                if ensemble_outputs is None:
                    ensemble_outputs = outputs
                else:
                    ensemble_outputs += outputs

            # Вибір класу з найвищим загальним передбаченням
            _, predicted = torch.max(ensemble_outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return accuracy


def calculate_accuracy(model, data_loader):
    correct = 0
    total = 0
    model.to(device)
    model.eval()

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    return correct / total

def get_predictions(model, test_loader, device):
    model.to(device)
    model.eval()
```

```
    predictions = []

    with torch.no_grad():
        for inputs, _ in test_loader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            predictions.extend(predicted.cpu().numpy())

    return predictions


device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


import numpy as np
from scipy.stats import mode


def vote(predictions):
    predictions_array = np.array(predictions)
    voted_predictions = mode(predictions_array, axis=0)[0]
    return voted_predictions[0]


import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

# Передобробка та завантаження даних CIFAR10
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Випадкове горизонтальне перевертання для
аугментації
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=128, shuffle=True,
num_workers=2)

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=128, shuffle=False,
num_workers=2)

# Визначення моделі ResNet18
resnet18_1 = models.resnet18(pretrained=True)
resnet18_2 = models.resnet18(pretrained=True)
resnet18_3 = models.resnet18(pretrained=True)
num_ftrs1 = resnet18_1.fc.in_features
num_ftrs2 = resnet18_2.fc.in_features
num_ftrs3 = resnet18_3.fc.in_features
resnet18_1.fc = nn.Linear(num_ftrs1, 10)  # CIFAR10 має 10 класів
resnet18_2.fc = nn.Linear(num_ftrs2, 10)  # CIFAR10 має 10 класів
```

```
resnet18_3.fc = nn.Linear(num_ftrs3, 10)  # CIFAR10 має 10 класів
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
resnet18_1.to(device)
resnet18_2.to(device)
resnet18_3.to(device)
# Визначення оптимізатора та функції втрат
optimizer1 = optim.Adam(resnet18_1.parameters(), lr=0.001)
optimizer2 = optim.Adam(resnet18_2.parameters(), lr=0.001)
optimizer3 = optim.Adam(resnet18_3.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

def train_model(model, criterion, optimizer, num_epochs=10):
    model.train()
    for epoch in range(num_epochs):
        running_loss = 0.0
        correct_predictions = 0
        total_predictions = 0

        for i, (inputs, labels) in enumerate(train_loader, 0):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            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_acc = (correct_predictions / total_predictions) * 100

        print(f'Epoch {epoch + 1}/{num_epochs}, Accuracy: {epoch_acc:.2f}%')

    print('Finished Training')

# Тренування моделі
train_model(resnet18_1, criterion, optimizer1, num_epochs=10)
train_model(resnet18_2, criterion, optimizer2, num_epochs=10)
train_model(resnet18_3, criterion, optimizer3, num_epochs=10)


ensemble_resnet18 = [resnet18_1,resnet18_2,resnet18_3]

accuracy_resnet18 = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_resnet18}%')

fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
```

```
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3, corrupt_layer=3, min_val=20, max_val=30
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2,
predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1,  min_val=10, max_val=30
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2,  min_val=10, max_val=30
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3,  min_val=10, max_val=30
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)
```

```
predictions2 = [predictions_resnet18_1, predictions_resnet18_2,
predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)

ensemble_resnet18_corrupt = [corrupt_resnet18_1,
corrupt_resnet18_2,corrupt_resnet18_3]
ensemble_resnet18 = [resnet18_1, resnet18_2,resnet18_3]

accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)


fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  zero_func_rand_weight(
            fi_resnet18_1
        )

corrupt_resnet18_2 =  zero_func_rand_weight(
            fi_resnet18_2
        )

corrupt_resnet18_3 =  zero_func_rand_weight(
            fi_resnet18_3
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2,
predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)

ensemble_resnet18_corrupt =
[corrupt_resnet18_1,corrupt_resnet18_2,corrupt_resnet18_3]
#ZEROS
```

```
accuracy = evaluate_ensemble_accuracy(ensemble_resnet18, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet18_corrupt, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)


import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

# Передобробка та завантаження даних CIFAR10
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Випадкове горизонтальне перевертання для
аугментації
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=128, shuffle=True,
num_workers=2)

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=128, shuffle=False,
num_workers=2)

# Визначення моделі ResNet50
resnet50_1 = models.resnet50(pretrained=True)
num_ftrs = resnet50_1.fc.in_features
resnet50_1.fc = nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів

resnet50_2 = models.resnet50(pretrained=True)
num_ftrs = resnet50_2.fc.in_features
resnet50_2.fc = nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів

resnet50_3 = models.resnet50(pretrained=True)
num_ftrs = resnet50_3.fc.in_features
resnet50_3.fc = nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
resnet50_1.to(device)
resnet50_2.to(device)
resnet50_3.to(device)
# Визначення оптимізатора та функції втрат

optimizer11 = optim.Adam(resnet50_1.parameters(), lr=0.001)
optimizer12 = optim.Adam(resnet50_2.parameters(), lr=0.001)
optimizer13 = optim.Adam(resnet50_3.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

```
# Функція для тренування моделі (аналогічна функції для ResNet18)
def train_model(model, train_loader, criterion, optimizer, device, num_epochs=10):
    model.to(device)
    model.train()
    for epoch in range(num_epochs):
        correct_predictions = 0
        total_predictions = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            _, predicted = torch.max(outputs.data, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

        accuracy = 100 * correct_predictions / total_predictions
        print(f'Epoch {epoch + 1}/{num_epochs}, Accuracy: {accuracy:.2f}%')

    print('Finished Training')


# Тренування моделі
train_model(resnet50_1,train_loader,  criterion, optimizer11, device)
train_model(resnet50_2, train_loader, criterion, optimizer12, device)
train_model(resnet50_3, train_loader, criterion, optimizer13, device)

fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, corrupt_layer=3, min_val=20, max_val=30
        )
```

```
predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, min_val=10, max_val=30
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, min_val=10, max_val=30
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, min_val=10, max_val=30
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)



fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
```

```
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, min_val=10, max_val=30
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, min_val=10, max_val=30
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, min_val=10, max_val=30
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)

fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  zero_func_rand_weight(
            fi_resnet50_1
        )

corrupt_resnet50_2 =  zero_func_rand_weight(
            fi_resnet50_2
        )

corrupt_resnet50_3 =  zero_func_rand_weight(
            fi_resnet50_3
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)

ensemble_resnet50 = [resnet50_1,resnet50_2,resnet50_3]
```

```
ensemble_resnet50_corrupt =
[corrupt_resnet50_1,corrupt_resnet50_2,corrupt_resnet50_3]
accuracy = evaluate_ensemble_accuracy(ensemble_resnet50, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_resnet50_corrupt, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)


import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim


# Визначення моделі MobileNet
mobilenet_1 = models.mobilenet_v2(pretrained=True)
num_ftrs = mobilenet_1.classifier[1].in_features
mobilenet_1.classifier = nn.Sequential(
    nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів
)
mobilenet_2 = models.mobilenet_v2(pretrained=True)
num_ftrs = mobilenet_2.classifier[1].in_features
mobilenet_2.classifier = nn.Sequential(
    nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів
)
mobilenet_3 = models.mobilenet_v2(pretrained=True)
num_ftrs = mobilenet_3.classifier[1].in_features
mobilenet_3.classifier = nn.Sequential(
    nn.Linear(num_ftrs, 10)  # CIFAR10 має 10 класів
)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
mobilenet_1.to(device)
mobilenet_2.to(device)
mobilenet_3.to(device)
# Визначення оптимізатора та функції втрат
optimizer21 = optim.Adam(mobilenet_1.parameters(), lr=0.001)
optimizer22 = optim.Adam(mobilenet_2.parameters(), lr=0.001)
optimizer23 = optim.Adam(mobilenet_3.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
def train_model(model, train_loader, criterion, optimizer, device, num_epochs=10):
    model.to(device)
    model.train()

    for epoch in range(num_epochs):
        correct_predictions = 0
        total_predictions = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
```

```
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            _, predicted = torch.max(outputs.data, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

        accuracy = 100 * correct_predictions / total_predictions
        print(f'Epoch {epoch + 1}/{num_epochs}, Accuracy: {accuracy:.2f}%')

    print('Finished Training')


# Тренування моделі
train_model(mobilenet_1,train_loader,  criterion, optimizer21, device, num_epochs=10)
train_model(mobilenet_2, train_loader, criterion, optimizer22, device, num_epochs=10)
train_model(mobilenet_3,train_loader,  criterion, optimizer23, device, num_epochs=10)


fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, corrupt_layer=3, min_val=20, max_val=30
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, corrupt_layer=3, min_val=20, max_val=30
        )

predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2,
predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)


fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  #   розмір пакету
```

```
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, min_val=10, max_val=30
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, min_val=10, max_val=30
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, min_val=10, max_val=30
        )

predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2,
predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)

fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  zero_func_rand_weight(
            fi_mobilenet_1
        )

corrupt_mobilenet_2 =  zero_func_rand_weight(
            fi_mobilenet_2
        )

corrupt_mobilenet_3 =  zero_func_rand_weight(
            fi_mobilenet_3
        )

predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
```

```
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2,
predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)

fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3, corrupt_layer=1, min_val=100, max_val=105
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2,
predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
```

```
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, corrupt_layer=1, min_val=100, max_val=105
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, corrupt_layer=1, min_val=100, max_val=105
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, corrupt_layer=1, min_val=100, max_val=105
        )

predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)


fi_resnet18_1 = pytorchfi.core.fault_injection(
    model=resnet18_1,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_2 = pytorchfi.core.fault_injection(
```

```
    model=resnet18_2,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet18_3 = pytorchfi.core.fault_injection(
    model=resnet18_3,
    batch_size=1,  # розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet18_1 =  random_weight_inj(
            fi_resnet18_1, min_val=50, max_val=55
        )

corrupt_resnet18_2 =  random_weight_inj(
            fi_resnet18_2, min_val=50, max_val=55
        )

corrupt_resnet18_3 =  random_weight_inj(
            fi_resnet18_3, min_val=50, max_val=55
        )

predictions_resnet18_1 = get_predictions(corrupt_resnet18_1, test_loader, device)
predictions_resnet18_2 = get_predictions(corrupt_resnet18_2, test_loader, device)
predictions_resnet18_3 = get_predictions(corrupt_resnet18_3, test_loader, device)

predictions2 = [predictions_resnet18_1, predictions_resnet18_2,
predictions_resnet18_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_resnet50_1 = pytorchfi.core.fault_injection(
    model=resnet50_1,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_2 = pytorchfi.core.fault_injection(
    model=resnet50_2,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_resnet50_3 = pytorchfi.core.fault_injection(
    model=resnet50_3,
    batch_size=1,  #  розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_resnet50_1 =  random_weight_inj(
            fi_resnet50_1, min_val=50, max_val=55
        )

corrupt_resnet50_2 =  random_weight_inj(
            fi_resnet50_2, min_val=50, max_val=55
        )

corrupt_resnet50_3 =  random_weight_inj(
            fi_resnet50_3, min_val=50, max_val=55
        )

predictions_resnet50_1 = get_predictions(corrupt_resnet50_1, test_loader, device)
predictions_resnet50_2 = get_predictions(corrupt_resnet50_2, test_loader, device)
```

```
predictions_resnet50_3 = get_predictions(corrupt_resnet50_3, test_loader, device)

predictions2 = [predictions_resnet50_1, predictions_resnet50_2,
predictions_resnet50_3]
final_predictions2 = vote(predictions2)
print(final_predictions2)


fi_mobilenet_1 = pytorchfi.core.fault_injection(
    model=mobilenet_1,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_2 = pytorchfi.core.fault_injection(
    model=mobilenet_2,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
fi_mobilenet_3 = pytorchfi.core.fault_injection(
    model=mobilenet_3,
    batch_size=1,  #   розмір пакету
    layer_types=[torch.nn.Conv2d],  # Типи шарів, до яких застосовуються помилки
)
corrupt_mobilenet_1 =  random_weight_inj(
            fi_mobilenet_1, min_val=50, max_val=55
        )

corrupt_mobilenet_2 =  random_weight_inj(
            fi_mobilenet_2, min_val=50, max_val=55
        )

corrupt_mobilenet_3 =  random_weight_inj(
            fi_mobilenet_3, min_val=50, max_val=55
        )

predictions_mobilenet_1 = get_predictions(corrupt_mobilenet_1, test_loader, device)
predictions_mobilenet_2 = get_predictions(corrupt_mobilenet_2, test_loader, device)
predictions_mobilenet_3 = get_predictions(corrupt_mobilenet_3, test_loader, device)

predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2,
predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)


predictions3 = [predictions_mobilenet_1, predictions_mobilenet_2,
predictions_mobilenet_3]
final_predictions3 = vote(predictions3)
print(final_predictions3)
ensemble_mobilenet = [mobilenet_1,mobilenet_2,mobilenet_3]
ensemble_mobilenet_corrupt =
[corrupt_mobilenet_1,corrupt_mobilenet_2,corrupt_mobilenet_3]
accuracy = evaluate_ensemble_accuracy(ensemble_mobilenet, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_mobilenet_corrupt, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
```

```
print(R)

ensemble_diff = [resnet50_1,resnet18_1,mobilenet_1]
ensemble_diff_corrupt = [corrupt_resnet50_1,corrupt_resnet18_1,corrupt_mobilenet_1]
accuracy = evaluate_ensemble_accuracy(ensemble_diff, test_loader, device)
print(f'Ensemble Accuracy: {accuracy}%')

accuracy_inject = evaluate_ensemble_accuracy(ensemble_diff_corrupt, test_loader,
device)
print(f'Ensemble Accuracy: {accuracy_inject}%')

R = 1 - (accuracy-accuracy_inject)/accuracy
print(R)
```