МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
_____
(підпис)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-наукової програми «Інформатика»
на тему: Інформаційна технологія глибокого навчання для детектування заборонених предметів під час митного контролю та митного оформлення
здобувача групи ІН.мз-31с Боровика Дмитра Олександровича

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

_____ Дмитро Боровик
(підпис)

Керівник,
кандидат технічних наук,
асистент кафедри
комп'ютерних наук
Дмитро ПРИЛЕПА

_____
(підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри комп'ютерних наук
_____ Оксана ШОВКОПЛЯС
_____ грудня 2024р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-наукової програми «Інформатика»
здобувача групи ІН.мз-31с Боровика Дмитра Олександровича

1. Тема роботи: Інформаційна технологія глибокого навчання для детектування заборонених предметів під час митного контролю та митного оформлення.
затверджую наказом по інституту від *«05» грудня 2024р. № 1267-VI*
2. Термін здачі студентом закінченого проекту (роботи) _____
3. Вхідні данні до проекту (роботи)_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Інформаційно-аналітичний огляд 2) Технології глибокого машинного навчання
3) Інформаційне та програмне забезпечення інтелектуальної технології для детектування заборонених предметів
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 _____

Керівник _____
Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Інформаційно-аналітичний огляд*		
2.	*Технології глибокого машинного навчання*		
3.	*Інформаційне та програмне забезпечення інтелектуальної технології для детектування заборонених предметів*		
4.	*Оформлення кваліфікаційної робота магістра*		

Здобувач вищої освіти _____
Керівник проекту _____

АНОТАЦІЯ

Записка: 81 стор., 28 рис., 5 табл., 3 додатка, 59 джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі детектування заборонених предметів шляхом розробки відповідних методів, моделей та інформаційної технології.

Об'єкт дослідження — процес детектування заборонених предметів на зображеннях скануючих пристроїв із застосуванням інформаційної технології глибокого навчання

Мета роботи — розробка, навчання інтелектуальної системи SecureScanNet (SSN) для детектування заборонених предметів.

Методи дослідження — методи аналізу та синтезу інтелектуальних систем, методи машинного навчання, методи розпізнавання образів, методи обробки зображень, методи оптимізації моделей глибокого навчання, методи оцінки функціональної ефективності інформаційної технології.

Результати — створено та проведено навчання інтелектуальної системи SecureScanNet (SSN) для детектування заборонених предметів на зображеннях, отриманих за допомогою скануючих систем. У процесі роботи створено комплекс програмних і алгоритмічних рішень для ключових компонентів системи автоматичного виявлення заборонених об'єктів, включно з моделями нейронних мереж. Особливу увагу приділено дослідженню існуючих моделей нейромереж їх недоліків та переваг у порівнянні однієї до одної, як результат для досягнення мети застосовано послідовне використання спроектованих моделей конволюційних нейронних мереж, а також здійснена оптимізація процесу навчання для досягнення високої точності та ефективності інтелектуальної системи без перенавчання та надано оцінку передбачуваних моделлю результатів. Проведено ретельне дослідження параметрів моделі для покращення детекції та мінімізації помилкових спрацювань. Програмна реалізація виконана із використанням редактору коду Microsoft Visual Studio Code, Jupiter Notebook, мови програмування Python та спеціалізованих бібліотек TensorFlow, Keras, OpenCV, NumPy, scikit-learn, Matplotlib.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи.

У сучасному світі, де обсяги міжнародних перевезень та переміщення громадян постійно зростають, питання забезпечення безпеки стає особливо важливим. Заборонені предмети, такі як зброя, вибухівка, наркотичні засоби та інші небезпечні об'єкти, становлять загрозу для громадян і можуть бути використані для протиправної діяльності. У зв'язку з цим виникає необхідність впровадження нових технологій, які б дозволяли автоматично виявляти такі об'єкти при митному контролі, зокрема за результатами відпрацювання інтелектуальних систем.

Актуальність.

Інтелектуальні системи на основі технологій глибокого навчання є одним із найбільш перспективних напрямків для автоматизації процесів детектування заборонених предметів. Вони дозволяють підвищити точність і швидкість перевірки, зменшити навантаження на працівників митних адміністрацій та мінімізувати кількість помилкових спрацювань. Технології глибокого навчання, зокрема нейронні мережі, показують високу ефективність у задачах розпізнавання образів, обробки зображень і виявлення аномалій, що робить їх особливо актуальними для вирішення задач митного контролю.

Об’єкт дослідження.

Процес детектування заборонених предметів на зображеннях, отриманих за допомогою скануючих пристроїв.

Предмет дослідження.

В роботі застосовуються наступні методи: методи аналізу та синтезу моделей нейронних мереж для розробки загальної архітектури системи, методи машинного навчання для побудови та тренування моделей нейронних мереж, що використовуються для класифікації та детекції об'єктів, методи обробки зображень, методи розпізнавання образів для ідентифікації заборонених предметів на зображеннях, методи оптимізації моделей глибокого навчання для покращення точності та запобігання перенавчанню.

Наукова новизна.

Наукова новизна полягає в розробці двоетапної системи детекції

заборонених предметів SecureScanNet, де перша модель виконує класифікацію об'єктів, а друга — локалізацію, лише якщо класифікація підтвердила наявність цільового об'єкта. Така комбінація підвищує ефективність процесу виявлення заборонених предметів, забезпечуючи високу точність класифікації, локалізації і мінімізуючи кількість помилкових спрацювань. Такий підхід забезпечує модульність, ефективність у використанні обчислювальних ресурсів.

Практична значимість дослідження полягає в можливості впровадження розробленої інтелектуальної системи на митницях та в інших сферах, де необхідно здійснювати контроль заборонених предметів. Розроблена система може бути використана для автоматизації процесів митного контролю, що дозволить зменшити навантаження на працівників, підвищити швидкість перевірок і мінімізувати кількість помилкових спрацювань.

Структура.

Дана робота складається із вступу, аналітичного огляду, постановки задачі трьох основних розділів. У першому розділі проводиться огляд сучасних технологій автоматизації митного контролю та інтелектуальних систем для детектування заборонених предметів. Другий розділ присвячено теоретичним аспектам глибокого машинного навчання, а також розгляду моделей і методів, що використовуються для детектування. У третьому розділі описується програмна реалізація розробленої системи, аналізуються результати тестування та оцінюється її ефективність. Також із висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасні комп'ютерні технології автоматизації митного оформлення

З часу свого заснування в 1952 році, Всесвітня митна організація (ВМО) займається розробкою сучасних підходів для підтримки ефективної роботи митних органів, аналізуючи митну політику та практики на глобальному рівні, а також співпрацюючи з країнами-членами, торговими спільнотами та міжнародними організаціями. Зусилля зі спрощення та гармонізації митних процедур увінчалися підписанням Кіотської конвенції, яку Всесвітня митна організація ухвалила у 1973 році, а чинності вона набула у 1974 році. [1] Звісно визначну роль в цьому відіграють інформаційні технології. «Глобалізація, швидкі зміни в міжнародних торгових схемах та досягнення в галузі інформаційних технологій (IT) з того часу змусили ВМО та її членів переглянути та оновити Конвенцію». [1, с. 53]. Отже один з розділів цієї Конвенції присвячений інформаційним технологіям та ним передбачається, що використання інформаційних технологій є ключовою вимогою для полегшення та уніфікації митних процедур і сприяння розвитку торгівлі. Цей розділ вимагає від митних адміністрацій впроваджувати інформаційні технології для підтримки операцій у випадках, коли це є економічно доцільним і ефективним як для самих митних органів, так і для торговельної спільноти [1].

Модернізація інформаційних технологій та значне розширення його використання стали важливими елементами митної реформи, оскільки це дозволило автоматизувати процес митного оформлення — від подачі вантажної та митної декларацій до випуску товарів. Інформаційні технології також сприяють впровадженню систем контролю, заснованих на оцінці ризиків, що забезпечує вибірковість при фізичних перевірках, можливість проведення аудитів та підвищують ефективність контролю за доходами. Оновлені інформаційні системи також дозволяють зменшити час на митне оформлення та підвищити ефективність операцій. Багато країн, що розвиваються, впровадили різноманітні системи електронного обміну даними (Electronic Data Interchange), адаптовані до їхніх потреб. Однією з найпоширеніших серед використаного програмного забезпечення є система ASYCUDA (Automated System for Customs Data).

Автоматизація приносить значні позитивні зміни в транзитні операції. Деякі рішення охоплюють майже всі аспекти процесу. Наприклад, Європейський Союз створив «Нову комп'ютеризовану транзитну систему» (New Computerised Transit System), яка повністю автоматизована.

Щодо економік, які розвиваються, Конференція ООН з торгівлі та розвитку (UNCTAD) розробила додаткові транзитні модулі для ASYCUDA. Модуль MODTRS (транзит) опрацьовує транзитні документи разом з іншими модулями функцій ASYCUDA++[1]. «Однак митниці в економіках, що розвиваються, не використовують цю перевагу повною мірою». [2, с. 25].

Всі ці автоматизовані системи насамперед передбачають :

- здійснення обробки великих обсягів даних, пов'язаних з митними операціями, забезпечуючи швидкий доступ до інформації про товари та угоди;

- використання аналітичних алгоритмів для виявлення потенційних ризиків і попередження митних органів про можливі порушення;

- використання електронного обміну даними між різними учасниками митного процесу (митниця, банки, суб’єкти зовнішньо-економічної діяльності), що знижує людський фактор і помилки;

- можливість підключення до національних і міжнародних баз даних для отримання та перевірки інформації про товари та учасників торгівлі.

Також впровадження більш сучасних підходів і методів автоматизації митного контролю і митного оформлення можуть передбачати обробку інформації і аналіз даних із застосуванням штучного інтелекту, роботизації, використання хмарних технологій, використання скануючих систем із імплементованими інтелектуальними системами, блокчейн технології.

1.2 Інтелектуальні технології детектування заборонених речей

1.2.1 Загальна інформація про скануючи системи.

Слід зазначити, що провідні світові виробники систем безпеки розробляють інноваційні скануючі технології для митного контролю та захисту кордонів. Такі компанії, як Smiths Detection, Rapiscan Systems, Leidos, Nuctech та Analogic Corporation, займаються створенням високотехнологічних рентгенівських

сканерів, систем для комп'ютерної томографії (КТ), а також рішень на основі штучного інтелекту (ШІ).

Сучасні скануючі системи для безпеки, використовуються в аеропортах, на митниці та в інших транспортних вузлах, базуються на принципах рентгенівського випромінювання, комп'ютерної томографії (КТ) та інтелектуальних алгоритмах обробки зображень. Ці системи призначені для автоматичного виявлення заборонених або потенційно небезпечних предметів, таких як зброя, вибухівка, наркотики або інші контрабандні товари. Основна мета таких систем — забезпечити високу швидкість та точність аналізу зображень без втручання людини, що значно підвищує ефективність контролю на кордонах.

Рентгенівські системи працюють на основі випромінювання рентгенівських променів, які проходять через об'єкти і створюють зображення на детекторах. Коли рентгенівські промені проходять через об'єкти, різні матеріали поглинають випромінювання по-різному залежно від їхньої щільності та складу. Щільніші матеріали, такі як метали або кераміка, поглинають більше випромінювання, утворюючи яскравіші ділянки на зображенні, тоді як менш щільні матеріали, такі як пластмаси або тканини, відображаються темнішими. Цей ефект дозволяє операторам і системам візуально або автоматично ідентифікувати підозрілі предмети [10].

Проте сучасні системи рентгенівського сканування інтегрують інтелектуальні алгоритми обробки зображень, що дозволяють автоматично аналізувати ці дані без необхідності людського втручання. Спеціальні алгоритми аналізують рентгенівські зображення, порівнюючи їх з шаблонами, що відповідають відомим загрозам. Наприклад, у системах безпеки в аеропортах алгоритми можуть розпізнавати загрозливі форми або контури заборонених предметів, таких як пістолети, ножі або вибухові пристрої. Оператори отримують тільки відфільтровану інформацію, що знижує ризик людських помилок і підвищує ефективність перевірки. Такі технології використовуються для аналізу багажу, вантажів, а також особистих речей пасажирів на кордонах та інших критичних об'єктах [11].

Комп'ютерна томографія (КТ) є ще більш передовою технологією для сканування багажу та вантажів, порівняно зі звичайними рентгенівськими

системами. На відміну від рентгенівських зображень, що створюють двовимірні (2D) зображення, КТ використовує серію зображень, зроблених під різними кутами, для побудови тривимірної (3D) моделі об'єкта. Це дозволяє системі краще ідентифікувати форми і структури всередині багажу або вантажу, оскільки вона може "бачити" об'єкт з усіх боків. Це особливо важливо для виявлення прихованих предметів, які можуть бути загорнуті або розташовані так, що їх не можна легко виявити за допомогою звичайних рентгенівських систем [12].

КТ-системи часто використовуються для аналізу підозрілих предметів з більшою точністю. Інтегровані алгоритми штучного інтелекту дозволяють КТ-сканерам автоматично визначати типи матеріалів на основі щільності об'єктів і їх структури. Це особливо корисно для виявлення вибухових речовин, які можуть бути приховані серед інших об'єктів, або наркотиків, що знаходяться в контейнерах. За допомогою 3D-зображення система може навіть автоматично ідентифікувати, чи є об'єкт загрозою, на основі його форми та розташування у багажі, зменшуючи кількість помилкових спрацювань та покращуючи точність роботи системи [12].

Алгоритми штучного інтелекту (ШІ) та глибокого навчання стали невід'ємною частиною сучасних скануючих систем для безпеки. Такі алгоритми, як конволюційні нейронні мережі (CNN), використовуються для аналізу зображень з рентгенівських або КТ-сканерів. Вони здатні автоматично розпізнавати і класифікувати об'єкти на основі їхніх форм і контурів. CNN, як один з типів нейронних мереж, спеціально розроблений для обробки візуальних даних і навчений розпізнавати складні візерунки та закономірності на зображеннях [13].

Алгоритми глибокого навчання здатні вивчати особливості загроз на основі великої кількості попередньо зібраних даних. Вони можуть автоматично адаптуватися до нових типів загроз і покращувати свої прогнози з часом. Наприклад, система може навчитися розпізнавати нові види вибухових речовин або зброї на основі зображень, які вона аналізує. Крім того, алгоритми здатні працювати в реальному часі, аналізуючи сотні або тисячі зображень за дуже короткий проміжок часу, що підвищує швидкість роботи системи і знижує потребу в людському втручанні [13].

Інтеграція машинного навчання з технологіями обробки неструктурованих даних дозволяє значно підвищити ефективність скануючих систем. Неструктуровані дані — це зображення, відео або текстові файли, що не мають чіткої форми для аналізу, але можуть містити важливу інформацію. Алгоритми класифікації, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit), використовуються для аналізу таких даних та автоматичного ухвалення рішень [13].

Наприклад, в контексті рентгенівських сканерів для безпеки, ці алгоритми можуть бути використані для аналізу зображень багажу або вантажу та автоматичної класифікації об'єктів на основі їхніх візуальних характеристик. Це допомагає системі швидко ідентифікувати, які об'єкти є безпечними, а які вимагають додаткового аналізу або перевірки. Також алгоритми можуть враховувати різні фактори ризику і вказувати на предмети, які можуть бути небезпечними або порушують правила, що робить процес безпеки більш швидким та ефективним [13].

Рентгенівські сканери, в поєднанні з інтелектуальними системами обробки даних, дозволяють підвищити рівень безпеки на транспортних вузлах і кордонах. Використання різних методів спектрального аналізу та аналізу щільності матеріалів дозволяє системам безпеки точно визначати типи об'єктів, їх хімічний склад і можливі загрози. Такі системи використовуються не лише для виявлення зброї чи наркотиків, а й для ідентифікації токсичних хімічних речовин або біологічних загроз [14].

1.2.2 Аналіз існуючих архітектур нейронних мереж.

ResNet є однією з найуспішніших архітектур глибокого навчання, що дозволяє будувати дуже глибокі нейронні мережі. Основна ідея ResNet полягає у використанні резидуальних блоків, які вирішують проблему зникнення градієнта під час тренування. Дозволяє створювати мережі глибиною до 152 шарів (ResNet-152), зберігаючи високу точність. Використовує «skip connections» для прямої передачі градієнта, зменшуючи ймовірність переобчислення. Часто використовується для виділення ознак із зображень у системах детекції. Для задач детекції заборонених об'єктів ResNet можна адаптувати через transfer learning або

fine-tuning на спеціалізованому датасеті[15]

YOLO є архітектурою реального часу, що дозволяє одночасно визначати об'єкти та їх місцезнаходження у зображенні. Розбиває зображення на сітку, у кожній комірці якої передбачає координати bounding boxes та класи об'єктів. Використовує Darknet-53 як основа, що складається із 53 згорткових шарів. Підходить для додатків, де важлива висока швидкість обробки. YOLO можна використовувати для аналізу великого потоку зображень (наприклад, у реальному часі на митниці). Модель легко адаптується до нестандартних класів шляхом повторного навчання на спеціалізованому датасеті[16].

SSD об'єднує переваги одночасного передбачення координат і класів об'єктів. Використовує декілька рівнів ознакових карт для роботи з об'єктами різного розміру. Баланс між швидкістю обробки і точністю. Основа моделі — VGG16 із додатковими згортковими шарами. Підходить для задач із великою варіацією розмірів заборонених предметів. Проте модель має певні обмеження при роботі з дрібними об'єктами через втрату інформації при згортках та недостатню деталізацію ознакових карт на ранніх рівнях. Може бути застосована для мобільних пристроїв завдяки оптимізації обчислювальних ресурсів[17].

EfficientDet є моделлю, що оптимізує точність і швидкість шляхом використання EfficientNet як базової архітектури. Використовує механізм BiFPN (Bi-directional Feature Pyramid Network) для об'єднання ознак. Низькі обчислювальні витрати і висока продуктивність. Масштабованість моделі дозволяє адаптувати її до задач з різними обчислювальними ресурсами. Ідеально підходить для мобільних систем детекції заборонених об'єктів, де важливі низькі затрати пам'яті. Легко адаптується до нових датасетів завдяки гнучкому підходу до налаштування моделі[18].

Сімейство моделей R-CNN (Fast R-CNN, Faster R-CNN) забезпечує високу точність детекції через виділення регіональних пропозицій. Використовує базову CNN-архітектуру (ResNet, VGG). Fast R-CNN прискорює процес за рахунок спільного обчислення ознак. Faster R-CNN додає Region Proposal Network (RPN) для автоматичного визначення регіонів. Найбільш ефективна у випадках, коли важлива висока точність локалізації заборонених предметів. Використовується для задач, де потрібна обробка зображень із деталізацією об'єктів[19].

Зведена інформація щодо основних характеристик, переваг та недоліків нейронних мереж зазначена у таблиці 1.1.

Таблиця 1.1 Порівняння основних характеристик нейромереж

Архітектура	Точність локалізації	Швидкість	Використання ресурсів	Переваги	Недоліки
ResNet	Висока	Низька	Високі	Стабільність, глибина аналізу	Обчислювально важка
YOLO	Середня	Висока	Середні	Робота в реальному часі	Точність нижча за R-CNN
SSD	Середня	Висока	Середні	Баланс швидкості і точності	Обмеження на дрібних об'єктах
EfficientDet	Висока	Висока	Низькі	Ефективність, масштабованість	Вимагає тонкого налаштування
R-CNN	Висока	Низька	Високі	Точність, детекція дрібних об'єктів	Не підходить для реального часу

У таблиці 1.2 ми можемо бачити що вищевказані моделі мають певні переваги та і одночасні недоліки щодо універсальності, балансу швидкості. Точність моделей не перевищує 90% а у деяких моделей 80%.

Таблиця 1.2 Порівняння технічних характеристик нейромереж

Модель	Шари	Кількість параметрів	Епохи	Регуляризація	Обчислювальні ресурси	Час навчання	Точність (%)	Швидкість (FPS)
ResNet	50–152	~25 млн (ResNet-50)	90–150	Dropout, BatchNorm	Tesla K80	2–3 тижні	80–85	10–15
YOLOv3	53	~62 млн	200–300	L2-регуляризація	Tesla V100	2–3 дні	70–80	45–60
SSD	~32	~24 млн	120–200	Dropout, Hard mining	GTX 1080	5–7 днів	75–80	30–40
EfficientDet	~200–250	~43 млн (D4)	200–300	DropConnect, MixUp	TPU	1–2 дні	80–85	35–50
Faster R-CNN	~128	~50 млн	90–120	Dropout, BatchNorm	Titan X	1–2 тижні	85–90	5–10

Також ми можемо бачити (таб. 1.2.) велику складність зазначених моделей, з великою кількістю шарів та параметрів, що вимагає великої кількості даних для навчання, часу та обчислювальних ресурсів.

1.3 Постановка задачі

Результати проведеного огляду сучасних технологій автоматизації митного

контролю підтверджують актуальність розробки інформаційної технології для автоматичного детектування заборонених предметів у багажі та вантажах. Зокрема, перспективним є використання методів глибокого машинного навчання для покращення точності й швидкості виявлення заборонених об'єктів та потенційно небезпечних аномалій, що знижує кількість помилкових спрацювань та навантаження на працівників.

Метою дослідження є розробка інтелектуальної системи SecureScanNet для автоматичної детекції заборонених предметів на зображеннях, отриманих за допомогою скануючих пристроїв.

Основні завдання роботи включають:

1. Провести аналіз існуючих архітектур нейронних мереж (ResNet, YOLO, SSD, EfficientDet, R-CNN) для вирішення задач детекції заборонених об'єктів.
2. Підготувати навчальну та тестову вибірки, у разі наявності дисбалансу у розподілі класів «заборонених» предметів здійснити аугментацію даних.
3. Створити моделі глибокого навчання здатні здійснювати детекцію заборонених предметів та відображати локалізацію таких предметів.
4. Провести тестування розробленої системи на реальних даних.
5. Здійснити аналіз результатів.

2 ТЕХНОЛОГІЇ ГЛИБОКОГО МАШИННОГО НАВЧАННЯ

2.1 Основні принципи глибокого машинного навчання

Останнім часом спостерігається стрімкий розвиток технологій, відомих як "глибоке навчання", що є частиною більш широкої сфери штучного інтелекту та машинного навчання. Машинне навчання базується на побудові моделей, що здатні навчатися з даних і робити прогнози або приймати рішення без явного програмування.

Для досягнення значних результатів у різних сферах, глибоке навчання стало основною технологією, що дозволяє автоматизувати складні завдання та знаходити приховані закономірності у великих обсягах даних. Зокрема, глибоке навчання для обробки природної мови демонструє значний прогрес у розв’язанні завдань, пов'язаних із розумінням та обробкою тексту [20]. Такі технології здатні аналізувати великі масиви даних. Глибокі нейронні мережі навчаються на історичних ринкових даних, включаючи цінові тренди, обсяги торгів та інші ринкові показники [21]. Глибоке навчання стає все більш актуальним інструментом у науці про Землю, оскільки воно надає потужні можливості для аналізу великих даних і виявлення нових наукових знань щодо екосистем і клімату [22]. Технології глибокого навчання та згорткових нейромереж є переломними у сфері комп'ютерного зору [23].

Ключовими принципами машинного навчання є навчання моделей на великих обсягах даних, автоматичне вилучення ознак, а також здатність моделі адаптуватися до нових умов. Завдяки цим принципам моделі машинного навчання можуть успішно знаходити закономірності в складних наборах даних і забезпечувати високу точність передбачень. Сучасні методи машинного навчання також передбачають використання глибоких нейронних мереж для навчання на величезних обсягах даних, що дозволяє значно підвищити точність та ефективність моделей.

У цьому розділі буде приділено увагу основним принципам машинного навчання, які є фундаментом для успішного застосування цих технологій у реальних сценаріях.

2.1.1 Навчання на великих обсягах даних

Сучасні моделі машинного навчання, особливо глибокі нейронні мережі, потребують значних обсягів даних для тренування. Використання великих наборів даних дозволяє досягати високої точності передбачень та забезпечує здатність моделі узагальнювати знання на нові дані. Важливість навчання на великих даних полягає в тому, що без достатньої кількості якісних даних модель може демонструвати високі результати на тренувальних прикладах, але виявлятися неефективною на нових тестових даних. Як зазначають Goodfellow та ін. (2016), великі обсяги даних забезпечують більшу стійкість моделей до перенавчання та дозволяють уникнути переобтяження на обмежених наборах даних [24].

Це особливо важливо для глибоких нейронних мереж, які можуть обробляти складні структури та залежності в даних завдяки своїм багатошаровим архітектурам [24]. Наприклад, Krizhevsky та ін. (2012) зазначають, що навчання нейронних мереж на великому наборі даних ImageNet дозволило значно покращити точність класифікації об'єктів на зображеннях, оскільки мережа мала доступ до мільйонів зразків для тренування [23].

Ще однією перевагою великих даних є можливість більш точної оцінки ризиків та побудови більш складних моделей, які здатні вирішувати завдання в реальному часі. Такі технології вже застосовуються у фінансовому секторі, де глибокі нейронні мережі використовуються для прогнозування ринкових трендів та аналізу великих обсягів історичних даних [26].

Незважаючи на численні переваги, навчання на великих наборах даних також має низку викликів. Одним із основних є висока обчислювальна складність. Для обробки мільйонів або навіть мільярдів зразків потрібні потужні апаратні ресурси, зокрема GPU та TPU, які здатні виконувати паралельні обчислення. Krizhevsky та ін. (2012) відзначають, що без використання графічних процесорів навчання їхньої моделі на ImageNet зайняло б кілька тижнів, тоді як використання GPU скоротило цей час до кількох днів [23]. Це вказує на те, що для ефективної роботи з великими даними потрібна відповідна інфраструктура.

Іншим викликом є зберігання та передача великих обсягів даних. Навіть якщо модель може обробляти такі дані, їхнє зберігання та організація можуть

вимагати значних ресурсів. Для цього використовуються сучасні хмарні платформи, такі як Google Cloud ML або AWS SageMaker, які дозволяють здійснювати навчання на великих наборах даних за допомогою хмарних ресурсів [24]. Крім того, для швидкої передачі даних застосовуються розподілені обчислювальні системи, наприклад Apache Spark, що забезпечує паралельну обробку даних.

Попередня обробка великих наборів даних також є важливим етапом у машинному навчанні. Оскільки великі обсяги даних часто містять пропущені значення, помилки або надлишкову інформацію, важливо здійснювати їх очистку та нормалізацію перед навчанням моделі. Як зазначає Géron (2019), правильно підготовлені дані значно підвищують ефективність моделей, оскільки покращують якість вхідних ознак [25]. Окрім цього, популярним підходом є аугментація даних — штучне збільшення кількості тренувальних прикладів через модифікацію наявних зразків (наприклад, для зображень можна застосовувати повороти, зміни яскравості тощо) [24].

2.1.2 Автоматичне вилучення ознак

Автоматичне вилучення ознак є одним із ключових аспектів глибокого машинного навчання та глибоких нейронних мереж. Воно полягає в тому, що система самостійно вивчає важливі ознаки (характеристики) з вхідних даних без необхідності їх попереднього визначення або ручної обробки. Цей процес значно полегшує аналіз складних даних, таких як зображення, текст або аудіо, оскільки дозволяє моделі самостійно виділяти релевантну інформацію з великих і складних наборів даних.

Автоматичне вилучення ознак дозволяє моделям машинного навчання уникати класичної проблеми ручного вибору ознак, яка була типовою для традиційних методів. Як зазначають Bengio Y., Courville A., та Vincent P. у своїй роботі "Representation Learning: A Review and New Perspectives" (2013), автоматичне вилучення ознак є особливо важливим для аналізу великих наборів даних, таких як зображення та текст, оскільки це дозволяє моделі самостійно "навчатися" корисним характеристикам даних без людського втручання [26]. У класичних методах машинного навчання процес створення ознак потребував

експертних знань, що обмежувало масштабованість і застосовність моделей до різних типів даних.

Глибокі нейронні мережі, завдяки своїм багатошаровим архітектурам, мають здатність автоматично вилучати все більш абстрактні та високорівневі ознаки з вхідних даних. На початкових рівнях мережа може навчитися розпізнавати простіші шаблони, такі як контури або градієнти на зображеннях, тоді як на більш глибоких рівнях — виявляти складні структури та об'єкти.

Автоенкодери є одним із найпоширеніших інструментів для автоматичного вилучення ознак. Автоенкодери успішно використовуються для вилучення латентних представлень даних, що дозволяє глибоким нейронним мережам навчатися корисним ознакам без втрати важливої інформації. Це є ключовою перевагою автоенкодерів у порівнянні з традиційними методами обробки даних [24].

LeCun Y та інші у своїй роботі "Deep Learning" (2015) також зазначають, що головною перевагою глибоких нейронних мереж є їх здатність автоматично вивчати ознаки на різних рівнях абстракції. Нижні шари мережі можуть навчитися розпізнавати базові патерни, такі як контури або текстури, тоді як верхні шари можуть вивчати більш складні структури та об'єкти [27]. Це особливо важливо для завдань комп'ютерного зору, таких як класифікація зображень або сегментація, де важливою є здатність мережі самостійно ідентифікувати ключові ознаки на різних рівнях деталізації.

Глибокі нейронні мережі використовують конволюційні шари, які здатні автоматично вилучати ознаки на основі локальних піксельних залежностей на зображеннях. Це робить їх дуже ефективними для задач обробки зображень, де розпізнавання об'єктів відбувається завдяки виявленню локальних характеристик, таких як границі, кути або текстури. Автоматичне вилучення ознак у таких мережах робить можливим обробку складних наборів даних без попередньої обробки вручну [24].

Глибокі нейронні мережі та автоенкодери дозволяють створювати ефективні системи, які можуть працювати з неструктурованими даними та вивчати ключові ознаки без людського втручання [25]. Це дозволяє значно скоротити час на підготовку даних і підвищити точність моделей.

2.1.3 Глибина мережі та нелінійність

Глибокі нейронні мережі (ГНМ) відрізняються від інших видів машинного навчання своєю здатністю обробляти складні залежності та структури даних завдяки використанню великої кількості шарів та нелінійних активаційних функцій. Глибина мережі дозволяє моделі вивчати більш складні відносини в даних, а нелінійні активаційні функції забезпечують можливість моделювати залежності, що неможливо відобразити за допомогою лінійних моделей.

Глибина мережі визначається кількістю шарів, що з'єднують вхідний сигнал із вихідним. Для шару l (від layer, в перекладі з англійської - шар) формула лінійного перетворення виглядатиме так:

$$y^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}, \quad (2.1)$$

де :

- $W^{(l)}$ – матриця ваг для l-го шару,
- $x^{(l-1)}$ – вхідні дані на l-му шарі, що є виходом з попереднього шару *(l-1)*,
- $b^{(l)}$ – вектор зміщення (bias) для l-го шару,
- $y^{(l)}$ – вихід на l-му шарі після лінійного перетворення.

Тоді, наприклад, для трьохшарової моделі (якщо шар l - середній) кожен шар виконує лінійне перетворення і попередній шар можна розписати таким чином:

$$y^{(l-1)} = W^{(l-1)}x^{(l-2)} + b^{(l-1)}, \quad (2.2)$$

підставляючи це у перше рівняння, отримаємо:

$$y^{(l)} = W^{(l)}(W^{(l-1)}x^{(l-2)} + b^{(l-1)}) + b^{(l)}. \quad (2.3)$$

Як бачимо, це рівняння є все ще лінійним. Якщо продовжити для наступного шару *(l-1)*, то результатом буде:

$$y^{(l+1)} = W^{(l+1)}(W^{(l)}(W^{(l-1)}x^{(l-2)} + b^{(l-1)}) + b^{(l)}) + b^{(l+1)}. \quad (2.4)$$

Тобто, всі перетворення є лінійними комбінаціями попередніх вхідних даних. У результаті використання глибоких багатошарових мереж без нелінійних функцій активації втрачає сенс, оскільки всі ці шари можна замінити одним лінійним шаром, по суті змінюється тільки кількість нейронів.

Якщо додається нелінійна функція активації f, то формула для шару з нелінійною активацією стане такою:

$$y^{(l)} = \mathrm{f}^{(l)}(W^{(l)}x^{(l-1)} + b^{(l)}), \quad (2.5)$$

де:

$f^{(l)}$ - нелінійна функція активації шару l.

Отже глибокі нейронні мережі використовують нелінійні функції активацій для забезпечення здатності моделі моделювати складні взаємозв'язки між вхідними та вихідними сигналами. Найбільш поширеними функціями активацій є:

- ReLU (рис. 2.1.) (Rectified Linear Unit): $f(x) = max(0, x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases}$, (2.6)

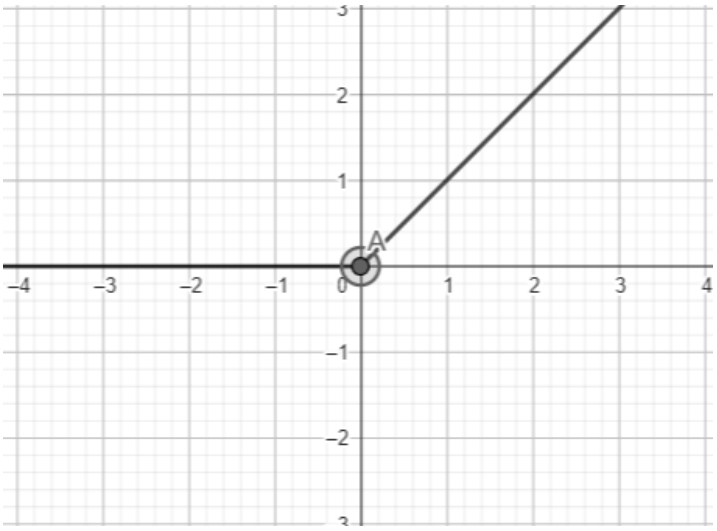

Рисунок 2.1 – Графік функції активації ReLU (Rectified Linear Unit)

- сигмоїдна функція (рис 2.2.): $f(x) = \frac{1}{1+e^{-x}}$, (2.7)

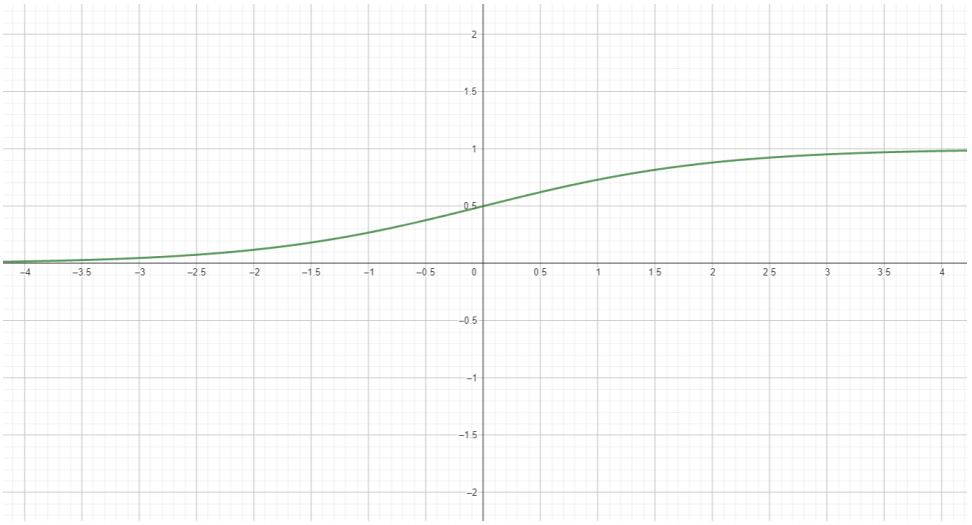

Рисунок 2.2 – Графік сигмоїдної функції активації

- tanh (гіперболічний тангенс) (рис. 2.3.): $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, (2.8)

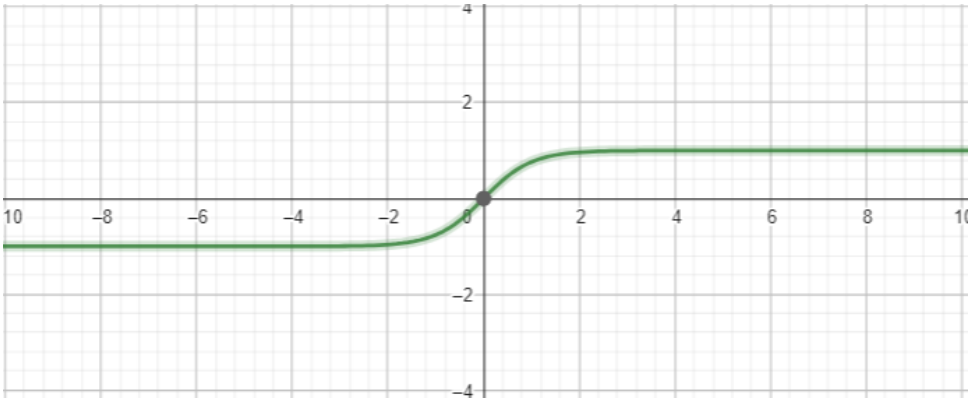

Рисунок 2.3 – Графік функції активації tanh «гіперболічний тангенс»

Як зазначає Ruder (2016), нелінійність є ключовим компонентом для побудови багатошарових мереж, що дозволяє їм ефективно вивчати більш складні шаблони в даних [28].

Нелінійні функції дозволяють мережі "ламати" лінійність між шарами, що робить можливим вивчення складних і багатовимірних взаємозв'язків.

Включення більшої кількості прихованих шарів дозволяє моделі будувати складніші функції та вивчати багатовимірні залежності в даних. Як зазначає Bottou (2010) у своїй роботі, глибокі мережі мають значно більшу здатність до узагальнення та забезпечують кращу продуктивність у порівнянні з менш глибокими моделями [29].

Важливо розуміти, що із збільшенням кількості шарів нейронної мережі виникають складнощі з оптимізацією. Зокрема, зростає ризик перенавчання, а також збільшується потреба в обчислювальних ресурсах для обробки великих даних і ефективної оптимізації ваг мережі.

2.1.4 Висока обчислювальна складність і необхідність масштабованості

Раніше у роботі вже зазначалось, що глибоке навчання є одним із найвибагливіших напрямів машинного навчання з точки зору обчислювальних ресурсів. Це пояснюється великою кількістю параметрів моделей та значними обсягами даних, що використовуються для їх навчання. Глибокі нейронні мережі складаються з кількох шарів нейронів, кожен з яких виконує численні операції обчислення. Унаслідок цього обчислювальна складність моделей глибокого

навчання зростає експоненційно зі збільшенням кількості шарів та розміром вхідних даних. У цьому контексті виникає потреба в масштабованості — можливості ефективно навчати моделі на великих обсягах даних та використовувати потужні обчислювальні ресурси для прискорення навчання.

Однією з основних проблем при навчанні великих моделей є розподілене навчання на кластерах комп'ютерів, що дозволяє паралельно обробляти великі набори даних та зменшувати час навчання моделі. Проблему масштабованості розподілених нейронних мереж, зокрема проблеми синхронізації між вузлами, обчислювальних навантажень та ефективної обробки великих наборів даних було досліджено у роботі Dean та ін. Важливим аспектом масштабованості є розподілені алгоритми навчання, такі як поширення зворотної похибки (backpropagation) на декілька обчислювальних пристроїв. Це дозволяє оптимізувати процес навчання моделі на декількох вузлах одночасно, що, в свою чергу, зменшує час навчання моделі [32].

Для прикладу, система TensorFlow (використовувалась для навчання моделей в даній роботі), була створена для ефективного навчання великих моделей глибокого навчання на великих кластерах серверів. Ця платформа дозволяє паралельно навчати моделі на кількох графічних процесорах (GPU) або центральних процесорах (CPU), що дозволяє швидко обробляти великі обсяги даних. TensorFlow також забезпечує високу масштабованість завдяки можливості налаштування моделі під конкретні ресурси. Крім того, ця система підтримує розподілене навчання, що робить її важливим інструментом для навчання моделей на великих наборах даних [32].

Одним із найвідоміших прикладів успішного використання високих обчислювальних ресурсів та масштабованості є модель AlexNet, розроблена Krizhevsky та ін. [23], яка досягла визначних результатів у задачі класифікації зображень на наборі даних ImageNet. Ця модель складалася з восьми шарів і використовувала графічні процесори для прискорення навчання, що дозволило обробляти мільйони зображень за відносно короткий час. Зокрема, модель AlexNet навчалась на двох графічних процесорах NVIDIA, що забезпечило можливість паралельної обробки даних та суттєво скоротило час навчання.

Математично, обчислювальна складність моделей глибокого навчання

залежить від кількості параметрів W у кожному шарі. Для простого шару з щільними зв'язками обчислювальна складність може бути оцінена як $O(nW)$, де:

n - це кількість нейронів у шарі, а

W - кількість вагових параметрів. При зростанні кількості шарів зростає, зростає і загальна обчислювальна складність моделі.

Основні підходи для зменшення обчислювальної складності включають використання меншої кількості параметрів (за рахунок компресії моделей або застосування спеціалізованих архітектур, таких як згорткові нейронні мережі) та використання потужних обчислювальних ресурсів, таких як графічні процесори (GPU) або спеціалізовані інтегральні схеми (ASIC).

Таким чином, високі вимоги до обчислювальних ресурсів та потреба в масштабованості є одними з ключових викликів при розробці та навчанні моделей глибокого машинного навчання. Використання потужних обчислювальних кластерів та розподілених систем, таких як TensorFlow, дозволяє ефективно обробляти великі обсяги даних та пришвидшувати процес навчання моделей, що є критично важливим у сучасних реаліях роботи з великими наборами даних.

2.1.5 Неперервне навчання та адаптація

Неперервне навчання (Continual Learning) є важливим аспектом глибокого машинного навчання, що дозволяє моделям пристосовуватися до нових даних та умов без необхідності повного перенавчання. У традиційному машинному навчанні моделі зазвичай навчаються на фіксованому наборі даних, однак у багатьох реальних застосуваннях дані постійно змінюються, і модель повинна бути здатною до адаптації без втрати попередньої інформації. Це особливо важливо у випадках, коли потрібно працювати в динамічних середовищах або коли доступ до даних обмежений і неможливо зберігати всі дані для подальшого навчання.

Однією з головних проблем неперервного навчання є "катастрофічне забування" (catastrophic forgetting), модель, навчаючись на нових даних, починає втрачати інформацію, отриману з попередніх наборів даних. Ця проблема полягає

в тому, що ваги нейронної мережі адаптуються до нових прикладів, але втрачають здатність коректно класифікувати раніше вивчені зразки. Дослідження методів подолання цієї проблеми, зокрема через введення додаткових регуляризуючих компонентів у функцію втрат, які контролюють зміни ваг для раніше вивчених класів привели до використання методу «Еластичної консолідації ваг» (Elastic Weight Consolidation (EWC)), який дозволяє зберігати важливі для попередніх завдань ваги моделі, зменшуючи ризик втрати знань під час навчання на нових даних [34]. Математичне вираження EWC через наступну модифіковану функцію втрат

$$L(\theta) = L_{new}(\theta) + \lambda \sum_i F_i(\theta_i - \theta_{i,old})^2 \quad (2.9)$$

,

де $L_{new}(\theta)$ — функція втрат для нових даних, λ — коефіцієнт регуляризації, F_i — наближене значення Фішера для ваги θ_i, а $\theta_{i,old}$ — попередні ваги. Ця формула показує, як модель зберігає важливі для попередніх завдань ваги, обмежуючи їх зміну під час навчання на нових даних.

Ця формула показує, як модель зберігає важливі для попередніх завдань ваги, обмежуючи їх зміну під час навчання на нових даних.

Існує кілька підходів до неперервного навчання, кожен з яких намагається вирішити проблему катастрофічного забування. Parisi та ін. [35] у своїй роботі описують три основні підходи:

1. Методи збереження ваг (Regularization-based methods): Ці методи, як, наприклад, вже згаданий EWC, використовують регуляризацію для контролю за змінами ваг моделі. Вони обмежують зміни ваг, які є важливими для попередніх завдань, тим самим зменшуючи ймовірність забування.

2. Репрезентаційні методи (Replay-based methods): У цих методах модель періодично переглядає приклади з попередніх задач, щоб зберегти знання. Для цього використовується буфер пам'яті, де зберігаються критичні приклади з минулого, які подаються моделі разом із новими даними під час навчання.

3. Архітектурні методи (Architectural methods): Ці методи передбачають адаптивну зміну архітектури моделі. Вони можуть додавати нові нейрони або цілі шари, коли з'являються нові завдання. Завдяки цьому нові знання інтегруються в існуючу архітектуру без значного впливу на вже вивчені знання.

Крім проблеми катастрофічного забування, неперервне навчання також передбачає здатність моделей покращувати свої навички на основі попереднього

досвіду — це відоме як "навчання навчатись" (Learning to Learn). Підходи, які дозволяють моделям машинного навчання краще адаптуватися до нових завдань, використовуючи знання, набуті з попередніх задач можуть включати мета-навчання, де модель вчиться вибирати оптимальні параметри або методи навчання для нових даних, базуючись на попередньому досвіді [36].

Такий підхід є особливо важливим для адаптивних систем, які повинні працювати в умовах постійно змінюваного середовища. Важливо зазначити, що «навчання навчатись» може істотно скоротити час адаптації моделі до нових завдань, оскільки модель використовує вже набуті знання.

2.1.6 Інтерпретація та пояснюваність моделей

У сучасних умовах глибокого навчання виникає потреба не лише в створенні високоточної моделі, але й у забезпеченні її інтерпретації та пояснюваності. Інтерпретація та пояснюваність моделей машинного навчання є ключовими аспектами, які дозволяють зрозуміти, як модель приймає рішення, що особливо важливо для критичних застосувань у медицині, фінансах та правовій системі. Одним з головних викликів у цьому контексті є складність глибоких нейронних мереж, яка значно ускладнює їх пояснення у порівнянні з традиційними алгоритмами, такими як лінійна регресія або дерева рішень.

Одним із провідних методів, що дозволяють пояснювати результати роботи моделей, є метод локальних апроксимацій, таких як LIME (Local Interpretable Model-agnostic Explanations) (рис. 2.4.). Локальні інтерпретації можна створювати для складних моделей шляхом апроксимації їх простими моделями, такими як лінійна регресія. Ідея полягає в тому, що для кожного окремого передбачення моделі створюється новий набір спрощених даних, на яких будується локально лінійна модель, що дозволяє зрозуміти, які фактори вплинули на передбачення у конкретному випадку [36]. Формула для побудови локальної інтерпретації може бути записана наступним чином:

$$\hat{f}(x) = \sum_{i=1}^{n} w_i \, x_i, \tag{2.10}$$

де:

$\hat{f}(x)$ – лінійна апроксимація складної моделі *f(x)* поблизу точки *x,*

w_i – ваги, що визначають важливість кожної ознаки x_i

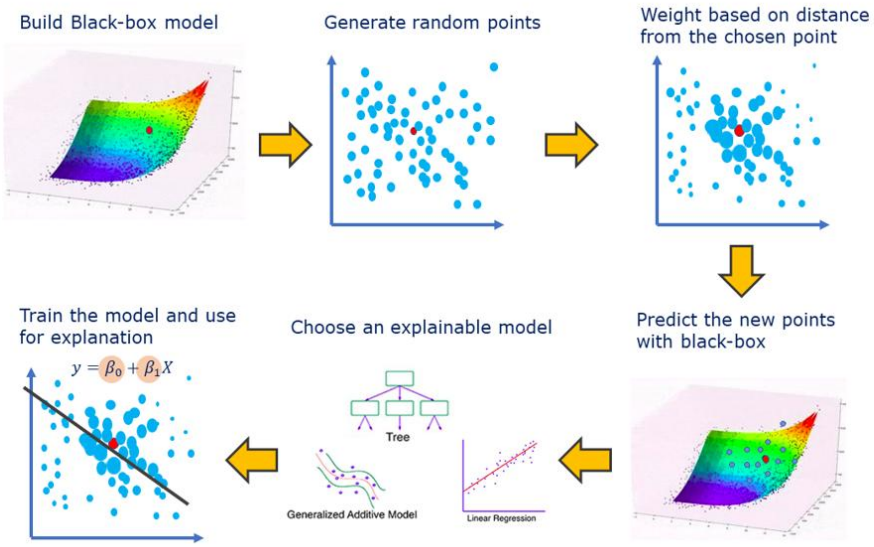

Рисунок 2.4 – Приклад локальної інтерпретації моделі з використанням LIME [38]

Крім того, важливу роль у пояснюваності моделей відіграє використання методів візуалізації внутрішніх процесів у нейронних мережах. Різні підходи до візуалізації, зокрема методи активаційних карт (activation maps) (рис. 2.5.) дозволяють візуалізувати, які частини вхідного зображення найбільше вплинули на рішення моделі. Цей підхід широко використовується для пояснення моделей у завданнях комп'ютерного зору, де необхідно зрозуміти, які частини зображення сприяли певній класифікації [39].

Методи візуалізації, такі як Grad-CAM (Gradient-weighted Class Activation Mapping), дозволяють визначити "гарячі" зони на зображенні, що були найбільш важливими для передбачення. Формально, Grad-CAM обчислює ваги градієнтів по відношенню до певного класу:

$$L^c_{Grad-CAM} = \text{ReLU}\left(\sum_k \alpha^c_k A^k\right) \tag{2.11}$$

де α^c_k — середнє значення градієнтів по активаціям A^k для певного класу c, а функція ReLU зберігає тільки позитивні значення, щоб виділити важливі активації.

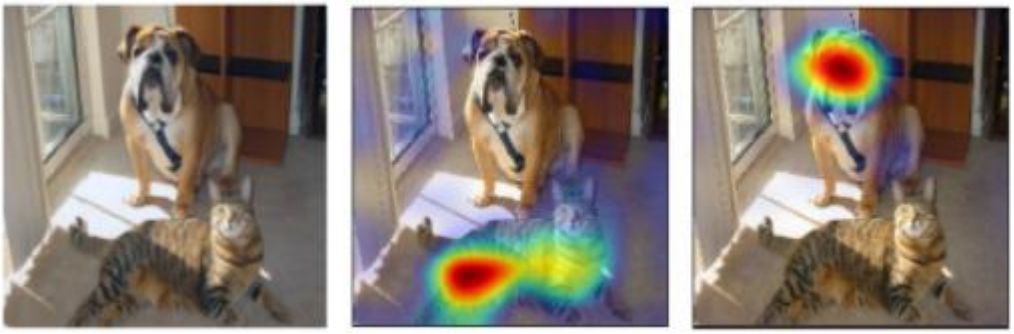

Рисунок 2.5 – Приклад активаційної карти з використанням Grad-CAM для пояснення класифікації зображень (зліва – оригінальне зображення, по центру активаційна карта для кота, праворуч – для собаки) [40]

Ще один підхід до пояснюваності моделей пов'язаний із створенням моделей, які від самого початку є інтерпретованими, або з подальшою розробкою методів для покращення інтерпретації вже існуючих моделей. Одним із запропонованих підходів є розробка спеціальних інтерпретованих архітектур, таких як дерева рішень (decision trees) (рис. 2.6.) для отримання простих та зрозумілих правил прийняття рішень.

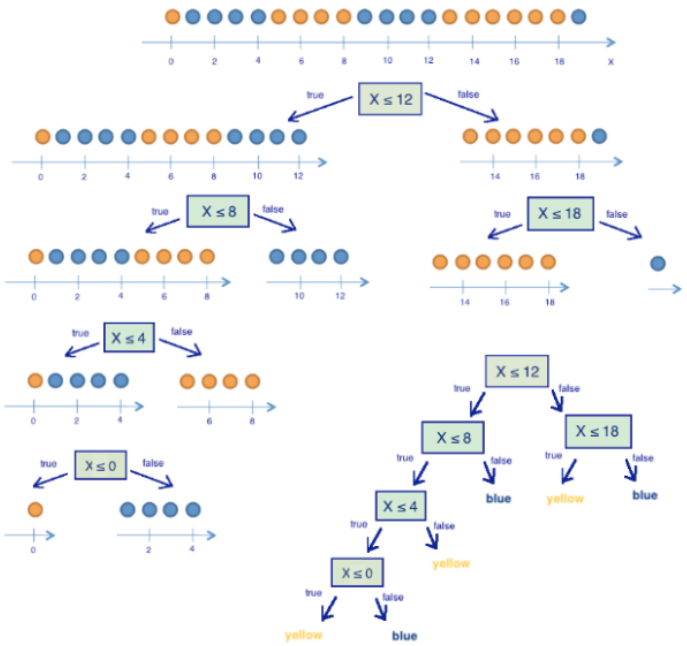

Рисунок 2.6 – Приклад інтерпретованого рішення у вигляді дерева рішень. Ілюстрація побудови повного дерева для задачі прогнозування кольору кульки по координаті «X» [41]

Пояснюваність моделей також є критично важливою для реальних систем, де прийняття рішень має значні наслідки. Наприклад, у медичних системах

пояснюваність дозволяє лікарям краще розуміти причини, чому модель зробила те чи інше передбачення, що допомагає у прийнятті обґрунтованих рішень. Тому важливо не лише створювати високоточні моделі, але й робити їх прозорими для кінцевого користувача, що дозволить краще довіряти результатам машинного навчання та приймати відповідні заходи.

2.2 Моделі глибокого машинного навчання

2.2.1 Згорткові нейроні мережі

Згорткові нейронні мережі (Convolutional Neural Networks, CNNs) є спеціально розробленими для обробки даних, що мають просторову структуру, зокрема зображень. CNN побудовані на концепції використання шарів згортки (convolutional layers), що дозволяє виявляти локальні особливості вхідних даних, та шарів об’єднання (pooling layers), що зменшують розмірність представлення та роблять модель менш чутливою до незначних зсувів об'єктів. Завдяки цим елементам CNN здатні автоматично виділяти особливості з вхідних даних, що робить їх надзвичайно ефективними для задач комп’ютерного зору, таких як класифікація зображень, розпізнавання об’єктів та сегментація [42].

Конволюційні нейромережі здобули популярність завдяки своєму успіху у різноманітних завданнях комп’ютерного зору, включаючи класифікацію зображень, розпізнавання об'єктів, сегментацію, виявлення об'єктів, аналіз відео та медичну діагностику на основі зображень. Наприклад, у роботі «ImageNet Classification with Deep Convolutional Neural Networks» [23] CNN були використані для класифікації зображень у великому наборі даних ImageNet, що стало значним досягненням у галузі, а також продемонструвало здатність CNN автоматично виділяти багаторівневі особливості із зображень.

Основні компоненти CNN включають:

- конволюційні шари: Це шари, які виконують операцію згортки над вхідними даними (рис. 2.7.). Згортка — це математична операція, що дозволяє виділяти локальні патерни з даних, такі як краєві або текстурні елементи зображення (рис. 2.8.). Кожен фільтр у згортковому шарі навчається виявляти певний набір особливостей, які можуть бути корисними для подальших рівнів мережі.

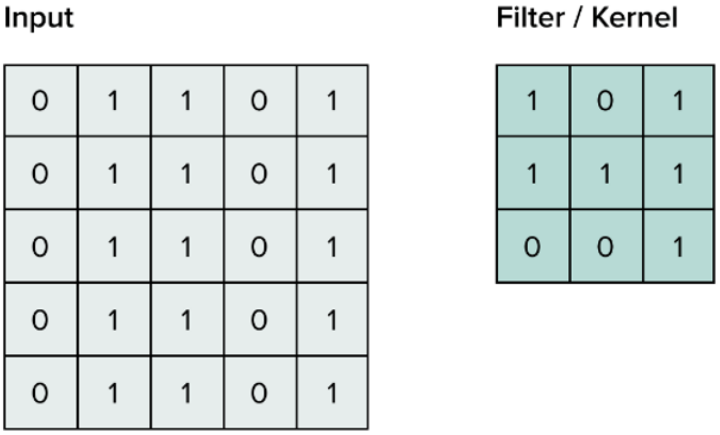

Рисунок 2.7 – Вхідні дані(зображення) зліва, фільтр або ядро праворуч

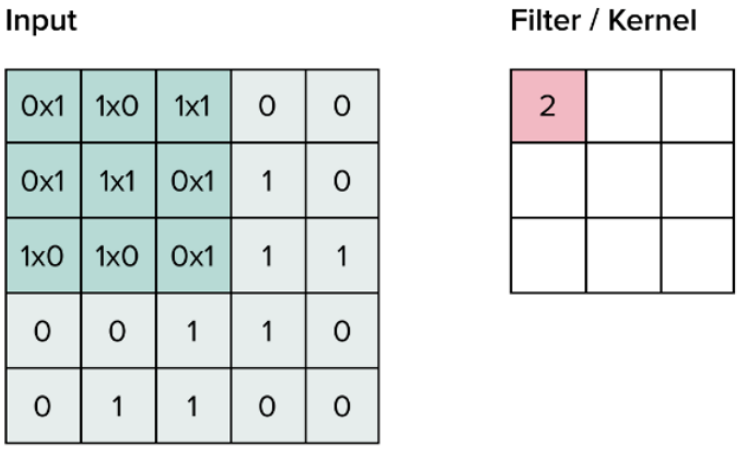

Рисунок 2.8 – Зліва операція згортки (конволюції), праворуч – результат

- шари об’єднання: Основна роль шарів об’єднання — зменшення розмірності простору ознак, що знижує обчислювальні витрати та підвищує стійкість мережі до незначних змін у положенні об'єктів. Найпоширенішими є максимальне об’єднання (max pooling) (рис. 2.9.) і середнє об’єднання (average pooling) (рис. 2.10.).

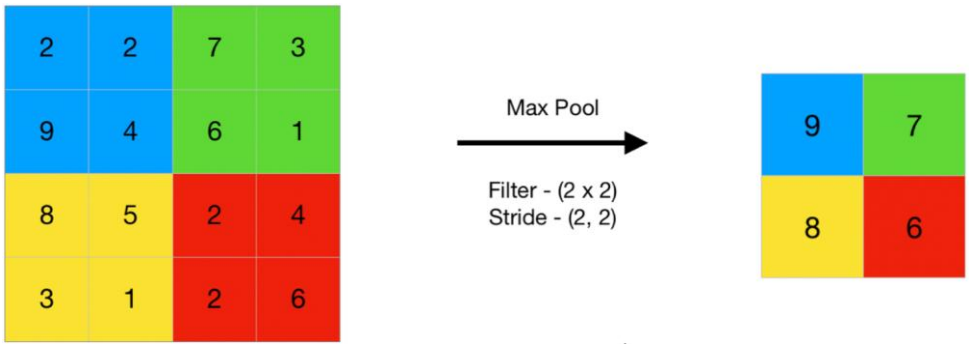

Рисунок 2.9 – Принцип роботи шару максимальне об’єднання

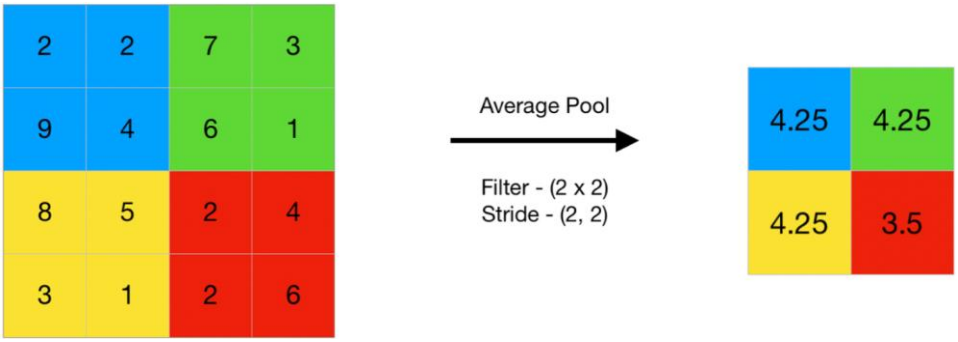

Рисунок 2.10 – Принцип роботи шару середнє об’єднання

- шари активації: Після кожної згортки застосовується нелінійна функція активації, наприклад, ReLU (Rectified Linear Unit) (рис. 2.1.), яка додає нелінійність до моделі. Це є важливим для виявлення складних патернів у даних [42, 43].

Серед найбільш відомих архітектур CNN можна виділити AlexNet, VGG і ResNet. Модель AlexNet, запропонована у 2012 році Кріжецьким та ін., стала значним проривом у задачі класифікації зображень, вигравши конкурс ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [23]. Її структура складається з декількох згорткових шарів, шарів об’єднання та повнозв’язних шарів, що дозволило досягти надзвичайно високих результатів.

Архітектура VGG, розроблена Симоняном та Зіссерманом, стала відома завдяки використанню дуже глибоких згорткових шарів однакової конфігурації, що значно покращує якість класифікації. Кожен шар у VGG містить невеликі фільтри (3x3), але завдяки великій глибині мережа може виявляти багатошарові патерни, що робить її дуже ефективною для обробки складних зображень [43].

Модель ResNet (рис. 2.11.), запропонована Хе та його колегами, відрізняється використанням залишкових блоків (residual blocks), що дозволяє зберігати інформацію з попередніх шарів без втрати якості навчання. Завдяки залишковим з’єднанням, модель може уникати проблем деградації, що виникають під час навчання дуже глибоких нейронних мереж [44]. ResNet показала видатні результати в задачах класифікації зображень і залишається однією з найбільш популярних архітектур CNN на сьогодні.

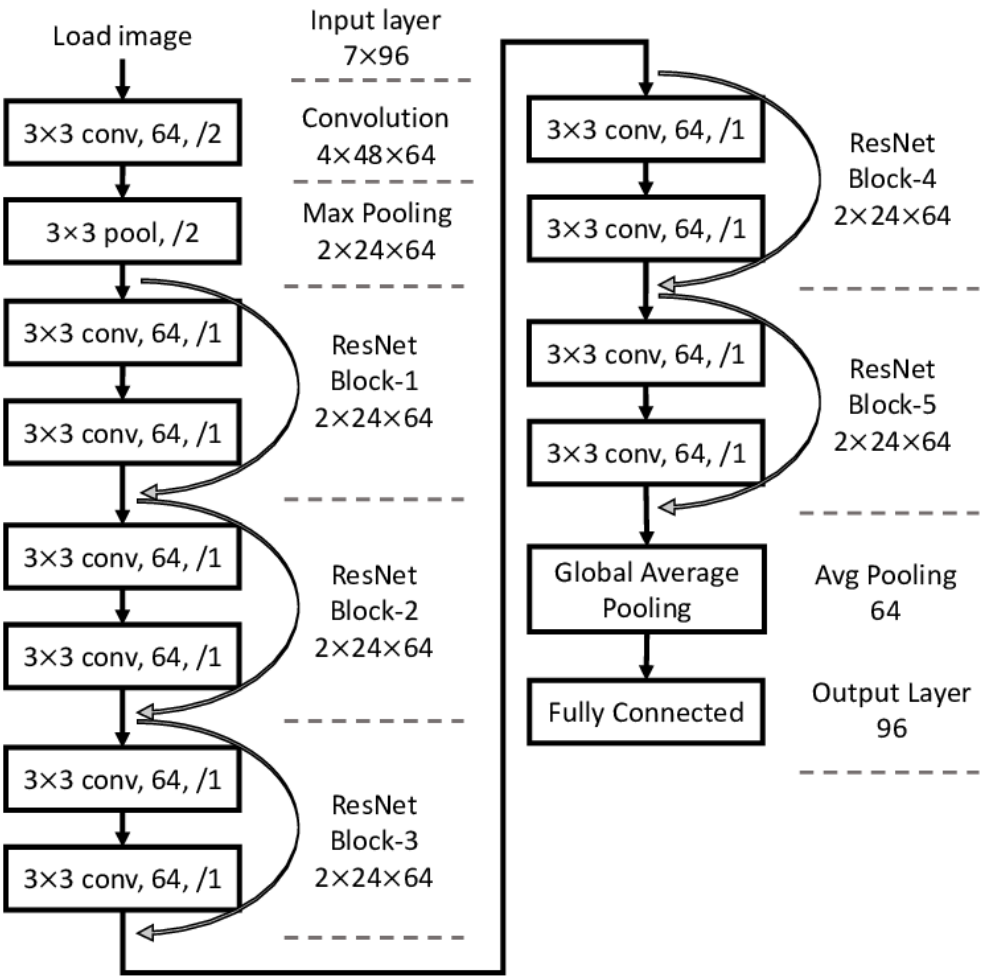

Рисунок 2.11 – Приклад архітектури моделі ResNet12

2.2.2 Рекурентні нейронні мережі

Рекурентні нейронні мережі (Recurrent Neural Networks, RNNs) – це вид нейронних мереж, призначений для обробки послідовних даних, таких як текст, звук або часові ряди. Вони відрізняються від традиційних нейронних мереж своєю здатністю зберігати контекст попередніх елементів послідовності завдяки зворотним зв'язкам. На кожному кроці RNN отримує як поточний вхід, так і інформацію про попередній стан, що дозволяє мережі запам’ятовувати важливу інформацію з попередніх кроків для прийняття рішень на наступних. Однак, базові RNN мають недолік: при обробці довгих послідовностей виникає проблема зникнення градієнтів, що ускладнює ефективне навчання на великих обсягах даних [44]. Проблема зникнення градієнтів знижує точність, оскільки нейронні мережі втрачають здатність зберігати корисну інформацію на віддалених етапах. Це робить RNN особливо корисними для обробки природної мови та роботи з часовими рядами, де попередні контексти є вирішальними для розуміння

поточних значень [45].

Рекурентні нейронні мережі використовуються для вирішення широкого спектру завдань, пов'язаних з послідовностями. Одним з основних напрямків є обробка природної мови (Natural Language Processing, NLP), де RNN застосовуються для аналізу тексту, машинного перекладу, аналізу настроїв та розпізнавання мови. В завданнях прогнозування часових рядів RNN дозволяють аналізувати історичні дані та передбачати значення на основі попередніх патернів. Наприклад, RNN ефективно використовуються для прогнозування цін на фондових ринках, де кожне нове значення залежить від попередніх. Ще одним популярним застосуванням є генерація тексту: навчена на великій кількості текстових даних, RNN може автоматично створювати тексти з урахуванням контексту та структури мови [45]. Деякі сучасні генеративні моделі, такі як GPT-4, використовують вдосконалені RNN-архітектури для генерації змістовних текстів, відповідей на запити та навіть для написання коду.

Основна особливість RNN полягає у здатності зберігати попередні стани в пам’яті, щоб забезпечити обробку послідовностей. Класичні RNN працюють за схемою, де кожен нейрон передає свої результати не тільки наступному шару, але й самому собі, зберігаючи пам’ять про попередні кроки. Це дозволяє накопичувати інформацію протягом послідовності. Проте через проблему зникнення градієнтів, коли під час зворотного поширення градієнти поступово зменшуються до нуля, класичні RNN неефективні для довгих послідовностей. Розробка архітектур LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit) стала вирішальним кроком для вирішення цієї проблеми, надавши RNN здатність зберігати довготривалі залежності [45].

Архітектура LSTM (Long Short-Term Memory) (рис. 2.12) запропонована у 1997 році Хохрайтером і Шмідхубером, і стала революційним рішенням проблеми зникнення градієнтів. На відміну від стандартних RNN, LSTM містить комірки пам'яті, які можуть зберігати або видаляти інформацію на основі вхідних сигналів. Завдяки механізмам затворів (input, forget, та output gates), LSTM контролює, які дані залишати в пам'яті, а які ігнорувати. Це дозволяє LSTM зберігати важливу інформацію про довготривалі залежності у послідовності, зокрема у завданнях обробки мовних послідовностей і прогнозування часових

рядів [45, 46]. LSTM стали основою для багатьох сучасних систем, включаючи Apple Siri та Google Assistant, де вони допомагають в обробці та генерації мовних команд.

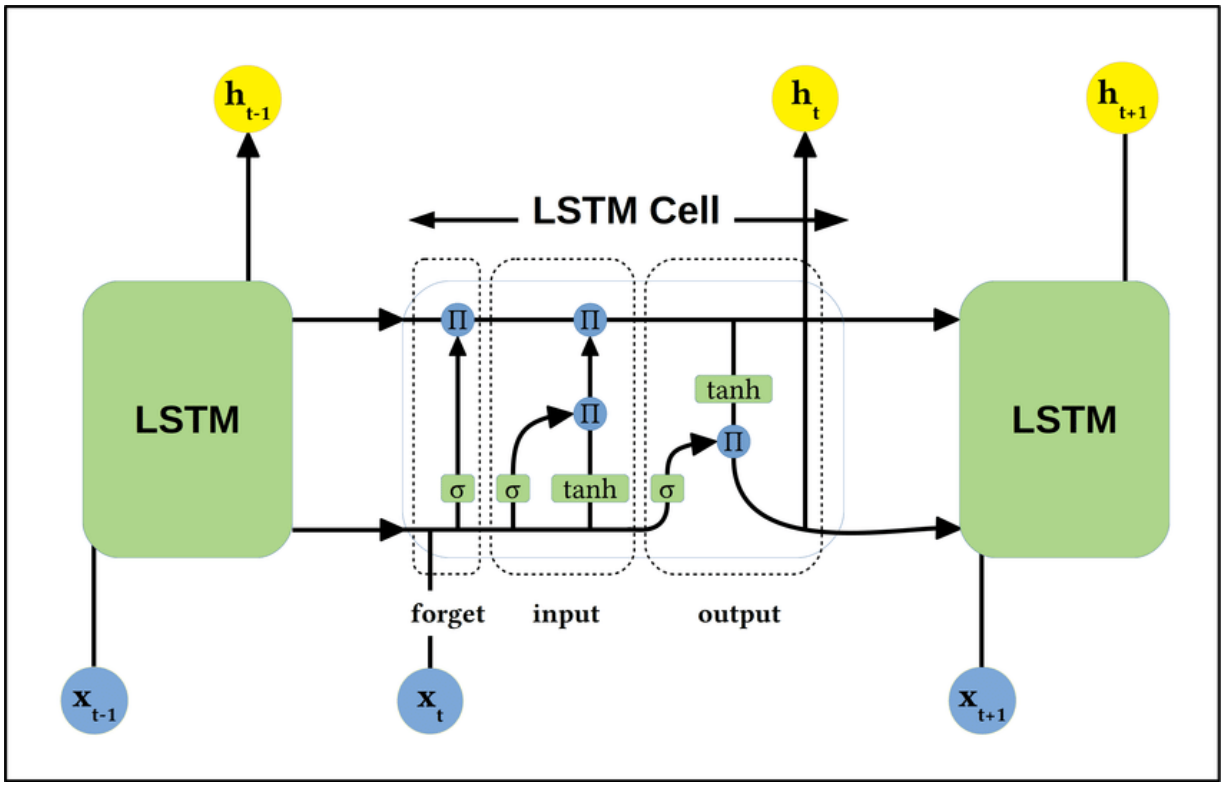

Рисунок 2.12 – Базова архітектура моделі Long Short-Term Memory

GRU (Gated Recurrent Unit) (рис. 2.13) запропонована командою під керівництвом Чо у 2014 році, є полегшеною версією LSTM, що поєднує деякі механізми затворів, спрощуючи процес навчання. GRU має лише два затвори – оновлення і скидання (update and reset gates), що знижує обчислювальні витрати і дозволяє швидше навчати модель, залишаючи при цьому здатність до роботи з довготривалими залежностями. GRU показали ефективність у багатьох завданнях, де довгі послідовності є критичними, як-от машинний переклад, де потрібно зберігати контекст всього речення [46]. GRU часто обирають для систем реального часу, де важливо забезпечити швидкість обробки даних.

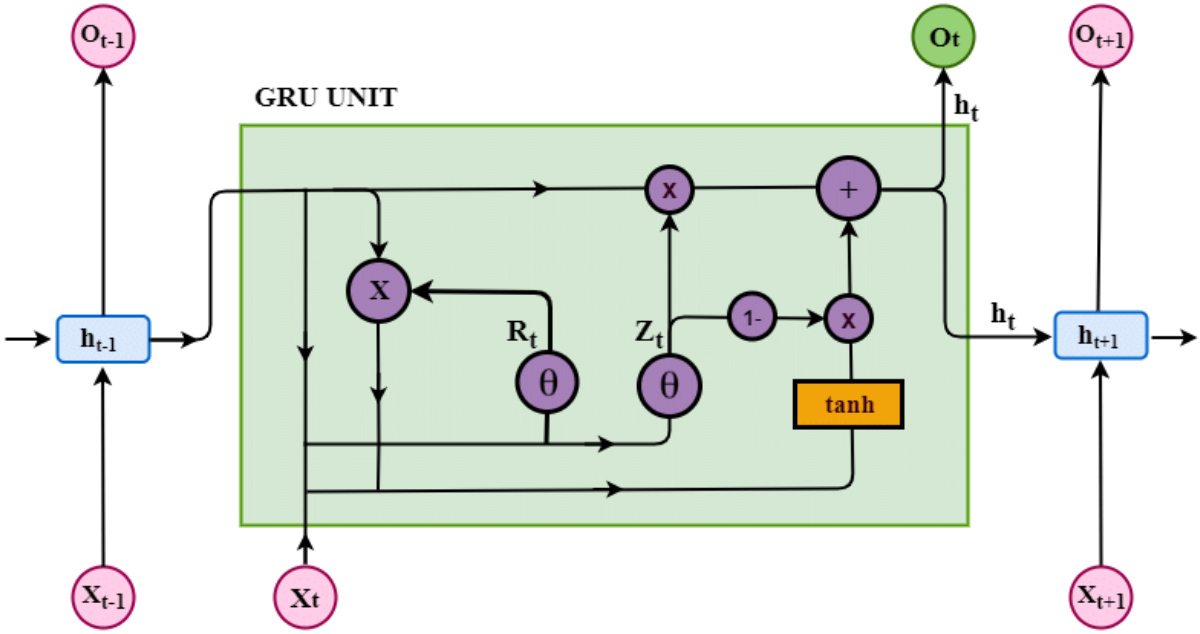

Рисунок 2.13 – Базова архітектура моделі Gated Recurrent Unit

Однією з головних переваг RNN є здатність ефективно працювати з послідовними даними будь-якої довжини та зберігати інформацію про попередні етапи, що дозволяє глибше аналізувати часові рядки і мовні патерни. Проте класичні RNN мають суттєвий недолік у вигляді зникнення градієнтів, який обмежує їхню здатність запам'ятовувати інформацію на довгих етапах. Використання архітектур LSTM і GRU значно покращує ситуацію, проте ці моделі є ресурсомісткими. LSTM і GRU потребують більше обчислювальних потужностей, особливо при роботі з великими наборами даних, що може бути обмеженням для деяких застосувань [45, 46, 47].

2.2.3 Автоенкодери (Autoencoders)

Автоенкодери (Autoencoders) — це тип нейронних мереж, який навчається кодувати вхідні дані в компактне, менш вимірне представлення (латентний простір) та відновлювати їх назад у початковий вигляд. На відміну від звичайних нейронних мереж, автоенкодери не вимагають наявності міток для навчання, оскільки вхідні дані одночасно є і мітками. Це дозволяє моделі знаходити найбільш важливі особливості даних, що можуть бути корисними для багатьох задач, таких як зменшення розмірності даних, виділення ознак та виявлення аномалій [24].

Ключова ідея автоенкодерів полягає у мінімізації відмінності між

початковими даними та їх відновленою версією. Автоенкодери навчаються мінімізувати помилку реконструкції, що стимулює їх до виділення основної інформації, уникаючи другорядних деталей. Така здатність автоенкодерів до "стискання" даних дозволяє використовувати їх у задачах, де є необхідність в аналізі великої кількості даних та в отриманні компактного, інформативного представлення [48].

Завдяки своїй здатності виділяти ключові особливості даних, автоенкодери широко застосовуються для зменшення розмірності даних. Наприклад, при обробці зображень автоенкодери можуть використовуватись для скорочення розмірності, зберігаючи основні деталі зображення. Це дозволяє зменшити кількість обчислювальних ресурсів, необхідних для подальшої обробки, а також може бути корисним для візуалізації та класифікації даних у задачах машинного навчання [24].

Іншим важливим застосуванням автоенкодерів є виявлення аномалій. Оскільки автоенкодери навчаються відновлювати типові (звичайні) дані з високою точністю, вони зазвичай дають велику помилку реконструкції для «аномальних» зразків. Це робить автоенкодери ефективними для виявлення незвичайних або підозрілих даних у системах фінансового моніторингу, мережевої безпеки, медичної діагностики та інших сферах, де аномалії мають велике значення [47].

Автоенкодери (рис. 2.14.) також застосовуються для видалення шуму з даних (денойзінг-автоенкодери). У таких випадках модель навчається відновлювати чисту версію даних із «зашумлених» зразків, що корисно для підвищення якості зображень або сигналів. Після додавання випадкового шуму до вхідних даних, автоенкодер навчається видаляти ці випадкові елементи, зберігаючи при цьому основну інформацію, що особливо корисно у завданнях обробки медичних зображень або даних супутникових знімків [48]. Нижче наведені основні компоненти:

Енкодер приймає вхідні дані та стискає їх, переводячи у компактне представлення у латентному просторі. Зазвичай енкодер складається з кількох шарів нейронів, які поступово зменшують розмірність даних, залишаючи найважливіші характеристики. Таким чином, енкодер відповідає за виділення

інформативних ознак, що є ключовими для відновлення початкових даних.

Декодер приймає стислий вигляд і реконструює його у початковий вигляд. Це зворотний процес до енкодера, який поступово збільшує розмірність даних, наближаючи їх до оригінальних вхідних даних. Мета декодера полягає у мінімізації втрати інформації та відтворенні структури вхідних даних якомога точніше [24].

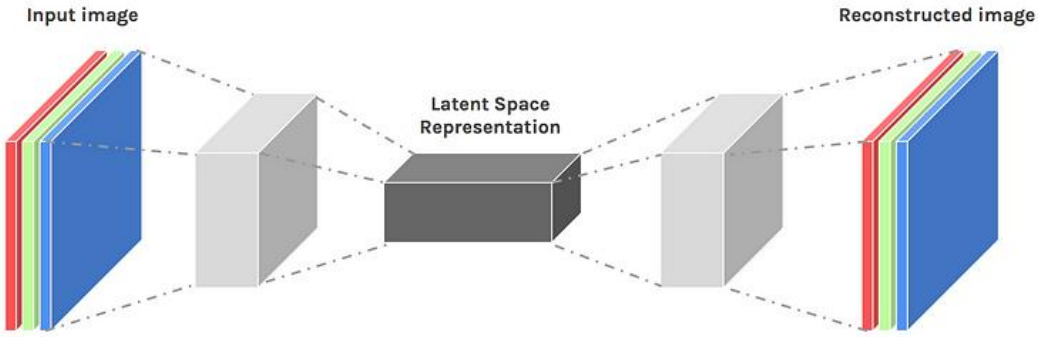

Рисунок 2.14 – Базова архітектура автоенкодера

Цей процес навчання дозволяє автоенкодерам знаходити баланс між ефективним стисненням даних та мінімізацією помилки реконструкції, що забезпечує їх ефективність для багатьох завдань обробки даних. Ось декілька варіантів архітектур автоенкодерів:

Варіаційні автоенкодери (VAE) є розширенням звичайних автоенкодерів, яке дозволяє моделі працювати з ймовірнісними представленнями. VAE додає випадковість до процесу кодування, що дозволяє генерувати нові дані, подібні до тих, що є у навчальному наборі. Вони використовуються для задач генеративного моделювання, таких як генерація зображень та текстів, і особливо корисні для створення нових зразків у складних доменах, де доступ до реальних даних обмежений [48].

Спарс-автоенкодери (Sparse Autoencoders) додають регуляризацію, яка обмежує кількість активних нейронів у латентному просторі. Це дозволяє моделі створювати більш зрозумілі представлення та зосереджуватися лише на найбільш важливих характеристиках даних, що забезпечує виділення інформативних ознак і мінімізацію надлишкової інформації. Такі автоенкодери особливо корисні у випадках, коли дані містять багато зайвої або неінформативної інформації [24].

Денойзінг-автоенкодери (Denoising Autoencoders) навчаються відновлювати

чисті дані з зашумлених вхідних даних. Ця архітектура була розроблена для задач, де важливо відновлювати сигнал, видаляючи випадковий шум. У процесі навчання до вхідних даних додається шум, і модель оптимізує реконструкцію чистих зразків. Денойзінг-автоенкодери широко використовуються у медицині для обробки рентгенівських зображень та у комп'ютерному зорі для покращення якості зображень [49].

Отже автоенкодери є важливим інструментом для зменшення розмірності, виділення ознак та видалення шуму з даних. Вони мають здатність відновлювати інформацію про основні характеристики, завдяки чому широко застосовуються у таких галузях, як комп'ютерний зір, фінанси та біомедичні дослідження. Варіаційні автоенкодери, спарс-автоенкодери та денойзінг-автоенкодери мають специфічні властивості, які дозволяють застосовувати їх до різноманітних завдань. VAE знаходять застосування у генеративному моделюванні, спарс-автоенкодери – у задачах класифікації та виявлення ознак, а денойзінг-автоенкодери – для підвищення якості даних шляхом видалення шуму.

2.2.4 Генеративні змагальні мережі (Generative Adversarial Networks, GANs)

Генеративні змагальні мережі (Generative Adversarial Networks, GANs) — це сучасна архітектура нейронних мереж, що складається з двох моделей: генератора (Generator) та дискримінатора (Discriminator), які працюють у конкурентному режимі (рис. 2.15.). Генератор навчається створювати зразки, які мають вигляд, максимально наближений до реальних, тоді як дискримінатор намагається відрізнити справжні зразки від тих, що були згенеровані генератором. Генератор приймає на вхід випадковий шум, перетворюючи його у вихідні дані, схожі на реальні. Дискримінатор же отримує як справжні, так і згенеровані дані і навчається розпізнавати підробки. У результаті процесу змагання дискримінатор поступово підвищує свою здатність розпізнавати фальшиві зразки, а генератор, у свою чергу, навчається створювати зразки, які все важче відрізнити від справжніх [50].

Такий механізм змагального навчання перетворює процес генерації даних на своєрідну гру з нульовою сумою, де кожна з моделей прагне покращити свої

результати, підвищуючи при цьому якість згенерованих зразків. Завдяки цьому GAN стали потужним інструментом у завданнях генерації зображень, покращення їх якості та створення аугментованих даних. З моменту свого створення GAN отримали широке застосування в багатьох сферах, де необхідно працювати з високоякісними візуальними даними, і продовжують вдосконалюватись [50].

Процес навчання GAN можна представити як гру з нульовою сумою, де генератор намагається мінімізувати помилку дискримінатора, тоді як дискримінатор намагається максимізувати її. Це створює складний процес оптимізації, оскільки поліпшення генератора призводить до зростання його здатності "обманювати" дискримінатор, а поліпшення дискримінатора робить йому складніше відрізняти підроблені зразки від справжніх.

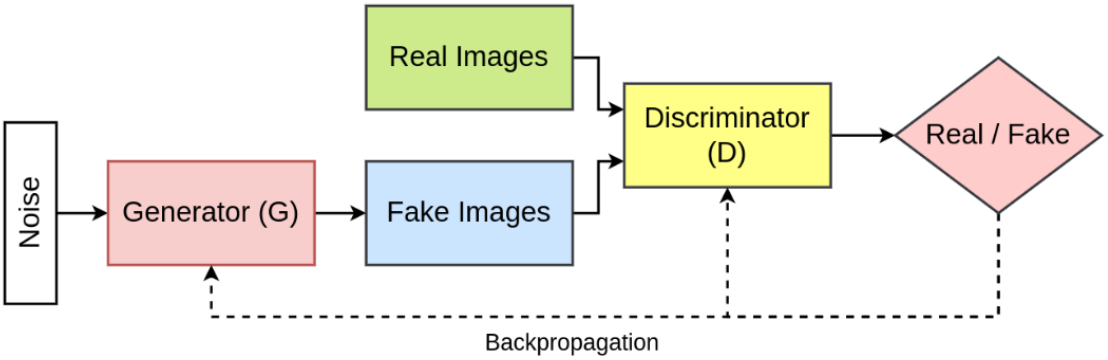

Рисунок 2.15 – Базова архітектура Генеративної змагальної мережі

GAN знайшли застосування у багатьох сферах завдяки їхній здатності генерувати реалістичні зразки. Основні напрямки застосування GAN включають:

- генерація зображень: GAN стали основою для створення нових зображень з реалістичними деталями, які є візуально подібними до справжніх фотографій. Це відкриває можливості для генерації фотореалістичних облич людей, які ніколи не існували, що корисно в кіноіндустрії та цифровому дизайні, а також для створення анонімних зображень або віртуальних персонажів.
- покращення якості зображень: Використання GAN для суперрезолюції зображень дозволяє збільшувати їх роздільну здатність, відновлюючи дрібні деталі. Це знайшло широке застосування в медичній діагностиці, де висока роздільна здатність зображень дозволяє отримувати точнішу інформацію для прийняття рішень.

- стилізація зображень: GAN також використовуються для стилізації зображень, наприклад, для перетворення фотографій у певний художній стиль (картинні фільтри, комікси тощо). Це особливо популярно у сфері цифрового мистецтва та дизайну, де стилізація дозволяє змінювати вигляд зображень під конкретний візуальний стиль.
- розширення наборів даних: GAN застосовуються для генерації додаткових зразків у навчальних наборах, що особливо корисно при роботі з невеликими наборами даних. У медицині, наприклад, де важко отримати велику кількість анонімізованих зображень, GAN можуть генерувати синтетичні зразки, які допомагають збільшити кількість даних для навчання моделей [51].

2.2.5. Трансформери (Transformers)

Трансформери — це сучасні архітектури нейронних мереж (рис. 2.16.), розроблені для ефективної обробки послідовних даних та взаємозалежностей між елементами в послідовності. Завдяки механізму уваги (attention mechanism) трансформери здатні ефективно обробляти довгі залежності, що є суттєвою перевагою перед рекурентними нейронними мережами (RNN), які обмежені при роботі з великими послідовностями через поступову втрату інформації на ранніх етапах обробки [53]. Трансформери дозволяють моделі одночасно враховувати усі позиції в послідовності за допомогою уваги, що суттєво покращує точність та швидкість обробки даних. Завдяки цій особливості, трансформери замінили RNN у багатьох завданнях, де необхідно обробляти великі послідовності даних.

Механізм уваги дозволяє трансформерам фокусуватися на різних частинах вхідних даних, визначаючи, які частини є найважливішими для поточного завдання. Це стало можливим завдяки Self-Attention, що забезпечує доступ до кожного елементу послідовності без обмеження на порядок проходження. Оскільки трансформери не залежать від послідовності обробки даних, вони використовують позиційне кодування для визначення позицій слів у реченні. Це дозволяє трансформеру враховувати порядок слів у послідовності, що є важливим для збереження граматичної та контекстуальної правильності. Після механізму уваги інформація передається через фідфорвардні шари, що дозволяє трансформеру здійснювати додаткову обробку та виділяти інформативні

особливості тексту [52]. Таким чином, трансформери є ефективним інструментом для обробки природної мови (Natural Language Processing, NLP) і виконання складних завдань, таких як машинний переклад, резюмування тексту та класифікація документів [52].

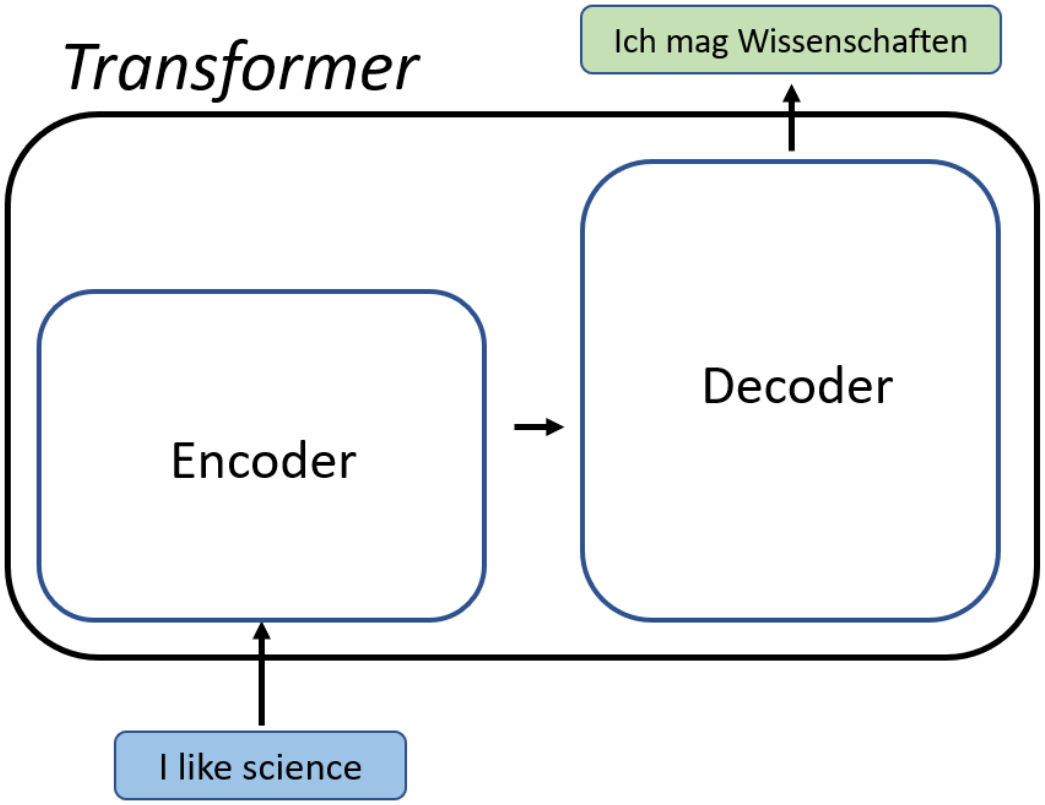

Рисунок 2.16 – Спрощене відображення моделі трансформера

Трансформери широко використовуються у задачах обробки природної мови та стали базою для багатьох передових NLP-моделей завдяки своїй універсальності та здатності працювати з довгими послідовностями. Основні напрями застосування трансформерів включають:

- обробка природної мови: Трансформери стали стандартом для вирішення задач NLP завдяки своїй здатності розуміти та обробляти текст, зберігаючи взаємозв'язок між словами у довгих послідовностях. Це дозволяє їм виконувати завдання, такі як аналіз настроїв, категоризація тексту та відповіді на запити.

- машинний переклад: У задачах перекладу трансформери використовуються для створення моделей, здатних генерувати високоточні переклади, що зберігають як граматичні, так і контекстуальні особливості мови.

Завдяки здатності моделювати залежності на різних рівнях, трансформери можуть перекладати складні фрази з урахуванням всього контексту [53].

- класифікація тексту: Трансформери також використовуються для класифікації документів та текстових даних. Моделі на основі трансформерів можуть розпізнавати та аналізувати ключові теми та зміст тексту, що дозволяє ефективно категоризувати великі обсяги текстових даних за допомогою попередньо навчених моделей.

Нижче наведені більш відомі приклади: BERT (Bidirectional Encoder Representations from Transformers, GPT (Generative Pre-trained Transformer) T5 (Text-To-Text Transfer Transformer).

2.3 Методи глибокого машинного навчання

2.3.1 Градієнтний спуск та його варіанти

Градієнтний спуск (Gradient Descent) — це один з найпоширеніших методів оптимізації, який використовується у задачах машинного та глибокого навчання для налаштування параметрів моделей. Основною ідеєю градієнтного спуску є мінімізація функції втрат, яка визначає, наскільки добре модель відповідає на навчальні дані. На кожній ітерації градієнт функції втрат обчислюється для параметрів моделі, що вказує на напрямок і величину, на які необхідно змінити ваги, щоб зменшити помилку (рис. 2.17.). Цей метод дозволяє моделі поступово поліпшуватися, наближаючись до оптимального рішення. Завдяки ефективності та гнучкості, градієнтний спуск став основою для навчання нейронних мереж і адаптивних алгоритмів [29]. Формула оновлення ваг моделі на кожній ітерації градієнтного спуску:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.12}$$

де:

- θ — параметри моделі (ваги),
- η — швидкість навчання (learning rate), яка визначає величину кроку для кожного оновлення,
- $J(\theta)$ — функція втрат, яка оцінює, наскільки добре модель відповідає на навчальні дані,
- $\nabla_\theta J(\theta)$ — градієнт функції втрат щодо параметрів моделі θ, який вказує на напрямок і величину зміни ваг, щоб зменшити значення функції втрат.

Стандартний градієнтний спуск працює шляхом обчислення середнього градієнта для всього набору даних. Проте такий підхід є обчислювально «затратним», особливо для великих обсягів даних, тому було розроблено кілька варіантів градієнтного спуску, які допомагають знизити обчислювальні витрати, підвищуючи ефективність оптимізації [28].

Варіанти градієнтного спуску:

Стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD): У стохастичному градієнтному спуску ваги моделі оновлюються на основі градієнта, обчисленого для одного випадкового зразка з навчального набору на кожній ітерації. Цей підхід значно знижує обчислювальні витрати, оскільки не вимагає обробки всього набору даних на кожному кроці. Однак, через випадковий вибір зразка оновлення можуть бути нестабільними, що додає шум до процесу оптимізації. Така "стрибкоподібність" у навчанні допомагає моделі уникати локальних мінімумів, дозволяючи досягти більш глобального мінімуму функції втрат. SGD використовується в багатьох задачах з великими наборами даних, таких як обробка зображень та тексту [29]. Проте основний недолік цього підходу — це коливання навколо оптимального рішення, що може уповільнити збіжність моделі до оптимальних значень.

Mini-batch Gradient Descent: Mini-batch градієнтний спуск є популярним компромісом між традиційним градієнтним спуском і стохастичним методом. У цьому підході модель оновлює ваги на основі градієнта, обчисленого для невеликої випадкової підмножини даних (пакету або mini-batch), що забезпечує баланс між швидкістю та стабільністю навчання. Mini-batch метод має ряд переваг: він знижує обчислювальні витрати, оскільки обробляється лише частина даних, а також зменшує нестабільність і шум у порівнянні з чистим SGD. Завдяки обробці невеликих пакетів даних, метод mini-batch дозволяє краще використовувати апаратне прискорення, таке як GPU, підвищуючи ефективність навчання для великих моделей [28].

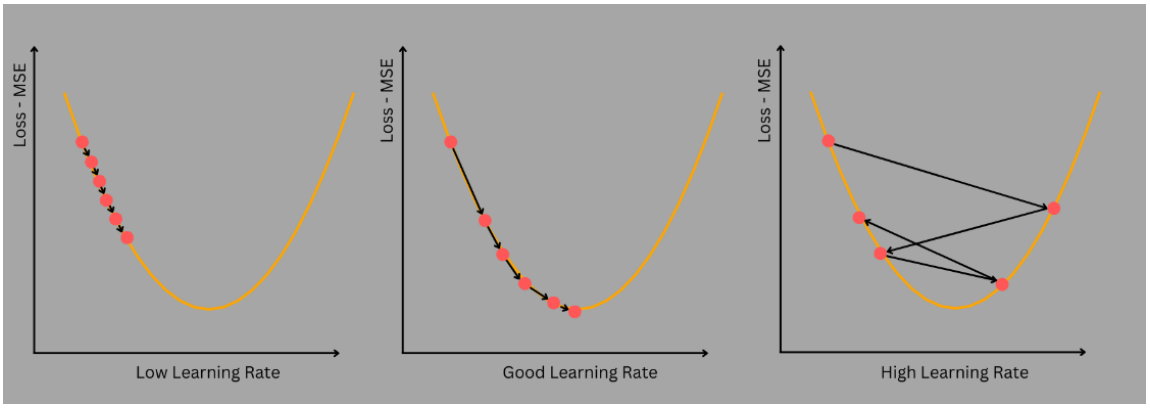

Рисунок 2.17 – Вплив швидкості навчання на процес збіжності градієнтного спуску

Адаптивна оцінка моментів (Adam - Adaptive Moment Estimation): Метод Adam був розроблений для адаптивного коригування швидкості навчання кожного параметра на основі накопичених моментів першого і другого порядку, що дозволяє враховувати не лише напрямок, але і масштаби зміни параметрів. На кожній ітерації Adam обчислює середнє значення градієнтів (момент першого порядку) і середньоквадратичне значення градієнтів (момент другого порядку) для кожного параметра, які використовуються для адаптивного налаштування кроків оновлення. Цей підхід дозволяє пришвидшити збіжність і підвищити стабільність процесу оптимізації, особливо для нерівномірних ландшафтів функцій втрат. Метод Adam, запропонований Кінгмою та Ба, став дуже популярним завдяки його ефективності у складних моделях з великою кількістю параметрів [54].

Adam є особливо корисним при роботі з глибокими нейронними мережами, оскільки дозволяє уникнути проблеми вибору оптимального кроку навчання. Метод автоматично підлаштовує швидкість навчання для кожного параметра залежно від його значення, що забезпечує стабільність та адаптивність при навчанні великих моделей. Крім того, Adam має властивість запам'ятовувати історію градієнтів, що допомагає уникнути різких стрибків і покращує збіжність [54].

Переваги та обмеження методів: Кожен з варіантів градієнтного спуску має свої сильні сторони, що робить їх оптимальними для певних типів задач. Стохастичний градієнтний спуск ефективний для великих наборів даних, але його нестабільність може вимагати додаткових епох навчання. Mini-batch градієнтний

спуск зменшує обчислювальні витрати та забезпечує збалансовану збіжність, що робить його стандартом для багатьох практичних застосувань. Метод Adam, у свою чергу, є адаптивним і підходить для завдань з великими моделями та складними ландшафтами функцій втрат. Однак адаптивні методи, такі як Adam, можуть мати проблему перенавчання при використанні з невеликими навчальними наборами, тому важливо ретельно налаштовувати параметри [28, 54].

2.3.2 Регуляризація для уникнення перенавчання

Регуляризація є одним з основних принципів боротьби з перенавчанням у машинному навчанні, і це особливо важливо для глибоких нейронних мереж, які мають велику кількість параметрів і можуть легко перенавчитися на тренувальних даних. Перенавчання (overfitting) виникає, коли модель стає занадто точною на тренувальних даних, але не в змозі коректно працювати з новими даними, що веде до зниження загальної продуктивності. Для того, щоб мінімізувати це явище, було запропоновано кілька технік регуляризації.

Однією з найпопулярніших технік регуляризації є Dropout, яка була запропонована Srivastava та ін. у 2014 році. Dropout полягає у випадковому «вимкненні» певного відсотка нейронів під час кожної ітерації навчання. Це дозволяє зменшити взаємозалежності між нейронами, змушуючи їх працювати більш незалежно один від одного, що значно покращує здатність моделі до узагальнення [30].

У глибокій нейронній мережі, кожен нейрон обчислює зважену суму своїх входів і, як вже зазначалось, передає її через нелінійну активаційну функцію. Однак під час навчання з Dropout випадково обирається підмножина нейронів, які «вимикаються», що означає, що їхній внесок у передбачення моделі не враховується. Формально, кожен нейрон зберігається з ймовірністю p, а решта нейронів обнуляються. Це забезпечує своєрідну "ансамблеву" модель, в якій під час кожної ітерації навчання використовується випадкова модель мережі. Формула для Dropout: $y = f(W \cdot x) \cdot m$, (2.13)

Де:

W — матриця ваг,

x — вхідні дані,

m — маска Dropout, яка випадково обирає 0 або 1 для кожного нейрона.

У результаті навчання з Dropout (рис. 2.17.) дозволяє уникнути ситуації, коли певні нейрони стають занадто залежними від інших і пристосовуються до конкретних патернів у даних, які можуть бути не релевантними. Srivastava та ін. відзначають, що ця техніка значно зменшує перенавчання та покращує продуктивність моделей, особливо при роботі з великими наборами даних [27].

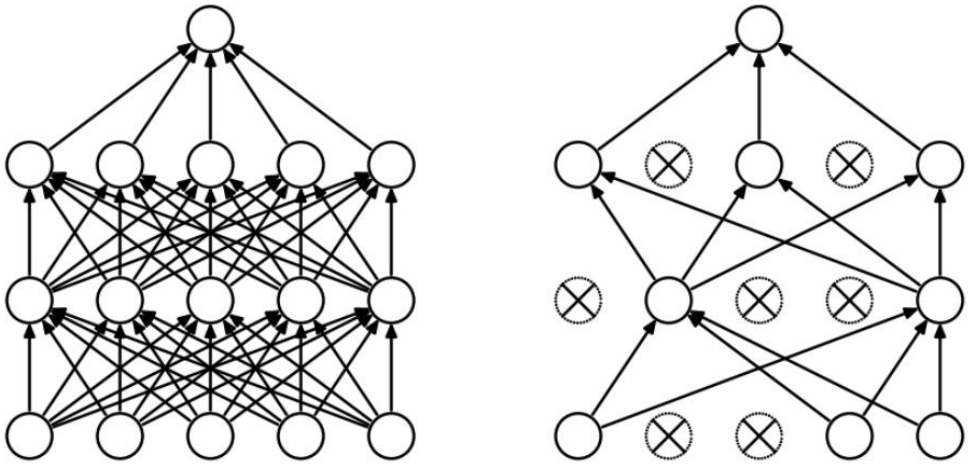

Рисунок 2.18 – Вихідна нейромережа(зліва), після застосування Dropout(праворуч)

Окрім Dropout, інші поширені методи регуляризації — це L1 та L2 регуляризація. Ці методи накладають штраф на величину ваг моделі, щоб запобігти надмірній адаптації моделі до тренувальних даних.

L1-регуляризація додає до функції втрат штраф, пропорційний сумі абсолютних значень ваг $L = L_0 + \lambda \sum_{i=1}^{n} |w_i|$, (2.14)

де:

L_0 - функція втрат без регуляризації, також відома як основна функція втрат (наприклад, крос-ентропія або середньоквадратична похибка),

λ - коефіцієнт регуляризації, який контролює вагу або інтенсивність штрафу за великі значення ваг. Гіперпараметр, який можна налаштувати під час навчання моделі,

w_i - вага i-того параметра в моделі, яку ми штрафуємо за допомогою

регуляризації,

n – загальна кількість параметрів моделі (кількість ваг).

L1-регуляризація сприяє тому, що деякі ваги стають нульовими, тим самим створюючи розріджену модель, що допомагає зменшити складність моделі та збільшити її здатність до узагальнення.

L2-регуляризація, також відома як ридж-регресія, додає штраф, пропорційний квадрату величини ваг: $L = L_0 + \lambda \sum_{i=1}^{n} w_i^2$, (2.15)

де:

L_0 - функція втрат без регуляризації, також відома як основна функція втрат (наприклад, крос-ентропія або середньоквадратична похибка),

λ - коефіцієнт регуляризації, який контролює вагу або інтенсивність штрафу за великі значення ваг. Гіперпараметр, який можна налаштувати під час навчання моделі,

w_i^2 – квадрат ваги i-того параметра в моделі, яку ми штрафуємо за допомогою регуляризації,

n – загальна кількість параметрів моделі (кількість ваг).

L2-регуляризація дозволяє зменшити великі ваги без їх обнулення, роблячи модель менш чутливою до невеликих змін у даних і шуму.

Як зазначено у роботі Goodfellow та ін. (2016), L1 та L2 регуляризації є важливими інструментами для підвищення стійкості моделей глибокого навчання, що дозволяє їм добре працювати навіть на складних завданнях [24].

2.3.3 Аугментація даних

Аугментація даних – це один із ключових методів у машинному навчанні, що дозволяє штучно розширювати тренувальний набір даних шляхом створення змінених версій наявних прикладів. Цей метод допомагає підвищити різноманітність даних, що використовується для навчання моделей, не вимагаючи збирання додаткових реальних зразків. Завдяки аугментації модель отримує змогу тренуватися на ширшому діапазоні прикладів, що дозволяє їй

краще узагальнювати результати на нових даних [55].

Аугментація стала необхідною технікою у завданнях глибокого навчання, де великі набори даних часто є критично важливими для успішного навчання складних моделей. Вона не лише допомагає компенсувати недоліки малого обсягу даних, але й зменшує ризик перенавчання, підвищуючи загальну стійкість моделі [55]. Методи аугментації можна поділити на різні категорії залежно від типу даних, що обробляються, зокрема зображення та текст.

Для роботи із зображеннями застосовуються різноманітні геометричні та колірні перетворення, які зберігають основні ознаки, але змінюють зовнішній вигляд прикладів. Основні методи включають:

- обертання: Зміна кута зображення, що дозволяє моделі краще розпізнавати об'єкти незалежно від їхнього орієнтування.
- масштабування: Зміна розміру зображення (збільшення або зменшення), яка допомагає моделі навчитися бути нечутливою до розмірів об'єктів.
- зсув: Переміщення об'єкта на зображенні по горизонталі чи вертикалі, що робить модель стійкішою до змін у розташуванні об'єктів.
- дзеркальне відображення: Відображення зображень по горизонталі чи вертикалі, що підвищує різноманітність навчальних даних [55].
- аугментація зображень стала стандартним інструментом у задачах комп’ютерного зору, зокрема для класифікації, сегментації та розпізнавання об’єктів. Наприклад, використання таких методів у моделі, що розпізнає автомобілі, дозволяє навчити її розпізнавати машини з різних ракурсів і положень [56].

Обробка текстових даних має свої унікальні підходи до аугментації, які фокусуються на зміні лексичних і синтаксичних структур тексту, зберігаючи загальний зміст. Основні методи включають:

- перефразування (Paraphrasing): Зміна формулювання тексту з тим самим змістом, що допомагає моделі розуміти синонімічні висловлювання.
- видалення слів (Word Deletion): Видалення неключових слів із тексту для навчання моделі розуміти контекст навіть за відсутності частини інформації.
- синонімічні заміни (Synonym Replacement): Заміна деяких слів їхніми

синонімами для збагачення текстових даних новими формулюваннями [55].

Аугментація тексту є критично важливою для задач обробки природної мови (Natural Language Processing, NLP), таких як класифікація тексту, аналіз настроїв і машинний переклад. Наприклад, у задачах класифікації текстів аугментація дозволяє створювати нові приклади для категорій, що мають недостатньо даних [57].

Особливо корисно, коли доступ до реальних даних обмежений або їх збирання є дорогим і тривалим процесом.

Моделі, треновані на розширених наборах даних, менш схильні до впливу випадкових шумів.

Аугментація даних дозволяє отримати кращі результати на тестових наборах, підвищуючи загальну продуктивність моделей [55].

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ТЕХНОЛОГІЇ ДЛЯ ДЕТЕКТУВАННЯ ЗАБОРОНЕНИХ ПРЕДМЕТІВ

3.1 Формування вхідних даних

Для тестування моделі SecureScanNet в наявності «негативні» кольорові зображення (які майже не містять заборонених предметів) в форматі jpeg в кількості 1,05 млн з великим розмахом роздільної здатності зображень від 30x38x3 до 4751x6400x3 де кожен піксель кожного каналу має значення в діапазоні від 0 до 255 і які не будуть використовуватись під час навчання (рис. 3.1.).

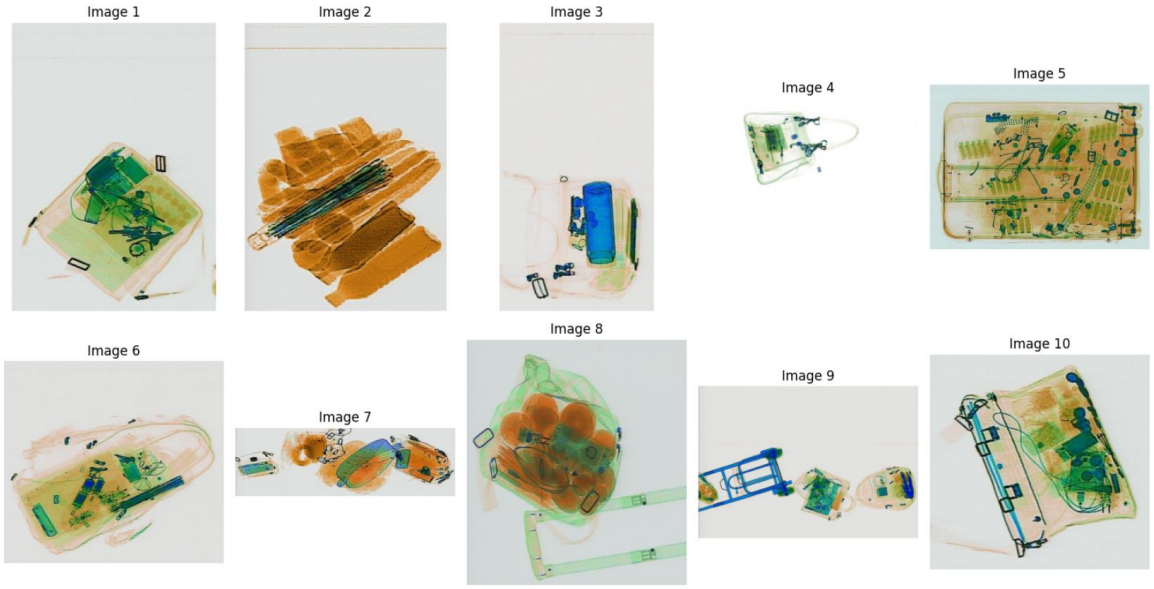

Рисунок 3.1 – Приклад «негативних» зображень

З метою зменшення математичного навантаження, а також враховуючи, що для цілей вказаної роботи важливий не колір а просторові ознаки і форми, для тестування моделі необхідні вхідні дані – матриці розмірності 360x180x1. Отже для тестування системи зменшена ширина великих зображень до 180 пікселів, та висота пропорційно. Також видалені всі зображення з висотою вище 360, видалені дублікати та всі зображення конвертовані в градації сірого кольору. До зображень, розмір яких менше ніж 360x180 застосовано заповнення невистачаючих пікселів нульовими значеннями (падінг). Як результат отримано датасет з 99651 шт, з яких 3000 шт (рис. 3.2.) використано для тестування безпосередньо.

Рисунок 3.2 – Приклад «негативних» зображень у градаціях сірого

Для навчання конволюційних нейронних мереж (для задачі класифікації та локалізації) в наявності датасет з 8827шт кольорових «позитивних» (із забороненими предметами) зображень в форматі в jpeg (рис. 3.3.) з великим розмахом роздільної здатності зображень від 234x144x3 до 1500x1035x3 де кожен піксель кожного каналу має значення від 0 до 255. Також для навчання конволюційної нейромережі в наявності 8827шт файлів з анотаціями (щодо назви класу забороненого предмету, та координат локалізації).

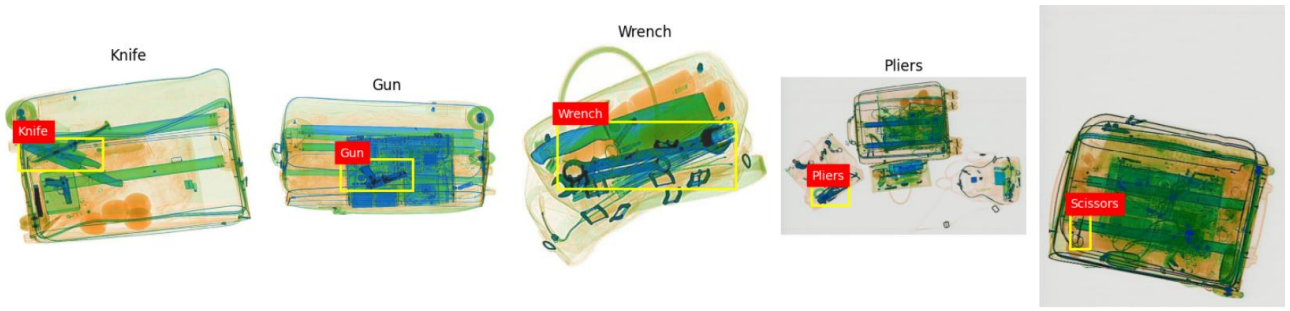

Рисунок 3.3 – Приклад «позитивних» зображень кожного класу

Серед вказаних зображень кількість об’єктів кожного класу: Knife(ніж) – 7268, Gun(пістолет) – 6693, Wrench(гайковий ключ) – 7147, Pliers(плоскогубці) – 6243, Scissors(ножиці) – 6749. Оскільки на одому зображенн зустрічаються від 1 до 10 предметів одночасно і вони можуть накладатись (перекривати) один-одного, а виборка для навчання не велика, в даній роботі прийнято рішення використовувати навчання на зображеннях де наявний лише один предмет. Для збільшення датасету застосовано такі типи аугментацій: дзеркальне відображення по вертикалі, дзеркальне відображення по горизонталі, масштабування, зсув(для задачі класифікації), зміна контрасту(для задачі локалізації). Загальна кількість

зображень з забороненими предметами після аугментації склала 12500 (по 2500 предметів на кожен клас). З метою зменшення математичного навантаження для навчання конволюційної нейромережі необхідні вхідні дані – матриці розмірності 360x180x1. Таким чином зменшена ширина та висота всіх зображень. До зображень які меншої розмірності застосовано заповнення невистачаючих пікселів нульовими значеннями (падінг). Також всі позитивні зображення конвертовані у градації сірого кольору.

Рисунок 3.4 – Приклад *.xml файлу з анотацією

Для навчання моделей нейромереж 12500 позитивних зображень розмірністю 360x180x1 розбиті на тренувальну виборку (70% - 8750шт), валідаційну (20% - 2500), тестову (10% - 1250) та така ж кількість відповідних анотацій – назв класів, що відповідає вхідній формі конволюційної нейронної мережі для класифікації предметів. Та відповідна кількість анотацій – координат предметів, для навчання конволюційної нейронної мережі задачі локалізації (рис. 3.4.). Всі дані (значення пікселів матриці та координат) приведені у діапазон значень від 0 до 1, тобто застосовано нормалізацію, для кращої поведінки моделі та зниження математичного навантаження під час навчання. Також кожній назві класу присвоєно значення від 0 до 4.

3.2 Короткий опис програмної реалізації

3.2.1 Конволюційна нейронна мережа, класифікація предметів

Для задачі класифікації створена конволюціна нейронна мережа здатна прийняти матрицю розміром 360x180x1 де кожен піксель в градації сірого від 0(чорний) до 255(білий) та назвами класів у вигляді міток від 0 до 4.

Конволюційна нейронна мережа складається з 14 шарів різних типів (рис. 3.5.), а саме:

- вхідний шар (Input Layer). Цей шар приймає зображення розміром 360×180×1.

Перший блок згортки.

- згортковий шар (Conv2D), кількість фільтрів – 36, розмір фільтрів: 3×3, застосована функція активації – relu. Зберігає розмірність 360×180×36, додаючи 36 нових каналів, які виділяють базові особливості зображення, такі як краї та кути.
- шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу - 2×2. Зменшує розмірність простору ознак до 180×90×36, зменшуючи розмір зображення наполовину та видаляючи «зайві» деталі.
- шар Dropout, ймовірність виключення нейронів – 10%. Запобігає перенавчанню моделі.

Другий блок згортки.

- згортковий шар (Conv2D), кількість фільтрів – 70, розмір фільтрів - 3×3, застосована функція активації – relu. Вихідний тензор розміром 180×90×70, що додає більше ознак до існуючих каналів.
- шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу - 2×2. Зменшує розмір до 90×45×70 ще більше стискаючи інформацію.
- шар Dropout, ймовірність виключення нейронів – 17.5%. Додатково забезпечує регуляризацію.

Третій блок згортки.

- згортковий шар (Conv2D), кількість фільтрів – 146, розмір фільтрів - 3×3, застосована функція активації – relu. Вихідний тензор розміром 90×45×146, що виділяє складніші особливості зображення.
- шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу -

2×2. В результаті операції отримано тензор розміром 45×22×146.

- шар Dropout, ймовірність виключення нейронів – 34%. Регуляризація для запобігання перенавчанню.

Повнозв'язний блок.

- шар Flatten, перетворює багатовимірний тензор розміром 45×22×146 в одновимірний вектор розміром 144540.
- повнозв'язний шар (Dense), кількість нейронів – 146, застосована функція активації – relu. Виділяє найважливіші ознаки для класифікації.
- четвертий шар Dropout, ймовірність виключення нейронів – 88%. Забезпечує стійкість до перенавчання.
- вихідний шар для класифікації (Dense), кількість нейронів – 5 (Knife, Gun, Wrench, Pliers, Scissors), застосована функція активації – softmax. Генерує ймовірності для кожного класу.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 360, 180, 1)	0
conv2d (Conv2D)	(None, 360, 180, 36)	360
max_pooling2d (MaxPooling2D)	(None, 180, 90, 36)	0
dropout (Dropout)	(None, 180, 90, 36)	0
conv2d_1 (Conv2D)	(None, 180, 90, 70)	22,750
max_pooling2d_1 (MaxPooling2D)	(None, 90, 45, 70)	0
dropout_1 (Dropout)	(None, 90, 45, 70)	0
conv2d_2 (Conv2D)	(None, 90, 45, 146)	92,126
max_pooling2d_2 (MaxPooling2D)	(None, 45, 22, 146)	0
dropout_2 (Dropout)	(None, 45, 22, 146)	0
flatten (Flatten)	(None, 144540)	0
dense (Dense)	(None, 146)	21,102,986
dropout_3 (Dropout)	(None, 146)	0
dense_1 (Dense)	(None, 5)	735

Рисунок 3.5 – Загальна структура моделі конволюційної нейромережі для задачі класифікації

Загальна кількість тренувальних параметрів – 21,218,95, розмір моделі - 80.94 МБ.

Лістинг коду на мові програмування Python розміщено у Додатку А.

3.2.2 Конволюційна нейронна мережа, локалізація предметів

Для задачі локалізації створена конволюціна нейронна мережа здатна прийняти матрицю розміром 360x180x1 де кожен піксель в градації сірого від 0(чорний) до 255(білий) та мітки у вигляді координат так званого bounding box навколо предмету, за формою [xmin, ymin, xmax, ymax].

У моделі використано 10 шарів різних типів (рис 3.6.), а саме:

- вхідний шар (Input Layer). Цей шар приймає зображення у відтінках сірого з розміром 360×180×1 пікселів.
- перший згортковий шар (Conv2D), кількість фільтрів – 32, розмір фільтрів – 3×3, застосована функція активації – relu. Вихідний тензор має розмірність 360×180×32, додаючи 32 канали, які виділяють базові особливості зображення, такі як краї та кути.
- шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу – 2×2. Зменшує розмірність простору ознак до 180×90×32, зменшуючи просторовий розмір зображення наполовину.
- другий згортковий шар (Conv2D), кількість фільтрів – 64, розмір фільтрів – 3×3, застосована функція активації – relu, параметр padding='same'. Вихідний тензор має розмір 180×90×64.
- другий шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу – 2×2. Зменшує розмір до 90×45×64, ще більше стискаючи інформацію.
- третій згортковий шар (Conv2D), кількість фільтрів – 128, розмір фільтрів – 3×3, застосована функція активації – relu, параметр padding='same'. Вихідний тензор має розмір 90×45×128.
- третій шар максимального пулінгу (MaxPooling2D), розмір вікна пулінгу – 2×2. Зменшує розмір до 45×22×128, створюючи компактне представлення просторових ознак.
- шар Flatten, перетворює багатовимірний тензор розміром 45×22×128 в одновимірний вектор розміром 126720.
- повнозв’язний шар (Dense), кількість нейронів – 256, застосована функція активації – relu. Виділяє високорівневі ознаки для передбачення координат bounding box.

- шар Dropout, ймовірність виключення нейронів – 50%. Використовується для регуляризації та запобігання перенавчанню.

- вихідний шар (Dense), кількість нейронів – 4, застосована лінійна функція активації (linear). Генерує координати bounding box у форматі [xmin, ymin, xmax, ymax].

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 360, 180, 1)	0
conv2d (Conv2D)	(None, 360, 180, 32)	320
max_pooling2d (MaxPooling2D)	(None, 180, 90, 32)	0
conv2d_1 (Conv2D)	(None, 180, 90, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 90, 45, 64)	0
conv2d_2 (Conv2D)	(None, 90, 45, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 45, 22, 128)	0
flatten (Flatten)	(None, 126720)	0
dense (Dense)	(None, 256)	32,440,576
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1,028

Рисунок 3.6 – Загальна структура моделі конволюційної нейромережі для задачі локалізації

Загальна кількість тренувальних параметрів – 32,534,276, розмір моделі - 124.11 МБ.

Лістинг коду на мові програмування Python розміщено у Додатку В.

3.3 Навчання, тестування на реальних даних, аналіз результатів

Навчання моделей відбувалось із застосуванням технології CUDA на відеокарті NVIDIA GeForce RTX 3080 Ti, із застосуванням технології CUDA на зображеннях SIXray[59].

Навчання моделі конволюційної нейронної мережі (для задач класифікацій) здійснювалось на протязі 120 епох з одночасним завантаженням 32 зображень у пакеті і тривало 50 хвилин. Параметри (ваги й зсуви) оновлюються за допомогою adam. Під час навчання для відстеження якості навчання моделі на тренувальних та валідаційних даних використовувались наступні метрики:

1. Точність (Accuracy) — це відношення кількості правильно передбачених класів до загальної кількості зразків.

2. Функція втрат – розріджена категоріальна крос-ентропія (sparse_categorical_crossentropy): обчислює крос-ентропію для багатокласових задач класифікації, де цільові мітки подаються у вигляді цілих чисел. Ця функція порівнює ймовірності, передбачені моделлю для кожного класу, з фактичним класом, мінімізуючи логарифмічну втрату.

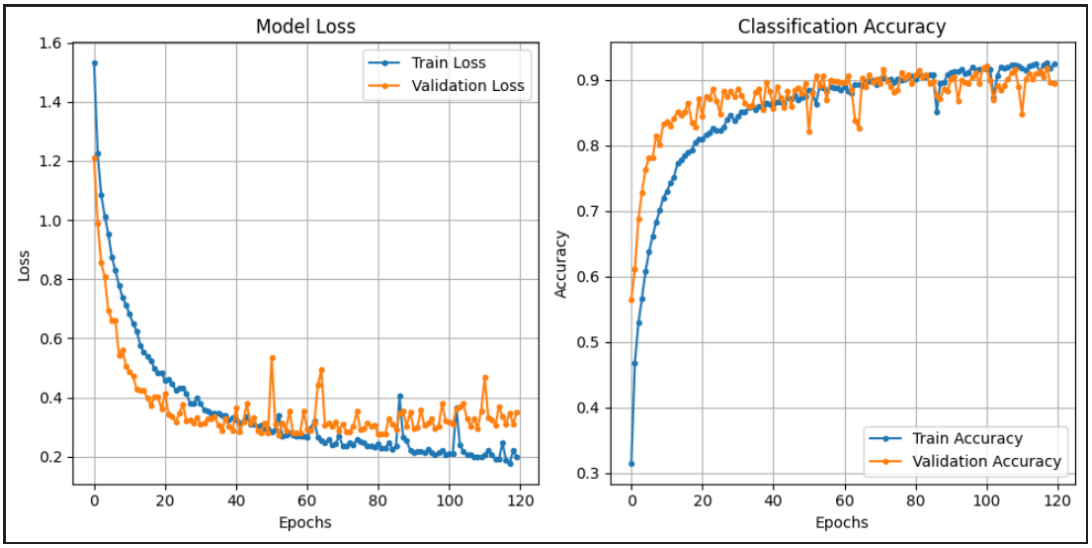

Рисунок 3.7 – Графічне відображення метрик

Як видно на рисунку 3.7, значення функції втрат зменшується і на 53 епосі становить 0.402 на тренувальних даних та 0.273 на валідаційних даних. Метрика точність на 53 епосі становить 0.8489 на тренувальних даних та 0.9068 на валідаційних даних. Вказанні значення є найкращими на валідаційних даних, у зв'язку з чим параметри моделі на 53 епосі були збережені. Дані краще на 53 епосі на валідаційній виборці ніж на тренувальних даних у зв'язку із застосуванням регуляризації, виключення нейронів з метою запобігання перенавчання.

На тестових даних, які не приймали участі у навчанні, модель показала достатньо високі результати (таб. 3.1.). Загальна точність моделі склала 90.16%.

Таблиця 3.1 Метрики точності класифікації по кожному класу предметів на тестових зображеннях

Клас	Вірно класифіковано (True)	Хибно класифіковано (False)	Точність (%)
Gun	249	1	99.60
Knife	219	31	87.60
Pliers	230	20	92.00

Продовження табл. 3.1

Scissors	216	34	86.40
Wrench	213	37	85.20
Разом	-	-	**90.16**

З таблиці 3.1 видно, що тестування проведено на 1250 позитивних фото по 250 на кожен клас заборонених предметів. Найвища точність у класу – «пістолет» і складає 99.6% вірно передбачено моделлю 249 зображень з 250, найнижчий показник у «гайкового ключа» і складає 85.2% вірно передбачено моделлю 213 з 250 зображень.

Навчання моделі конволюційної нейронної мережі (для задачі локалізації) здійснювалось на протязі 50 епох з одночасним завантаженням 32 зображень у пакеті і тривало до 20 хвилин. Параметри (ваги й зсуви) оновлюються за допомогою Adam. Під час навчання для відстеження якості навчання моделі на тренувальних та валідаційних даних використовувались наступні метрики:

1. MAE (Mean Absolute Error) використовується для оцінки середньої похибки в локалізації, тобто для визначення, наскільки точно модель передбачає координати меж (bounding boxes) у порівнянні зі справжніми значеннями. Ця метрика обчислює середню абсолютну різницю між передбаченими координатами та фактичними і показує середню величину помилки в одиницях вимірювання координат. Вона забезпечує простий і зрозумілий спосіб моніторингу прогресу моделі під час навчання або тестування, даючи можливість швидко оцінити, наскільки модель поліпшує свої результати з кожною ітерацією.

2. Smooth L1 Loss (Гладка L1-функція втрат) — це функція втрат, яка поєднує властивості L1 Loss (Mean Absolute Error) і L2 Loss (Mean Squared Error), забезпечуючи баланс між чутливістю до малих помилок і стійкістю до великих викидів. Для малих помилок використовується квадратичний режим. Це дозволяє зробити функцію більш гладкою та сприяє стабільності під час оптимізації. Для великих помилок переходить до лінійного режиму, що зменшує вплив великих похибок (викидів) на процес навчання.

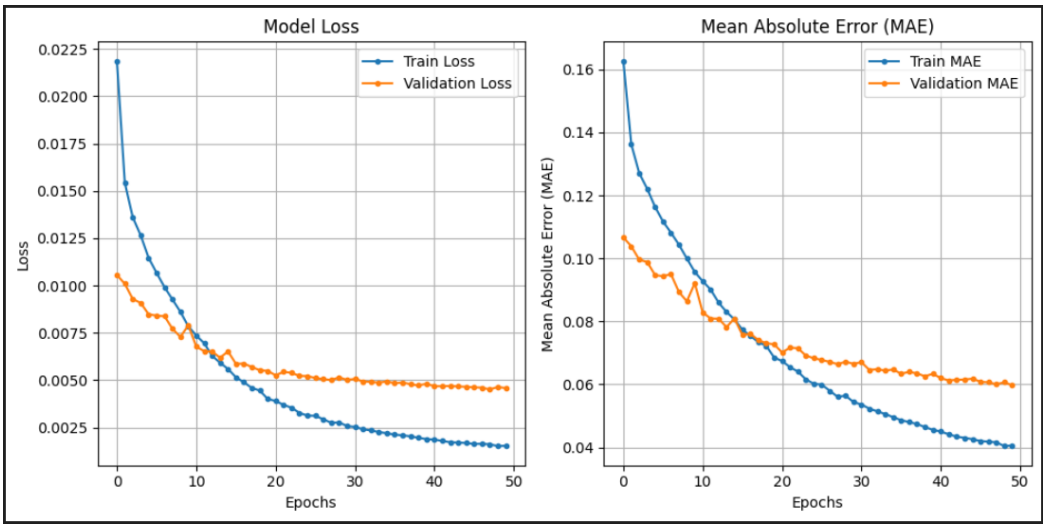

Рисунок 3.8 – Графічне відображення метрик

На рисунку 3.8, значення функції втрат зменшується і на 48 епосі становить 0.0016 на тренувальних даних та 0.005 на валідаційних даних. Метрика похибки на 48 епосі становить 0.0415 на тренувальних даних та 0.06 на валідаційних даних. Вказанні значення є найкращими на валідаційних даних, у зв'язку з чим параметри моделі 48 епохи були збережені. Дані краще на 48 епосі на валідаційній виборці ніж на тренувальних даних у зв'язку із застосуванням регуляризації, виключення нейронів з метою запобігання перенавчання.

При тестуванні на тестових даних (1255 зображень по 251 зображенню на клас), які не приймали участі у навчанні модель показала достатньо високі результати.

Таблиця 3.2 Значення похибки у відсотках по кожній координаті кожного класу предметів на тестових зображеннях

Клас	xmin (%)	ymin (%)	xmax (%)	ymax (%)	Середня похибка (%)
Gun	7.76	3.37	7.95	3.11	5.55
Knife	7.66	3.12	7.36	3.33	5.87
Pliers	9.81	3.87	9.85	4.27	6.95
Scissors	8.42	4.15	8.64	3.77	6.25
Wrench	7.62	3.41	7.76	3.68	5.62

Наведені у таблиці 3.2 результати відображають середню абсолютну похибку (MAE) для координат (xmin, ymin, xmax, ymax) у відсотках та загальну похибку для всіх класів. Загальна середня похибка (Mean Error) складає 5.95%.

Для класу Knife середня похибка по кожній координаті становить 7.66%, 3.12%, 7.36%, 3.33%. Цей клас демонструє найменшу похибку серед горизонтальних координат (xmin, xmax) і вертикальних (ymin, ymax). Можливо, ножі мають більш стандартизовану форму та розташування, що полегшує локалізацію. MAE (xmin, ymin, xmax, ymax) для класу Pliers становить 9.81%, 3.87%, 9.85%, 4.27%. Найвища похибка серед усіх класів, особливо у горизонтальних координатах (xmin, xmax). Модель недостатньо вивчила цей клас через недостатню кількість даних або складність для екстракції ознак. MAE (xmin, ymin, xmax, ymax) для класу Gun становить 7.76%, 3.37%, 7.95%, 3.11%. Похибка знаходиться на середньому рівні. Вертикальні координати (ymin, ymax) прогнозуються точніше, ніж горизонтальні. MAE (xmin, ymin, xmax, ymax) для класу Scissors становить 8.42%, 4.15%, 8.64%, 3.77%. Похибка трохи вища, особливо для вертикальних координат (ymin, ymax), можливо, через різноманітність положення ножиць у зображеннях. MAE (xmin, ymin, xmax, ymax) для класу Wrench становить 7.62%, 3.41%, 7.76%, 3.68%. Схожий рівень похибки з класом Gun. Похибка залишається в межах допустимого. Горизонтальні координати (xmin, xmax) в середньому мають більшу похибку (~7-10%), ніж вертикальні (~3-4%). Це може свідчити про те, що модель складніше прогнозує ширину об'єктів. Можливо, горизонтальні межі об'єктів є більш варіативними або накладаються на інші об'єкти. Вертикальні координати (ymin, ymax) прогнозуються точніше (~3-4%), що може свідчити про те, що висота об'єктів є менш варіативною. Приклади локалізації передбачуваних моделлю у порівнянні з оригіналом наведені на рисунках 3.9 та 3.10.

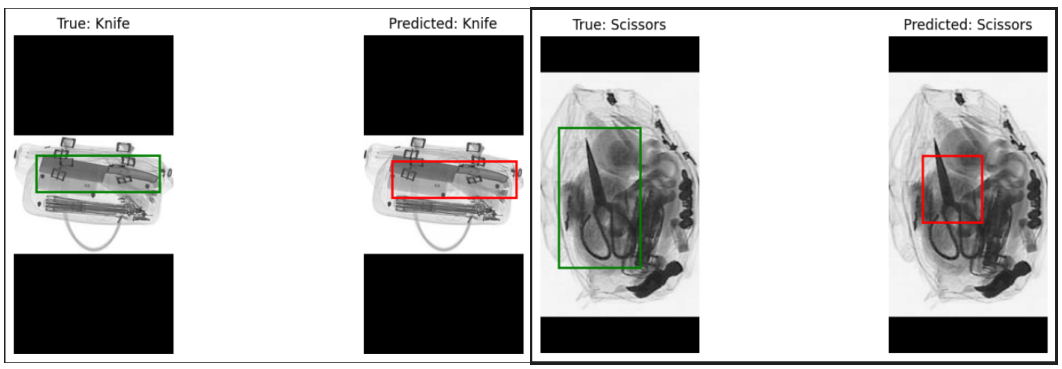

Рисунок 3.9 – Приклади локалізації об’єктів класів Knife і Scissors у рентгенівських зображеннях

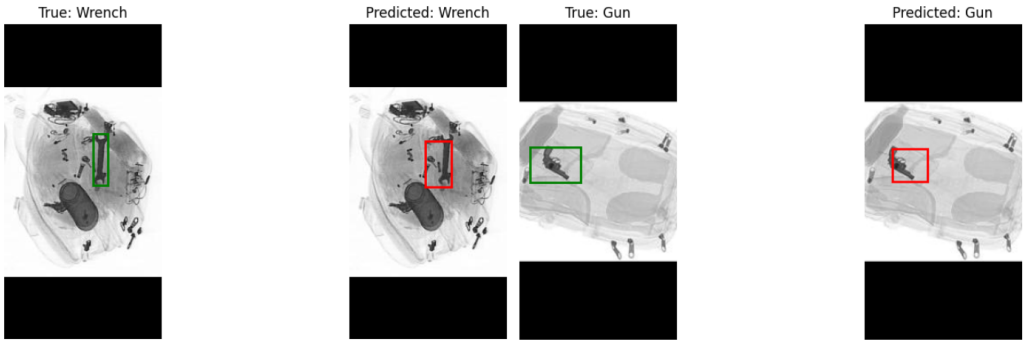

Рисунок 3.10 – Приклади локалізації об’єктів класів Wrench і Gun у рентгенівських зображеннях

Під час тестування класифікаційної моделі конволюційної нейронної мережі на «негативних» фото було використано здатність функції Softmax повертати вірогідність по кожному класу. Оскільки модель також здійснює спроби класифікації на «негативних» фото, то вона повертає вірогідності приналежності зображень до того чи іншого класу, але оскільки зображення не містять заборонених предметів, то ці вірогідності нижче ніж на «позитивних» зображеннях. В даній роботі найнижча вірогідність по класу гайковий ключ (wrench) і складає 85.2% отже для визначення наявності заборонених предметів на «негативних» фото застосована межа у 83%. Результати тестування наведенні у таблиці 3.3.

Таблиця 3.3 Результати тестування на негативних зображеннях

Classification Result	Count	Percentage (%)
Correctly Classified	2610	87.00
Incorrectly Classified	390	13.00
Total	3000	100.00

Як бачимо, модель показує достатньо високі результати та у 87% при межі 84% вірно визначає відсутність на зображеннях заборонених предметів.

Лістинг коду на мові програмування Python розміщено у Додатку С.

Порівняння переваг і недоліків :

- ResNet

Переваги: Висока точність локалізації, стабільність роботи та глибокий

аналіз зображень завдяки резидуальним блокам.

Недоліки: Обчислювально важка модель із низькою швидкістю.

SecureScanNet пропонує баланс між точністю та швидкістю завдяки адаптованій архітектурі, що складається з меншої кількості шарів. Легша для обчислювальних ресурсів, ніж ResNet, але поступається в глибині аналізу.

- YOLO

Переваги: Висока швидкість обробки та здатність працювати в реальному часі.

Недоліки: Точність нижча за R-CNN; не підходить для складних задач із високою деталізацією.

SecureScanNet має вищу точність класифікації (90.16%) та локалізації (5.95% середньої похибки) порівняно з YOLO. Швидкість нижча, ніж у YOLO, проте модель забезпечує краще опрацювання деталей.

- SSD

Переваги: Баланс між швидкістю та точністю, здатність працювати з об'єктами різного розміру.

Недоліки: Обмеження при детекції дрібних об'єктів.

В моделі SecureScanNet, завдяки регуляризації та оптимізації архітектури покращено роботу з дрібними об'єктами. Має схожу продуктивність у швидкості, але кращу точність класифікації.

- EfficientDet

Переваги: Висока ефективність, низькі обчислювальні витрати, масштабованість моделі.

Недоліки: Потребує ретельного налаштування.

SecureScanNet забезпечує подібну точність для класифікації та локалізації, але потребує менше обчислювальних ресурсів. Простішу структуру легко адаптувати без складного налаштування.

- R-CNN

Переваги: Висока точність локалізації та детекція дрібних об'єктів.

Недоліки: Низька швидкість; не підходить для задач реального часу.

SecureScanNet надає високу точність локалізації (середня похибка 5.95%), близьку до R-CNN, але має вищу швидкість. Модель адаптована для

використання в реальних умовах із меншими вимогами до ресурсів.

Ключові переваги SecureScanNet – це баланс точності та швидкості, ефективність роботи з дрібними об'єктами, завдяки оптимізації регуляризації та використанню Dropout покращено роботу з дрібними об'єктами, модульність і легкість адаптації, простішу структуру моделі можна легко адаптувати до нових класів або датасетів, зниження обчислювального навантаження.

ВИСНОВКИ

В кваліфікаційній магістерській роботі розв’язано практичну задачу розробки інтелектуальної системи SecureScanNet для автоматичної детекції заборонених предметів на зображеннях, отриманих за допомогою скануючих пристроїв. Для досягнення поставленої мети були використані сучасні підходи глибокого машинного навчання та розпізнавання образів, які були адаптовані до задачі митного контролю.

При цьому виконано такі основні завдання:

1) Проведено аналіз сучасних архітектур нейронних мереж (ResNet, YOLO, SSD, EfficientDet, R-CNN), їх переваг, недоліків і доцільності застосування для задач класифікації та локалізації заборонених предметів. За результатами аналізу обґрунтовано вибір архітектури для створення моделі SecureScanNet.

2) Сформовано вхідний набір даних, що включає 99651 "негативних" та 12500 "позитивних" зображень із застосуванням методів масштабування, конвертації в градації сірого, нормалізації та аугментацій. Дані розділено на тренувальну, валідаційну та тестову вибірки у співвідношенні 70/20/10.

3) Розроблено дві моделі глибокого навчання:

- модель для класифікації заборонених предметів із загальною точністю 90.16%, яка складається з 14 шарів.
- модель для локалізації заборонених предметів, яка прогнозує координати bounding boxes ([xmin, ymin, xmax, ymax]) із середньою похибкою (Mean Error) 5.95%. Модель складається з 10 шарів.

4) Проведено навчання моделей у середовищі Python із застосуванням TensorFlow та CUDA, що забезпечило ефективне використання обчислювальних ресурсів.

5) Перевірено працездатність системи SecureScanNet на реальних даних:

Модель класифікації продемонструвала високу точність для всіх класів, зокрема, для класу Gun точність склала 99.6%, а для класу Wrench — 85.2%.

Модель локалізації забезпечила точне прогнозування координат bounding boxes із середньою похибкою для координат не більше 10%.

Система SecureScanNet показала ефективність при тестуванні на "негативних" фото, демонструючи 87% правильного визначення відсутності заборонених предметів при пороговому значенні ймовірності 84%.

Результати магістерської роботи підтверджують, що система SecureScanNet може бути впроваджена для автоматизації процесів митного контролю. Це дозволить значно зменшити навантаження на працівників, підвищити точність перевірок і знизити ризик пропуску небезпечних предметів.

Розробка системи SecureScanNet є перспективним напрямком, що потребує подальшого вдосконалення. Зокрема, можливим є розширення набору класів заборонених предметів, додавання нових методів аугментації для покращення роботи з дрібними об'єктами, оптимізація системи для роботи в умовах реального часу на мобільних пристроях або слабких обчислювальних платформах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. De Wulf L., Sokol J. B. Customs Modernization Handbook / L. De Wulf, J. B. Sokol. – Washington: The World Bank, 2005. – 345 p.

2. Customs Matters: Strengthening Customs Administration in a Changing World. – Washington: The World Bank, 2005. – 301 p.

3. Фрадинський О. А., Брендак А. І. Зарубіжний досвід застосування митних інформаційних технологій // Науковий вісник Університету державної фіскальної служби України. – 2020. – № 5-6. – С. 22-30. – DOI: 10.37332/2309-1533.2020.5-6.22.

4. Івашова Л. М., Кийда Л. І. Діджиталізація митних процедур: сучасний стан та перспективи розвитку митної справи // Публічне управління та митне адміністрування. – 2019. – № 3. – С. 218-230. – DOI: 10.32836/2310-9653-2019-3-218-230.

5. Smiths Detection. X-ray systems for security and control [Електронний ресурс]. – Режим доступу: https://www.smithsdetection.com/. – Назва з екрану. – Дата звернення: 07 жовтня 2024 р.

6. Rapiscan Systems. Technologies for baggage and cargo scanning [Електронний ресурс]. – Режим доступу: https://www.rapiscansystems.com/. – Назва з екрану. – Дата звернення: 07 жовтня 2024 р.

7. Leidos. Security solutions for airports and customs checkpoints [Електронний ресурс]. – Режим доступу: https://www.leidos.com/. – Назва з екрану. – Дата звернення: 07 жовтня 2024 р.

8. Nuctech. Innovative scanning systems for cargo and vehicles [Електронний ресурс]. – Режим доступу: http://www.nuctech.com/en. – Назва з екрану. – Дата звернення: 07 жовтня 2024 р.

9. Analogic Corporation. Computed tomography systems for security [Електронний ресурс]. – Режим доступу: https://www.analogic.com/. – Назва з екрану. – Дата звернення: 07 жовтня 2024 р.

10. Russo P. X-Ray Imaging: Fundamentals, Industrial Techniques and Applications [Текст] / Paolo Russo. – CRC Press, 2017. – 1344 с.

11. McKay D. Security Screening Technologies in the 21st Century [Текст] /

David McKay. – Taylor & Francis, 2016. – 488 с.

12. Hsieh J. Computed Tomography: Principles, Design, Artifacts, and Recent Advances [Текст] / Jiang Hsieh. – 3-е вид. – SPIE Press, 2016. – 456 с.

13. Chio C., Freeman D. Machine Learning and Security: Protecting Systems with Data and Algorithms [Текст] / Clarence Chio, David Freeman. – O'Reilly Media, 2018. – 384 с.

14. Garito A. F. Detection Technologies for Chemical Warfare Agents and Toxic Vapors [Текст] / Anthony F. Garito. – Springer, 2013. – 305 с.

15. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 770–778

16. Redmon, J., & Farhadi, A. (2018). "YOLOv3: An Incremental Improvement." *arXiv preprint arXiv:1804.02767*

17. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). "SSD: Single Shot Multibox Detector." In *European Conference on Computer Vision (ECCV)* (pp. 21–37)

18. Tan, M., Pang, R., & Le, Q. V. (2020). "EfficientDet: Scalable and Efficient Object Detection." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 10781–10790)

19. Ren, S., He, K., Girshick, R., & Sun, J. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In *Advances in Neural Information Processing Systems (NIPS)* (pp. 91–99)

20. Young, T., Hazarika, D., Poria, S., Cambria, E. Recent trends in deep learning based natural language processing [Текст] / Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria // IEEE Computational Intelligence Magazine. – 2018. – Vol. 13, No. 3. – С. 55–75.

21. Lee, S. M., Kang, S. G., Shin, H. J. Artificial intelligence and financial markets: Can machines predict market movements? [Текст] / Sang M. Lee, Sang G. Kang, Hyo J. Shin // Journal of Computational Finance. – 2019. – Vol. 22, No. 3. – С. 77–101.

22. Reichstein, M., Camps-Valls, G., Stevens, B., et al. Deep learning and process understanding for data-driven Earth system science [Текст] / Markus

Reichstein, Gustavo Camps-Valls, Bjorn Stevens та ін. // Nature. – 2019. – Vol. 566, No. 7743. – C. 195–204.

23. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, 1097–1105.

24. Goodfellow I., Bengio Y., Courville A. Deep Learning [Текст] / I. Goodfellow, Y. Bengio, A. Courville. — MIT Press, 2016. — 775 с.

25. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow [Текст] / A. Géron. — O'Reilly Media, 2019. — 856 с.

26. Bengio Y., Courville A., Vincent P. Representation Learning: A Review and New Perspectives [Text] / Yoshua Bengio, Aaron Courville, Pascal Vincent // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2013. – Vol. 35, No. 8. – P. 1798-1828.

27. LeCun Y., Bengio Y., Hinton G. Deep Learning [Text] / Y. LeCun, Y. Bengio, G. Hinton // Nature. – 2015. – Vol. 521, No. 7553. – P. 436–444.

28. Ruder S. "An Overview of Gradient Descent Optimization Algorithms" [Text] // arXiv, 2016. – 16 p.

29. Bottou L. "Large-Scale Machine Learning with Stochastic Gradient Descent" [Text] / Léon Bottou // // Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT), 2010. – P. 177–186.

30. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting // Journal of Machine Learning Research. — 2014. — Vol. 15. — P. 1929–1958.

31. Dean C., Bengio S., Hardt M., et al. "Understanding Deep Learning Requires Rethinking Generalization" — arXiv, 2017

32. Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., & Ng, A. Y. (2012). Large Scale Distributed Deep Networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, 1223–1231.

33. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S.,

Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 265–283.

34. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13), 3521–3526.

35. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 113, 54–71.

36. Thrun, S., & Pratt, L. (1998). *Learning to Learn*. Springer.

37. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why Should I Trust You? Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1135–1144.

38. CRIF Digital. Інтернет-ресурс. URL: https://www.crif.digital/ (дата звернення: 05.11.2024)

39. Samek, W., Montavon, G., Vedaldi, A., Hansen, L. K., & Müller, K. R. (2019). *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer.

40. GlassBox Medicine. Інтернет-ресурс. URL: https://glassboxmedicine.com/ (дата звернення: 05.11.2024)

41. Москаленко В. Вступ до науки про дані: Дерева рішень та модель найближчого сусіда: Лекція 5 з курсу "Вступ до науки про дані". Особистий конспект лекцій.

42. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

43. Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.

44. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 770–778

45. Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780

46. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*

47. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In Proceedings of Interspeech, 1045–1048.

48. Kingma, D. P., Welling, M. *Auto-Encoding Variational Bayes* // arXiv preprint arXiv:1312.6114, 2013

49. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P. A. *Extracting and Composing Robust Features with Denoising Autoencoders* // Proceedings of the 25th international conference on Machine learning (ICML), 2008. – P. 1096–1103.

50. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. *Generative Adversarial Nets* // Advances in Neural Information Processing Systems (NIPS), 2014. – P. 2672–2680.

51. Radford, A., Metz, L., Chintala, S. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* // arXiv preprint arXiv:1511.06434, 2015

52. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. *Attention Is All You Need* // Advances in Neural Information Processing Systems (NIPS), 2017. – P. 5998–6008

53. Devlin, J., Chang, M. W., Lee, K., Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* // arXiv preprint arXiv:1810.04805, 2018.

54. Kingma, D. P., Ba, J. *Adam: A Method for Stochastic Optimization* // arXiv preprint arXiv:1412.6980, 2014.

55. Shorten, C., Khoshgoftaar, T. M. *A survey on image data augmentation for deep learning* // Journal of Big Data, 2019. – Vol. 6, No. 60. – P. 1–48.

56. Taylor, L., Nitschke, G. *Improving deep learning with generic data augmentation techniques* // Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2018. – P. 1542–1547.

57. Wei, J., Zou, K. *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks* // arXiv preprint arXiv:1901.11196, 2019

58. Information Technology — Generic Coding of Moving Pictures and Associated Audio Information (ISO/IEC 13818-2). International Standard. Geneva: International Organization for Standardization, 2000. 166 p.

59. Miao, C., Xie, L., Wan, F., Su, C., Liu, H., Jiao, J., Ye, Q. SIXray: A Large-scale Security Inspection X-ray Benchmark for Prohibited Item Discovery in Overlapping Images // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

ДОДАТОК А

```
import tensorflow as tf
from tensorflow.keras import layers, models, Input
# Input parameters
input_shape = (360, 180, 1)  # Image dimensions (height, width, number of channels)
num_classes = 5  # Number of classes (Knife, Gun, Wrench, Pliers, Scissors)
# Function to create the classification model
def create_classification_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    # Convolutional block 1
    x = layers.Conv2D(36, (3, 3), activation='relu', padding='same')(inputs)  # First convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to reduce spatial dimensions
    x = layers.Dropout(0.1)(x)  # Dropout for regularization
    # Convolutional block 2
    x = layers.Conv2D(70, (3, 3), activation='relu', padding='same')(x)  # Second convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to further reduce dimensions
    x = layers.Dropout(0.175)(x)  # Dropout for regularization
    # Convolutional block 3
    x = layers.Conv2D(146, (3, 3), activation='relu', padding='same')(x)  # Third convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to compress feature maps
    x = layers.Dropout(0.34)(x)  # Dropout for regularization
    # Transition to fully connected layers
    x = layers.Flatten()(x)  # Flattening the feature maps
    x = layers.Dense(146, activation='relu')(x)  # Fully connected layer for feature extraction
    x = layers.Dropout(0.88)(x)  # Dropout for regularization
    # Classification output layer
    classification_output = layers.Dense(num_classes, activation='softmax')(x)  # Output layer with softmax activation

    # Build the model
    model = models.Model(inputs=inputs, outputs=classification_output)
    return model
# Create the model
model = create_classification_model(input_shape, num_classes)
# Compile the model
```

```
model.compile(
    optimizer='adam',  # Adam optimizer
    loss='sparse_categorical_crossentropy',  # Sparse categorical cross-entropy loss
    metrics=['accuracy']  # Metric to evaluate accuracy
)
# Display the model's architecture
model.summary()
```

ДОДАТОК В

```
import tensorflow as tf
from tensorflow.keras import layers, models, Input
#Create a convolutional neural network for object localization.
def create_localization_model(input_shape):
    inputs = Input(shape=input_shape)  # Input layer with specified input shape
    # Block 1: Convolution + MaxPooling
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)  # First convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to reduce spatial dimensions
    # Block 2: Convolution + MaxPooling
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)  # Second convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to further reduce spatial dimensions
    # Block 3: Convolution + MaxPooling
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)  # Third convolutional layer
    x = layers.MaxPooling2D((2, 2))(x)  # Max pooling to compress feature maps
    # Transition to fully connected layers
    x = layers.Flatten()(x)  # Flattening the feature maps into a single vector
    x = layers.Dense(256, activation='relu')(x)  # Fully connected layer for feature extraction
    x = layers.Dropout(0.5)(x)  # Dropout for regularization to prevent overfitting
    # Output layer for bounding box coordinates prediction
    localization_output = layers.Dense(4, activation='linear')(x)  # Predicts [xmin, ymin, xmax, ymax]
    # Build the model
    model = models.Model(inputs=inputs, outputs=localization_output)
    return model
# Model parameters
input_shape = (360, 180, 1)  # Image dimensions (height, width, number of channels)
# Create the localization model
localization_model = create_localization_model(input_shape)
# Display the model's architecture
localization_model.summary()
from tensorflow.keras.callbacks import ModelCheckpoint
# Function to implement Smooth L1 Loss
def smooth_l1_loss(y_true, y_pred, delta=1.0):
    abs_error = tf.abs(y_true - y_pred)  # Calculate absolute error
```

```
        quadratic = tf.minimum(abs_error, delta)  # Quadratic term for small errors
        linear = abs_error - quadratic  # Linear term for large errors
        loss = 0.5 * tf.square(quadratic) + delta * linear  # Combine quadratic and linear terms
        return tf.reduce_mean(loss)  # Compute mean loss over the batch
    # Compile the model
    localization_model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),  # Adam optimizer with a learning rate of 0.0001
        loss=lambda y_true, y_pred: smooth_l1_loss(y_true, y_pred, delta=1.0),  # Use Smooth L1 Loss for training
        metrics=['mae']  # Include Mean Absolute Error (MAE) as a performance metric
    )
```

ДОДАТОК С

```
import os
import numpy as np
import cv2
from tensorflow.keras.models import load_model
# Path to negative images
negative_images_path =
"/home/dmitry/repos/AutoEncoderCourseProject/data/ResizedImages"

# Load classification model
classification_model = load_model("best_classification_model.keras")

# Parameters
IMAGE_WIDTH = 180
IMAGE_HEIGHT = 360
CLASSIFICATION_THRESHOLD = 0.84  # Classification threshold for negative images
MAX_NEGATIVE_IMAGES = 3000  # Maximum number of negative images to process

# Classification results
negative_correct = 0
negative_incorrect = 0

# Function to preprocess images with padding
def preprocess_image(image_path):
    #Loads an image, applies padding to 360x180, and normalizes it.
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    original_height, original_width = img.shape

    # Calculate scaling factor for proportional resizing
    scale = min(IMAGE_WIDTH / original_width, IMAGE_HEIGHT / original_height)
    new_width = int(original_width * scale)
    new_height = int(original_height * scale)

    # Resize the image
   resized_img = cv2.resize(img, (new_width, new_height), interpolation=cv2.INTER_AREA)

    # Create a 360x180 canvas with a black background
```

```
        padded_img = np.zeros((IMAGE_HEIGHT, IMAGE_WIDTH), dtype=np.uint8)

        # Calculate offsets for centering the image
        x_offset = (IMAGE_WIDTH - new_width) // 2
        y_offset = (IMAGE_HEIGHT - new_height) // 2

        # Place the image at the center of the canvas
        padded_img[y_offset:y_offset + new_height, x_offset:x_offset + new_width] =
resized_img
        # Normalize the image
        padded_img = padded_img / 255.0  # Normalize to range [0, 1]
        padded_img = np.expand_dims(padded_img, axis=-1)  # Add channel dimension
        return padded_img
    # Function to classify an image
    def classify_image(image_path):
        #Classifies an image as positive or negative.
        img = preprocess_image(image_path)
        img = np.expand_dims(img, axis=0)  # Add batch dimension
        prediction = classification_model.predict(img)[0]
        max_prob = np.max(prediction)
        return max_prob

    # Process negative images
    def process_negative_images(image_paths):
        #Processes negative images: classifies them and counts correct/incorrect classifications.
        global negative_correct, negative_incorrect

        for image_path in image_paths:
            max_prob = classify_image(image_path)

            if max_prob < CLASSIFICATION_THRESHOLD:  # Correctly classified as negative
                negative_correct += 1
            else:  # Incorrectly classified as positive
                negative_incorrect += 1

    # Get the list of negative image files
    negative_files = [os.path.join(negative_images_path, f) for f in
```

```
os.listdir(negative_images_path) if f.endswith(".jpg")]

    # Limit the number of negative images
    negative_files = negative_files[:MAX_NEGATIVE_IMAGES]

    # Process negative images
    process_negative_images(negative_files)

    # Output classification results for negative images
    print("\nNegative image classification results:")
    print(f"Correctly classified = {negative_correct}")
    print(f"Incorrectly classified = {negative_incorrect}")

    import tensorflow as tf

    def evaluate_model_predictions_with_counts_and_accuracy(model_path, images_test,
labels_test, label_encoder):

        # Load the best saved model
        model = tf.keras.models.load_model(model_path)
        print(f"Loaded model from {model_path}")

        # Get predictions for the test images
        predictions = model.predict(images_test, verbose=1)

        # Convert prediction probabilities to class indices
        predicted_classes = np.argmax(predictions, axis=1)

        # Decode class indices into class names
        predicted_labels = label_encoder.inverse_transform(predicted_classes)
        real_labels = label_encoder.inverse_transform(labels_test)

        # Initialize counters for True and False predictions per class
        counts = {
            "True": Counter(),
            "False": Counter(),
            "Accuracy": {}
```

```
    }

    # Count True and False predictions for each class
    for real, pred in zip(real_labels, predicted_labels):
        if real == pred:
            counts["True"][real] += 1
        else:
            counts["False"][real] += 1

    # Calculate accuracy for each class
    for class_name in label_encoder.classes_:
        true_count = counts["True"].get(class_name, 0)
        false_count = counts["False"].get(class_name, 0)
        total = true_count + false_count
        accuracy = (true_count / total) * 100 if total > 0 else 0
        counts["Accuracy"][class_name] = accuracy

    # Print class-wise results
    print("\nClass-wise Results:")
    print(f"{'Class':<15} {'True':<5} {'False':<5} {'Accuracy (%)':<10}")
    print("=" * 40)
    for class_name in label_encoder.classes_:
        true_count = counts["True"].get(class_name, 0)
        false_count = counts["False"].get(class_name, 0)
        accuracy = counts["Accuracy"][class_name]
        print(f"{class_name:<15} {true_count:<5} {false_count:<5} {accuracy:<10.2f}")

    # Calculate overall accuracy
    overall_accuracy = np.mean(predicted_classes == labels_test) * 100
    print(f"\nOverall Accuracy on Test Data: {overall_accuracy:.2f}%")

    return counts

# Function call to evaluate the model
counts = evaluate_model_predictions_with_counts_and_accuracy(
    "best_classification_model.keras",  # Path to the saved model
    images_test,
```

```
    labels_test,
    label_encoder
)

import tensorflow as tf

def evaluate_model_predictions_with_counts_and_accuracy(model_path, images_test,
labels_test, label_encoder):

    # Load the best saved model
    model = tf.keras.models.load_model(model_path)
    print(f"Loaded model from {model_path}")

    # Get predictions for the test images
    predictions = model.predict(images_test, verbose=1)

    # Convert prediction probabilities to class indices
    predicted_classes = np.argmax(predictions, axis=1)

    # Decode class indices into class names
    predicted_labels = label_encoder.inverse_transform(predicted_classes)
    real_labels = label_encoder.inverse_transform(labels_test)

    # Initialize counters for True and False predictions per class
    counts = {
        "True": Counter(),
        "False": Counter(),
        "Accuracy": {}
    }

    # Count True and False predictions for each class
    for real, pred in zip(real_labels, predicted_labels):
        if real == pred:
            counts["True"][real] += 1
        else:
            counts["False"][real] += 1
```

```
    # Calculate accuracy for each class
    for class_name in label_encoder.classes_:
        true_count = counts["True"].get(class_name, 0)
        false_count = counts["False"].get(class_name, 0)
        total = true_count + false_count
        accuracy = (true_count / total) * 100 if total > 0 else 0
        counts["Accuracy"][class_name] = accuracy

    # Print class-wise results
    print("\nClass-wise Results:")
    print(f"{'Class':<15} {'True':<5} {'False':<5} {'Accuracy (%)':<10}")
    print("=" * 40)
    for class_name in label_encoder.classes_:
        true_count = counts["True"].get(class_name, 0)
        false_count = counts["False"].get(class_name, 0)
        accuracy = counts["Accuracy"][class_name]
        print(f"{class_name:<15} {true_count:<5} {false_count:<5} {accuracy:<10.2f}")

    # Calculate overall accuracy
    overall_accuracy = np.mean(predicted_classes == labels_test) * 100
    print(f"\nOverall Accuracy on Test Data: {overall_accuracy:.2f}%")

    return counts

# Function call to evaluate the model
counts = evaluate_model_predictions_with_counts_and_accuracy(
    "best_classification_model.keras",  # Path to the saved model
    images_test,
    labels_test,
    label_encoder
)
```