МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри ПМ та МСС
_____ Коплик І. В.

____ _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

зі спеціальності 113 «Прикладна математика ,
освітньо-професійної_ програми «Наука про дані та моделювання складних систем»
на тему: «Розробка та навчання нейронної мережі для виявлення ракових пухлин на знімках легень»

Здобувачки групи ПМ.м-21 Чухно Віри Юріївни

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

_____ Віра Чухно
(підпис)

Керівник кандидат фіз.-мат. наук, доцент Аліна Дворниченко _____
(підпис)

Суми – 2023

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**
Кафедра **прикладної математики та моделювання складних систем**
Рівень вищої освіти другий (магістр)

Галузь знань **11 «Математика та статистика»**
Спеціальність **113 «Прикладна математика»**
Освітня програма **освітньо-професійна «**Наука про дані та моделювання складних систем**»**

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМтаМСС
Коплик І. В._____
«___» _____20__ р.

І Н Д И В І Д У А Л Ь Н Е З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧЕВІ ВИЩОЇ ОСВІТИ

Чухно Віра Юріївна
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та навчання нейронної для виявлення ракових пухлин легень

Керівник роботи Дворниченко Аліна Василівна
канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада)

затверджено наказом по факультету ЕлІТ від «07»листопада 2023 р.№1237-VI

2. Термін подання роботи здобувачем «16» грудня 2023р.

3. Вихідні дані до роботи Набір КТ знімків грудної клітини зі злоякісними пухлинам

4. Зміст розрахунково-пояснювальної записки (перелік питань, для розроблення): 1) Літературний огляд і аналіз існуючих підходів; 2) Теоретичні аспекти нейронних мереж; 3) Підготовка бази даних; 4) Розробка моделі; 5) Оцінка результатів та порівняння з існуючими методами.

5. Перелік графічного матеріалу: схеми архітектури нейронних мереж, графіки та діаграма ефективності різних моделей, знімки результатів вивчення,

рисунки результатів прогнозування.

6. Консультанти проєкту (роботи) із зазначенням розділів проєкту, що їх стосується

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1)	Літературний огляд і аналіз існуючих підходів	06.11 – 12.11	
2)	Ознайомлення з теоретичними аспектами нейронних мереж	13.11 – 20.11	
3)	Створення бази даних для подальшої розробки моделі	21.11 – 25.11	
4)	Розробка моделі для виявлення пухлин	26.11 – 05.12	
5)	Оцінка результатів та порівняння з існуючими методами	06.12 – 16.12	

Здобувач вищої освіти _____ Чухно В.Ю
(підпис)

Керівник роботи _____ Дворниченко А.В.
(підпис)

АНОТАЦІЯ

Звіт містить: 62 с., 23 рис., 28 джерел, 2 додатки.

Мета роботи: розробка та навчання нейронної мережі для виявлення ракових пухлин на медичних зображеннях легень, зокрема на знімках комп'ютерної томографії грудної клітини.

Об’єкт досліджень: медичні зображення легень, які містять інформацію про наявність чи відсутність пухлин.

Предмет дослідження: нейронні мережі, спрямовані на виявлення ракових пухлин на медичних зображеннях легень, їхні різні конфігурації та параметри.

У роботі аналізуються основні аспекти нейронних мереж, їхні різні конфігурації та параметри з метою покращення точності виявлення пухлин.

Отримані знання використовуються для розробки та навчання моделі, що виявляє ракові пухлини на знімках комп'ютерної томографії грудної клітини. Висновки та фінальні розрахунки презентовані на завершальних сторінках роботи. До дипломної роботи додаються додатки, що містять вихідний код програм, використаних у дослідженні.

Ключові слова: РАК ЛЕГЕНЬ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, CNN, INCEPTIONV3, RESNET-50.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ANN – Штучна нейронна мережа

CNN – Згорткова нейронна мережа

FC – Повністю зв’язний рівень

ReLU – Випрямлена лінійна одиниця

ResNet – Residual Network

RNN – Рекурентна нейронна мережа

SVM – Метод опорних векторів

VGG – Visual Geometry Group

ДРЛ – Дрібноклітинний рак легень

КТ – Комп’ютерна томографія

МРТ – Магнітно-резонансна томографія

НДРЛ – Недрібноклітинний рак легень

ПЕТ – Позитронно-емісійна томограф

ЗМІСТ

ВСТУП

Сучасна медична діагностика стикається з низкою викликів, зокрема в сфері виявлення ракових захворювань. Рак легень, як одна з найпоширеніших та смертоносних форм раку, вимагає особливої уваги та пошуку нових методів для ранньої та ефективної діагностики. Актуальність цієї проблеми визначається не лише високою поширеністю хвороби, але й необхідністю покращення точності та швидкості діагностики.

Несприятливий вплив раку легень на здоров'я населення та загальні медичні витрати робить цю проблему надзвичайно актуальною. Покращення методів виявлення пухлин та рання діагностика можуть суттєво підвищити ефективність лікування та збільшити шанси на повне одужання.

На сьогоднішній день, розпізнавання ракових пухлин на знімках легень в основному базується на ручному аналізі лікарів-рентгенологів. Хоча існують певні програмні рішення та системи допомоги у діагностиці, але їх ефективність часто обмежена. Зокрема, недоліки можуть включати низьку точність, особливо на ранніх стадіях хвороби, та обмеженість у роботі з великими обсягами даних. Таким чином, важливо розробити нові методи, які враховують ці обмеження та забезпечують кращу роботу в умовах реальної клінічної практики.

Метою даної роботи є розробка та навчання нейронної мережі для виявлення ракових пухлин на знімках легень. Підхід спрямований на подолання недоліків попередніх методів шляхом використання сучасних методів машинного навчання. Дана робота ставить за мету підвищення точності та швидкості діагностики, зменшення втручання людини та розробку методу, що ефективно працює з великими обсягами медичних зображень.

Таким чином, дане дослідження заповнює прогалини в існуючих методологіях, враховуючи новітні досягнення у сфері машинного навчання та нейронних мереж, і створює основу для подальших досліджень в цій важливій галузі.

ПОСТАНОВКА ЗАДАЧІ

1. Провести огляд актуальних досліджень в області виявлення ракових пухлин легень за допомогою нейронних мереж.
2. Проаналізувати різноманітні архітектури нейронних мереж, такі як згорткові нейронні мережі, InceptionV3, ResNet-50 та інші.
3. Підготувати базу даних з медичними знімками для подальшого використання.
4. Вибрати оптимальну архітектуру для моделі виявлення ракових пухлин на КТ знімках легень; розробити та навчити нейронну мережу, використовуючи вибрану архітектуру.
5. Провести експерименти та оцінити результати розробленої моделі; здійснити порівняльний аналіз з існуючими моделями.

РОЗДІЛ 1. АНАЛІТИЧНИ ОГЛЯД

1.1. Рак легень

Рак легень є третім за поширеністю видом раку в країні, поступаючись лише раку молочної залози у жінок і раку передміхурової залози у чоловіків [1]. Незважаючи на високий рівень складності у лікуванні, медичні фахівці та дослідники досягають постійних успіхів у виявленні патології на ранніх етапах.

Процес формування раку легень виникає тоді, коли аномальні клітини починають розвиватися у легенях, а потім швидко розповсюджуються за межі, які імунна система не може ефективно контролювати. Це призводить до утворення пухлин, які в свою чергу перешкоджають нормальному функціонуванню легеневих тканин. Зазвичай рак легень розпочинається в дихальних шляхах, таких як бронхи чи бронхіоли, або в невеликих повітряних мішках, які називаються альвеолами, розташованими у ваших легенях. Після цього він може поширитися на інші органи.

Застосування новітніх методів діагностики та розвиток ефективних терапевтичних стратегій дозволяють виявляти рак легень на ранніх стадіях, що істотно підвищує шанси на успішне лікування. Важливими факторами у боротьбі з цією хворобою є попередження ризикових факторів, вчасне виявлення симптомів та регулярні медичні обстеження.

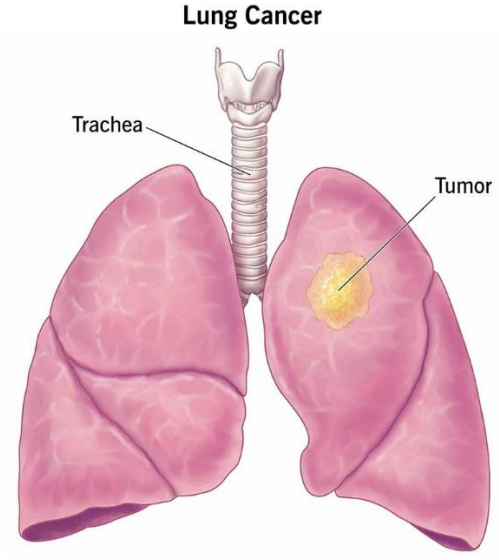

Рисунок 1.1. – Приклад легень з пухлиною

1.1.1. Фактори ризику та симптоми раку легень

Основними факторами ризику раку легень є:

1) *Паління:* Близько 90% випадків раку легень пов'язані з курінням. Тютюновий дим містить речовини, що можуть пошкодити клітини, які вистилають легені. Відмова від куріння може значно знизити ризик розвитку раку та інших захворювань, але майже 60% випадків раку легень діагностується у тих, хто колись курив.

2) *Пасивне куріння*: Вплив пасивного куріння призводить до тисяч випадків раку легень щороку в Україні.

3) *Стать:* В Україні рак легень частіше діагностується у чоловіків. Близько 59 випадків на 100 000 серед чоловіків порівняно з приблизно 47 випадками на 100 000 серед жінок. Показники можуть варіюватися залежно від вікової категорії та етнічної групи.

4) *Вік:* На момент діагностики більше двох третин пацієнтів мають 65 років і старше.

5) *Навколишнє середовище:* Вплив забруднюючих речовин, таких як вихлопи транспортних засобів та хімічні пари, може підвищити ризик. Вплив газу радону, що може виникати внаслідок розпаду урану в ґрунті, також може збільшити ризик.

6) *Сімейна історія:* Успадковані гени можуть збільшити ризик для тих, хто курить і тих, хто не курить. Якщо є історія раку легень в сім'ї або якщо ви вважаєте себе високо ризиковою особою, експертне генетичне консультування та оцінка ризику можуть допомогти керувати цією ситуацією.

Симптоми раку легень можуть варіювати в залежності від типу раку та стадії захворювання. Ось деякі загальні симптоми, які можуть вказувати на можливість наявності раку легень:

1) Постійний кашель: Особливо тривалий кашель, який не піддається лікуванню і може супроводжуватися виплутуванням крові.

2) Зміни у диханні: Задишка або важкість у диханні, яка може з'явитися навіть при невеликих фізичних навантаженнях.

3) Біль у грудях: Неприємні відчуття або біль у грудях, який може поглиблюватися при кашлі або глибокому диханні.

4) Втрата ваги і апетиту: Неконтрольована втрата ваги та відчуття втрати апетиту.

5) Слабкість і втомленість: Загальна слабкість, втомленість та втрата енергії.

6) Системні симптоми: Лихоманка, підвищена температура тіла, незаспокоєна нічний піт та інші системні ознаки.

7) Голосові зміни: Зміни в голосі або хриплість, які можуть бути пов'язані з ураженням дихальних шляхів. Погіршення загального стану здоров'я: Загальне погіршення самопочуття та здоров'я, що не може бути пояснене іншими причинами.

1.1.2. Типи раку легень

Існує широкий спектр видів раку, які можуть вражати легені, але загалом ми використовуємо термін "рак легень" для двох основних форм: недрібноклітинного раку легень (НДРЛ) [2] і дрібноклітинного раку легень (ДРЛ)[3]. НДРЛ є найпоширенішим типом, становлячи понад 80% всіх випадків раку легень. ДРЛ розвивається швидше і має вищий рівень ускладнення в лікуванні порівняно з НДРЛ. Зазвичай він виявляється як невелика пухлина легень, яка вже має метастазувати на інші частини тіла.

До НДРЛ належать такі типи[4]:

- Аденокарцинома: Це найпоширеніший підтип, особливо серед некурців. Зазвичай він розвивається в периферійних частинах легень.
- Плоскоклітинна карцинома: Частіше пов'язаний з курінням і виникає в центральних частинах легень.
- Аденосквамозна карцинома: Це комбінований підтип, який включає елементи аденокарциноми та плоскоклітинної карциноми.

- Саркоматоїдна карцинома: Це рідкісний підтип, який характеризується високим рівнем агресивності.

До ДРЛ належать [5]:

- Дрібноклітинний рак (карцинома вівса): Це агресивний підтип, який швидко поширюється і часто виявляє метастази на момент діагностики.
- Комбінований дрібноклітинний рак: Містить елементи інших типів раку.

Інші типи раку, пов'язані з легенями:

- Лімфома: Це рак у лімфатичних вузлах, який може впливати на легені.
- Саркома: Це рак у кістках або м'яких тканинах, який може виникнути в легенях або навколишніх областях.

1.2. Інструментальні методи дослідження грудної клітини

Діагностика раку легень проходить в кілька етапів, починаючи зі збору анамнестичних даних пацієнта, фізичного огляду, лабораторних аналізів і інструментальних методів дослідження[6].

Для підтвердження та встановлення заключного діагнозу використовують такі інструментальні методи: рентгенографія грудної клітини, комп’ютерна томографія (КТ), біопсія, магнітно-резонансна томографія (МРТ), позитронно-емісійна томографія під наглядом КТ (ПЕТ/КТ).

1.2.1. Рентген грудної клітки

Рентген грудної клітки - це один із методів образної діагностики, який використовується для вивчення структур в області грудної клітки, зокрема легень. Цей метод може бути застосований для діагностики раку легень, але важливо враховувати його обмеження та можливості[7].

Принцип рентгенівського дослідження полягає в тому, що рентгенівські промені проходять через тканини тіла, але поглиблюються різноманітним чином

в залежності від щільності тканин. Коли промені проходять через легені, вони вбираються менше, що створює зображення, на якому видно контури структур.

Щодо діагностики раку легень, рентген грудної клітки може допомагати виявити зміни в легенях, такі як темні або неоднорідні області, які можуть бути підозрілі на наявність пухлини або інших патологічних процесів.

Проте, важливо враховувати, що рентген грудної клітки має свої обмеження. Він може не завжди точно визначати тип чи стадію раку, інші методи, такі як КТ чи ПЕТ часто використовуються для більш точної діагностики та визначення стадії раку легень.

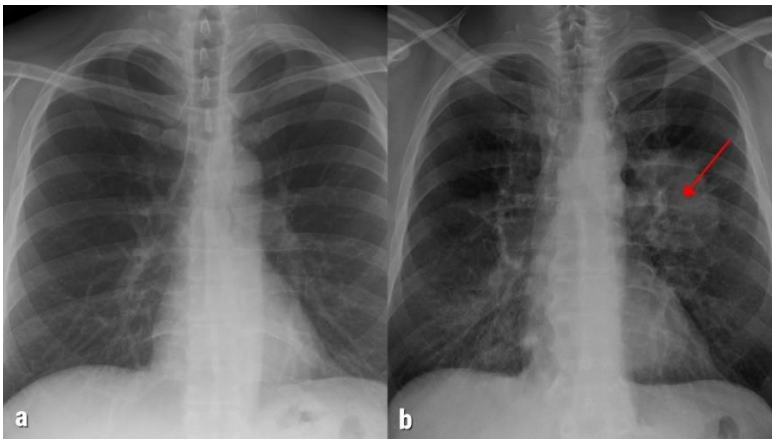

Рисунок 1.2 – Рентген грудної клітки; a – здорові легені; b – легені з раковою пухлиною.

1.2.2. Комп’ютерна томографія

КТ використовує рентгенівські промені для створення детальних зображень поперечних перерізів тіла. У порівнянні зі звичайним рентгеном, який робить лише 1-2 знімки, сканер КТ виконує багато знімків, а комп'ютер об'єднує їх для створення зображень досліджуваної області.

КТ має високу ефективність у виявленні пухлин легень порівняно із звичайним рентгеном грудної клітки. Вона дозволяє визначити розмір, форму та положення пухлини, а також виявляти збільшені лімфатичні вузли, які можуть бути заражені раком [8].

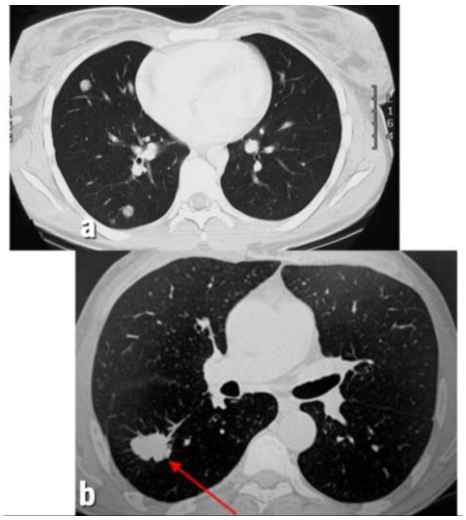

Рисунок 1.3 – КТ легень; a – здорові легені; b – легені з раковою пухлиною.

1.2.3. Магнітно-резонансна томографія

МРТ є одним із методів діагностики, який може бути використаний для виявлення та оцінки раку легень. Цей нетравматичний метод надає детальні зображення внутрішніх структур тіла за допомогою магнітних полів і радіохвиль [7].

МРТ надає дуже детальні зображення тканин, що дозволяє лікарям ретельно досліджувати структури легень та навколишні органи. На відміну від рентгенівських методів, МРТ не використовує іонізуюче випромінювання, що дозволяє уникнути потенційних шкідливих впливів. МРТ може бути особливо корисною для визначення розміру та ступеня ураження пухлини, а також виявлення можливих ускладнень, таких як метастази чи обструкція бронхів.

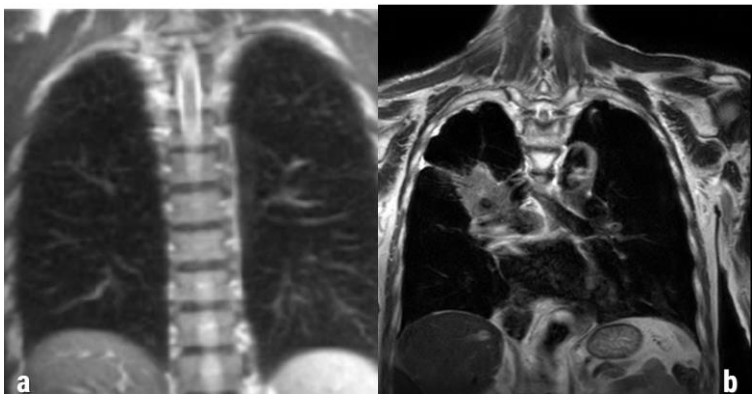

Рисунок 1.4 – МРТ легень; a – здорові легені; b – легені з раковою пухлиною.

1.2.4. Позитронна-емісійна томографія

ПЕТ- це діагностичний метод, який використовує радіоактивні речовини для визначення функціональної активності в тканинах організму. У контексті діагностики раку легень, ПЕТ зазвичай поєднується з комп'ютерною томографією (ПЕТ/КТ) для отримання детальних структурних і функціональних зображень [9].

ПЕТ може виділяти області в організмі, де є підвищений обмін речовин, що часто характерно для злоякісних новоутворень. Це дозволяє виявляти ракові клітини у ранніх стадіях. Інформація, отримана з ПЕТ/КТ, може бути використана для точного планування лікування, включаючи радіотерапію та хірургічні втручання.

1.3. Застосування машинного навчання для діагностики раку легень

Машинне навчання - це набір методів, які дозволяють комп'ютерам виконувати складні завдання, включаючи аналіз великих обсягів даних. Зростання обчислювальної потужності, доступності сховищ і пам'яті, а також збільшення обсягів даних сприяло розвитку цієї галузі. У сфері медицини машинне навчання використовується для діагностики, прогнозування результатів, аналізу зображень і обробки текстової інформації.

Алгоритми машинного навчання успішно використовуються для класифікації раку та прогнозування розвитку діабету. Зазвичай машинне навчання поєднується з обробкою природної мови для аналізу неструктурованих текстових даних, таких як звіти клінічних випадків та відгуки пацієнтів. Ключова роль машинного навчання в розвитку систем охорони здоров'я полягає в створенні навчальних систем, які об'єднують різноманітні джерела даних та алгоритми для оптимізації біомедичних досліджень і надання медичної допомоги.

У медицині методи машинного навчання застосовуються для аналізу клінічних даних, обробки зображень, діагностики та прогнозування захворювань. Деякі основні методи включають в себе використання нейронних

мереж, таких як зображення з використанням Convolutional Neural Networks (CNN) та обробка послідовних даних за допомогою Recurrent Neural Networks (RNN). Також використовуються дерева рішень (Random Forests), ансамблеві методи (Gradient Boosting), методи опорних векторів (SVM), кластеризація[10].

Застосування машинного навчання для діагностики раку легень включає в себе використання алгоритмів та моделей машинного навчання для обробки медичних даних та виявлення патернів, які можуть вказувати на наявність або ризик розвитку раку.

Застосування машинного навчання у діагностиці раку легень допомагає покращити швидкість, точність та індивідуалізацію процесу діагностики, сприяючи покращенню результатів лікування та розумінню характеристик цієї хвороби.

РОЗДІЛ 2. МЕТОДИКА ДОСЛІДЖЕНЬ

2.1. Нейронні мережі

Нейронна мережа - це система обчислювальних елементів, які намагаються моделювати роботу людського мозку. Ці мережі складаються з великої кількості взаємопов'язаних штучних нейронів, які працюють разом для вирішення конкретних завдань.

Ідея полягає в тому, що ці штучні нейрони обмінюються інформацією через зв'язки, які мають ваги. Коли мережа навчається на великій кількості даних, ваги змінюються так, щоб оптимально вирішувати поставлену задачу. Навчання відбувається методом вагового налаштування на основі великої кількості прикладів[11].

Існує кілька типів нейронних мереж, таких як прямі (повністю з'єднані) мережі, згорткові мережі (використовуються для обробки зображень), рекурентні мережі (використовуються для роботи з послідовностями даних, такими як мова чи часові ряди) та інші[12].

Нейронні мережі використовуються в різних областях для:

- Ідентифікація об’єктів, обличчя і розуміння розмовної мови в таких програмах, як безпілотні автомобілі та голосові помічники.
- Аналіз і розуміння людської мови, увімкнення аналізу настроїв, чат-ботів, переклад і генерація тексту.
- Діагностика захворювань за допомогою медичних зображень, прогнозування результатів пацієнтів і відкриття ліків.
- Прогнозування цін на акції, оцінка кредитного ризику, виявлення шахрайства та алгоритмічна торгівля.
- Персоналізація вмісту та рекомендацій в електронній комерції, потокових платформах і соціальних мережах.
- Забезпечення робототехніки та автономних транспортних засобів шляхом обробки даних датчиків і прийняття рішень у реальному часі.

- Удосконалення штучного інтелекту в іграх, створення реалістичної графіки та створення захоплюючих віртуальних середовищ.
- Моніторинг та оптимізація виробничих процесів, прогнозне обслуговування та контроль якості.
- Аналіз складних наборів даних, моделювання наукових явищ і допомога в дослідженнях у різних дисциплінах.
- Створення музики, мистецтва та іншого творчого контенту.

Існує 3 типи навчання в нейронних мережах[11]:

1. *Контрольоване навчання:* Цей тип навчання, також відомий як навчання з учителем, передбачає наявність супервізора чи учителя, який надає набори вхідних даних і відповідних вихідних результатів. Модель порівнює свій вихід із бажаним результатом, а потім коригує ваги, враховуючи помилку. Цей процес триває до тих пір, поки модель не досягне високої точності.

2. *Навчання без контролю:* Навчання без контролю відбувається без супервізора чи конкретних бажаних результатів. Мережа самостійно групує вхідні дані в класи, визначаючи подібність між членами кожного класу. Відсутність зворотного зв'язку і бажаного результату дозволяє моделі самостійно вивчати структури та патерни в даних.

3. *Навчання з підкріпленням:* Цей тип навчання отримує вплив як з контрольованого, так і з неконтрольованого навчання. Він використовує критичний фідбек замість точних вихідних результатів, щоб вдосконалювати модель. Модель навчається на основі критичної інформації, що дозволяє йому самостійно вдосконалювати свої рішення. Це подібно до контрольованого навчання, але з критичним фідбеком замість точної відповіді.

2.2. Штучна нейронна мережа

Штучна нейронна мережа (ANN) - це нейронна мережа прямого зв'язку, де вхідні дані передаються в прямому напрямку. Вона може містити приховані шари, що роблять модель ще більш компактною. Зазвичай використовується для

текстових або табличних даних. Один з популярних застосувань - розпізнавання обличчя[13].

Архітектура ANN включає в себе кілька ключових елементів, які визначають її структуру та функціональність. Основні компоненти архітектури нейронної мережі включають вхідний шар, приховані шари (якщо вони присутні), і вихідний шар (рис. 2.1.). Ось короткий огляд кожного елемента[14]:

1. *Вхідний шар:*
    - Приймає вхідні дані у визначеному форматі.
    - Скільки нейронів в цьому шарі залежить від кількості вхідних параметрів або функцій.
2. *Приховані шари:*
    - Виконують обчислення та виявлення прихованих особливостей у вхідних даних.
    - Кількість та розміщення прихованих шарів може варіюватися. Кожен нейрон в прихованому шарі з'єднаний з кожним нейроном попереднього та наступного шарів.
3. *Вихідний шар:*
    - Генерує вихідні дані після обчислень, проведених в прихованих шарах.
    - Кількість нейронів у вихідному шарі зазвичай відповідає кількості класів (у випадку задач класифікації) або вихідних значень.
4. *Зв'язки та ваги:*
    - Нейрони в одному шарі з'єднані з нейронами в сусідніх шарах за допомогою зв'язків.
    - Кожен зв'язок має вагу, яка визначає його вплив на передачу сигналу від одного нейрона до іншого. Ваги змінюються під час процесу навчання.
5. *Функції активації:*
    - Визначає, як сигнал, отриманий нейроном, буде перетворений у вихідний сигнал.

- Функції активації додають нелинійність до моделі, дозволяючи нейронам виявляти складніші залежності у вхідних даних.

6. *Байеси та функції втрат:*
   - Використовуються для оцінки точності та коригування параметрів мережі під час навчання.

Ці компоненти спільно створюють структуру та функціональність штучної нейронної мережі, яка може бути налаштована для вирішення різних завдань.

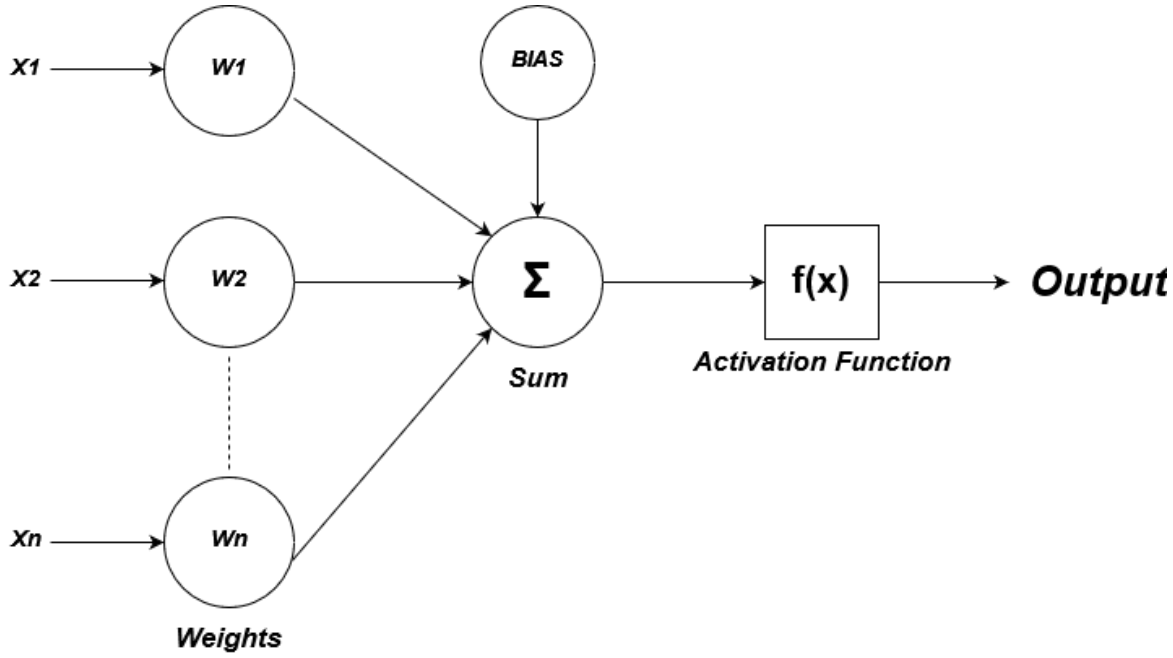

Рисунок 2.1. – Архітектура ANN

Штучний нейрон є основним компонентом нейронних мереж, моделюючи основні функції природного нейрона (рис. 2.1). Під час своєї роботи нейрон отримує одночасно багато вхідних сигналів, кожен з яких має свою власну синаптичну вагу. Ця вага визначає вплив кожного входу, необхідний для суматора обробки в нейроні. Ваги служать мірою сили вхідних зв'язків і відображають різноманітні синаптичні сили біологічних нейронів. Суттєві вхідні сигнали підсилюються, тоді як ваги несуттєвих входів зменшуються, що визначає інтенсивність вхідного сигналу[15].

Ваги можуть змінюватись залежно від навчальних прикладів, топології мережі та навчальних правил. Вхідні сигнали (позначені як $x_1, x_2, \dots x_n$)

зважуються ваговими коефіцієнтами з'єднання (позначеними як $w_1, w_2, \dots, w_n$), сумуються, пройшовши через передатну функцію, і генерують результат, який виводиться. Цей процес є ключовим у функціонуванні нейронів та нейронних мереж, де ваги визначають, як важливі різні входи для формування відповіді.

Для наведеної вище загальної моделі штучної нейронної мережі (рис.2.1) чистий вхід можна розрахувати таким чином:

$$y_{im} = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b, \quad \text{тобто } y_{im} = \sum_i^n (x_i w_i) + b. \qquad (2.1)$$

Зсув (bias), позначений як b, є додатковим параметром у кожному нейроні штучної нейронної мережі. Це константа, яка додається до зваженого сумарного входу перед застосуванням функції активації.

Додавання зсуву дозволяє моделі нейронної мережі зміщувати функцію активації вгору чи вниз, що є важливим для того, щоб нейрони могли вчитися правильно вирішувати завдання. Зсув дозволяє моделі легше адаптуватися до різних умов та навчатися оптимальним вагам для вхідних сигналів.

Вихід можна розрахувати шляхом застосування функції активації до чистого входу:

$$Y = F(y_{im}). \qquad (2.2)$$

Функція активації грає ключову роль у роботі нейронних мереж. Основні функції активації вводять нелінійність у модель, що робить їх здатними вирішувати більш складні завдання. Існує кілька основних функцій активації[16]:

1) Сигмоїдальна функція (Sigmoid):

$$\sigma(x) = \frac{1}{1+e^{-x}}. \qquad (2.3)$$

Приймає будь-яке значення і "стискає" його в діапазон від 0 до 1. Це дозволяє використовувати її для прогнозування ймовірностей у бінарних класифікаційних задачах.

2) Гіперболічний тангенс (tanh):

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}. \qquad (2.4)$$

Аналогічно сигмоїдній функції, але "стискає" значення в діапазон від -1 до 1. Також використовується у завданнях класифікації та регресії.

3) ReLU (Rectified Linear Unit):

$$f(x) = \max(0, x)\,. \qquad (2.5)$$

Приймає значення x, і якщо воно від'ємне, видає 0; в іншому випадку повертає саме x. ReLU є однією з найпопулярніших функцій активації, особливо в глибоких нейронних мережах.

4) Лінійна функція:

$$f(x) = x. \qquad (2.6)$$

Не вводить нелінійність, але іноді використовується у вихідному шарі для регресії, коли потрібен прямий вихід без обмежень.

2.3. Рекурентна нейронна мережа

Рекурентна нейронна мережа (RNN) - це тип штучної нейронної мережі, призначений для роботи з послідовностями даних та моделювання залежностей в часі. Основна ідея RNN полягає в тому, що вона має внутрішню пам'ять, яка дозволяє їй зберігати і використовувати інформацію з попередніх кроків в обробці нових вхідних даних[17].

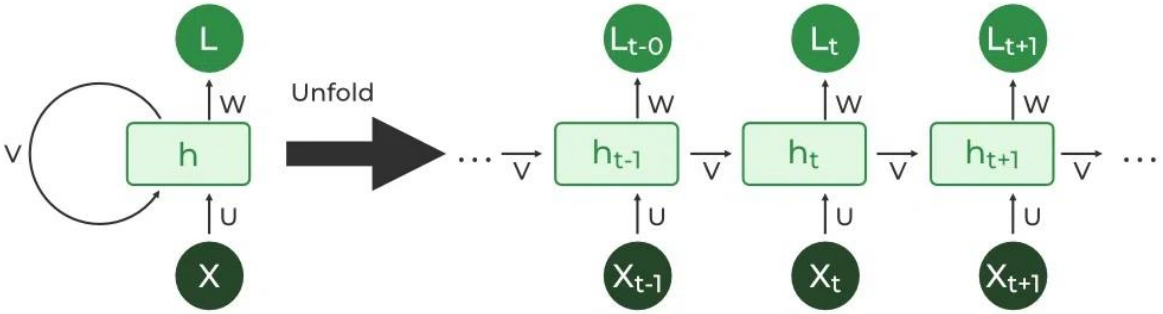

Рисунок 2.2 - RNN як у згорнутому, так і в розгорнутому вигляді

На рис.2.2 є кілька окремих компонентів, серед яких найважливіші:

- *x*: вхід. Це може бути слово в реченні або інший тип послідовних даних.

- L: Вихід. Наприклад, те, що мережа думає, наступне слово в реченні має бути надано попереднім словам.
- h: Основний блок RNN. Він містить ваги та функції активації мережі.
- V: представляє комунікацію від одного часового кроку до іншого.

Згорнуте та розгорнуте зображення мережі на малюнку еквівалентні. Іноді корисно розгорнути мережу, щоб краще зрозуміти, що відбувається на кожному кроці.

Рекурентні нейронні мережі використовують однакові ваги для кожного елемента послідовності, зменшуючи кількість параметрів і дозволяючи моделі узагальнювати послідовності різної довжини. RNN узагальнюють структуровані дані, відмінні від послідовних даних, наприклад географічні або графічні дані, через їх дизайн.

Повторювана нейронна мережа складається з кількох фіксованих функціональних одиниць активації, по одній на кожен часовий крок. Кожна одиниця має внутрішній стан, який називається прихованим станом одиниці. Цей прихований стан означає минулі знання, які мережа зберігає на даний момент часу. Цей прихований стан оновлюється на кожному кроці часу, щоб вказати на зміну знань мережі про минуле. Прихований стан оновлюється за допомогою наступного відношення повторення:

$$h_t = f(h_{t-1}, x_t), \tag{2.7}$$

де h_t – поточний стан, h_{t-1} – попередній стан, x_t – вхідний стан.

Формула для застосування функції активації (tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t), \tag{2.8}$$

де W_{hh} - вага на рекурентному нейроні, W_{xh} - вага на вхідному нейроні.

Формула розрахунку виходу:

$$y_t = W_{hy}h_t, \tag{2.9}$$

де y_t – вихід, W_{hy} – вага на вхідному шарі.

Ці параметри оновлюються за допомогою зворотного поширення. Однак, оскільки RNN працює з послідовними даними, то необхідно використовувати оновлене зворотне поширення, яке відоме як зворотне поширення в часі.

Архітектура RNN може відрізнятися залежно від проблеми, яку ви намагаєтеся вирішити. Від тих із одним входом і виходом до тих із багатьма (з варіаціями між ними)[18].

2.4. **Згорткові нейронні мережі**

Згорткова нейронна мережа (CNN) - це тип нейронних мереж, який часто використовується для обробки та розпізнавання зображень. Основна ідея полягає в тому, щоб використовувати згорткові шари для автоматичного виявлення важливих особливостей (фільтрів) у вхідних зображеннях, а потім піддавати ці виокремлені особливості пулінговим шарам для зменшення розмірності та забезпечення інваріантності до масштабу та позиції[19].

CNN були вперше розроблені та використані приблизно у 1980-х роках. Найбільше, що могла тоді CNN, це розпізнавати рукописні цифри. Здебільшого він використовувався в поштових секторах для зчитування поштових індексів, пін-кодів тощо. Важливо пам’ятати про будь-яку модель глибокого навчання: для її навчання потрібна велика кількість даних, а також багато обчислювальних ресурсів. Це було головним недоліком для CNN у той період, тому CNN обмежувалися лише поштовими секторами, і їм не вдалося увійти у світ машинного навчання [20].

У 2012 році Алекс Крижевський зрозумів, що настав час повертати гілку глибокого навчання, яка використовує багаторівневі нейронні мережі. Наявність великих наборів даних, а точніше, наборів даних ImageNet з мільйонами позначених зображень і велика кількість обчислювальних ресурсів дозволили дослідникам відродити CNN.

Архітектура згорткової нейронної мережі (CNN) включає дві основні компоненти [21]:

1. *Мережа виділення ознак:* Інструмент згортки використовується для розділення та ідентифікації різних характеристик зображення в процесі, відомому як вилучення функцій.

2. *Повністю пов'язаний рівень:* Повністю пов'язаний рівень отримує вихідні дані від процесу згортання та передбачає клас зображення на основі виділених ознак.

Модель вилучення ознак у CNN спрямована на зменшення кількості ознак у наборі даних та створення нових функцій, які узагальнюють існуючі. Ця архітектура має багато рівнів, які включають в себе згорткові, об'єднуючі та повністю пов'язані шари, і кожен рівень грає свою роль у процесі ефективного вилучення та використання важливих ознак для класифікації (рис.2.3). Окрім цих трьох рівнів, є ще два важливі параметри, а саме рівень вилучення та функція активації, визначені нижче.

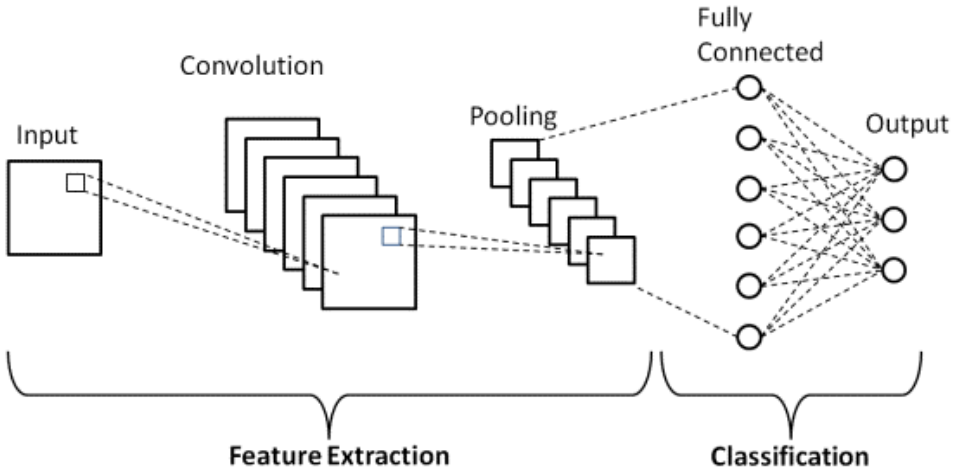

Рисунок 2.3 – Базова архітектура CNN [21]

2.4.1 Архітектура CNN

Згортковий рівень (Convolutional Layer) є основним будівельним блоком у згортковій нейронній мережі (CNN), відіграючи центральну роль у виконанні більшості обчислень. Він спирається на кілька ключових компонентів, включаючи вхідні дані, фільтри та карти функцій [22].

Згортка зберігає просторове співвідношення між пікселями, вивчаючи особливості зображення за допомогою невеликих квадратів вхідних даних. Це

тензорна операція (скалярний добуток), де два тензори служать вхідними даними, а результуючий тензор генерується як вихідні дані. Цей рівень використовує плитковий підхід до фільтрації вхідного тензора за допомогою маленького вікна, відомого як ядро . Ядро вказує конкретні характеристики, які операція згортки прагне відфільтрувати, генеруючи значну відповідь, коли вона виявляє бажані функції.

Згортковий шар обчислює скалярний добуток між значенням фільтра та значеннями пікселів зображення , а матриця, сформована ковзанням фільтра по зображенню, називається згорнутою функцією, картою активації або картою функцій.

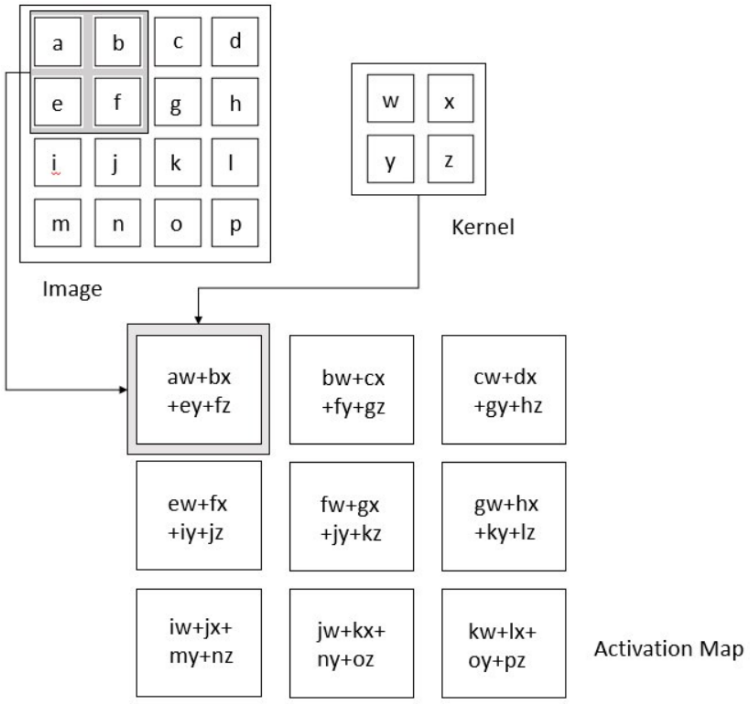

Рисунок 2.4 – Операція згортки

Кожен елемент з одного тензора (піксель зображення) множиться на відповідний елемент (елемент у тій же позиції) другого тензора (значення ядра), а потім усі значення підсумовуються, щоб отримати результат.

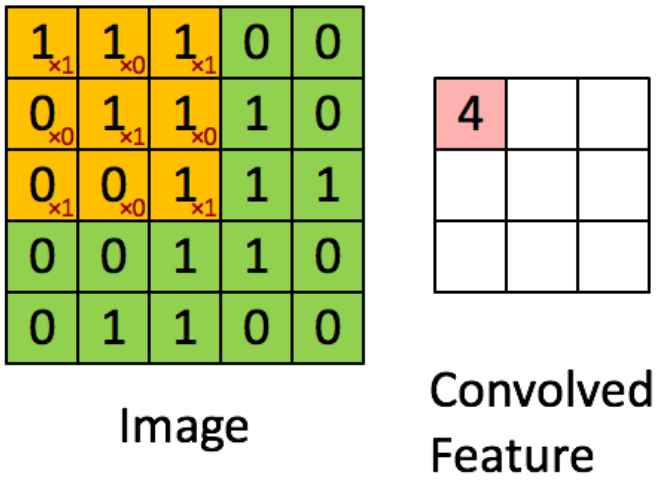

Рисунок 2.5 – Вилучення інформації про зображення

Поведінка згорткового шару в основному регулюється такими основними гіперпараметрами:

- *Kernel size (Розмір ядра):* Він визначає розмір розсувного вікна. Зазвичай рекомендується використовувати менші розміри вікон, бажано непарні значення, такі як 1, 3, 5 і іноді, рідко 7.
- *Stride:* Параметр stride визначає кількість пікселів, на яку переміщатиметься вікно ядра під час кожного кроку згортки. Зазвичай для нього встановлюється значення 1, щоб на зображенні не було пропущено жодного місця. Однак його можна збільшити, якщо планується одночасно зменшити розмір вхідних даних.
- *Padding:* Відступ відноситься до техніки додавання нулів до межі зображення. Застосовуючи доповнення, ядро може повністю відфільтрувати кожну позицію вхідного зображення, забезпечуючи належну обробку навіть країв.
- *Number of filters /Depth:* Кількість фільтрів у згортковому шарі визначає кількість шаблонів або особливостей, які цей шар намагатиметься ідентифікувати. Іншими словами, він керує кількістю окремих характеристик або елементів, на виявленні яких буде зосереджено згортковий рівень.

Вихідний об’єм згорнутого шару визначається кількома факторами, включаючи вхідний розмір, розмір ядра, крок і відступ. Формула для розрахунку вихідного розміру така:

$$V_{out} = 1 + \frac{V_{in}+2p-K}{S}, \quad (2.10)$$

де V_{in} – розмір вхідного об’єму, K – розмір ядра, p – величина відступу (padding), S – крок згортки.

Наступним шаром є Pooling Layer (рівень об’єднання). Він використовується для зменшення розмірності матриць карти ознак, які отримані зі згорткового шару. Його головною метою є зменшення складності моделі шляхом зменшення розмірів, зберігаючи при цьому ключові особливості зображення. Це робиться шляхом зменшення розмірів матриці по горизонталі та вертикалі, але зберігаючи кількість каналів (глибина) постійною [20].

Найпоширенішим методом об'єднання є «Максимальне об'єднання», де для кожного регіону у вихідній матриці вибирається найбільше значення. Це допомагає зберегти найважливіші особливості зображення, зменшуючи при цьому його розмір.

Ще одним методом є «Середнє об’єднання», де для кожного регіону вихідної матриці обчислюється середнє значення. Цей метод також допомагає зменшити розмір матриці, але використовує середні значення замість максимальних.

Обидва методи сприяють зменшенню кількості параметрів, що дозволяє зменшити обчислювальне навантаження на мережу. Крім того, вони сприяють мінімізації обсягу пам'яті, використованої для зберігання інформації про особливості зображення. Процес, який виконується шляхом взяття середнього значення в матрицях, називається «Середнє об’єднання». Ця структура наведена на рисунку 2.6.

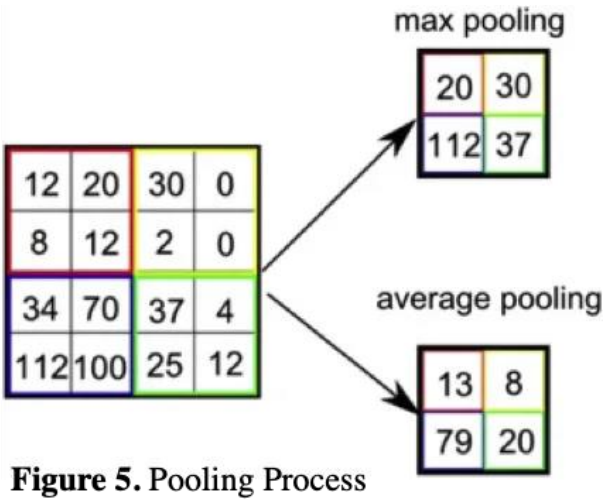

Figure 5. Pooling Process

Рисунок 2.6 – Процес об’єднання

Розмір вихідного об'єму після пулінгу обчислюється як:

$$V_{out} = \frac{V_{in} - poll\_size}{S} + 1. \quad (2.11)$$

Рівень Fully Connected (FC) є третім шаром в архітектурі CNN, який складається з ваг, зміщень та нейронів і використовується для повного з'єднання нейронів між двома різними шарами мережі. Зазвичай ці шари розташовані перед вихідним шаром і є частиною завершальних етапів архітектури CNN.

У цьому рівні вхідне зображення з попередніх шарів вирівнюється та подається на рівень FC. Сплощений вектор потім проходить через декілька шарів FC, де зазвичай виконуються операції математичних функцій. На цьому етапі починається процес класифікації, оскільки FC-шари визначають зв'язки між входами та визначають, до якого класу відноситься вхідне зображення [23].

Причина використання двох FC шарів полягає в тому, що вони можуть взаємодіяти та вивчати складні залежності між різними функціями або ознаками. Повністю з'єднані шари можуть допомагати моделі краще узагальнювати та робити прогнози на основі здобутих знань. Використання FC-шарів сприяє автоматизації процесу класифікації та зменшенню впливу людського контролю в тих аспектах, де потрібно виявлення високорівневих ознак або взаємодій в даних.

Вихід O з повністю зв'язаного шару обчислюється як лінійне відображення вхідного вектора X за допомогою матриці ваг W та додаванням зсуву b, а потім застосуванням функції активації A:

$$O = A(WX + b). \quad (2.12)$$

2.4.2 Додаткові параметри архітектури CNN

Поряд із зазначеними вище рівнями існують додаткові терміни, які є частиною архітектури CNN.

Функція активації в CNN грає ключову роль у введенні нелінійності в мережу. Без функцій активації нейронна мережа зі складовими шарами, такими як лінійні згортки і повністю з'єднані шари, буде просто виконувати лінійні перетворення вхідних даних, що робить їх еквівалентними одному лінійному шару. Додавання нелінійності дозволяє мережі вивчати складніше, не лінійне представлення даних. Зазвичай в CNN використовують такі функції активації як ReLU, Sigmoid, Tanh та Leaky ReLU [16].

Dropout Layer є шаром, який застосовує техніку dropout до вихідних значень нейронів. Цей шар допомагає уникнути перенавчання та покращити загальну здатність моделі до узагальнення на нові дані. Dropout випадковим чином "вимикає" (встановлює в нуль) вибрані нейрони під час тренування, тим самим змушуючи мережу навчатися більш резилієнтним функціональним представленням.

Математично, операція Dropout виглядає наступним чином:

$$\begin{cases} 0, \text{з ймовірністю } Dropout\ Rate \\ \frac{X}{1-Dropout\ Rate}, \text{інакше} \end{cases}, \quad (2.13)$$

Де X - вхідний тензор або вектор (вихід з Pooling Layer), Dropout Rate - ймовірність вимкнення нейрона (зазвичай визначається користувачем, наприклад, 0.25 означає, що кожен нейрон має ймовірність 25% бути вимкненим).

Важливо відзначити, що Dropout використовується лише під час тренування. Під час тестування всі нейрони залишаються активними, але їх вихід

змінюється так, щоб врахувати відсутність вимикання, яке було застосовано під час тренування.

Важливою складовою архітектури CNN є функція втрат (loss function). Функція втрати у зв'язку з CNN використовується для оцінки різниці між прогнозованими значеннями моделі та фактичними (вірними) значеннями. Задача функції втрати полягає в тому, щоб змінювати параметри моделі так, щоб ця різниця стала як найменшою. Основна ідея полягає в тому, щоб навчати модель на основі втрати та вдосконалювати її прогнозуючі здібності [24].

Основні види функцій втрати для різних типів задач:

1) Класифікація:
   - *Крос-ентропійна функція втрати (Cross-Entropy Loss):*
     $$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i), \quad (2.14)$$
     де y – фактичний розподіл класів, $\hat{y}$ – передбачений розподіл класів. Використовується для багатокласової класифікації
   - Бінарна крос-ентропійна функція втрати (Binary Cross-Entropy Loss):
     $$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})], \quad (2.15)$$
     де y – фактичний клас (0 або 1), $\hat{y}$ – передбачений ймовірнісний клас (муж 0 та 1). Використовується для бінарної класифікації.
2) Регресія:
   - Середньоквадратична функція втрати (Mean Squared Error - MSE):
     $$L(y, \hat{y}) = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2 , \quad (2.16)$$
     де y – фактичні значення, $\hat{y}$ – передбачені значення. Використовується для задач регресії.

2.5 Аналіз існуючих архітектур CNN для класифікацій зображень

Згорткові нейронні мережі мають різні архітектури, які ефективно застосовуються для обробки зображень. Ось кілька різних архітектур:

1) *LeNet-5:* Розроблена Яном ЛеКуном у 1990-х, ця архітектура була однією з перших CNN, яка застосовувалася для розпізнавання рукописних цифр.

2) *AlexNet:* Представлена у 2012 році, ця архітектура мала значний вплив на розвиток глибокого навчання. Вона має велику кількість шарів і вперше показала ефективність глибокого навчання на великих наборах даних, таких як ImageNet.

3) *VGG (Visual Geometry Group) Networks:* Вони мають дуже глибокі архітектури зі згортковими шарами, що складаються з невеликих фільтрів 3x3. Ця проста структура з декількома повторюваними шарами зробила їх популярними та ефективними.

4) *GoogLeNet / Inception:* Ця архітектура використовує модулі Inception, які виконують згортки з різними розмірами ядер одночасно і об'єднують їх в один шар. Це дозволяє ефективніше використовувати обчислювальні ресурси.

5) *ResNet (Residual Network):* Ця архітектура вирішує проблему зникновення градієнту в глибоких мережах за рахунок використання "residual connections" або "skip connections", що дозволяють передавати градієнти безпосередньо через деякі шари.

6) *MobileNet:* Це ефективна архітектура CNN, спеціально розроблена для мобільних пристроїв з обмеженими ресурсами. Вона використовує глибокі згорткові мережі, оптимізовані для ефективної роботи на пристроях з обмеженим обсягом пам'яті та потужністю обчислень.

7) *EfficientNet:* Це архітектура, яка використовує метод оптимізації для балансу між точністю моделі та ресурсами, необхідними для її тренування та запуску.

Це лише кілька прикладів з численних архітектур CNN, кожна з яких має свої переваги та застосування в залежності від завдання, обсягу даних та обчислювальних можливостей.

2.5.1 InceptionV3

InceptionV3 - це архітектура глибокого навчання, розроблена командою вчених компанії Google в рамках проекту Inception. Ця модель призначена для

завдань класифікації зображень і використовується для розпізнавання об'єктів на зображеннях.

Основна ідея InceptionV3 полягає в тому, щоб одночасно використовувати фільтри різних розмірів та об'єднувати їх результати, щоб модель могла вивчати ширший спектр шаблонів та абстракцій у зображеннях. Це дозволяє досягти високої точності класифікації для різних типів зображень [25].

InceptionV3 була натренована на великому обсязі даних, зокрема на наборі даних ImageNet, і є дуже потужною для різноманітних завдань в області комп'ютерного зору. Модель використовується як основа для передових досліджень та застосувань у сферах розпізнавання обличчя, медичного зображення, автономних транспортних засобів та інших областях машинного зору.

Архітектура мережі InceptionV3 будується поетапно, крок за кроком, як пояснюється нижче [25,26]:

1. *Факторизовані згортки:* використання факторизованих згорток допомагає знизити кількість параметрів в мережі, що сприяє більш ефективному використанню обчислювальних ресурсів та поліпшує продуктивність мережі.
2. *Менші згортки:* заміна більших фільтрів згорток меншими сприяє прискоренню навчання.

   Посередині на рис. 2.7 зображено згортку 3x3, а внизу повністю зв’язаний шар. Оскільки обидві згортки 3x3 можуть розподіляти ваги між собою, кількість обчислень можна зменшити.

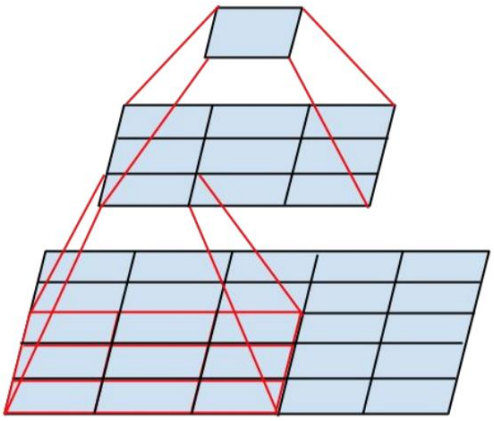

Рисунок 2.7 - Міні-мережа замінює згортки 3×3 на шарі згортки 3 × 1 з 3 вихідними одиницями у нижньому рівні.

3. *Асиметричні згортки:* згортку 3 × 3 можна замінити згорткою 1 × 3, а потім згорткою 3 × 1. Якщо згортку 3 × 3 замінити згорткою 2 × 2, кількість параметрів буде трохи більшою, ніж запропонована асиметрична згортка.

Рисунок 2.8 – a - оригінальний модуль Inception; b - початкові модулі з розширеним банком виходів фільтрів.

4. *Допоміжний класифікатор:* допоміжний класифікатор - це невелика CNN, яка вставляється між рівнями основної мережі під час тренування. Втрати, які виникають в результаті цього допоміжного класифікатора, додаються до загальних втрат мережі. В InceptionV3, допоміжний класифікатор виступає як регуляризатор, допомагаючи запобігти перенавчанню та підвищити загальну стійкість мережі [27].
5. *Зменшення розміру сітки:* зменшення розміру сітки зазвичай виконується шляхом об’єднання операцій. Однак для боротьби з вузькими місцями обчислювальних витрат пропонується ефективніша методика: початковий

модуль одночасно зменшує розмір сітки та розширює банки фільтрів. Це вигідне та вартісне рішення, яке дозволяє уникнути вузького місця у представленні, відповідно до принципу 1.

Усі вищезазначені концепції об’єднані в остаточну архітектуру.

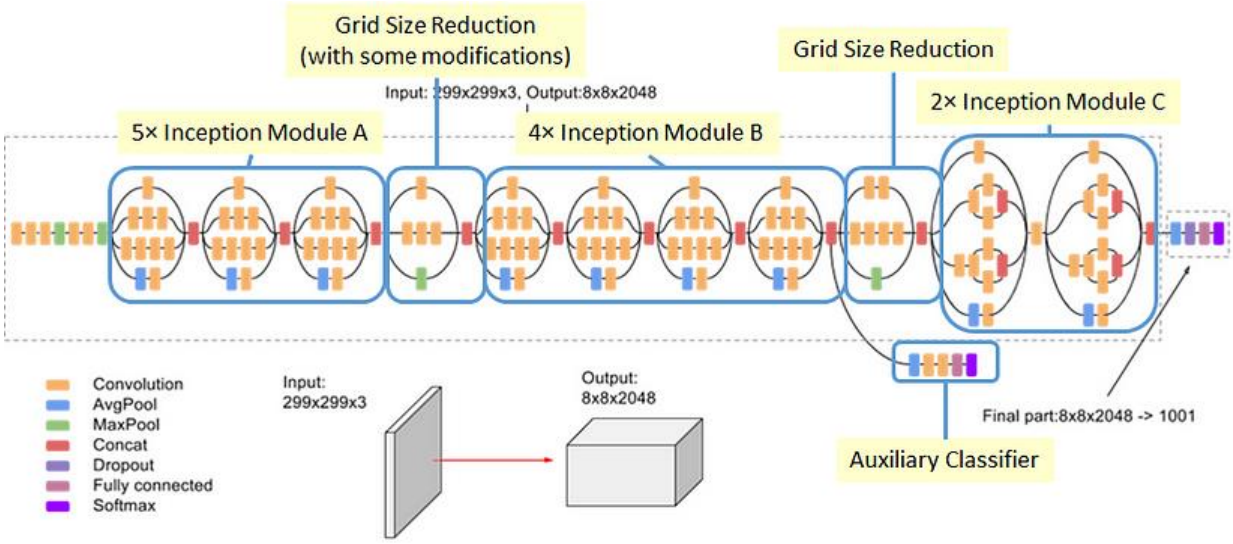

Рисунок 2.9 – Архітектура InceptionV3

2.5.2 RESNet50

ResNet-50 представляє собою важливий тип згорткової нейронної мережі (CNN), який відзначився в області глибокого навчання. Вперше представлений в 2015 році командою від Kaiming He та інших в Microsoft Research Asia, ResNet, що вказує на залишкову мережу, революціонізував підхід до конструювання глибоких нейронних мереж.

ResNet-50 базується на структурі глибокого залишкового навчання, яка дозволяє навчати дуже глибокі мережі із сотнями рівнів.

Архітектуру ResNet було розроблено у відповідь на дивовижне спостереження в дослідженнях глибокого навчання: додавання додаткових рівнів до нейронної мережі не завжди покращує результати.

Це було неочікувано, оскільки додавання рівня до мережі має дозволити йому дізнатися принаймні те, що навчилася попередня мережа, а також додаткову інформацію. Щоб вирішити цю проблему, команда ResNet на чолі з Кайміном Хе

розробила нову архітектуру, яка включає пропускні з’єднання . Ці з’єднання дозволили зберегти інформацію з попередніх рівнів, що допомогло мережі дізнатися краще представлення вхідних даних. Завдяки архітектурі ResNet вони змогли навчити мережі з аж 152 рівнями [28].

Результати ResNet були революційними: рівень помилок у наборі даних ImageNet становив 3,57%, а також посіли перше місце в кількох інших конкурсах, зокрема у змаганнях із виявлення об’єктів.

Це продемонструвало потужність і потенціал архітектури ResNet у дослідженнях і програмах глибокого навчання.

ResNet-50 складається з 50 шарів, які розділені на 5 блоків, кожен з яких містить набір залишкових блоків. Залишкові блоки дозволяють зберегти інформацію з попередніх рівнів, що допомагає мережі вивчати кращі представлення вхідних даних.

Основні компоненти архітектури ResNET:

1. Згорткові шари: Перший рівень мережі — це згортковий шар, який виконує згортку вхідного зображення. Далі слідує шар максимального об’єднання, який знижує дискретизацію результату згорткового шару. Вихід шару максимального об’єднання потім пропускається через серію залишкових блоків.

2. Залишкові блоки: Кожен залишковий блок складається з двох згорткових шарів, за кожним із яких іде шар пакетної нормалізації та функція активації випрямленої лінійної одиниці (ReLU). Потім вихідні дані другого згорткового рівня додаються до входу залишкового блоку, який потім пропускається через іншу функцію активації ReLU. Потім вихідні дані залишкового блоку передаються до наступного блоку.

3. Повністю підключений рівень: Останній рівень мережі — це повністю зв’язаний рівень, який приймає вихідні дані останнього залишкового блоку та відображає їх у вихідних класах. Кількість нейронів у повністю зв'язаному шарі дорівнює кількості вихідних класів.

З’єднання пропуску, також відомі як з’єднання ідентифікації, є ключовою функцією ResNet-50. Вони дозволяють зберігати інформацію з попередніх рівнів, що допомагає мережі вивчати кращі представлення вхідних даних.

З’єднання пропуску реалізуються шляхом додавання виводу попереднього шару до виводу пізнішого шару.

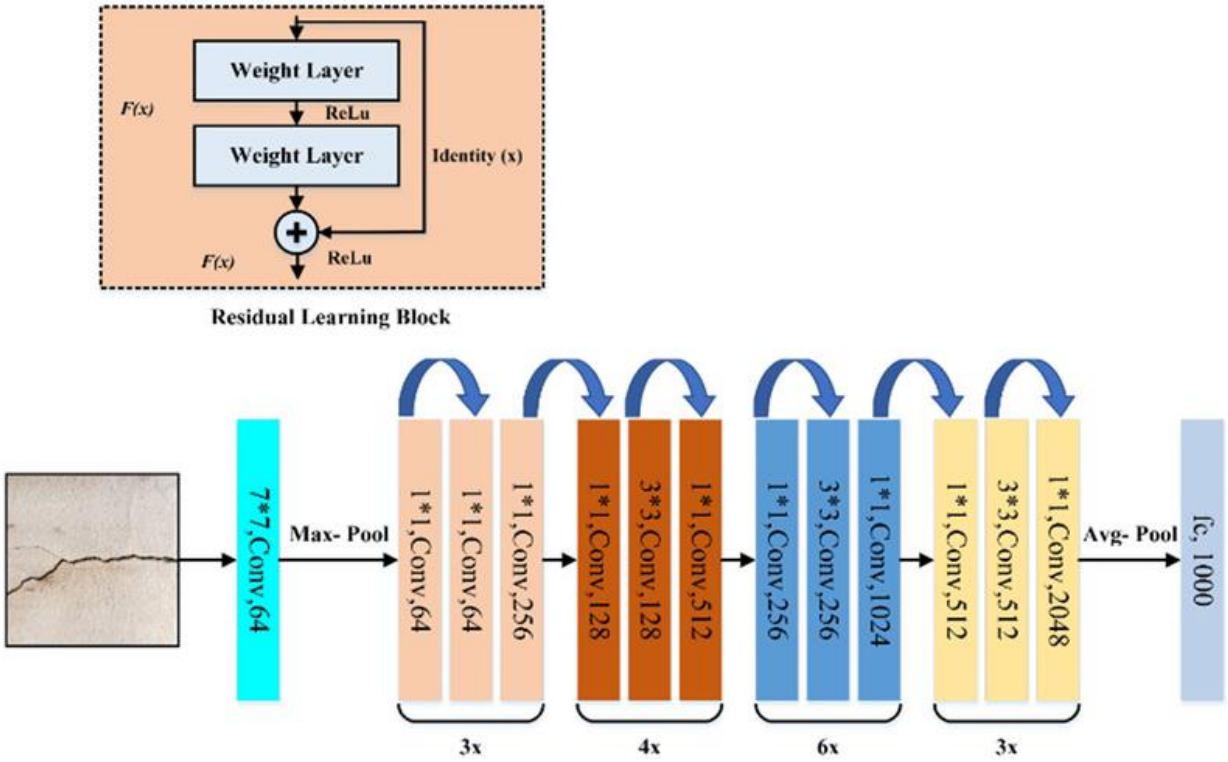

Рисунок 2.10 – Архітектура RESNet50

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОГО ЕКСПЕРИМЕНТУ

3.1 Опис набору даних

Головною метою даної роботи є розробка нейронної мережі для виявлення ракових пухлин на КТ знімках легень. КТ знімки забезпечують високу роздільну здатність, дозволяючи отримувати деталізовані зображення внутрішньої структури легень. Саме використання знімків КТ дозволяє розробляти нейронні мережі, які ефективно виявляють аномалії, характерні для цієї хвороби.

Для підготовки моделі дані було зібрано з різних джерел. В датасеті містяться зображення формату png, який відповідає вимогам моделі. Дані містять чотири категорії:

- Аденокарцинома T2, N0, M0, Ib: середній розміром пухлини, без віддалених метастаз, без ураження лімфатичних вузлів;
- Великоклітинний рак T2, N2, M0, IIIb: середній розмір пухлини, з ураженням лімфатичних вузлів, без метастаз;
- Плоскоклітинний рак T1, N2, M0, IIIa: малий розмір пухлини, з ураженням лімфатичних вузлів, без метастаз;
- легень без ракових пухлин.

Набір даних розділений на три основні категорії: навчальний (train), тестовий (test) та валідаційний (valid). Навчальний набір становить 70% від загального обсягу даних, тестовий - 20%, а валідаційний - 10%. Це дозволить ефективно навчати та перевірити точність моделі на різних наборах даних.

3.2 Розробка CNN моделі

На початковому етапі розробки системи була реалізована базова згорткова нейронна мережа (CNN) з шістьма шарами. Процес побудови моделі включав такі кроки:

1. Завантаження та обробка даних.
2. Визначення архітектури моделі:
    - Визначені функції активації ReLU та Softmax.

- Створено клас SimpleCNN з конструктором, що ініціалізує розміри вхідних даних, кількість класів, розміри фільтрів, кількість фільтрів, розміри пулінгу та повністю зв'язаних шарів.
- Архітектура мережі включала 2 згорткові, 2 пулінгові та 2 повністю зв'язані шари з функцією активації ReLU.

3. Тренування моделі:
   - Використано метод зворотнього поширення та градієнтного спуску для навчання моделі.
   - Ваги та зсуви оновлювалися за допомогою градієнтного спуску.
   - Модель тренувалася на тренувальних даних протягом певної кількості епох.
4. Побудова графіків втрат та точності:
   - Під час тренування виводилися метрики, такі як втрати та точність, для кожної епохи.
   - Побудовано графіки втрат та точності для оцінки ефективності моделі.

Модель була тренована на тренувальному та валідаційному наборах даних. Графік залежності функції втрат від номеру епохи представлений на рисунку 3.1. Для оцінки роботи моделі було обрано 10 епох. На рисунку 3.2 відображено графік, на якому видно зміну точності моделі відносно номеру епохи.

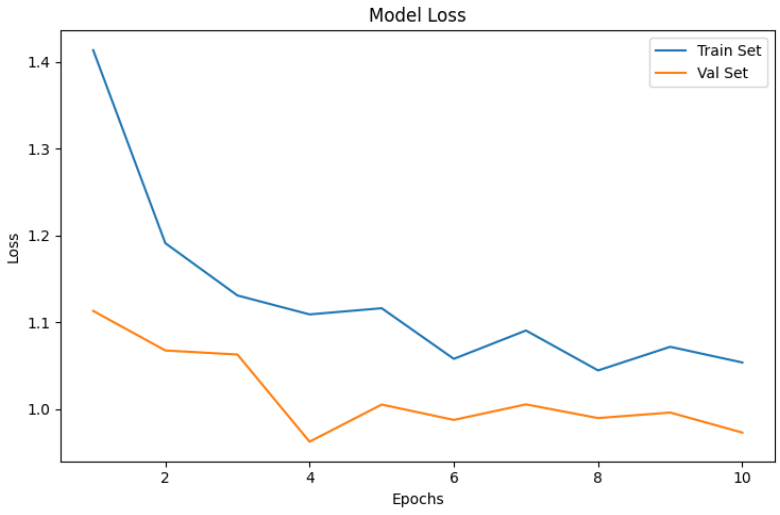

Рисунок 3.1 – Графік залежності функції втрат від номеру епохи

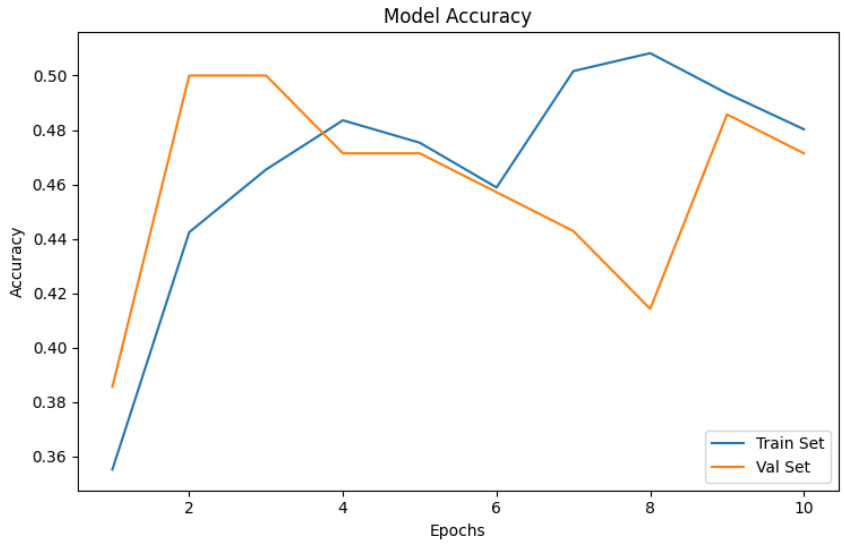

Рисунок 3.2 – Графік залежності точності від номеру епохи

Результати базової моделі виявилися не задовільними, з точністю всього 50%. У зв'язку з цим було прийнято рішення покращити модель.

3.3 Покращення CNN моделі

У вдосконаленій моделі CNN було внесено кілька змін, таких як використання трьох згорткових шарів із різною кількістю фільтрів (64, 128, 256), три шари пулінгу, і два повністю зв'язаних шари . В якості функції втрат обрано категоріальну крос-ентропію, а для оптимізації ваг та зсувів моделі використано алгоритм Adam.

Тренування моделі відбувалося протягом 40 епох на тренувальному та валідаційному наборах даних. Результати покращеної моделі показали схильність до перенавчання, що може бути пов'язано з обмеженою кількістю даних у валідаційному наборі. Точність на тренувальному наборі становила 89%, а значення функції втрат склало 1.1.

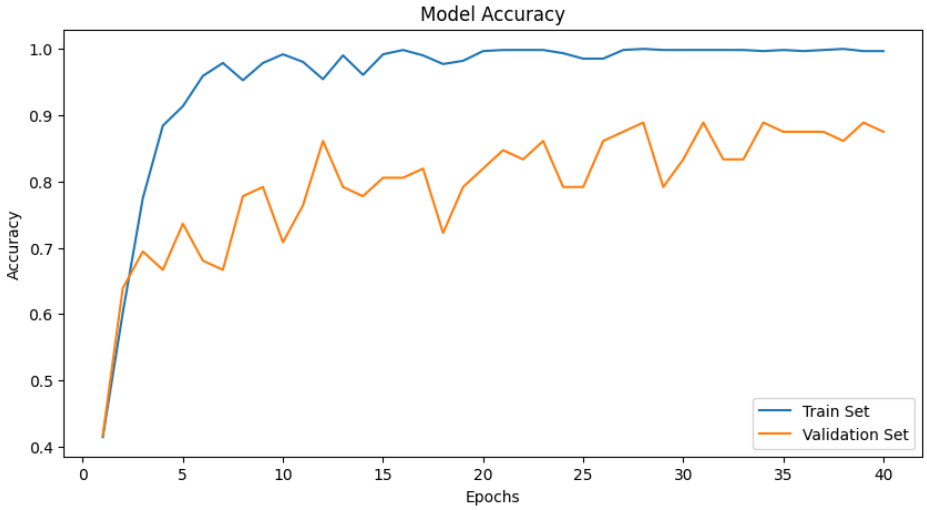

Рисунок 3.3 – Графік залежності точності моделі від номеру епохи

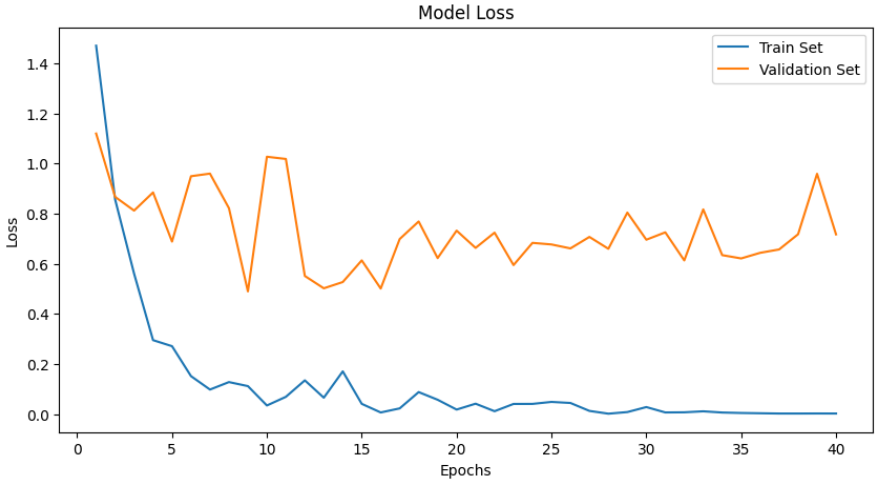

Рисунок 3.4 – Графік залежності функції втрати від номеру епохи

Після тренування моделі на тренувальному та валідаційному наборах, було проведено оцінку її ефективності на тестовому наборі даних. Прогнозування моделі на тестовому наборі дозволяє визначити, наскільки добре вона взагалі узагальнює свої навички на нових, раніше не бачених даних.

Аналізуючи результати прогнозування на тестовому наборі (рис.3.5) для різних типів пухлин, можна зробити наступні спостереження:

1) Точність прогнозування для аденокарциноми становить 88,21%. З врахуванням середнього розміру пухлини та відсутності ураження лімфатичних вузлів, модель до певної міри успішно визначає даний тип пухлини.

2) Результати прогнозування для великоклітинного раку складають 92,10%. Враховуючи середній розмір пухлини та ураження лімфатичних вузлів, модель успішно визначає цей тип пухлини. Також, стадія IIIa вказує на більш розповсюджену хворобу, і модель ефективно розпізнає цю характеристику.
3) Для нормальних зображень точність прогнозування становить 100%. Це логічно, оскільки в цьому випадку не виявлено патології.
4) Прогноз для плоскоклітинного раку складає 68,63%. Модель розпізнає невеликий розмір пухлини та ураження лімфатичних вузлів, але точність є меншою порівняно з іншими типами.

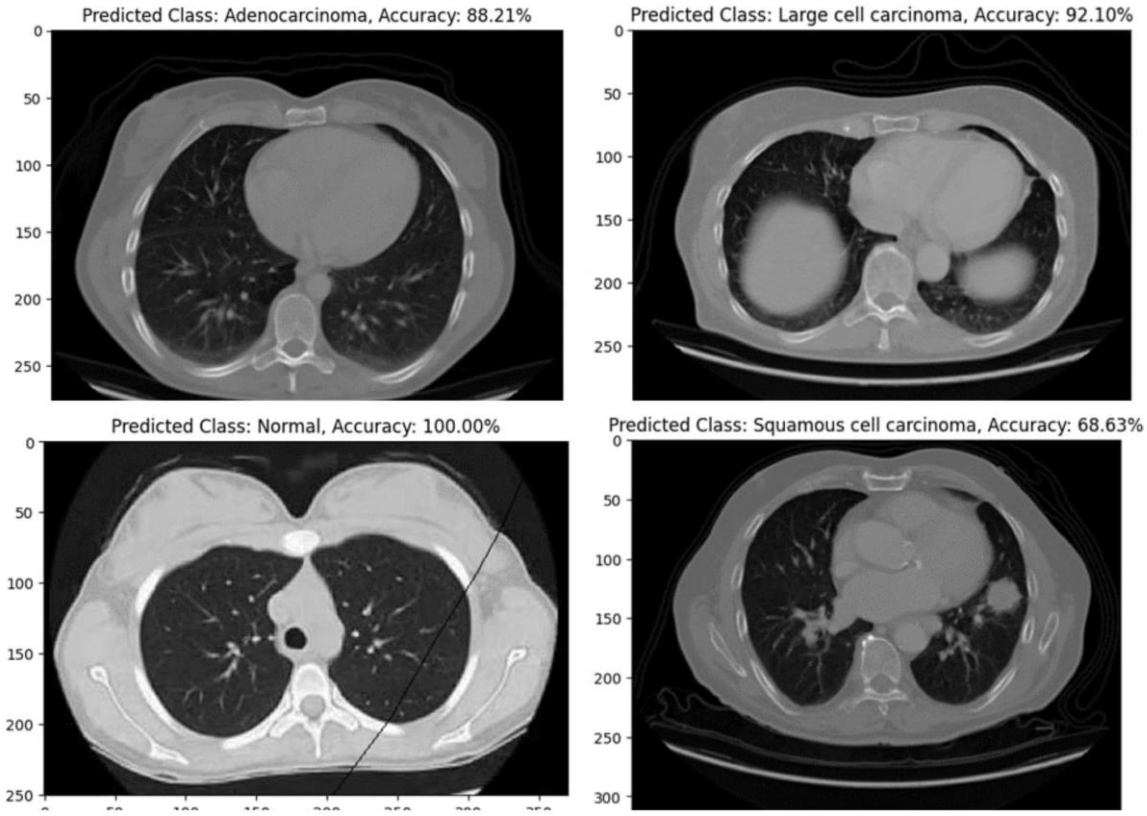

Рисунок 3.5 – Результат прогнозування CNN

3.4 Порівняння з існуючими моделями

Порівняння архітектур, таких як InceptionV3 і ResNet50, з власною моделлю допоже визначити, яка є найбільш ефективною для прогнозування ракових пухлин на КТ знімках легень.

Результати навчання моделі InceptionV3 на тренувальній та валідаційній вибірках представлені на рис. 3.6, 3.7. Незважаючи на більшу складність архітектури цієї моделі порівняно з попередньою CNN із 8 шарами, InceptionV3 демонструє високу узагальнюючу здатність, показуючи значно кращі результати та не схильна до перенавчання, з максимальною точністю 92% та значенням функції втрат 0.8.

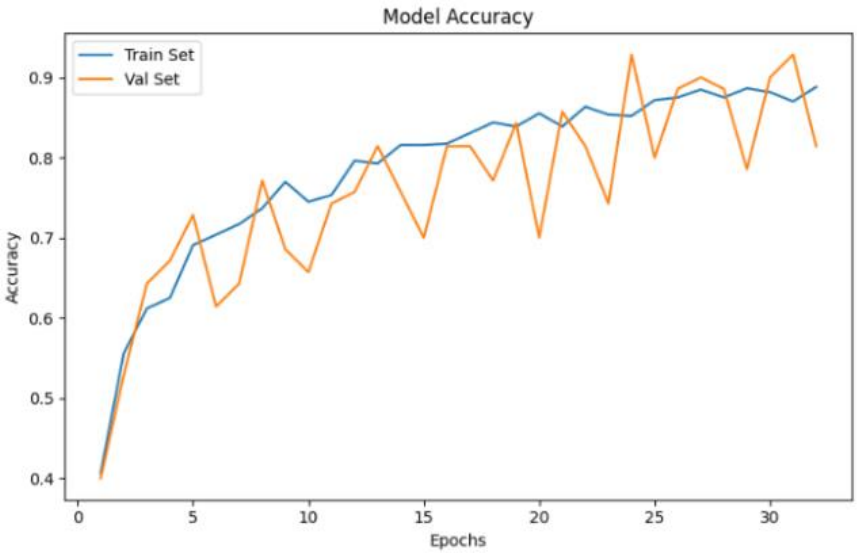

Рисунок 3.6 – Залежність точності моделі від номеру епохи

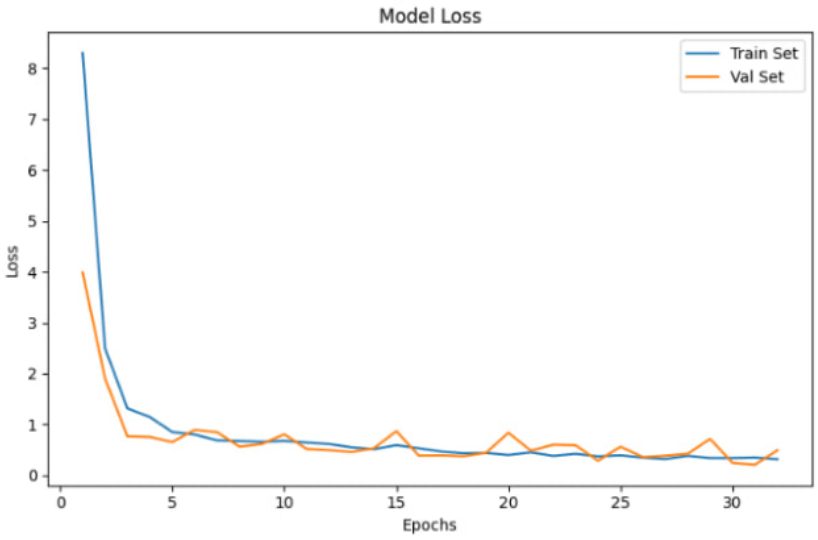

Рисунок 3.7 – Залежність функції втрат від номеру епохи

Результати тренування моделі ResNet-50 на тренувальній та валідаційній вибірці представлені на рис. 3.8, 3.9. Варто відзначити, що ця модель, починаючи

з 5 епохи, виявляє ознаки перенавчання, але досягає точності на рівні 96%, зі значенням функції втрат 0.75.

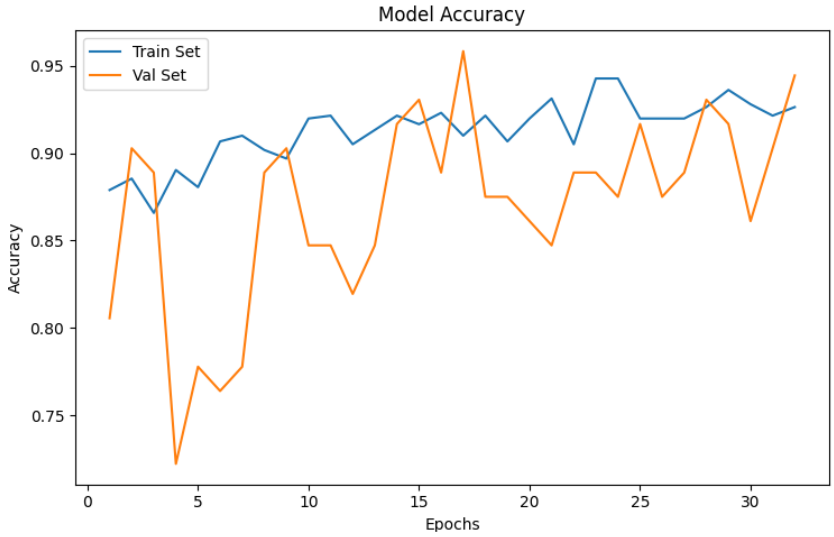

Рисунок 3.8 – Залежність точності моделі від номеру епохи

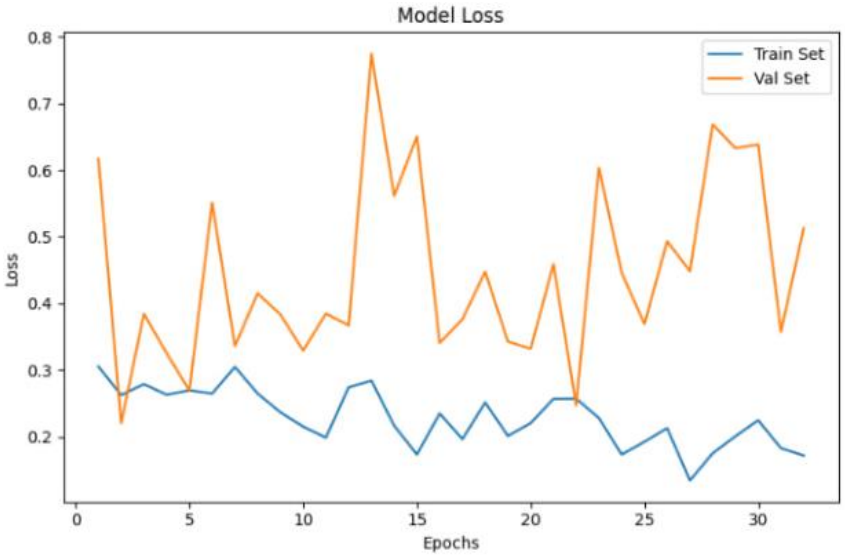

Рисунок 3.9 – Залежність функції втрат від номеру епохи

На тестовому наборі модель ResNet-50 показала точність прогнозування, зазначену для різних типів раку легень:

- Аденокарцинома: 99,92%

- Великоклітинний рак: 100%

- Здорові легені (Normal): 99,98%

- Плоскоклітинний рак: 100%

Ці відсотки вказують на те, як добре модель здатна класифікувати тестові зображення для кожної з розглянутих категорій.

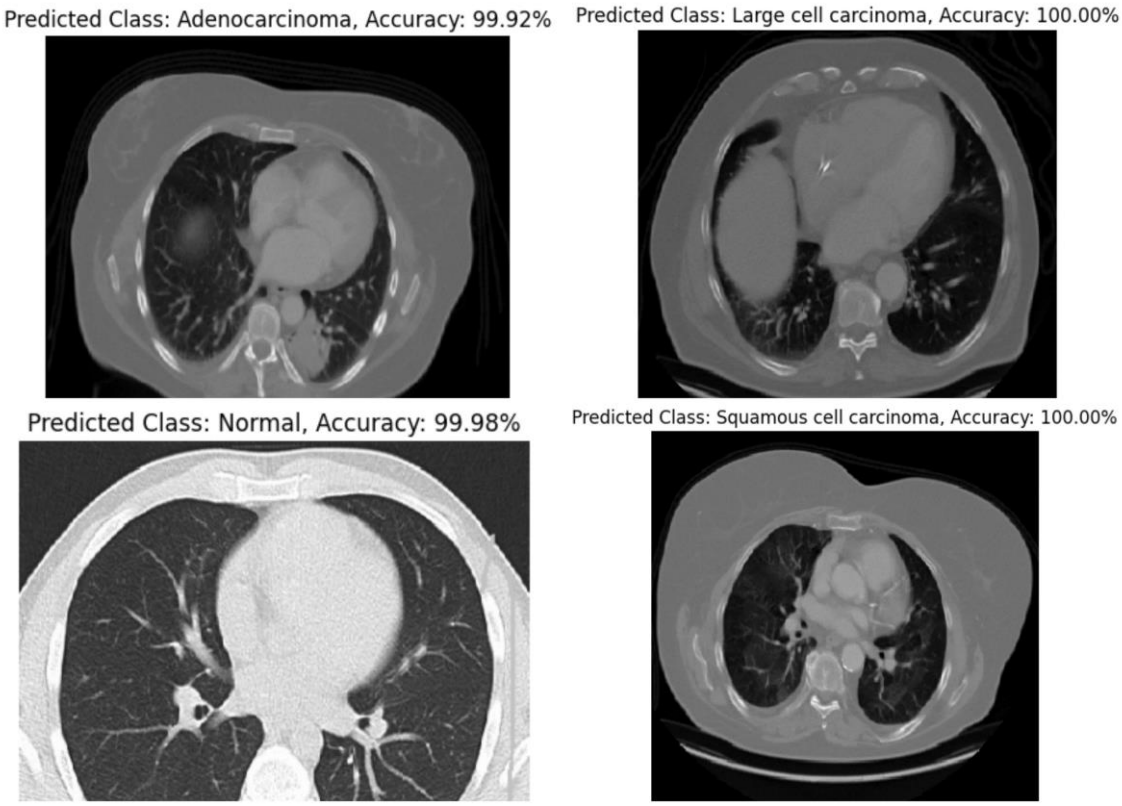

Рисунок 3.10 – Результат прогнозування ResNet50

Модель ResNet50 виявилася найефективнішою моделлю для поставленої задачі з високою точністю та придатністю для медичного використання.

ВИСНОВКИ

У ході дослідження було здійснено аналіз та порівняння різних моделей нейронних мереж для виявлення ракових пухлин на зображеннях легень. Починаючи з базової CNN, яка виявилася менш ефективною, експериментували з більш складними архітектурами, такими як InceptionV3 та ResNet-50.

Проаналізувавши результати власно створеної CNN, було визначено:

1. Модель успішно розпізнає аденокарциному, зосереджуючись на середньому розмірі пухлини та відсутності ураження лімфатичних вузлів. Точність становить 88,21%.
2. Модель ефективно розпізнає великоклітинний рак, враховуючи середній розмір пухлини та ураження лімфатичних вузлів, включаючи стадію IIIa. Точність цього типу раку становить 92,10%.
3. Модель ідеально розпізнає відсутність патології на зображеннях легень. Точність для нормальних зображень складає 100%.
4. Точність для плоскоклітинного раку становить 68,63%, що може бути пов'язано з меншим розміром пухлини та ураженням лімфатичних вузлів.

Порівняно із існуючими моделями InceptionV3 та ResNet-50, власно створена CNN показала меншу точність, але важливо враховувати обмежені ресурси для навчання.

Усе ж, ResNet-50 виявилася найефективнішою, надійною та високоточною моделлю для виявлення ракових пухлин на медичних зображеннях легень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Murray & Nadel's textbook of respiratory medicine. / ed. by M. J. F. 1927, M. R. J. 5th ed. Philadelphia : Elsevier Saunders, 2010.

2) Froesch P, Mark M, Rothschild SI, et al. Binimetinib, pemetrexed and cisplatin, followed by maintenance of binimetinib and pemetrexed in patients with advanced non-small cell lung cancer (NSCLC) and KRAS mutations. The phase 1B SAKK 19/16 trial. *Lung Cancer*. 2021;156:91-99.

3) Non-Small Cell Lung Cancer Treatment. National Cancer Institute. URL: https://www.cancer.gov/types/lung/patient/non-small-cell-lung-treatment-pdq (date of access: 05.11.2023).

4) Small Cell Lung Cancer Treatment. National Cancer Institute. URL: https://www.cancer.gov/types/lung/patient/small-cell-lung-treatment-pdq (date of access: 06.11.2023).

5) Lee A. J. X., Lee S. M. Small Cell Lung Cancer. Reference Module in Biomedical Sciences. 2020. URL: https://doi.org/10.1016/b978-0-08-102723-3.00035-4 (date of access: 06.11.2023).

6) Early Lung Cancer Action Project: overall design and findings from baseline screening / C. I. Henschke et al. The Lancet. 1999. Vol. 354, no. 9173. P. 99–105. URL: https://doi.org/10.1016/s0140-6736(99)06093-6 (date of access: 06.11.2023).

7) Cancer Imaging Program (CIP). Cancer Imaging Program (CIP). URL: https://imaging.cancer.gov/imaging_basics/cancer_imaging/uses_of_imaging.htm (date of access: 07.11.2023).

8) Lung Cancer Detection using CT Scan Images / S. Makaju et al. Procedia Computer Science. 2018. Vol. 125. P. 107–114. URL: https://doi.org/10.1016/j.procs.2017.12.016 (date of access: 07.11.2023).

9) Analysis on Early Detection of Lung Cancer by PET/CT Scan / H.-Q. Wang et al. Asian Pacific Journal of Cancer Prevention. 2015. Vol. 16, no. 6. P. 2215–

2217. URL: https://doi.org/10.7314/apjcp.2015.16.6.2215 (date of access: 07. 11.2023).

10) Sidey-Gibbons J. A. M., Sidey-Gibbons C. J. Machine learning in medicine: a practical introduction. BMC Medical Research Methodology. 2019. Vol. 19, no. 1. URL: https://doi.org/10.1186/s12874-019-0681-4 (date of access: 07.11.2023).

11) VARIN A. Introduction to Neural Network in Machine Learning. Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/ (date of access: 09.11.2023).

12) Awad M., Khanna R. Deep Neural Networks. Efficient Learning Machines. Berkeley, CA, 2015. P. 127–147. URL: https://doi.org/10.1007/978-1-4302-5990-9_7 (date of access: 10.11.2023).

13) Artificial neural networks in medical diagnosis / F. Amato et al. Journal of Applied Biomedicine. 2013. Vol. 11, no. 2. P. 47–58. URL: https://doi.org/10.2478/v10136-012-0031-x (date of access: 10.11.2023).

14) Pasini A. Artificial neural networks for small dataset analysis. J Thorac Dis. 2015 May;7(5):953-60.

15) Artificial Neural Network: Understanding the Basic Concepts without Mathematics / S.-H. Han et al. Dementia and Neurocognitive Disorders. 2018. Vol. 17, no. 3. P. 83. URL: https://doi.org/10.12779/dnd.2018.17.3.83 (date of access: 10.11.2023).

16) Baheti P. Activation Functions in Neural Networks [12 Types & Use Cases]. V7 | The AI Data Engine for Computer Vision & Generative AI. URL: https://www.v7labs.com/blog/neural-networks-activation-functions (date of access: 10.11.2023).

17) Tsantekidis A., Passalis N., Tefas A. Recurrent neural networks. Deep Learning for Robot Perception and Cognition. 2022. P. 101–115. URL: https://doi.org/10.1016/b978-0-32-385787-1.00010-5 (date of access: 11.11.2023).

18) Beysolow II T. Recurrent Neural Networks (RNNs). Introduction to Deep Learning Using R. Berkeley, CA, 2017. P. 113–124. URL: https://doi.org/10.1007/978-1-4842-2734-3_6 (date of access: 11.11.2023).

19) Recent Advances in Deep Learning Techniques for Face Recognition / M. T. H. Fuad et al. IEEE Access. 2021. Vol. 9. P. 99112–99142. URL: https://doi.org/10.1109/access.2021.3096136 (date of access: 11.11.2023).

20) C-CNN: Contourlet Convolutional Neural Networks / M. Liu et al. IEEE Transactions on Neural Networks and Learning Systems. 2020. P. 1–14. URL: https://doi.org/10.1109/tnnls.2020.3007412 (date of access: 12.11.2023).

21) V. H. Phung, E. J. Rhee et al., “A high-accuracy model average ensembleof convolutional neural networks for classification of cloud image patcheson small datasets,” Applied Sciences, vol. 9, no. 21, p. 4500, 2019

22) (PDF) Recent Advances in Deep Learning Techniques for Face Recognition. Available from: https://www.researchgate.net/publication/350326846_Recent_Advances_in_Deep_Learning_Techniques_for_Face_Recognition [accessed Dec 01 2023].

23) Milosevic N. Introduction to Convolutional Neural Networks. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-5648-0 (date of access: 12.11.2023).

24) Understanding Loss Function in Deep Learning. Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/?utm_source=reading_list&utm_medium=https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/ (date of access: 13.11.2023).

25) Rethinking the Inception Architecture for Computer Vision / C. Szegedy et al. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),

Las Vegas, NV, USA, 27–30 June 2016. 2016. URL: https://doi.org/10.1109/cvpr.2016.308 (date of access: 13.11.2023).

26) Overview of Models. Site not found · GitHub Pages. URL: https://jabocken.github.io/ML2017Fall/docs/models/ (date of access: 13.11.2023).

27) S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of The 32nd International Conference on Machine Learning, pages 448–456, 2015.

28) Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures / L. Ali et al. Sensors. 2021. Vol. 21, no. 5. P. 1688. URL: https://doi.org/10.3390/s21051688 (date of access: 14.11.2023).

ДОДАТОК А

Розробка базової моделі CNN

```
import numpy as np
from scipy.signal import convolve2d
from scipy.ndimage import maximum_filter
from PIL import Image
import matplotlib.pyplot as plt
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from torchvision import transforms
from torch.nn.functional import conv2d
import torch
import torch.nn.functional as F

# Функція активації ReLU
def relu(x):
    return np.maximum(0, x)

# Функція активації Softmax
def softmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / np.sum(exp_x, axis=0)

# Повністю зв'язаний шар (Fully Connected Layer)
def fully_connected(x, weights, bias):
    return np.dot(x, weights) + bias

# Функція для "згладжування" даних
def flatten(x):
    return x.flatten()

# Операція максимального пулінгу
def max_pooling(x, pool_size):
    return maximum_filter(x, size=pool_size)

# Функція втрати для категоріальної крос-ентропії
def categorical_cross_entropy_loss(output, target):
    epsilon = 1e-15
    output = np.clip(output, epsilon, 1 - epsilon)
    return -np.sum(target * np.log(output))

# Функція обчислення точності моделі
def accuracy(output, target):
    predictions = np.argmax(output, axis=1)
    true_labels = np.argmax(target, axis=1)
    correct_predictions = np.sum(predictions == true_labels)
    total_samples = len(target)
    return correct_predictions / total_samples

# Клас простої згорткової нейронної мережі (CNN)
class SimpleCNN:
```

```
    # Конструктор класу
    # Ініціалізація розмірів вхідних даних, кількості класів, розмірів фільтрів,
     # кількості фільтрів, розмірів пулінгу та розмірів повністю зв'язаних шарів.
    def __init__(self, input_shape, num_classes, filter_sizes, num_filters, pool_size, fc_size):
        self.input_shape = input_shape      # Розмір вхідних даних
        self.num_classes = num_classes      # Кількість класів
        self.filter_sizes = filter_sizes    # Розміри фільтрів для кожного шару згортки
        self.num_filters = num_filters      # Кількість фільтрів для кожного шару згортки
        self.pool_size = pool_size          # Розмір пулінгу
        self.fc_size = fc_size              # Розмір повністю зв'язаного шару

         # Ініціалізація ваг та зсувів для кожного шару згортки
        self.weights = [np.random.randn(filter_sizes[i], filter_sizes[i], input_shape[2], num_filters[i])
for i in range(len(num_filters))]
        self.biases = [np.zeros((1, 1, num_filters[i])) for i in range(len(num_filters))]

        # Визначення розмірів для повністю зв'язаного шару
        fc_input_size = (input_shape[0] // (2 ** len(num_filters))) * (input_shape[1] // (2 **
len(num_filters))) * num_filters[-1]

        # Ініціалізація ваг та зсувів для повністю зв'язаного шару
        self.fc_weights = np.random.randn(fc_input_size, fc_size)
        self.fc_bias = np.zeros((1, fc_size))

        # Ініціалізація ваг та зсувів для вихідного шару
        self.output_weights = np.random.randn(fc_size, num_classes)
        self.output_bias = np.zeros((1, num_classes))

    # Метод прямого поширення (forward pass)
    def forward_pass(self, input_data):
        # Перетворюємо тензор PyTorch у NumPy-масив, якщо потрібно
        if isinstance(input_data, torch.Tensor):
            input_data = input_data.numpy()

        for i in range(len(self.num_filters)):
            # Розширення розмірності вхідних даних та параметрів для шару згортки
            input_data_reshaped = np.expand_dims(input_data, axis=0)
            weights_reshaped = np.expand_dims(self.weights[i], axis=0)
            biases_reshaped = np.expand_dims(self.biases[i], axis=0)

            # Застосування операцій згортки та активації ReLU
            input_data = relu(conv2d(input_data_reshaped.transpose(0, 3, 1, 2),
                                     weights_reshaped.transpose(3, 2, 0, 1),
                                     bias=biases_reshaped.transpose(0, 3, 1, 2),
                                     padding=(1, 1)).transpose(0, 2, 3, 1))
            # Застосування операції максимального пулінгу
            input_data = max_pooling(input_data, pool_size=(self.pool_size, self.pool_size))

        # Згладжування даних перед передачею у повністю зв'язаний шар
        input_data = flatten(input_data)
        # Застосування активації ReLU для повністю зв'язаного шару
        input_data = relu(fully_connected(input_data, self.fc_weights, self.fc_bias))
```

```
    # Застосування Softmax для отримання вихідних ймовірностей
    output = softmax(fully_connected(input_data, self.output_weights, self.output_bias))
    return output

# Метод зворотнього поширення (backward pass)
 # Обчислення градієнтів для оновлення ваг та зсувів (biases)
def backward_pass(self, output, target):
    # Розрахунок градієнтів для вихідного шару
    output_grad = output - target
    self.output_weight_grad = np.dot(self.fc_input_activation.T, output_grad)
    self.output_bias_grad = np.sum(output_grad, axis=0, keepdims=True)

    # Розрахунок градієнтів для повністю зв'язаного шару
    fc_input_grad = np.dot(output_grad, self.output_weights.T)
    fc_input_grad = fc_input_grad * (self.fc_input_activation > 0)
    self.fc_weight_grad = np.dot(self.input_flatten.T, fc_input_grad)
    self.fc_bias_grad = np.sum(fc_input_grad, axis=0, keepdims=True)

     # Розрахунок градієнтів для шарів згортки та пулінгу
    conv_input_grad = np.dot(fc_input_grad, self.fc_weights.T)
    conv_input_grad = conv_input_grad.reshape(self.conv_output_shape)
    conv_input_grad = conv_input_grad * (self.conv_input_activation > 0)

    pool_input_grad = np.zeros_like(self.conv_input_activation)

    # Розрахунок градієнтів для шарів пулінгу
    for i in range(self.num_filters[-1]):
        for j in range(self.pool_output_shape[0]):
            for k in range(self.pool_output_shape[1]):
                pool_input_grad[j * self.pool_size : (j + 1) * self.pool_size,
                                k * self.pool_size : (k + 1) * self.pool_size, i] = np.kron(
                    conv_input_grad[j, k, i],
                    (self.pool_input_activation[j * self.pool_size : (j + 1) * self.pool_size,
                                                k * self.pool_size : (k + 1) * self.pool_size, i] == np.max(
                        self.pool_input_activation[j * self.pool_size : (j + 1) * self.pool_size,
                                                   k * self.pool_size : (k + 1) * self.pool_size, i])
                    )
                )

    # Розрахунок градієнтів для шарів згортки після операції пулінгу
    conv_input_grad = np.zeros_like(self.pool_input_activation)

    for i in range(self.num_filters[-1]):
        conv_input_grad[:, :, i] = np.sum(pool_input_grad[:, :, i][:, :, np.newaxis, np.newaxis] *
                                          self.pool_input_activation[:, :, i][:, :, np.newaxis, np.newaxis],
                                          axis=(0, 1))

    return conv_input_grad

# Побудова графіків втрат та точності
def plot_metrics(self, train_losses, val_losses, train_accuracies, val_accuracies):
    plt.figure(figsize=(12, 4))
```

```
    # Графік втрат на тренувальному та валідаційному наборі даних
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Set')
    if val_losses:
        plt.plot(val_losses, label='Val Set')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # Графік точності на тренувальному та валідаційному наборі даних
    plt.subplot(1, 2, 2)
    plt.plot(train_accuracies, label='Train Set')
    if val_accuracies:
        plt.plot(val_accuracies, label='Val Set')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.show()

# Тренування моделі на тренувальних даних
def train(self, train_data, epochs, learning_rate=0.001, val_data=None)
    # Ініціалізація списків для зберігання втрат та точності
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(epochs):
        # Ініціалізація показників втрат та точності для кожної епохи
        total_train_loss = 0
        total_train_accuracy = 0
        num_train_samples = len(train_loader.dataset)

        for batch in train_loader:
            image_data, label = batch

            # Пряме поширення (forward pass)
            output = self.forward_pass(image_data)
            # Обчислення втрат та градієнтів за допомогою методу зворотнього поширення
            loss = categorical_cross_entropy_loss(output, label.numpy())
            gradients = self.backward_pass(output, label.numpy())

            # Оновлення ваг та зсувів (biases) за допомогою градієнтного спуску
            self.update_weights(gradients, learning_rate)

            # Акумуляція втрат та точності
            total_train_loss += loss
            total_train_accuracy += accuracy(output, label.numpy())
```

```
            # Середні значення втрат та точності для тренувального набору даних
            avg_train_loss = total_train_loss / num_train_samples
            avg_train_accuracy = total_train_accuracy / num_train_samples

            # Збереження метрик для подальшого відображенн
            train_losses.append(avg_train_loss)
            train_accuracies.append(avg_train_accuracy)

            # Обчислення метрик для валідаційного набору даних, якщо він вказаний
            if val_loader:
                total_val_loss = 0
                total_val_accuracy = 0
                num_val_samples = len(val_loader.dataset)

                for val_batch in val_loader:
                    val_image_data, val_label = val_batch
                    val_output = self.forward_pass(val_image_data.numpy())

                    val_loss = categorical_cross_entropy_loss(val_output, val_label.numpy())
                    total_val_loss += val_loss
                    total_val_accuracy += accuracy(val_output, val_label.numpy())

                avg_val_loss = total_val_loss / num_val_samples
                avg_val_accuracy = total_val_accuracy / num_val_samples

                # Виведення інформації про поточну епоху
                print(f'Epoch {epoch + 1}, Train Loss: {avg_train_loss}, Train Accuracy:
{avg_train_accuracy}, '
                      f'Validation Loss: {avg_val_loss}, Validation Accuracy: {avg_val_accuracy}')
            else:
                # Виведення інформації про поточну епоху (без валідації)
                print(f'Epoch {epoch + 1}, Train Loss: {avg_train_loss}, Train Accuracy:
{avg_train_accuracy}')

        # Відображення графіків метрик
        self.plot_metrics(train_losses, val_losses, train_accuracies, val_accuracies)
    # Метод оновлення ваг та зсувів (biases) за допомогою градієнтного спуску
    def update_weights(self, gradients, learning_rate):
        # Оновлення ваг та зсувів для кожного шару згортки
        for i in range(len(self.num_filters)):
            self.weights[i] -= learning_rate * gradients[i]['weights']
            self.biases[i] -= learning_rate * gradients[i]['biases']
        # Оновлення ваг та зсуву для повністю зв'язаного шару
        self.fc_weights -= learning_rate * gradients[-2]
        self.fc_bias -= learning_rate * gradients[-1]
        # Оновлення ваг та зсуву для вихідного шару
        self.output_weights -= learning_rate * gradients[-3]
        self.output_bias -= learning_rate * gradients[-4]

    # Метод оцінки моделі на тестових даних
    def evaluate(self, test_images, test_labels):
```

```
        # Ініціалізація показників втрат та точності
        total_loss = 0
        total_accuracy = 0
        num_samples = len(test_images)
        for i in range(num_samples):
            # Отримання вхідних даних та відповіді для кожного зображення
            image_data = test_images[i]
            label = np.eye(self.num_classes)[test_labels[i]]

            # Пряме поширення для отримання вихідних ймовірностей
            output = self.forward_pass(image_data)

            # Підрахунок втрат та точності для кожного зображення
            total_loss += categorical_cross_entropy_loss(output, label)
            total_accuracy += accuracy(output, label)

        # Середні значення втрат та точності для тестового набору даних
        avg_loss = total_loss / num_samples
        avg_accuracy = total_accuracy / num_samples

        return avg_loss, avg_accuracy
# Завантаження та підготовка даних
# Визначення трансформацій для обробки та підготовки зображень
transform = transforms.Compose([
    transforms.Resize((350, 350)),      # Зміна розмірів зображення
    transforms.ToTensor(),              # Перетворення зображення у тензор
])

# Завантаження тренувального набору даних та створення DataLoader
train_set = ImageFolder(train_path, transform=transform)
train_loader = DataLoader(train_set, batch_size=5, shuffle=True)

# Завантаження валідаційного набору даних та створення DataLoader
val_set = ImageFolder(val_path, transform=transform)
val_loader = DataLoader(val_set, batch_size=5, shuffle=True)

# Завантаження тестового набору даних та створення DataLoader
test_set = ImageFolder(test_path, transform=transform)
test_loader = DataLoader(test_set, batch_size=5, shuffle=True)
# Визначення параметрів моделі
input_shape = (350, 350, 3)   # Розмір вхідних зображень: 350x350 пікселів, 3 канали кольору
(RGB)
num_classes = 4             # Кількість класів у задачі класифікації
filter_sizes = [3, 3]      # Розміри фільтрів для кожного згорткового шару
num_filters = [32, 64]   # Кількість фільтрів для кожного згорткового шару
pool_size = 2               # Розмір пулінгу (пулінг-шару)
fc_size = 128               # Розмір повністю зв'язаного шару

# Ініціалізація та тренування моделі
model = SimpleCNN(input_shape, num_classes, filter_sizes, num_filters, pool_size, fc_size)
# Тренування моделі на тренувальних даних з використанням DataLoader
model.train(train_loader, epochs=10, learning_rate=0.001)
```

ДОДАТОК Б

Покращення моделі CNN

```
# Функція для завантаження та обробки зображень
def load_images_labels(data_path):
    images = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(data_path):
        label_path = os.path.join(data_path, label)
        label_id = label_encoder.fit_transform([label])[0]
        for image_file in os.listdir(label_path):
            image_path = os.path.join(label_path, image_file)
            image = cv2.imread(image_path)
            image = cv2.resize(image, (350, 350))
            images.append(image)
            labels.append(label_id)
    return np.array(images), np.array(labels)

# Клас для простої CNN-моделі
class SimpleCNN:
    def __init__(self):
        self.conv1 = ConvLayer(3, 64, kernel_size=3, stride=1, padding=1)
        self.pool = MaxPool2D(kernel_size=2, stride=2, padding=0)
        self.conv2 = ConvLayer(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = ConvLayer(128, 256, kernel_size=3, stride=1, padding=1)
        self.fc1 = FullyConnected(256 * 43 * 43, 128)
        self.fc2 = FullyConnected(128, 4)
        self.relu = ReLU()
        self.dropout = Dropout(0.2)

    def forward_pass(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.relu(self.conv3(x))
        x = self.pool(x)
        x = x.reshape(x.shape[0], -1)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

    def get_params(self):
        return [self.conv1.weights, self.conv1.bias,
                self.conv2.weights, self.conv2.bias,
                self.conv3.weights, self.conv3.bias,
                self.fc1.weights, self.fc1.bias,
                self.fc2.weights, self.fc2.bias]

# Класи для розрахунку операцій CNN
```

```
class ConvLayer:
    def __init__(self, in_channels, out_channels, kernel_size, stride, padding):
        self.weights = torch.tensor(np.random.randn(out_channels, in_channels, kernel_size,
kernel_size), requires_grad=True)
        self.bias = torch.tensor(np.zeros((out_channels, 1)), requires_grad=True)
        self.stride = stride
        self.padding = padding

    def __call__(self, x):
        # Convolutional layer application
        x_padded = np.pad(x, ((0, 0), (0, 0), (self.padding, self.padding), (self.padding, self.padding)))
        output_height = (x.shape[2] - len(self.weights[0][0]) + 2 * self.padding) // self.stride + 1
        output_width = (x.shape[3] - len(self.weights[0][0]) + 2 * self.padding) // self.stride + 1
        output = np.zeros((x.shape[0], len(self.weights), output_height, output_width))

        for i in range(output_height):
            for j in range(output_width):
                x_slice = x_padded[:, :, i * self.stride: i * self.stride + len(self.weights[0][0]),
                                   j * self.stride: j * self.stride + len(self.weights[0][0])]
                for k in range(len(self.weights)):
                    output[:, k, i, j] = np.sum(x_slice * self.weights[k], axis=(1, 2, 3)) + self.bias[k, 0]
        # Debugging print statements
        print("x_slice shape:", x_slice.shape)
        print("self.weights shape:", self.weights.shape)

        return output

class MaxPool2D:
    def __init__(self, kernel_size, stride, padding):
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def __call__(self, x):
        # Застосування пулінгу
        output_height = (x.shape[2] - self.kernel_size) // self.stride + 1
        output_width = (x.shape[3] - self.kernel_size) // self.stride + 1
        output = np.zeros((x.shape[0], x.shape[1], output_height, output_width))

        for i in range(output_height):
            for j in range(output_width):
                x_slice = x[:, :, i * self.stride: i * self.stride + self.kernel_size,
                            j * self.stride: j * self.stride + self.kernel_size]
                output[:, :, i, j] = np.max(x_slice, axis=(2, 3))

        return output

class FullyConnected:
    def __init__(self, in_features, out_features):
        self.weights = torch.tensor(np.random.randn(out_features, in_features), requires_grad=True)
        self.bias = torch.tensor(np.zeros((out_features, 1)), requires_grad=True)
```

```
    def __call__(self, x):
        # Застосування повнозв'язаного шару
        return np.dot(self.weights, x.T).T + self.bias.T

class ReLU:
    def __call__(self, x):
        # Застосування функції активації ReLU
        return np.maximum(0, x)

class Dropout:
    def __init__(self, p):
        self.p = p

    def __call__(self, x):
        # Застосування Dropout (випадково вимикаємо нейрони)
        mask = (np.random.rand(*x.shape) > self.p) / (1 - self.p)
        return x * mask

class AdamOptimizer:
    def __init__(self, model, learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.learning_rate = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m = [np.zeros_like(param) for param in model.get_params()]
        self.v = [np.zeros_like(param) for param in model.get_params()]
        self.t = 0

    def zero_grad(self):
        for m_param, v_param in zip(self.m, self.v):
            m_param.fill(0)
            v_param.fill(0)

    def step(self):
        self.t += 1
        lr_t = self.learning_rate * np.sqrt(1 - self.beta2 ** self.t) / (1 - self.beta1 ** self.t)

        for i in range(len(self.m)):
            self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1) * model.get_params()[i].grad
            self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2) * model.get_params()[i].grad ** 2
            model.get_params()[i] -= lr_t * self.m[i] / (np.sqrt(self.v[i]) + self.epsilon)

class CrossEntropyLoss:
    def __init__(self):
        self.criterion = nn.CrossEntropyLoss()

    def __call__(self, predictions, targets):
        if not isinstance(predictions, torch.Tensor):
            predictions = torch.from_numpy(predictions)
        if not isinstance(targets, torch.Tensor):
            targets = torch.from_numpy(targets).long()
```

```
        return self.criterion(predictions, targets)
def train_model(model, train_data, val_data, num_epochs=10, learning_rate=0.001):
    optimizer = AdamOptimizer(model, learning_rate=0.001)
    criterion = CrossEntropyLoss()

    train_accuracies = []
    val_accuracies = []
    train_losses = []
    val_losses = []

    for epoch in range(num_epochs):
        running_loss = 0.0
        correct_train = 0
        total_train = 0

        for i, (inputs, labels) in enumerate(train_data, 1):
            optimizer.zero_grad()
            outputs = model.forward_pass(inputs)

            # Convert labels to PyTorch tensor and ensure they are of type long
            labels = labels.long()

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

            running_loss += loss.item()
            predicted = torch.argmax(outputs, axis=1).numpy()
            total_train += labels.shape[0]
            correct_train += np.sum(predicted == labels.numpy())

        train_accuracy = correct_train / total_train
        train_accuracies.append(train_accuracy)
        train_losses.append(running_loss / i)

        val_loss = 0.0
        correct_val = 0
        total_val = 0

        for i, (inputs, labels) in enumerate(val_data, 1):
            outputs = model.forward_pass(inputs)
            labels = labels.long()
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            predicted = torch.argmax(outputs, axis=1).numpy()
            total_val += labels.shape[0]
            correct_val += np.sum(predicted == labels.numpy())

        val_accuracy = correct_val / total_val
        val_accuracies.append(val_accuracy)
        val_losses.append(val_loss / i)
```

```
        print(f'Epoch {epoch + 1}/{num_epochs}, '
              f'Train Loss: {running_loss / i:.4f}, Train Accuracy: {train_accuracy:.4f}, '
              f'Validation Loss: {val_loss / i:.4f}, Validation Accuracy: {val_accuracy:.4f}')

    # Print final results
    print("Training finished. Final results:")
    print(f'Train Accuracy: {train_accuracies[-1]}, Validation Accuracy: {val_accuracies[-1]}')

    # Збереження моделі після тренування
    save_path = '/working/model_final.pth'
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    torch.save(model.state_dict(), save_path)
    print(f'Final model saved to {save_path}')

# Ініціалізація моделі та завантаження даних
model = SimpleCNN()
transform = transforms.Compose([
    transforms.Resize((350, 350)),
    transforms.ToTensor(),
])

train_set = ImageFolder(train_path, transform=transform)
train_loader = DataLoader(train_set, batch_size=16, shuffle=True)

val_set = ImageFolder(val_path, transform=transform)
val_loader = DataLoader(val_set, batch_size=16, shuffle=True)

test_set = ImageFolder(test_path, transform=transform)
test_loader = DataLoader(test_set, batch_size=16, shuffle=True)

# Виклик функції для тренування моделі
train_accuracies, val_accuracies, train_losses, val_losses = train_model(model, train_loader,
val_loader, num_epochs=10)

# Виведення графіків
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Завантаження збереженої моделі
model = SimpleCNN()
model.load_state_dict(torch.load('/kaggle/working/model_final.pth'))
model.eval()

# Функція для прогнозування та виведення зображення з написом
def predict_and_display(image_path, model, transform, dataset):
    # Завантаження та обробка зображення
    image = Image.open(image_path).convert("RGB")
    input_tensor = transform(image)
    input_batch = input_tensor.unsqueeze(0)

    # Прогнозування класу
    with torch.no_grad():
        output = model(input_batch)

    # Отримання індексу класу з найвищим значенням ймовірності
    _, predicted_idx = torch.max(output, 1)

    # Custom class names
    class_names = ["Adenocarcinoma", "Large cell carcinoma", "Normal", "Squamous cell
carcinoma"]

    # Predicted class name
    predicted_class = class_names[predicted_idx.item()]

    # Отримання ймовірності для прогнозу
    probabilities = torch.nn.functional.softmax(output, dim=1)
    predicted_probability = probabilities[0, predicted_idx].item() * 100

    # Виведення зображення з написом та точністю прогнозу
    plt.imshow(image)
    plt.title(f'Predicted Class: {predicted_class}, Accuracy: {predicted_probability:.2f}%')
    plt.show()

# Шлях до зображення, яке ви хочете прогнозувати
image_path_to_predict = '/kaggle/input/chest-ctscan-
images/Data/test/large.cell.carcinoma/000113.png'

# Прогнозування та виведення
predict_and_display(image_path_to_predict, model, transform, test_set)
```