МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ Коплик І.В
«__»_____20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

спеціальність 113 «Прикладна математика»

освітньо-професійна програма «Наука про дані та моделювання складних систем»

тема роботи **«Штучний інтелект для скринінгу та діагностики грудної залози»**

Виконавець
студент факультету ЕлІТ
Васильченко Микола Анатолійович _____

Науковий керівник
кандидат фізико-математичних наук
Дворниченко Аліна Василівна_____

Суми 2021

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.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Васильченко Микола Анатолійович

1. Тема роботи Штучний інтелект для скринінгу та діагностики грудної залози

Керівник роботи Кандидат фізико-математичних наук, Дворниченко Аліна Василівна

затверджую наказом по факультету ЕлІТ від «08» жовтня 2021 р. № 0687-VI

2. Термін подання роботи студентом «16» грудня 2021р.

3. Вихідні данні до роботи 1) Набір зображень ЕКГ з доброякісними і злоякісними пухлинами;
2)Табличні дані хворих на ракову пухлину

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Розробка методу дослідження зображень для аналізу зображень знімків ЕКГ молочної залози, що дозволить пришвидшити аналізування проблеми пухлини

5. Перелік графічного матеріалу

1) Приклад пухлини головного мозоку.

2) Приклад злоякісної і доброякісної пухлини.

3) Схематичний вигляд перциптрону.

4) Найпростіша нейронна марежа з прихованими шарами.

5) Сігмоїдальна функція.

6) Приклад локалізації зображення.

7) Застосування фільтру до зображення.

8) Субдискретизація зображення .

за методом максимізації.

9) лінійна та нелінійна регресія.

10) Приклад дерева рішень.

11) -14) Пришлади підготування зображень.

15) Схематичний вигляд нейронної мережі.

16) – 19) Результати навчання нейронної мережі.

20) Табличний вигляд даних представлених у датасеті.

21) Співвідношення злоякісної і доброякісної пухлини.

22) Графік середньої точності розрахунків різних моделей.

23) Гістограма середньої точності прогнозування даних.

3. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

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

7. Дата видачі завдання «01» листопада 2021р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1)	Ознайомлення з поставленою задачею	01.11.2021-07.11.2021	
2)	Пошук і аналіз літератури	08.11.2021-11.11.2021	
3)	Проведення розрахунків	11.11.2021-18.11.2021	
4)	Моделювання та опис отриманих розрахунків	18.11.2020-26.11.2021	

Здобувач вищої освіти _____ М.А. Васильченок

Керівник роботи _____ А.В. Дворниченко

РЕФЕРАТ

Кваліфікаційна робота: 66 с., 23 рис., 2 таблиці, 9 джерел.

Ключові слова: НЕЙРОННІ МЕРЕЖІ, ПУХЛИНА, ЗГОРТОЧНІ НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ, ЗОБРАЖЕННЯ.

Мета роботи: дослідити способи виявлення ракової пухлини у молочній залозі. Для цього будуть розроблені декілька моделей штучного інтелекту і таблично-аналітичній спосіб прогнозування. Будуть встановлені елементи зображення, які вказують на тип пухлини. Також представлені декілька нейронних мереж: проса нейронна мережа (перциптрон), згорточна нейронна мережа та модифікована нейронна мережа від компанії google. Всі вони будуть застосовані для аналізу зображень пухлин на молочній залозі. Ще одним способом буде статистичний аналі табличних даних, яки буде складатись з побудови моделі дерева рішень, регресійного аналізу та «ліса дерев».

В роботі будуть проаналізовані отримані дані, для виявлення оптимального способу аналізу пухлини. Так буде з’ясовано, чи є актуальним заносити параметри пухлини в таблицю для аналізу його типу, чи можно проаналізувати одразу зі знімка.

Додаються додатки в яких міститься код програм для розрахунку і візуалізації даних.

ЗМІСТ

ВСТУП

Метою цієї роботи є розробка методу дослідження зображень для аналізу зображень знімків ЕКГ молочної залози, що дозволить пришвидшити аналізування проблеми пухлини. З впевненістю не можна стверждувати, що машинний аналіз є точніший, але він дає змогу прибрати людські недоліки (неуважність, психіку і переваг до пацієнтів).

Тому спробуємо не тільки розробити, але і порівняти різні способи аналізу. Для цього в дану роботу будуть добавлені як більш проста модель нейронної системи, так і гараздо більш точна, яку я використаю в якості порівняння, і не треба забувати про велику частину аналізу даних у вигляді табличних значень.

Оскільки онкологічні захворювання є доволі великою проблемою в нашому світі, і швидкість розпізнавання може значно підвищити шанси людини до виздоровлення. Тому я вважаю, що дана тема і технологія має сенс до нашого реального життя.

Саме для підвищення швидкості аналізу і конкурентноспроможності з людимою, в даній роботі використовується потужна нейронна мережа під назвою «згорточна». За основу була взята класична нейронна мережа яка має вхідні нейрони, приховані шари та вихідні нейрони, які виступають кінцевим рішенням. Також до цієї мережі застосовано згортка пікселів зображення задопомогою фільтрів, які мають властивість налаштовуватись (навчатись) автоматично.

1. ПОСТАНОВКА ЗАДАЧІ

1. Підготувати медичні знімки та табличні данні для зручного використання перед обробкою штучним інтелектом
2. Проаналізувати і використати методи аналізу зображень
3. Проаналізувати різними методами табличні значення захворюваності
4. Визначити більш точний і зручний метод для виявлення захворіння

2. ЛІТЕРАТУРНИЙ ОГЛЯД

2.1. Проблематика онкозахворювання

2.1.1. Основні поняття онкозахворювання

Починаючи дослідження даного захворювання спершу потрібно зрозуміти її сутність і в чому саме проблема.

Пухлина— патологічний процес,представлений новостворено тканиною,в якій зміни генетичного апарату клітин призводять до порушення регуляції їхнього росту і диференціювання, див. рис 1 [1].

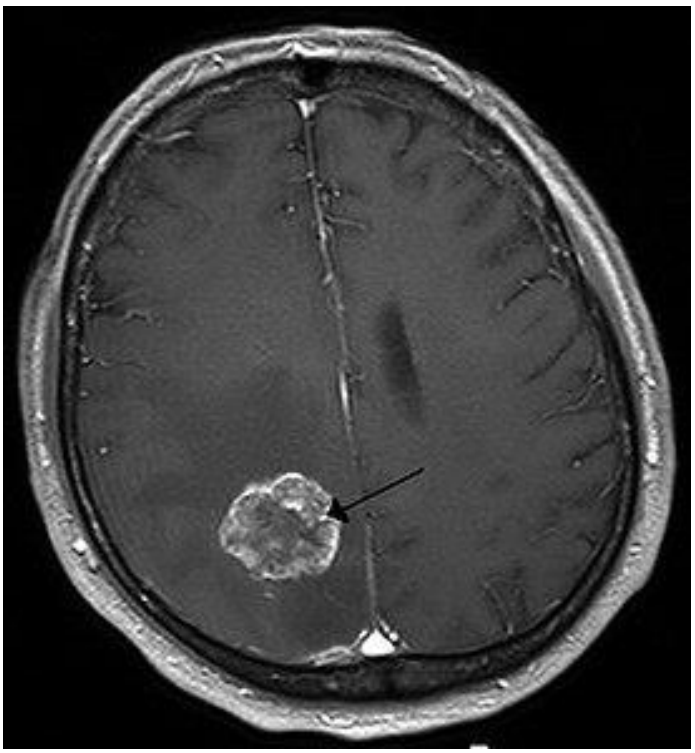


Рис. 1 – Приклад пухлини головного мозоку.

В природі існує два види пухлини: доброякісна і злоякісна.

Злоякісна пухлиина (див. рис. 2.а) — патологічний процес, зумовлений неконтрольованим розмноженням клітин, інвазією та, іноді, метастазуванням. Залежно від типу тканини, клітини якої перетворились на злоякісні, розрізняють рак, саркому, лімфому та інші види злоякісних пухлин [1].

Доброякісні пухлини (див. рис. 2.б) характерезуються повільним ростом, при цьому без проростання у сусідні тканини (експанисивний ріст), в більшості випадків обмежені капсулою. При пальпації мають гладку

поверхню, не болять, легко зміщуються, не метастазують. Під впливом деяких факторів можуть перетворюватись у злоякісні пухлини [2].

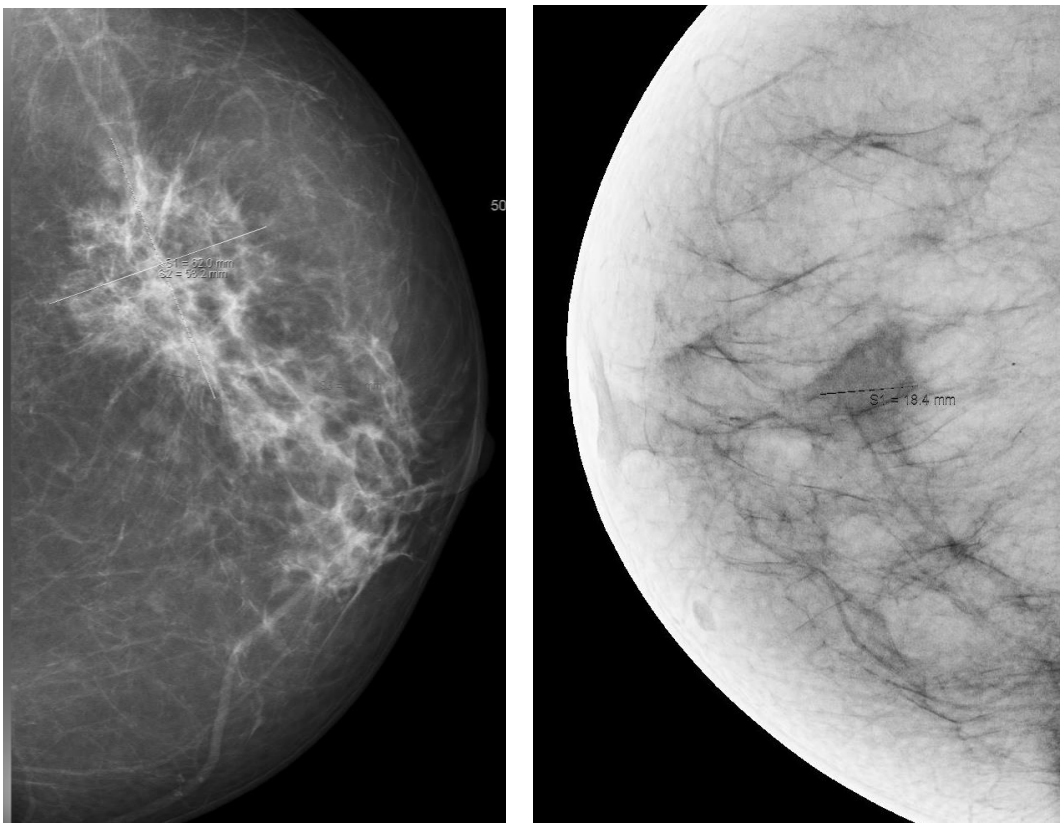


Рис. 2 – а) злоякісна пухлина у молочній залозі;

б) доброякісна пухлина у молочній залозі.

Загальною характеристикою злоякісних пухлин є їх виражені тканинні аплазія, атипізм (тобто втрата клітинами здатності до диференціювання із порушенням структури тканини, з якої розвивається пухлина), агресивне зростання з ураженням, як самого органу, так і інших прилеглих органів, схильність до метастазування, тобто до поширення клітин пухлини із током лімфи або крові по всьому організму з утворенням нових вогнищ пухлинного росту (метастазів) в багатьох органах, віддалених від первинної пухлини. За стрімкістю зростання, більшість злоякісних пухлин перевершують доброякісні та, як правило, можуть досягати значних розмірів у короткі терміни. Розрізняють також вид місцево деструктивних злоякісних пухлин, що ростуть з утворенням інфільтрату в товщі тканини, призводячи до її руйнування, але, як правило, не метастазують. До таких форм відносять, наприклад, базаліому шкіри.

Існує поверхневий пухлинно-специфічний антиген, поява якого пов'язується не із утворенням раковою клітиною яких-небудь нових білків, а із втратою нею частини поверхневою структури. Перешкодою до розпізнавання лімфоцитами є «слабка антигенність» багатьох спонтанних пухлин. Це може бути викликано дестабілізацією поверхневої мембрани клітин за злоякісного переродження. Антигенні детермінанти виявляються вільно плаваючими у «рідкій» мембрані й не можут бути впізнаними відповідними лімфоцитами.

Основну роль у реакції неприйняття пухлинного трансплантату відіграють спцифічні Т-кілери, а також (на перших етапах) цитотоксичні антитіла. Взаємодія між лімфоцитом-кілером й клітиною здійснюється в три етапи:

- встановлення специфічного контакту між клітинами;
- летальний удар, який носить назву «поцілунок смерті»;
- деструкція клітин-мішеней.

У цій реакції одна Т-клітина здатна нанести удар по декільком пухлинним клітинам. Транскрипційний чинник, регулюючий активність генів, які є відповідальними за диференціювання Т-клітин [3].

Аспекти взаємодії специфічних імунних сил та злоякісного новоутворення можна з'ясувати при дослідженні простої моделі, яка містить динаміку активних клітин пухлини x та лімфоцитів-кілерів y. Модель записується у вигляді:

$$\frac{\mathrm{dx}}{\mathrm{dt}} = \mu_{\mathrm{x}}\mathrm{x} - \gamma_{\mathrm{x}}\mathrm{xy}, \frac{dy}{dt} = \mu_y(x - \beta x^2)y - x_y y + v_y \,. \qquad (1)$$

Тут припускається, що клітини пухлини розмножуються із сталою питомою швидкістю μ_{x},обмеження швидкості росту пухлини відбувається через знищення злоякісних клітин лімфоцитами-кілерами ($\gamma_{\mathrm{x}}\mathrm{xy}$). Рівняння динаміки лімфоцитів містить член, який описує їх розмноження, $\mu_{\mathrm{y}}(\mathrm{x} - \beta\mathrm{x}^2)\mathrm{y}$. У моделі припускається, що за малого x пухлина стимулює проліферацію лімфоцитів, а за великого — пригнічує. Два інших члени у

другому рівнянні відповідають природній смерті лімфоцитів й сталому притокові попередників із «стовбурових клітин». Смерть лімфоцитів за взаємодії із пухлинними клітинами тут явно не враховується, оскільки припускається, що один лімфоцит може знищити декілька пухлинних клітин.

Вважаючи кількість пухлинного антигену G пропорційним числу клітин пухлини G=gx, можна записати швидкість виробництва лімфоцитів в залежності від кільності антигену:

$$\left(\frac{\mathrm{dy}}{\mathrm{dt}}\right)_{+} = \bar{\mu}_{\mathrm{y}}\mathrm{yG}(\mathrm{k_G} + \mathrm{G})^{-1} = \bar{\mu}_y yx(x_0 + x)^{-1}, \mathrm{X}_0 = \frac{\mathrm{k_G}}{\mathrm{g}} \qquad (2)$$

де $\overline{\mu_{\mathrm{y}}}$— коефіцієнт розмноження лімфоцитів за максимальної стимуляції $(\mathrm{G} \to \infty)$. Константа $\mathrm{K_G}$ характеризує антигенність пухлини: за $\mathrm{K_G}$ імунна відповідь є послабленою.

Можливість ракових клітин функціонувати в умовах низької концентрації глюкози вказує на їх високу спорідненість до неї. Відтак гормони надниркової залози (глюкокортикоїди), які стимулюють процеси глюконеогенезу, у організмі із злоякісною пухлиною можуть пригнічувати імунні сили організму. Відтак проблемою онкоімнології є розробка ефективних методів імунотерапії злоякісних пухлин. Одним з таких методів є імунотерапія пухлин, заснована на ад'ювантній дії мікроорганізмів — представників різних видів Corynebacterium, Mycobacterium, Streptococcus тощо. Вони посилюють Т-кілерну активність[4].

Ракові клітини мають здатність до більш швидкого перенесення молекул глюкози через мембрану. Це створює перевагу ракових клітин у конкурентній боротьбі із іншими клітинами організму за глюкозу. Пухлина дії як «пастка глюкози» (а також вітамінів, азотистих основ й інших метаболітів), конкуруючи із здоровими клітинами за життєво важливі ресурси[5].

2.1.2. Природа канцеру

Сучасна медицина знає багато факторів, які можуть запускати механізми розвитку раку. Речовини або фактори навколишнього середовища, що мають цю властивість, називаються канцерогенами.

- Хімічні канцерогени - до них відносяться різні групи поліциклічних та гетероциклічних ароматичних вуглеводнів, ароматичні аміни, нітрозосполуки, афлатоксини, а також вінілхлорид, метали, пластмаси і т. д. Їх загальною характеристикою є здатність реагувати з ДНК клітин і цим викликати їх злоякісне переродження.

- Канцерогени фізичної природи – це різні типи іонізуючого випромінювання (α, β, γ-випромінювання, рентгенівське випромінювання).

- Біологічні фактори канцерогенезу – це різні типи вірусів – вірус герпесу людини 4-го типу – вірус Епштейна-Барра (лімфома Беркітта), вірус герпесу людини 8-го типу (саркома Капоші), вірус папіломи людини (рак шийки матки), віруси гепатиту та С (печінка), які мають у своїй структурі специфічні онкогени, що сприяють зміні генетичного матеріалу клітини з подальшим її злоякісним новоутворенням.

- Гормональні фактори – деякі типи людських гормонів (наприклад, статеві гормони) можуть викликати злоякісне переродження тканин, чутливих до цих гормонів (рак грудей, рак яєчок, рак простати).

- Генетичні чинники - різні молекулярні і генетичні аномалії, включаючи мутації, делеції, перебудови генів тощо. буд., можуть призвести до розвитку злоякісних новоутворень. Генетичні аномалії можуть виникати спорадично або передаватися у спадок (наприклад, мутація у протоонкогені RET, що призводить до розвитку медулярного раку щитовидної залози).

В цілому впливаючи на клітину, канцерогени спричинюють певні порушення її структури і функції (особливо ДНК), що називають ініціацією. Пошкоджена клітина таким чином набуває виражений потенціал до малігнізації. Повторний вплив канцерогену (того ж, що викликав ініціацію, або будь-якого іншого) призводить до незворотних порушень механізмів, які контролюють розподіл, ріст і диференціювання клітин, в результаті яких клітина набуває ряд здібностей, не властивих нормальним клітинам організму — промоція.

Зокрема, пухлинні клітини набувають здатності до безконтрольного поділу, втрачають тканиноспецифічну структуру та функціональну активність, змінюють свій антигенний склад, тощо. Зростання пухлини (пухлинна прогресія) характеризується поступовим зниженням диференціювання і збільшенням здатності до безконтрольного поділу, а також зміною взаємозв'язку «пухлинна клітина — організм», що призводить до утворення метастаз. Метастазування відбувається переважно лімфогенним шляхом (тобто із током лімфи) в регіонарні лімфовузли, або ж гематогенний шляхом (з током крові) з утворенням метастазів у різних органах (легені, печінка, кістки, тощо).

Ознаки злоякісних пухлин:

- Пухлинні клітини характеризуються втратою диференціювання, спрощенням і атиповістю будови.
- Швидкий ріст пухлини, що спонтанно не зупиняється, мимовільна регресія таких пухлин не відома.
- Інвазивний ріст — інфільтрують навколишні тканини.
- Зазвичай дають метастази.
- При відсутності ефективного лікування призводять до смерті.[6]

2.1.3. Статистичні показники захворюваності

Згідно з даними Національного інституту раку, протягом 2018 року в Україні на обліку перебувало 975 301 пацієнт, у якого було діагностовано злоякісні новоутворення (ЗН), зокрема:

- понад 147 000 випадків — у молочних залозах;
- більш ніж 42 000 — у щитовидних залозах;
- 29 000 випадків — у трахеї, бронхах, легенях.

Таблиця 1. – Кількість хворих на обліку,
показник на 100 тис. населення.

Локалізація ЗН	Особи		Чоловіки		Жінки	
	кількість*	%₀₀₀₀	кількість*	%₀₀₀₀	кількість*	%₀₀₀₀
всі	**975301**	**2719.3**	**341668**	**2050.1**	**633633**	**3300.2**
всі за виключ. немеланомних ЗН шкіри	***793592***	***2212.7***	***269500***	***1617.1***	***524092***	***2729.6***
молочна залоза	147192	410.4	875	5.3	146317	762.1
немеланомні ЗН шкіри	181709	506.6	72168	433.0	109541	570.5
тіло матки	76968	214.6	-	-	76968	400.9
шийка матки	53847	150.1	-	-	53847	280.5
передміхурова залоза	39891	111.2	39891	239.4	-	-
щитовидна залоза	42765	119.2	6451	38.7	36314	189.1
сечовий міхур	31469	87.7	24895	149.4	6574	34.2
ободова кишка	50670	141.3	22299	133.8	28371	147.8
яєчник	27816	77.6	-	-	27816	144.9
трахея, бронх, легеня	29093	81.1	20893	125.4	8200	42.7

2.2. Огляд методів дослідження

2.2.1. Нейронні мережі

Що таке нейромережа? Спочатку розглянемо найпростіший тип штучного нейрона – перцептрон.

Перцептрон приймає на вхід кілька двійкових чисел x1, x2, … і видає одне двійкове число (див. рис. 3):

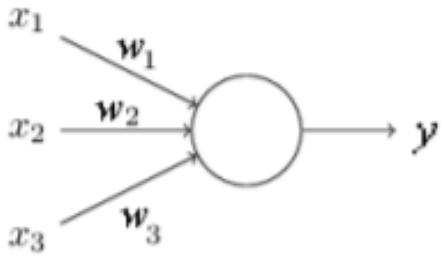


Рис. 3. – Схематичний вигляд перциптрону.

Ваги, w1, w2, ... - Речові числа, що виражають важливість відповідних вхідних чисел для результатів. Вихід нейрона, 0 (нейрон не був активований) або 1 (нейрон активувався і передав сигнал далі), визначається тим, менше або більше якогось порога threshold зважена сума x1w1+x2w2+…+xnwn (n – у входів). Як і ваги, поріг – дійсне число (параметр нейрона). Говорячи математичними термінами:

$$y = \begin{cases} 0, \text{якщо } \sum_j w_j x_j \leq thresold, \\ 1, \text{якщо } \sum_j w_j x_j > thresold\,. \end{cases} \quad (3)$$

У цій мережі перший стовпець перцептронів – те, що ми називаємо першим шаром перцептронів – приймає три дуже прості рішення, зважуючи вхідні дані. Перцептрони другого шару приймають рішення, зважуючи результати першого шару прийняття рішень. Таким способом перцептрон другого шару може прийняти рішення на більш складному та абстрактному рівні, порівняно з перцептроном першого шару. А ще складніші рішення можуть приймати перцептрони на третьому шарі.

Допустимо, у нас є мережа перцептронів, яку ми хочемо використовувати для вирішення завдання розпізнавання рукописних символів-цифр. У цьому випадку вхідними даними мережі будуть пікселі

відсканованого зображення рукописної цифри. При розмірі чорно-білого зображення 28x28 пікселів (вхідні пікселі чорно-білі, при цьому значення 0 позначає білий колір, 1 позначає чорний, а проміжні значення позначають все темніші відтінки сірого.), кількість вхідних нейронів – 784 (по одному на кожен пік), кількість нейронів на виході - 10 (по одному на кожну цифру). Якщо активується перший нейрон вихідного шару, тобто його вихідне значення ≈ 1, це говорить про те, що мережа вважає, що на вході був 0. Якщо активується другий нейрон, мережа вважає, що на вході була одиниця. І так далі. Строго кажучи, ми нумеруємо вихідні нейрони від 0 до 9, і дивимося, у якого значення було максимальним. Якщо це, скажімо, нейрон №6, тоді наша мережа вважає, що на вході була цифра 6. І так далі.

Як показано нижче, для коректної роботи мережі необхідний додатковий шар нейронів (можливо не один!) виділення основних характеристик зображення. Цей шар серед фахівців прийнято називати прихованим. У найпростішому випадку така мережа матиме вигляд рис. 4.

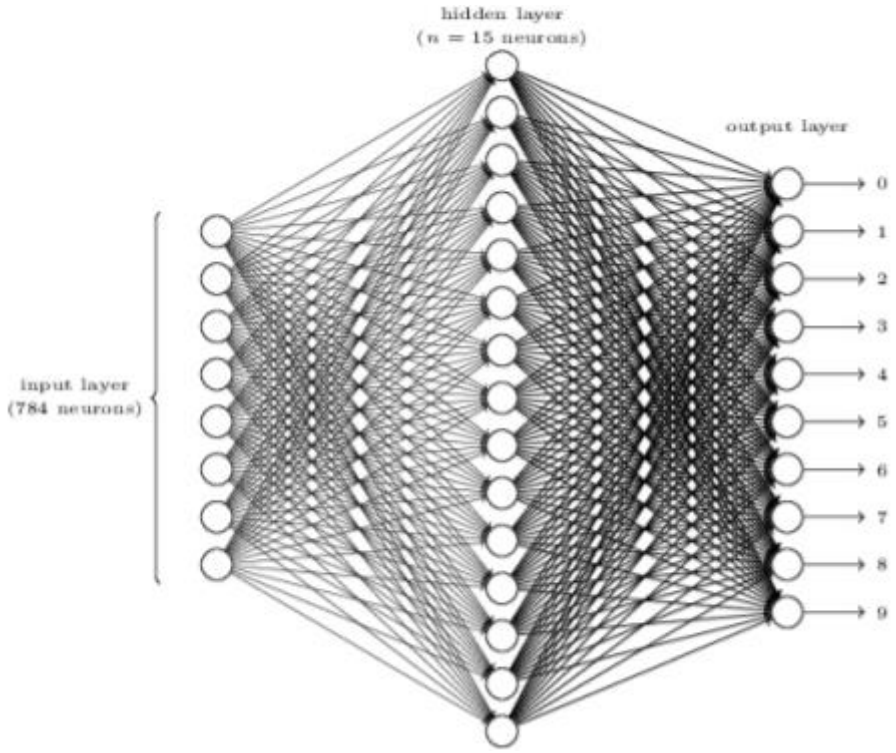


Рис. 4 – Приклад простої нейнонної марежі з прихованими шарами.

Розберемо алгоритм автоматичного підбору терезів і зсувів, необхідні правильної класифікації цифри.

У нього так само є вхідні дані x1, x2, ... Входи (виходи) сигмоїдального нейрона можуть набувати будь-якого значення в проміжку від 0 до 1. Наприклад, величина 0,638 буде допустимим значенням вхідних даних. Сігмоїдальна (або логістична функція), як відомо, має вигляд представлений на рис. 5.

$$f(z) = \frac{1}{1+e^{-z}} \tag{4}$$

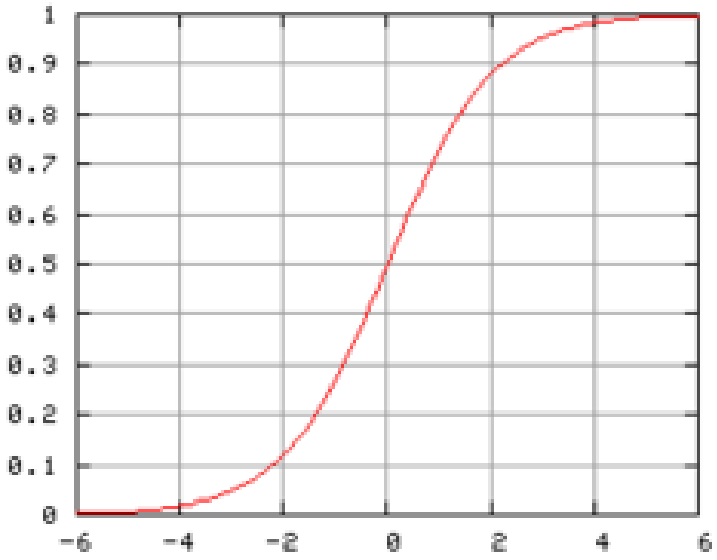


Рис. 5. – Вигляд сігмоїдальної функції.

На відміну від елементарного перцептрону вихід сигмоїдального нейрона у формується шляхом пропускання сумарного вхідного сигналу z=x1w1+x2w2+…+xnwn+b через сигмоїдальну функцію, y=f(z). Сигмоїдальна функція в нашому випадку називається активаційною функцією нейрона.

Навчання, побудоване на основі градієнтного спуску.

Нам хочеться знайти алгоритм, що дозволяє нам шукати такі ваги та зсуву, щоб вихід мережі z наближався до y(x) для всіх навчальних вхідних x. Щоб кількісно оцінити наближення цієї мети, визначимо функцію вартості.

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - z\| \tag{5}$$

Тут w позначає набір ваги мережі, b – набір зсувів, n – кількість навчальних вхідних даних, z – вектор вихідних даних, який відповідає вектору вхідних даних x; підсумовування проводиться за всіма навчальними вхідними даними x (за набором прикладів зображень різних цифр). Позначення || || означає довжину вектор. Ми називатимемо C квадратичною функцією вартості; іноді її ще називають середньоквадратичною помилкою, або MSE. Вартість C(w,b) стає малою, коли вектор y приблизно дорівнює еталонному вектору z для всіх навчальних вхідних даних x. Так що наш алгоритм працює добре, якщо він дозволяє знайти ваги та зміщення такі, що C(w,b) ≈ 0. І навпаки, працює погано, якщо C(w,b) >>0 – це означає, що y(x) не збігається з виходом для великої кількості вхідних даних. Виходить, мета навчального алгоритму – знайти набір терезів і зсувів, що мінімізує значення функції вартості C. Ми робитимемо це за допомогою алгоритму під назвою градієнтний спуск. Ми можемо обчислити похідні та спробувати використати їх для пошуку екстремуму.

Щоб градієнтний спуск працював правильно, потрібно вибрати досить мале значення швидкості навчання η. В іншому випадку, в околиці мінімуму може вийти, що C > 0 (ми перестрибнемо через мінімум!). У той же час, не потрібно, щоб η була надто маленькою, оскільки тоді зміни Δv будуть крихітними, і алгоритм працюватиме надто повільно. Насправді значення η беруть з діапазону (0,1).

Для коригування значень ваг та байєсів використовуються формули:

$$\omega'_K = w_k - \eta \frac{\partial C}{\partial \omega_k},$$

$$b'_I = b_I - \eta \frac{\partial c}{\partial b_I}, \qquad (6)$$

На практиці для обчислення градієнта ∇C нам потрібно обчислювати градієнти ∇Cx окремо для кожного навчального входу x, а потім усереднювати їх. На жаль, коли кількість вхідних даних буде дуже великою, таке навчання проходитиме дуже повільно. Для прискорення навчання можна використати стохастичний градієнтний спуск. Ідея в тому, щоб приблизно обчислити

градієнт ∇C, обчисливши ∇Cx для невеликої випадкової вибірки навчальних вхідних даних (така вибірка називається міні-пакетом). Порахувавши їхнє середнє, ми можемо швидко отримати хорошу оцінку істинного градієнта ∇C.

$$\nabla C = \frac{1}{m}\sum_{j=1}^{m} \nabla C_{xj}, \tag{7}$$

,де m – кількість навчальних прикладів пакеті, Xj – приклад j з набору навчальних прикладів пакета. В результаті формули коригування ваг набувають вигляду.

$$\omega_K' = w_k - \frac{\eta}{m}\frac{\partial C}{\partial \omega_k},$$

$$b_I' = b_I - \frac{\eta}{m}\frac{\partial c}{\partial b_I}, \tag{8}$$

2.2.2. Глибокі нейронні мережі. Згорточна нейронна мережа

Під неглибокої СР зазвичай розуміють мережу з одним прихованим шаром. Відповідно, ДПС - це мережа з двома і більш прихованими шарами. І чим більше прихованих шарів, тим глибшою вважається нейронна мережа.

Але використання великої кількості прихованих шарів або великої кількості вхідних нейронів може бути критичним при розрахунку великої кількості даних. Для цього було розроблено згорткову мережу.

Ідея згорткових нейронних мереж полягає в чергуванні прихованих згорткових шарів (C-layers), прихованих субдискретизуючих шарів (S-layers) та наявності повнозв'язних (F-layers) шарів на виході. Така архітектура містить у собі 3 основні парадигми:

1) Локальне сприйняття.

2) Роздільні ваги.

3) Субдіскретизація.

Розглянемо ці парадигми детальніше.

Локальне сприйняття передбачає, що у вхід одного нейрона внутрішнього шару подається в повному обсязі зображення (чи виходи попереднього шару), лише деяка його область. Такий підхід дозволив зберігати топологію зображення від шару до шару. Тобто. якщо у нас є на вході

зображення розмірами 32x32 пікселя, то кожен з нейронів наступного шару прийме на вхід лише невелику ділянку цього зображення розміром, наприклад, 5x5. Зверніть увагу, ми не представляємо вихідне зображення у вигляді вектора стовпця, обробляючи двовимірний масив: топологія зображення не втрачається.

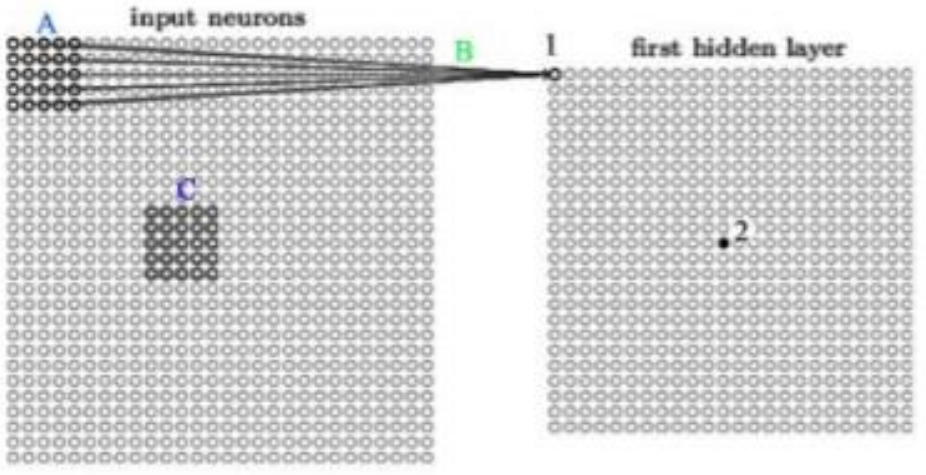


Рис. 6. – Приклад локалізації зображення.

Концепція ваг, що розділяються, передбачає, що для великої кількості зв'язків використовується дуже невеликий набір ваг. Фактично кожен нейрон наступного внутрішнього шару (наприклад, нейрон 1 на малюнку вище), “відповідає” за певну ділянку попереднього шару (матриця A). Вхід цього нейрона розраховується аналогічно до того, як це робилося раніше. Маємо 25 зв'язків, причому ваги цих зв'язків занесені в матрицю розміром 5x5 (на початковому етапі ця матриця формується за допомогою генератора випадкових чисел). Як це робилося раніше, поелементно перемножуємо ваги на вхідний сигнал (A B), підсумовуємо і отримуємо вхід нейрона 1. Та ж процедура виконується для кожної ділянки вхідної матриці (попереднього шару), наприклад, твір B формує вхідний сигнал нейрона 2 на рис. 2.2.2.6. Важливо! Матриця B (набір терезів) називається фільтром або ядром; вона не змінюється для формування сигналів на входах наступного шару (first hidden layer на рис. вище). Фактично, ваги зв'язків нейронів наступного шару з нейронами попереднього шару однакові

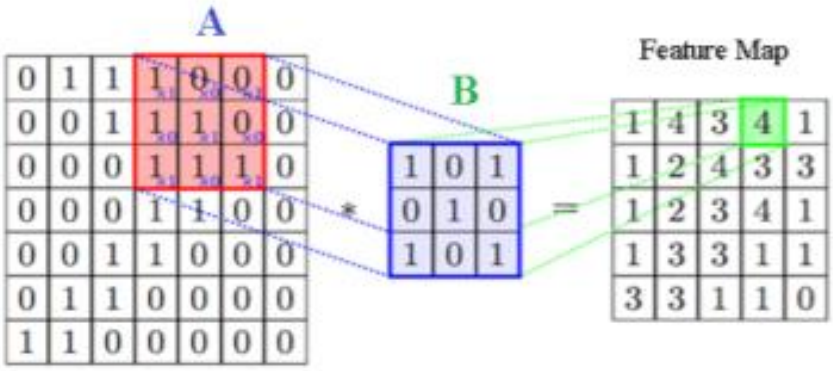


Рис. 7 – Застосування фільтру до зображення.

Як бачимо, матриця B застосовується до зображення за допомогою математичної операції, званої згорткою (звідси назва згортковий шар (C-layers)). Суть цієї операції в тому, що кожен фрагмент зображення множиться на матрицю B (ядро) згортки поелементно і підсумовується результат і записується в аналогічну позицію вихідного зображення (подається на вхід нейрона наступного шару). Основна властивість фільтрів

полягає в тому, що значення їх виходу тим більше, чим більше фрагмент зображення схожий на сам фільтр. Таким чином, зображення, згорнуте з якимось ядром (фільтром, матрицею ваг) B дасть нам інше зображення, кожен піксел якого означатиме ступінь схожості фрагмента зображення на фільтр B. Саме тому отриманий прихований шар називається картою ознак. Фільтрів може бути дуже багато, причому кожен із них відповідає за елемент, з яких складається вихідне зображення. Отже, прихований шар буде представляти з себе набір двовимірних карт ознак (матиме шарувату структуру), причому кількість карт співпадатиме з кількістю фільтрів.

Суть субдискретизації та S-шарів полягає у зменшенні просторової розмірності зображення. Тобто. вхідне зображення грубо (середнення) зменшується в задану кількість разів. Найчастіше в 2 рази (крок дорівнює 2), хоча може бути і не рівномірна зміна, наприклад, 2 по вертикалі та 3 по горизонталі. Субдискретизація (або даунсемплінг) потрібна для забезпечення інваріантності до масштабу, прискорення процесу навчання та зменшення

споживання обчислювальних ресурсів. Існує кілька способів виконати субдискретизацію. Розглянемо найпростіший і найпоширеніший з них - максимальне об'єднання (max pooling). Операція максимального об'єднання полягає в тому, що вздовж даних переміщується так зване вікно просіювання. З пікселів, що потрапляють у поле зору, відбирається максимальний і переміщається в результуючу матрицю. Крок (або страйд) для цього вікна може бути відмінним від одиниці; все залежить від того, якого розміру вихідну матрицю потрібно отримати, і з яким ступенем потрібно "стиснути" дані.

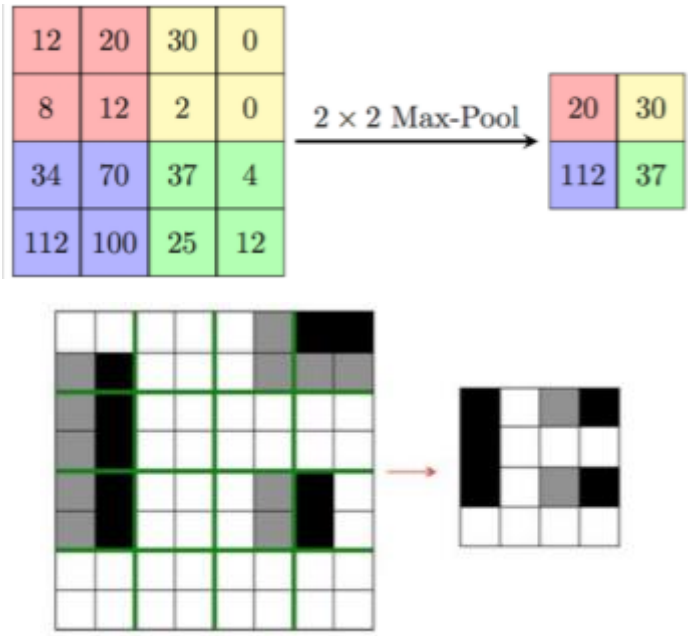


Рис. 8. – Субдискретизація зображення за методом максимізації.

Чергування шарів дозволяє складати карти ознак із карт ознак, що на практиці означає здатність розпізнавання складних ієрархій ознак. Фактично, мережа згортка складається з двох частин: перша частина являє собою комбінацію згорткових і пулінгових S-шарів (кількість цих шарів і порядок чергування визначає архітектор нейромережі); друга частина - звичайна нейромережа з одного або декількох повнозв'язкових шарів та вихідного шару.

2.2.3. Статистичний аналіз табличних даних

2.2.3.1. Регресійний аналіз

До класичних методів статистичного аналізу належать: кореляційний аналіз, регресійний аналіз, перевірка гіпотез, метод порівняння середніх, аналіз часових рядів та інші. Оскільки попередньо була визначена цільова змінна – рівень заробітної плати IT-спеціаліста, то найбільш прийнятним методом моделювання є регресійний аналіз.

Регресійний аналіз, який використовує вибраний метод оцінки, залежну змінну та одну або кілька незалежних змінних для створення рівняння, яке оцінює значення залежної змінної. Регресійна модель включає вихідні дані, такі як R2 і p-значення, які можна використовувати, щоб зрозуміти, наскільки добре модель оцінює залежну змінну [7]. Існує два основних типи регресії - лінійна і нелінійна.

$$R^2 = 1 - \frac{\sigma^2}{\sigma_y^2} \tag{9}$$

Регресія називається простою, якщо вхідна змінна дорівнює одиниці. Однак така модель є занадто грубою апроксимацією реальності, і на практиці зазвичай використовують залежності від кількох змінних.

Для лінійної регресії коефіцієнти повинні бути перевірені та задоволені за допомогою методу MNC. Для МНК коефіцієнти регресії визначаються як унікальні значення, які мінімізують суму квадратів помилок є в межах вибірки даних, що вивчаються в моделі.

Також однією з необхідних умов використання МНК при оцінці параметрів моделі є гомоскедастичність, тобто постійність дисперсії випадкових залишків для кожного спостереження. Якщо дисперсія залишків змінюється для кожного спостереження, це явище називається гетероскедастичністю. За наявності гетероскедастичності оцінки параметрів моделі будуть розумними, неупередженими, але неефективними.

Відповідність вимозі гомоскедастичності випадкових залишків можна перевірити візуально, на основі графіка залишків.

У лінійній регресії функція залежить від незалежних параметрів лінійно. З рис. 9.a видно, що цей тип регресії має вигляд лінії. Якщо представити лінійну регресію у вигляді залежності x від y, то отримаємо формулу [8]:

$$y = \beta_0 + \beta_1 x + \varepsilon, \quad (10)$$

де y та x – деякі змінні, β_0 та β_1 – коефіцієнти моделі, ε – помилка.

Тоді як для нелінійної регресії функція повинна залежати не тільки від суми незалежних змінних, але мати варіації множення, піднесення та ін. приклад:

$$y = \beta_0 x^2 + \beta_1 x + \varepsilon \quad (11)$$

Тоді цей тип регресії буде виглядати як крива, як показано на рисунку .9.b.

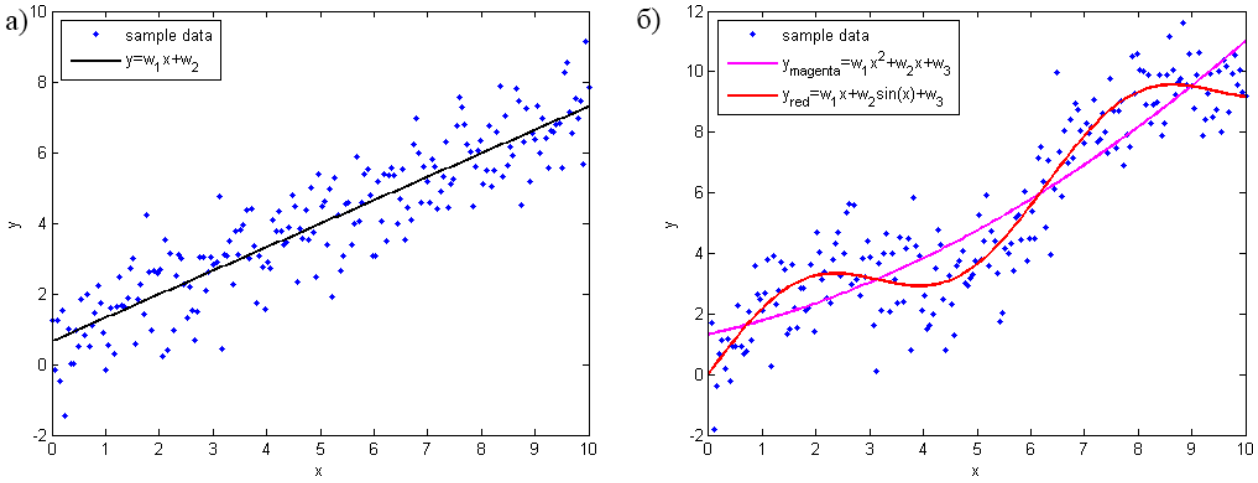


Рис. 9. - а) лінійна регресія; б) нелінійна регресія [8].

Для мого випадку буде доречно побудувати регресійні моделі та вибрати параметри для прогнозування зарплати працівника.

Для вибору правильної моделі регресії необхідно:

- Підготовка та очищення даних, усвідомлення їх походження;
- Дослідження описової статистики для розуміння загальних закономірностей даних та виявлення проблем з якістю даних;

- Застосовуйте відповідні перетворення даних, якщо є докази нелінійних залежностей або шуму, який не є нормально розподіленим або не залежить від часу;
- Порівняння різних моделей та їх удосконалення;
- Перевіряє, чи достатньо задовольняються припущення певної моделі, чи пропонується альтернативна модель;
- Вибір серед прийнятних моделей для заданого рівня надійності;
- Отримання корисних висновків з усього процесу.

Після побудови моделі ми маємо наступні показники, які необхідні для перевірки адекватності моделі та для її подальшого опису та аналізу:

a) Відрізок - якщо наша модель представлена у вигляді $\beta 0 + \beta 1\ x + \varepsilon$, то $\beta 0$ є точкою перетину прямої з віссю координат, або перерізом.
b) R-квадрат - коефіцієнт детермінації показує, наскільки близьким є зв'язок між коефіцієнтами регресії і залежною змінною, відношення пояснених сум квадратів збурень до непояснених. Чим ближче до 1, тим більш виражена залежність.
c) Коригований R-квадрат – проблема з R2 полягає в тому, що він зростає з будь-якою кількістю факторів, тому високе значення цього коефіцієнта може ввести в оману, якщо в моделі є багато факторів. Щоб вилучити цю властивість з коефіцієнта кореляції, був винайдений скоригований коефіцієнт детермінації.
d) F-статистика - використовується для оцінки значущості регресійної моделі в цілому, являє собою відношення чіткої дисперсії до неясної. Якщо модель лінійної регресії побудована успішно, вона пояснює значну частину дисперсії, залишаючи невелику частину в знаменнику. Чим вище значення параметра, тим краще.
e) t-значення є критерієм, заснованим на розподілі t Стьюдента. Значення параметра в лінійній регресії вказує на важливість фактора, вважається, що при $t > 2$ фактор є значущим для моделі.

f) p-value — це ймовірність нульової гіпотези, яка стверджує, що незалежні змінні не пояснюють динаміку залежної змінної. Якщо значення p-значення нижче t.

2.2.3.2. Дерева рішень

Дерева рішень є одним з найбільш ефективних інструментів інтелектуального аналізу даних, які дозволяють вирішувати завдання класифікації і регресії.

Метод дерева рішень - це графічний метод автоматичного аналізу величезних масивів даних. Мета всього процесу побудови дерева прийняття рішень - створити модель, по якій можна було б класифікувати випадки і вирішувати, які значення може приймати цільова функція, маючи на вході кілька змінних. У методиці використовується ієрархічна структурна схема, що складається з елементів двох типів - вузли (node) і листки (leaf) [9].

Процес побудови дерева рішень полягає в послідовному, рекурсивному розбитті навчальної множини на підмножини з застосуванням вирішальних правил в вузлах. Процес розбиття триває до тих пір, поки всі вузли в кінці всіх гілок не будуть листям. Оголошення вузла листом може статися природним чином (коли він буде містити єдиний об'єкт, або об'єкти тільки одного класу), або після досягнення деякого умови зупинки, що задається користувачем (наприклад, мінімально допустиму кількість прикладів у вузлі або максимальна глибина дерева).

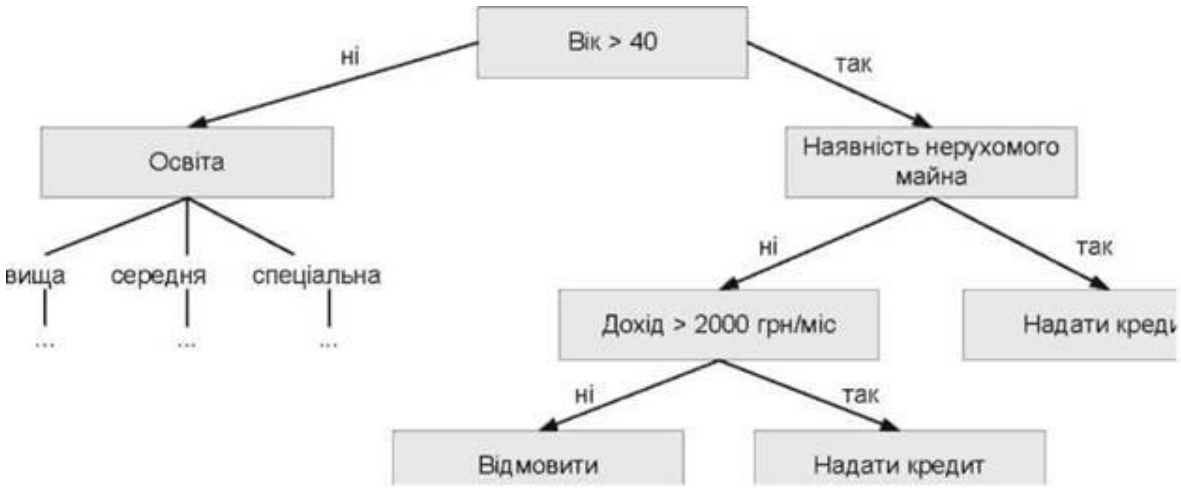


Рис. 10 – Приклад дерева [9].

На схемі верхнє положення займає кінцева мета розв'язання проблеми. Для кожного дерева рішень будується матриця. Зазвичай вводяться коефіцієнти взаємної корисності рішень, вони показують вплив ступеня важливості одних рішень на інші.

Random Forest («випадковий ліс») – алгоритм, що для отриманих даних він створює безліч дерев прийняття рішень і потім усереднює результат їх прогнозів. Важливим моментом тут є елемент випадковості у створенні кожного дерева. Алгоритм побудови дерева прийняття рішень дуже швидкий. І тому нам не складно зробити стільки дерев, скільки буде потрібно.

Алгоритм random forest має по-суті лише один параметр: розмір випадкової підмножини обраного на кожному кроці побудови дерева. Хоча цей алгоритм є доволі простим, він дає дуже хороші результати в реальних задачах, тому «випадковий ліс» є одним з часто використовуваних алгоритмів data mining.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Розпізнавання пухлини за допомогою простої нейронної мережі

Першим етапом є розпізнавання знімків ЕКГ молочної залози жінок. Для цього було сформовано та відсортовано близько 150 реальних знімків злоякісної та доброякісної пухлини. Оскільки для непотужного ШІ така кількість знімків не є достатньою, то зробимо деякі перетворення:

а) Оскільки маємо два види (зверху та збоку), то можна розділити їх як два окремих знімки

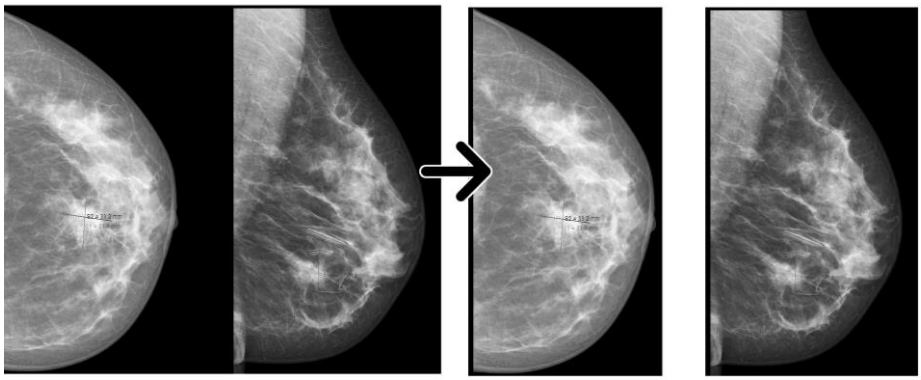


Рис. 11 – Розділення зображень.

б) Через те, що програма не розуміє в якому положенні знаходиться зображення, можна перегортати його різними сторонами, та відзеркалювати

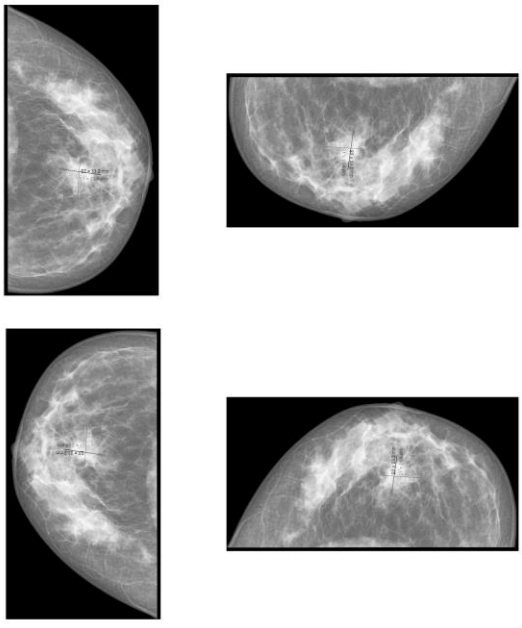


Рис. 12 – Модифікація одного і того ж зображення.

в) І останнє, оскільки знімки бувають як в негативі, та і в позитиві, то можна робити реверс негативу для збільшення різноманітності.

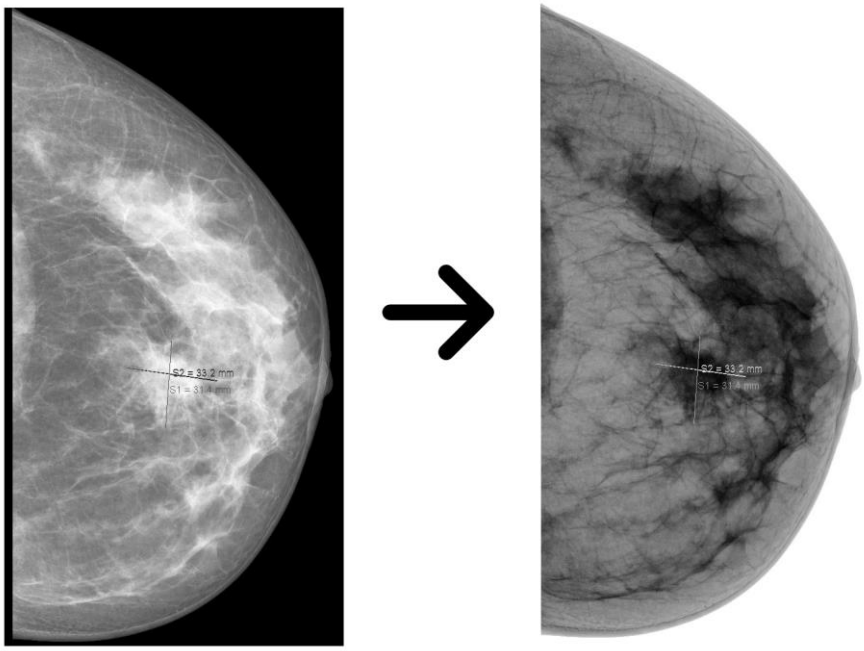


Рис. 13 – Реверс кольорів зображення.

Отже після всіх перетворювань база зображень налікує приблизно 2000 картинок.

Наступним кроком при застосуванні звичайного штучного інтелекту є зменьшення розмірності збражень. Оскільки кожне зображення має розмірність 1000x1000 пікселів всередньому, то кількість вхідних нейроннів складатиме 1000000, що є забагато для такого інтелекту. Зменшуємо розмірність методом пережимання картинки (що зменшить точність прогнозування).

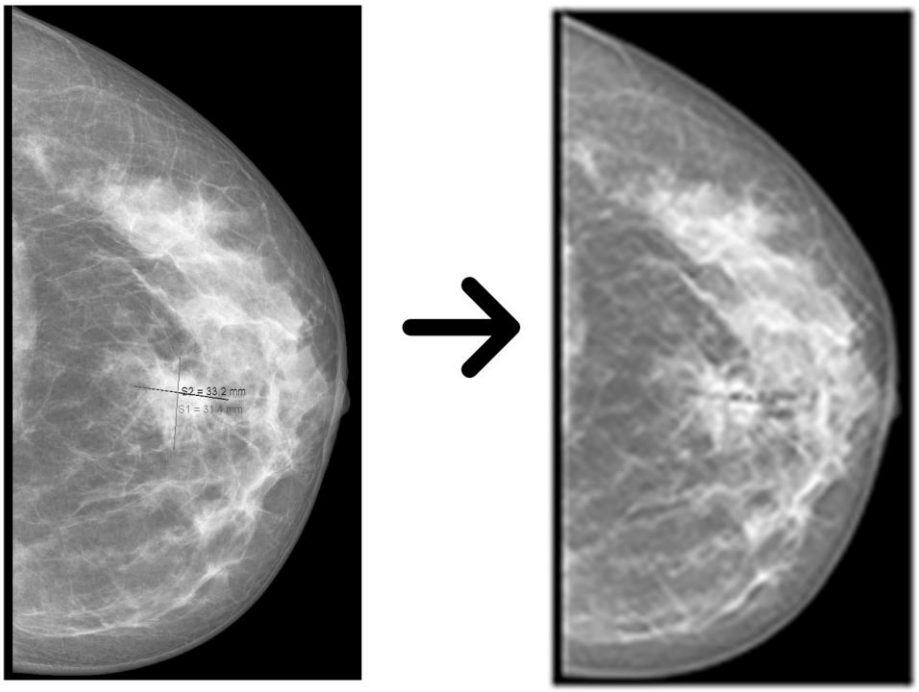


Рис. 14 – Зменьшення розмірності картинки.

Тепер кожна картинка має фіксований розмір 86x86 пікселів, що є біль приємним для розрахунків.

Схематичний вигляд моделі штучного інтелекту вказаний нижче, також початкові данні для налаштування нейронної мережі.

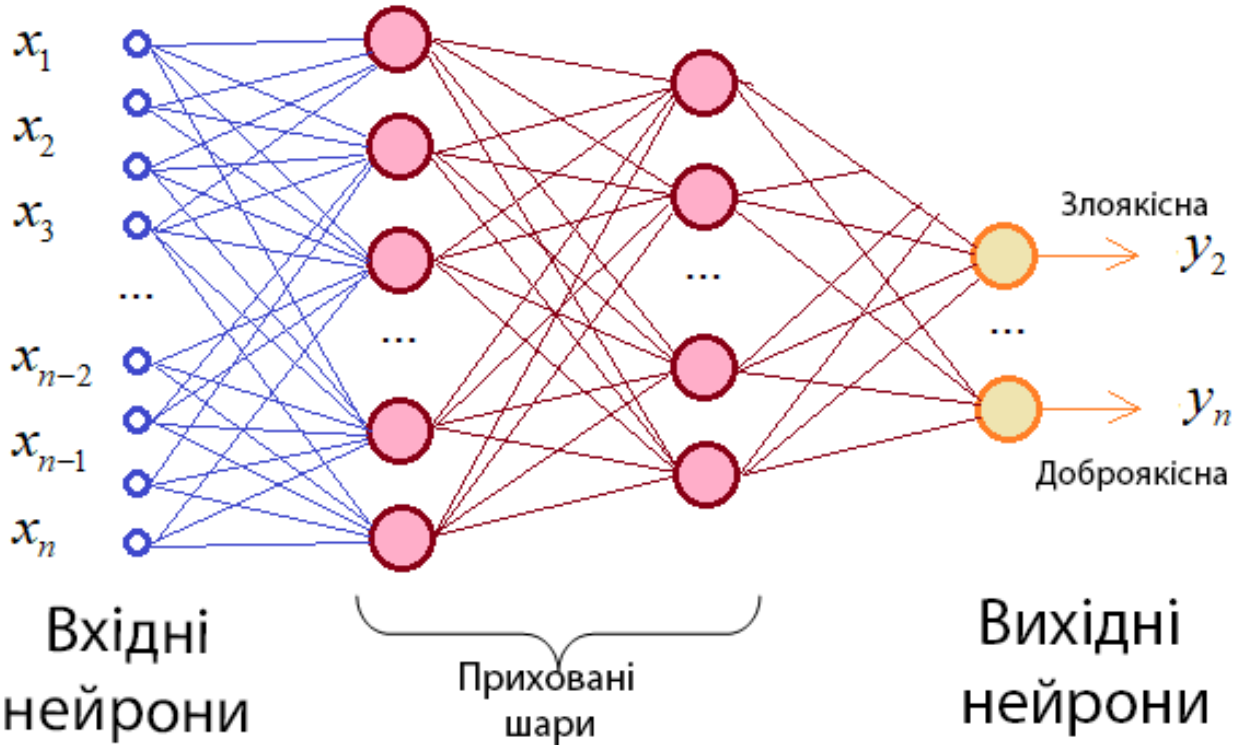


Рис. 15 – Схематичний вигляд нейронної мережі.

NN nn = new NN(0.01f, 7396, 256, 128, 10, 2);

,де 0,01 – крок або швидкість навчання ШІ

7396 – кількість вхідних нейронів (86x86 пікселів)

256, 128, 10 – приховані нейронні шари

2 – кількість вихідних нейронів (злоякісна або доброякісна)

Маючи на руках всі потрібні дані, можна запускати програму штучного інтелекту.

Нижче представлені скріншоти роботи. Для навчання я взяв 30 епох, що символізують еволюційні етапи. В кожному з цих етапів представлено по 100 єкземплярів, які беруться із загальної вибірки оброблених знімків.

```
Epoch: 1; Correct: 3; Error: 3022
Epoch: 2; Correct: 8; Error: 3020
Epoch: 3; Correct: 14; Error: 3015
Epoch: 4; Correct: 10; Error: 3011
Epoch: 5; Correct: 8; Error: 3009
Epoch: 6; Correct: 11; Error: 3004
Epoch: 7; Correct: 9; Error: 3002
Epoch: 8; Correct: 6; Error: 2998
Epoch: 9; Correct: 12; Error: 2993
Epoch: 10; Correct: 18; Error: 2992
Epoch: 11; Correct: 21; Error: 2988
Epoch: 12; Correct: 29; Error: 2987
Epoch: 13; Correct: 33; Error: 2984
Epoch: 14; Correct: 42; Error: 2982
Epoch: 15; Correct: 51; Error: 2981
Epoch: 16; Correct: 60; Error: 2979
Epoch: 17; Correct: 57; Error: 2976
Epoch: 18; Correct: 66; Error: 2973
Epoch: 19; Correct: 62; Error: 2970
Epoch: 20; Correct: 61; Error: 2968
Epoch: 21; Correct: 56; Error: 2967
Epoch: 22; Correct: 56; Error: 2963
Epoch: 23; Correct: 56; Error: 2961
Epoch: 24; Correct: 62; Error: 2957
Epoch: 25; Correct: 66; Error: 2956
Epoch: 26; Correct: 63; Error: 2954
Epoch: 27; Correct: 70; Error: 2949
Epoch: 28; Correct: 78; Error: 2945
Epoch: 29; Correct: 85; Error: 2941
Epoch: 30; Correct: 88; Error: 2939
```

Рис. 16 – Результати навчання простої нейронної мережі.

Прогнавши через нейронну мережу 3000 картинок, отримуємо результат вказаний нижче. Можна побачити, що з пачки рисунків, яки було використано в якості тестувальних, нейронна мережа розпізнає всередньому 50%.

```
Batch: 100; Correct: 45
```

```
Batch: 100; Correct: 48
```

```
Batch: 100; Correct: 54
```

На другому етапі спробуємо підвищити точність кроку та кількість прихованих нейронів. Вигляд вхідних даних:

NN nn = new NN(0.001f, 7396, 256, 128, 10, 2);

,де 0,001 – крок або швидкість навчання ШІ

7396 – кількість вхідних нейронів (86x86 пікселів)

512, 128, 64 – приховані нейронні шари

2 – кількість вихідних нейронів (злоякісна або доброякісна)

Повторюємо процедуру з 30 епохами по 100 рисунків. І як бачимо на рисунку ** точність розпізнавання рисунків виросла, але не набагато. В пункті 3.4 цієї роботи будуть приведені порівння різних методів.

```
Epoch: 1; Correct: 11; Error: 3666
Epoch: 2; Correct: 2; Error: 3665
Epoch: 3; Correct: 16; Error: 3664
Epoch: 4; Correct: 7; Error: 3663
Epoch: 5; Correct: 0; Error: 3658
Epoch: 6; Correct: 0; Error: 3654
Epoch: 7; Correct: 0; Error: 3653
Epoch: 8; Correct: 3; Error: 3652
Epoch: 9; Correct: 0; Error: 3648
Epoch: 10; Correct: 5; Error: 3647
Epoch: 11; Correct: 5; Error: 3646
Epoch: 12; Correct: 4; Error: 3645
Epoch: 13; Correct: 23; Error: 3640
Epoch: 14; Correct: 27; Error: 3637
Epoch: 15; Correct: 40; Error: 3632
Epoch: 16; Correct: 35; Error: 3628
Epoch: 17; Correct: 41; Error: 3624
Epoch: 18; Correct: 43; Error: 3623
Epoch: 19; Correct: 37; Error: 3619
Epoch: 20; Correct: 42; Error: 3615
Epoch: 21; Correct: 46; Error: 3612
Epoch: 22; Correct: 46; Error: 3611
Epoch: 23; Correct: 43; Error: 3607
Epoch: 24; Correct: 58; Error: 3604
Epoch: 25; Correct: 51; Error: 3599
Epoch: 26; Correct: 70; Error: 3595
Epoch: 27; Correct: 73; Error: 3590
Epoch: 28; Correct: 71; Error: 3586
Epoch: 29; Correct: 89; Error: 3581
Epoch: 30; Correct: 80; Error: 3576
```

Рис. 17 – Результати навчання простої нейронної мережі.

Прогнавши через нейронну мережу 3000 картинок, отримуємо результат вказаний нижче. Можна побачити, що з пачки рисунків, яки було використано в якості тестувальних, нейронна мережа розпізнає всередньому 58%.

```
Batch: 100; Correct: 69
```

```
Batch: 100; Correct: 47
```

```
Batch: 100; Correct: 67
```

3.2. Розпізнавання пухлини за допомогою згорточної нейронної мережі.

Оскількі згорточні нейронні мережі є більш точними впорівнянні зі звичайними нейронними мережами, то можна відкинути декілька допущень прийнятих в попередньому етапі. Тепер 150 оригінальних зображень перетворимо за правилами а), б), в) з пункту 3.1. Отже отримуємо 2000 зображень кожне з яких має розмірність 800x800 пікселів.

Відмова від погіршення якості для зменьшення кількості вхідних пікселів несе за собою покращення точності розпізнавання. Наступним кроком в поліпшенні якості прогнозування є використання 3-х наступних правил:

а).Локальне сприйняття.

Тепер замість всієї картинки на нейрони подаються лише окремі ділянки. В моєму випадку, я розділяю картинку 800x800 пікселів на 10 зон, які мають розмірність 80x80

б). Роздільні ваги.

Оскільки основною задачею згорточнох нейронної мережі є використання згорточних фільтрів, то для даного завдання будуть використовуватись згортки розміром 7x7. Що дозволить значно зменшувати розміри картинки при кожному проході через згортку.

в). Субдіскретизація.

Останнім кроком буде зменшення розміру картинки , для цього я обираю вибір максимального значення для пікселів з кроком 2x2.

Для наступних кроків використовується попередня модель нейронної мережі, але модифікована під вхідні нейрони:

NN nn = new NN(0.01f,7,5, 2 289, 256, 128, 10, 2);

,де 0,01 – крок або швидкість навчання ШІ

7,5 – розміри згорточних фільтрів (на першому та другому кроках згортки)

2 – розмір фільтра зменьшення

289– кількість вхідних нейронів (17x17 пікселів)

512, 128, 64 – приховані нейронні шари

2 – кількість вихідних нейронів (злоякісна або доброякісна)

Для навчання я взяв 30 епох, що символізують еволюційні етапи. В кожному з цих етапів представлено по 100 єкземплярів, які беруться із загальної вибірки оброблених знімків.

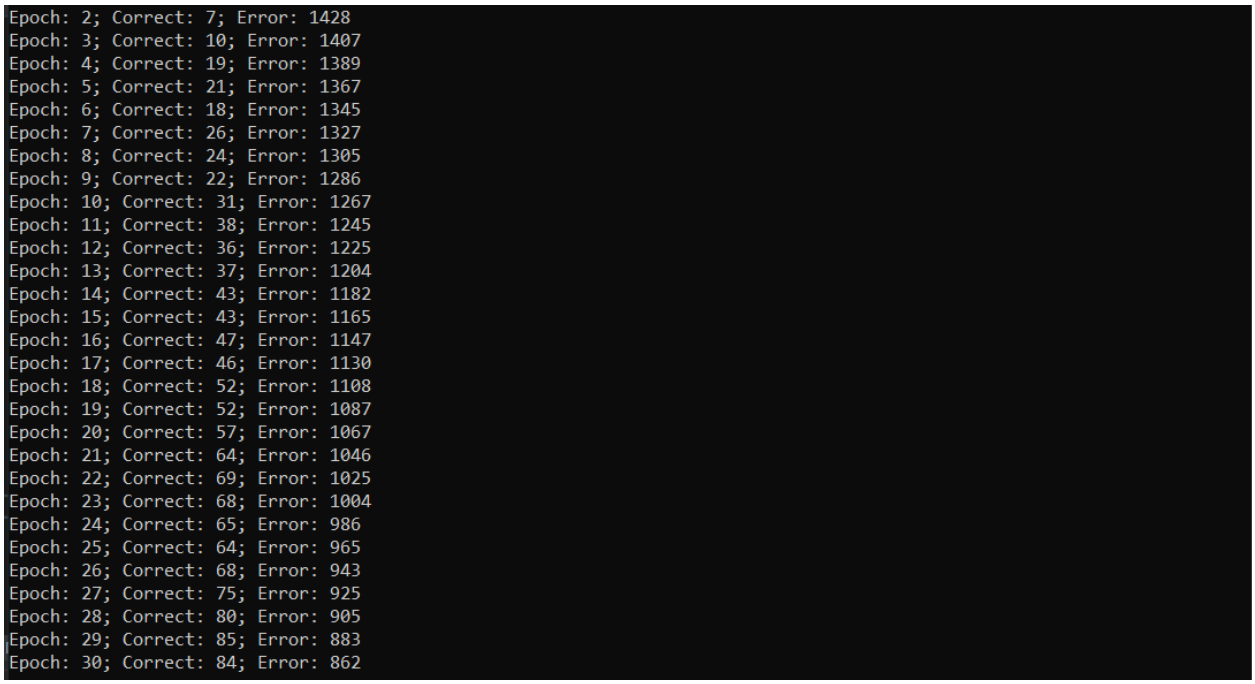


Рис. 18 – Результати навчання згорточної нейронної мережі.

Прогнавши через нейронну мережу 3000 картинок, отримуємо результат вказаний нижче. Можна побачити, що з пачки рисунків, яки було використано в якості тестувальних, нейронна мережа розпізнає всередньому 77%.

```
Batch: 100; Correct: 67
```

```
Batch: 100; Correct: 79
```

```
Batch: 100; Correct: 82
```

Якщо трохи підвищити точність для градієнтрого спуску і кількість прихованих нейронів, то отримаємо наступні результати.

NN nn = new NN(0.001f, 7396, 256, 128, 64, 2);

,де 0,001 – крок або швидкість навчання ШІ

7396 – кількість вхідних нейронів (86x86 пікселів)

256, 128, 64 – приховані нейронні шари

2 – кількість вихідних нейронів (злоякісна або доброякісна)

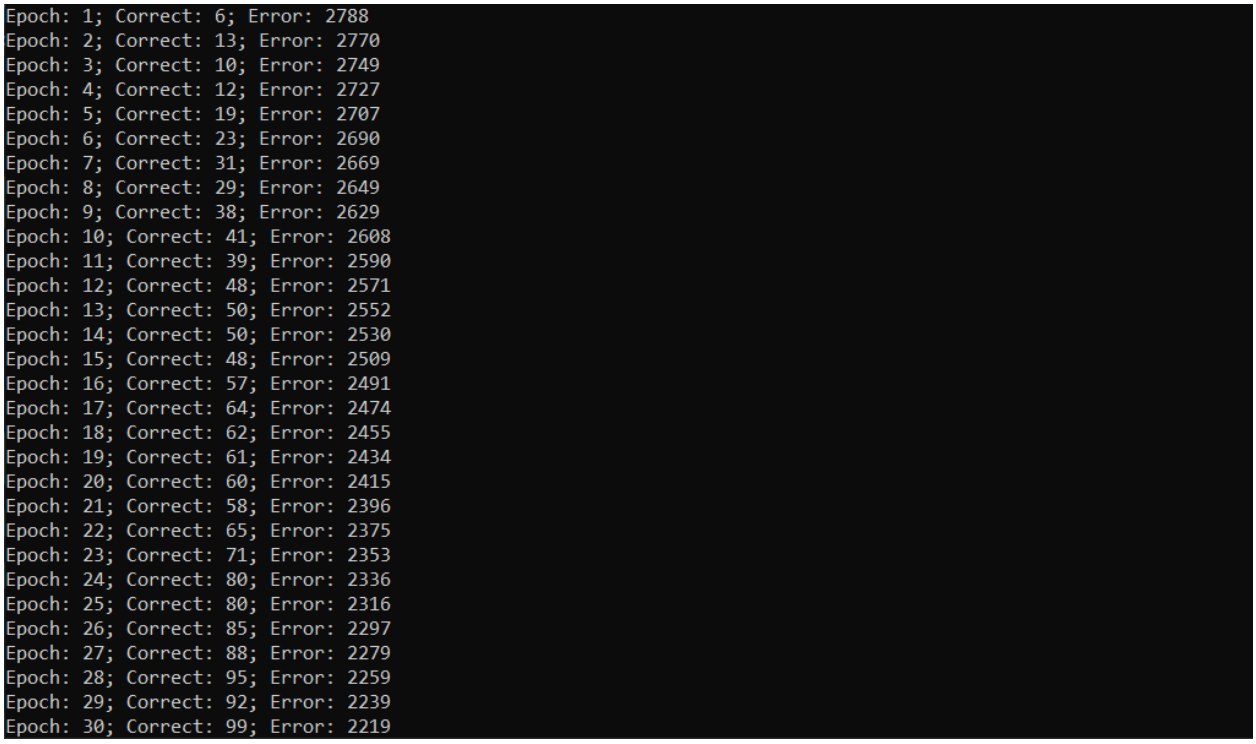
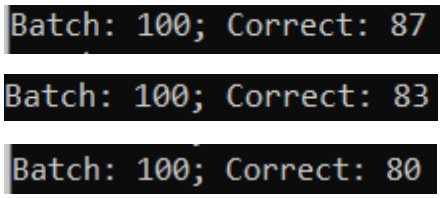


Рис. 19 – Результати навчання згорточної нейронної мережі.

Прогнавши через нейронну мережу 3000 картинок, отримуємо результат вказаний нижче. Можна побачити, що з пачки рисунків, яки було використано в якості тестувальних, Середня точність виросла на 7-10% і складає 83%.



В пункті 3.4 цієї роботи будуть приведені порівння різних методів.

3.3. Розпізнавання пухлини за допомогою регресійного методу та дерева рішень

Дані для статистичного обчислення було взято з відкритих медичних резурсів. Характеристики обчислюються на основі оцифрованого зображення аспірату тонкої голки (FNA) грудної залози. Вони описують характеристики ядер клітин, присутніх на зображенні.

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	tex
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

Рис. 20. – Табличний вигляд даних представлених у датасеті.

Інформація про атрибути:

1) Ідентифікаційний номер

2) Діагноз (M = злоякісний, B = доброякісний)

а) радіус (середнє відстань від центру до точок по периметру)

б) текстура (стандартне відхилення значень шкали сірого)

в) периметр

г) площа

е) гладкість (локальна зміна довжини радіусу)

е) компактність (периметр^2 / площа - 1,0)

g) увігнутість (виразність увігнутих ділянок контуру)

h) увігнуті точки (кількість увігнутих ділянок контуру)

і) симетрія

j) фрактальна розмірність ("наближення берегової лінії" - 1)

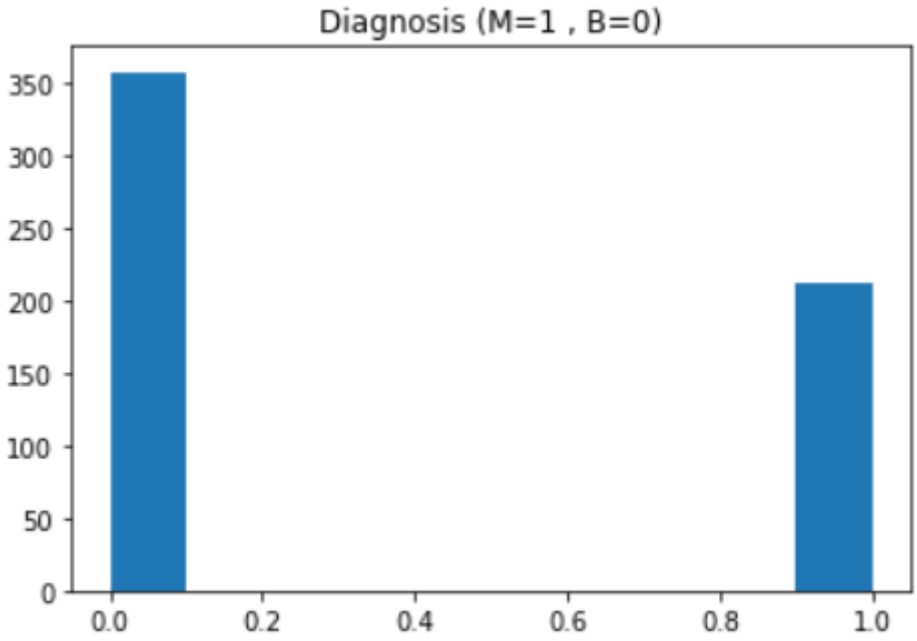


Рис. 21. – Співвідношення злоякісної і доброякісної пухлини.

Розділив дані на тестові і тренувальні в пропорції 30% на 70% відповідно, можна обрати найбільш впливові параметри. Очевидно до них будуть входити таки колонки:

- radius - середнє відстань від центру до точок по периметру
- perimeter
- area
- compactness - периметр^2 / площа - 1,0
- concave points - кількість увігнутих ділянок контуру

Виявивши основні показники впливу можна приступати до аналізу даних. Першим методом буде логістична регресія, яка має наступні показники передбачення:

Accuracy : 87.940%

Cross-Validation Score : 90.000%

Cross-Validation Score : 88.750%

Cross-Validation Score : 88.333%

Cross-Validation Score : 88.718%

Cross-Validation Score : 87.684%

Такі саме умови використовуємо і для дерева прийняття рішень:

Accuracy : 85.000%

Cross-Validation Score : 87.500%

Cross-Validation Score : 87.500%

Cross-Validation Score : 85.000%

Cross-Validation Score : 84.636%

Cross-Validation Score : 84.924%

Ще одним різновидом дерева рішень є ліс рішень, який відрізняється більшою точністю через використання не одного дерева, а декількох, результати такого «лісу» представлені нижче:

Accuracy : 94.724%

Cross-Validation Score : 93.750%

Cross-Validation Score : 92.500%

Cross-Validation Score : 91.250%

Cross-Validation Score : 90.906%

Cross-Validation Score : 90.953%

Детальне порівняння приводиться в пункті 3.4 даної роботи.

3.4. Порівняння методів дослідження

Для порівняння моделей було зроблено декілька ітерацій при навчанні та перевірці даних. Так для нейронних мереж зроблено 10 замірювань по 30 епох, і по 5 естових запусків для табличного аналізу. Так була сформована таблиця, де найнижча строка визначає середню точність кожного з методів.

Таблиця 2. – Порівняння результатів різних методів дослідження.

	Проста нейронна мережа		Згорточна нейронна мережа		Штучний інтелект (TensorFlow)	Табличний аналіз		
	точність кроку 0.01	точність кроку 0.001	точність кроку 0.01	точність кроку 0.001		Регресійний	Дерево	Ліс
Епохи	0	1	2	0	8			
	0	0	3	9	10			
	1	1	4	13	12			
	5	10	3	16	9			
	2	10	7	16	6			
	4	18	11	23	8			
	14	28	9	23	13			
	24	31	19	33	18			
	28	32	17	32	17			
	29	32	22	34	21			
	27	34	23	38	26			
	32	33	30	38	34			
	35	39	37	46	39			
	32	41	41	46	47			
	34	50	42	52	44			
	35	48	42	52	52			
	33	48	40	55	55			
	40	49	39	61	61			
	46	53	39	64	65			
	48	63	43	67	63			
	49	64	43	70	67			
	47	62	50	74	72			
	54	62	50	80	76			
	51	62	56	78	83			
	51	70	66	84	86			
	50	75	71	84	87			
	53	75	72	83	88			
	62	74	81	91	85			
	72	74	81	96	89			
	82	79	84	95	93			
	81	88	87	93	97			
Середнє значення точності прогнозу (%)	50	58	77	83	95	88	85	94

Оскільки табличні значення та голі цифри не дуже підходять для аналізу людиною і вимагать більше часу для розуміння, тому перенесу їх у вигляді графіків для зручного аналізу.

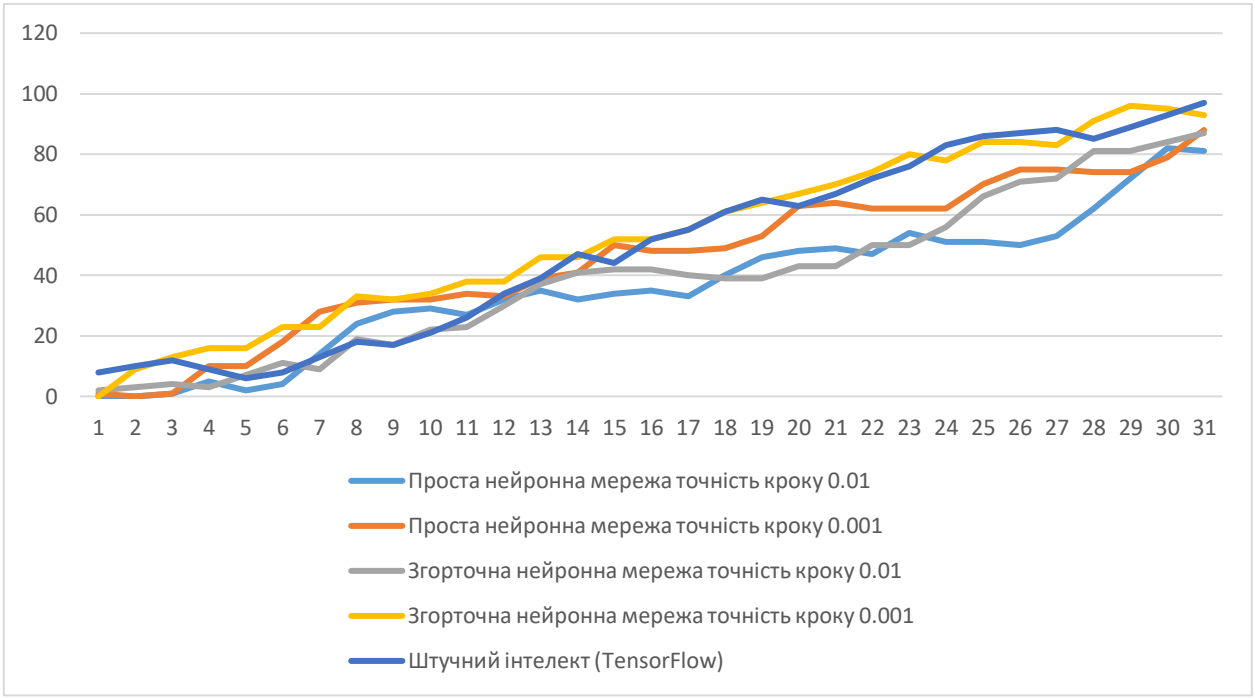


Рис. 22 – Графік середньої точності розрахунків різних моделей.

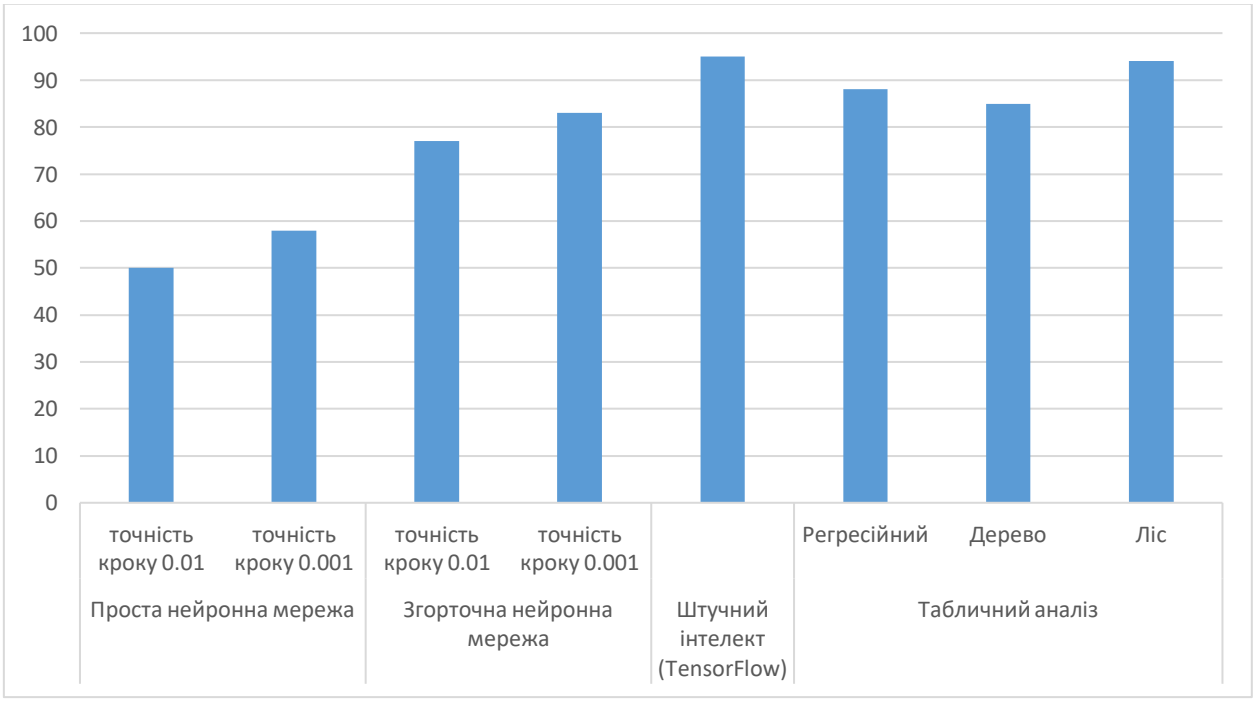


Рис. 23. – Гістограма середньої точності прогнозування даних.

ВИСНОВКИ

В даній роботі було проведено аналіз зображень знімків ЕКГ молочної залози з метою виявлення наявності ракової пухлини. Для цього було побудовано дві моделі нейронної мережі, а саме проста нейронна мережа і згорточна. Також додано аналіз табличних значень, таких як регресійний аналіз, дерево та ліс рішень. Для проведення оцінки спроможності побудованих моделей проводити відповідний аналіз зображень було використано існуючу згорточну модель, яка вбудована в бібліотеки мови програмування Python. Ці моделі були навчені на зображеннях, які взяті з реальних знімків ЕКГ молочної залози, а табличні значення взяті з реальних досліджень медичних закладів.

Після навчання та тестування обраних моделей, було зроблено порівняння точності проведення розрахунків. Показано, що розроблені у даній роботі моделі нейронних мереж ілюструють досить великий збіг отриманих результатів з результатами тестування на існуючій розробці компанії google та їх TensorFlow. Похибка у отриманих результатах не перевищує 10%, що є прийнятним для числового аналізу. Серед переваг побудованих моделей слід відзначити використання простих елементів таких як: перциптрон та згорточні матриці. Розроблені в роботі алгоритми можуть бути вдосконалені з метою зменшення похибки при прогнозуванні. Побудовані моделі можуть використовуватися для аналізу ЕКГ зображень з виявлення ракових пухлин молочної залози.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Злоякісна пухлина [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Злоякісна_пухлина
2. Скрипниченко Д. Ф. Хірургія: Підручник. — 4-е вид., випр. і доповн. — Київ: Вища шк., 1992. — 581 с.:іл. ISBN 5-11-003837-6(с.350)
3. Churina Ye.G., Novitsky V.V., Urazova O.I. - Immunosuppression factors under various pathologies.
4. Семерников, Валерий Андреевич - Локализация и свойства антигенов Bac.Megaterium H, перекрестно реагирующих с антигенами опухолей.
5. Андрей Чантурия, Франтишек Висмонт - Патологическая физиология, с. 335.
6. Романовский Ю.М. и др. - Математическая биофизика, 1984.
7. Класифікація і кластерізація [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.intuit.ru/studies/courses/6/6/lecture/166?page=4
8. Дрейпер Н., Сміт Г. Прикладний регресійний аналіз. М .: Видавничий дім «Вільямс». 2007.
9. Дерева рішень [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://pidruchniki.com/12280805/informatika/dekompozitsiya_tsiley_informatsiynoyi_tehnologiyi

ДОДАТОК А

Код програми для простої нейронної мережі

```
using System;
using UnityEngine;
using System.Drawing;
using System.IO;
using UnityEngine.UI;

using Color = System.Drawing.Color;
using Random = System.Random;

public class MainScript : MonoBehaviour
{
    NeuralNetwork nn; //NeuralNetwork object

    [SerializeField] private Text numbersWeightLeft;
    [SerializeField] private Text numbersWeightRight;

    [SerializeField] private Text number;

    [SerializeField] private int sample = 3000; //Number of Images for treaning
    [SerializeField] private int sampleTest = 200; //Number of Images for test
    [SerializeField] private int epochs = 100; //Number of Epochs

    private const int numbersCount = 10; //number of output neurons
    private int imageSize = 28; // the size of the picture is 28 because it is square

    private void Start()
    {
        DrawScript.Inputs += CheckingNN;
        //nn = new NeuralNetwork();
        Digits();
        //Test();
    }

    private void TestingNN()
    {
        string[] files;
        string path = @"F:/Download/test";
        files = Directory.GetFiles(path, "*.png");
        int correctResult = 0;
        for (int imgIndex = 0; imgIndex < sampleTest; imgIndex++)
        {
            int targetNumber = GetNumber(imgIndex, files, 27);
            double[] outputs = nn.FeedForward(GetImage(imgIndex, files));
            int estimatedNum = WhichNumber(outputs);
            if (targetNumber == estimatedNum)
            {
                correctResult++;
            }
        }
```

```
        Debug.Log(correctResult);
    }

    private int GetNumber(int index, string[] files, int size)
    {
        return Convert.ToInt32(files[index].Substring(size, 1));
    }

    private void Digits()
    {
        Random rnd = new Random();

        nn = new NeuralNetwork(0.001f, 7684,256,64, 10); //Creating a neural network object

        //Loading files into an array
        string[] files;
        string path = @"F:/Download/train";
        files = Directory.GetFiles(path, "*.png");


        for (int i = 1; i < epochs; i++)
        {
            int correctResult = 0;
            double error = 0;
            int batchSize = 100; //Number of packs of pictures
            for (int j = 0; j < batchSize; j++)
            {
                int imgIndex = (int)(rnd.NextDouble() * sample); //Random Image

                //Get target number
                double[] targets = new double[numbersCount] { 0,0,0,0,0,0,0,0,0,0};
                int targetNumber = GetNumber(imgIndex, files, 28);
                targets[targetNumber] = 1;

                //Run the picture through the neural network and get an array of 10 numbers at the
output
                double[] outputs = nn.FeedForward(GetImage(imgIndex, files));

                //Determine which number has the highest value and compare with the target number
                int estimatedNum = WhichNumber(outputs);
                if (targetNumber == estimatedNum)
                {
                    correctResult++;
                }

                // Calculate error (Target[i] - Output[i])^2
                for (int k = 0; k < numbersCount; k++)
                {
                    error += (targets[k] - outputs[k]) * (targets[k] - outputs[k]);
                }

                //Backpropagate with an error
```

```
            nn.BackPropagation(targets);
        }
        Debug.Log($"Epoch: {i}; Correct: {correctResult}; Error: {error}");
    }
    nn.SaveGame();
}

public void CheckingNN(double[] inputs)
{
    double[] outputs = nn.FeedForward(inputs);
    //Debug.Log($"{Math.Round(outputs[0], 2)} {Math.Round(outputs[1], 2)}
{Math.Round(outputs[2], 2)} {Math.Round(outputs[3], 2)} {Math.Round(outputs[4], 2)}
{Math.Round(outputs[5], 2)} {Math.Round(outputs[6], 2)} {Math.Round(outputs[7], 2)}
{Math.Round(outputs[8], 2)} {Math.Round(outputs[9], 2)}");
    int num = WhichNumber(outputs);
    numbersWeightLeft.text = string.Join("",
     Math.Round(outputs[0], 3)+ "\n" +
     Math.Round(outputs[1], 3) + "\n" +
     Math.Round(outputs[2], 3) + "\n" +
     Math.Round(outputs[3], 3) + "\n" +
     Math.Round(outputs[4], 3));
    numbersWeightRight.text = string.Join("",
     Math.Round(outputs[5], 3) + "\n" +
     Math.Round(outputs[6], 3) + "\n" +
     Math.Round(outputs[7], 3) + "\n" +
     Math.Round(outputs[8], 3) + "\n" +
     Math.Round(outputs[9], 3));
    number.text = string.Join("", num);
    //Debug.Log(num);
}

public int WhichNumber(double[] outputs)
{
    int maxDigit = 0;
    double maxDigitWeight = -1f;
    for (int k = 0; k < numbersCount; k++)
    {
        if (outputs[k] > maxDigitWeight)
        {
            maxDigitWeight = outputs[k];
            maxDigit = k;
        }
    }
    return maxDigit;
}

private double[] GetImage(int index, string[] files)
{
    Color clr;
    double[] inputs = new double[imageSize* imageSize];

    Bitmap btm = new Bitmap(files[index]);
```

```
        for (int i = 0; i < btm.Width; i++)
        {
            for (int j = 0; j < btm.Height; j++)
            {
                clr = btm.GetPixel(j, i);
                inputs[j + i * imageSize] = (clr.ToArgb() & 0xff) / 255.0f;
            }
        }
        return inputs;
    }
}

using System.Collections;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections.Generic;
using UnityEngine;
using Random = System.Random;

public class NeuralNetwork
{
    private double learningRate;
    private Layer[] layers; // Number of layers
    private Random rnd = new Random();

    //Constructor for the first training
    public NeuralNetwork(double learningRate, params int[] sizes)
    {
        this.learningRate = learningRate;
        layers = new Layer[sizes.Length];
        for (int i = 0; i < sizes.Length; i++)
        {
            int nextSize = 0;
            if (i < sizes.Length - 1) nextSize = sizes[i + 1];
            layers[i] = new Layer(sizes[i], nextSize);
            for (int j = 0; j < sizes[i]; j++)
            {
                layers[i].biases[j] = rnd.Next(-10000, 10000) / 10000f;
                for (int k = 0; k < nextSize; k++)
                {
                    layers[i].weights[j, k] = rnd.Next(-10000, 10000) / 10000f;
                }
            }
        }
    }

    //Constructor for a trained neural network
    public NeuralNetwork()
    {
        LoadGame();
```

```
}

//Getting output values
public double[] FeedForward(double[] inputs)
{

   System.Array.Copy(inputs, 0, layers[0].neurons, 0, inputs.Length);
   for (int i = 1; i < layers.Length; i++)
   {
      Layer l = layers[i - 1];
      Layer l1 = layers[i];
      for (int j = 0; j < l1.size; j++)
      {
         l1.neurons[j] = 0;

         for (int k = 0; k < l.size; k++)
         {
            l1.neurons[j] += l.neurons[k] * l.weights[k, j];
         }
         l1.neurons[j] += l1.biases[j];
         l1.neurons[j] = Sigmoid(l1.neurons[j]);

      }
   }

   return layers[layers.Length - 1].neurons;
}


private double Sigmoid(double x)
{
   return 1 / (1 + Mathf.Exp(-(float)x));
}

private double Dsigmoid(double x)
{
   return x * (1 - x);
}

//Backpropagation method to increase accuracy and reduce guessing error
public void BackPropagation(double[] targets)
{
   double[] errors = new double[layers[layers.Length - 1].size];
   for (int i = 0; i < layers[layers.Length - 1].size; i++)
   {
      errors[i] = (targets[i]-layers[layers.Length - 1].neurons[i]);
   }
   for (int k = layers.Length - 2; k >= 0; k--)
   {
      Layer l = layers[k];
      Layer l1 = layers[k + 1];
```

```
            double[] errorsNext = new double[l.size];
            double[] gradients = new double[l1.size];
            for (int i = 0; i < l1.size; i++)
            {
                gradients[i] = errors[i] * Dsigmoid(layers[k + 1].neurons[i]);
                gradients[i] *= learningRate;
            }
            double[,] deltas = new double[l1.size, l.size];
            for (int i = 0; i < l1.size; i++)
            {
                for (int j = 0; j < l.size; j++)
                {
                    deltas[i, j] = gradients[i] * l.neurons[j];
                }
            }
            for (int i = 0; i < l.size; i++)
            {
                errorsNext[i] = 0;
                for (int j = 0; j < l1.size; j++)
                {
                    errorsNext[i] += l.weights[i, j] * errors[j];
                }
            }
            errors = new double[l.size];
            System.Array.Copy(errorsNext, 0, errors, 0, l.size);
            double[,] weightsNew = new double[l.weights.GetLength(0), l.weights.GetLength(1)];
            for (int i = 0; i < l1.size; i++)
            {
                for (int j = 0; j < l.size; j++)
                {
                    weightsNew[j, i] = l.weights[j, i] + deltas[i, j];
                }
            }
            l.weights = weightsNew;
            for (int i = 0; i < l1.size; i++)
            {
                l1.biases[i] += gradients[i];
            }
        }
    }
}

[Serializable]

public class Layer
{
    public int size;
    public double[] neurons;
    public double[] biases;
    public double[,] weights;

    public Layer(int size, int nextSize)
```

```
    {
        this.size = size;
        neurons = new double[size];
        biases = new double[size];
        weights = new double[size,nextSize];
    }
}
```

ДОДАТОК Б

Код програми для згорточної нейронної мережі

```
using System;
using System.Drawing;
using System.IO;

namespace ConsoleApp2
{
   class Program
   {
      static Neuralnetwork nn;
      private static string[] files;
      private static int imageSize = 28;
      static string pathTrain = @"F:\Repository\NumbersData\train";
      static string pathTest = @"F:\Repository\NumbersData\test";
      static void Main(string[] args)
      {
         int[] sizes = new int[3] { 1, 5, 5 };
         int[] layersSizes = new int[4] { 2500, 512,32,10 };
         NeuralNetwork nn = new NeuralNetwork(0.001f, 5, 784, sizes, layersSizes) ;
         Digits();
         Test();
      }




      private static void Print(double[] inputs)
      {
         for (int i = 0; i < inputs.Length / 28; i++)
         {
            for (int j = 0; j < inputs.Length / 28; j++)
            {
               Console.Write(String.Format("{0,4}", Math.Round(inputs[j + i * 28], 2)));
            }

            Console.WriteLine();
         }
         Console.WriteLine();
      }

      public static void Digits()
      {
         Random rnd = new Random();
         nn = new Neuralnetwork(0.001f, 784, 512,256, 128, 64, 32, 10);
         int sample = 60000;

         files = Directory.GetFiles(pathTrain, "*.png");

         int epochs = 1000;
         for (int i = 1; i < epochs; i++)
```

```
    {
        int right = 0;
        double error = 0;
        int batchSize = 50;
        for (int j = 0; j < batchSize; j++)
        {
            int imgIndex = (int)(rnd.NextDouble() * sample);
            double[] targets = new double[10];
            int number = GetNumber(imgIndex, pathTrain.Length);
            targets[number] = 1;
            double[] outputs = nn.FeedForward(GetImage(imgIndex));
            int maxDigit = WhichNumber(outputs);

            if (number == maxDigit)
            {
                right++;
            }
            for (int k = 0; k < 10; k++)
            {
                error += (targets[k] - outputs[k]) * (targets[k] - outputs[k]);
            }
            nn.BackPropagation(targets);
        }
        Console.WriteLine($"Epoch: {i}; Correct: {right}; Error: {error}");
    }
    //nn.SaveGame();
}
private static int GetNumber(int index, int rathLength)
{
    int number = Convert.ToInt32(files[index].Substring(rathLength + 11, 1));
    return number;
}
public static void Test()
{
    nn = new Neuralnetwork();
    files = Directory.GetFiles(pathTest, "*.png");
    int sampleTest = 10000;
    int correctResult = 0;
    for (int imgIndex = 0; imgIndex < sampleTest; imgIndex++)
    {
        int targetNumber = GetNumber(imgIndex, pathTest.Length);
        double[] outputs = nn.FeedForward(GetImage(imgIndex));
        int estimatedNum = WhichNumber(outputs);
        if (targetNumber == estimatedNum)
        {
            correctResult++;
        }
    }
    Console.WriteLine($"{(float)correctResult / sampleTest * 100f}%");
}

public static int WhichNumber(double[] outputs)
```

```
        {
            int maxDigit = 0;
            double maxDigitWeight = -1f;
            for (int k = 0; k < 10; k++)
            {
                if (outputs[k] > maxDigitWeight)
                {
                    maxDigitWeight = outputs[k];
                    maxDigit = k;
                }
            }
            return maxDigit;
        }

        private static double[] GetImage(int index)
        {
            Color clr;
            double[] inputs = new double[imageSize * imageSize];

            Bitmap btm = new Bitmap(files[index]);

            for (int i = 0; i < btm.Width; i++)
            {
                for (int j = 0; j < btm.Height; j++)
                {
                    clr = btm.GetPixel(i, j);
                    inputs[j + i * imageSize] = (clr.ToArgb() & 0xff) / 255.0f;
                }
                //Console.WriteLine();
            }
            //Print(inputs);

            return inputs;
        }
    }
}

using System.Collections;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections.Generic;
using Random = System.Random;

namespace ConsoleApp2
{
    class NeuralNetwork
    {
        private double learningRate;
        private Layer[] layers;
        private Cluster[] clusters;
        double[,,] maxi;
```

```
    public NeuralNetwork(double learningRate, int filterSize, int inputSize, int[] sizes, int[]
lsize)
    {
        int clusterSize = sizes.Length;
        int[] inputs = new int[clusterSize];
        clusters = new Cluster[clusterSize];
        for(int i = 0; i < clusterSize; i++)
        {
            inputs[i] = inputSize - (int)Math.Truncate((decimal)sizes[i] / 2) * i * 2;
        }

        for(int i = 0; i < clusterSize; i++)
        {
            int newSize = 0;
            int oldSize = 1;
            if (i < clusterSize - 1) newSize = sizes[i + 1];
            if (i != 0) oldSize = sizes[i - 1];
            clusters[i] = new Cluster(inputs[i], sizes[i] * oldSize, filterSize, sizes[i] * newSize);
            for(int k = 0; k < newSize; k++)
            {
                clusters[i].biases[k] = (new Random()).Next(-20000, 20000) / 10000f;
                for(int x = 0; x < filterSize; x++)
                {
                    for(int y = 0; y < filterSize; y++)
                    {
                        clusters[i].filters[x,y,k] = (new Random()).Next(-20000, 20000) / 10000f;
                    }
                }
            }
        }
        this.learningRate = learningRate;
        layers = new Layer[lsize.Length];
        for (int i = 0; i < lsize.Length; i++)
        {
            int nextSize = 0;
            if (i < lsize.Length - 1) nextSize = lsize[i + 1];
            layers[i] = new Layer(lsize[i], nextSize);
            for (int j = 0; j < lsize[i]; j++)
            {
                layers[i].biases[j] = (new Random()).Next(-10000, 10000) / 10000f;
                for (int k = 0; k < nextSize; k++)
                {
                    layers[i].weights[j, k] = (new Random()).Next(-10000, 10000) / 10000f;
                }
            }
        }

    }
    public NeuralNetwork()
    {
        LoadGame();
```

```
        }
        private void Print(double[,,] inputs, int c)
        {
            //Console.WriteLine(inputs.GetLength(0));
            for (int i = 0; i < inputs.GetLength(0); i++)
            {
                for (int j = 0; j < inputs.GetLength(1); j++)
                {
                    Console.Write(String.Format("{0,4}", Math.Round(inputs[i, j, c], 1)));
                }

                Console.WriteLine();
            }
            Console.WriteLine();
        }

        public double[] ClusteringImage(double[,] inputs)
        {
            for(int i = 0; i < inputs.GetLength(0); i++)
            {
                for (int j = 0; j < inputs.GetLength(1); j++)
                {
                    clusters[0].neurons[i, j, 0] = inputs[j, i];
                }
            }
            for (int i = 1; i < clusters.Length; i++) //Проход по всем слоям
            {
                for(int j = 0; j < clusters[i-1].layerSize; j++)// проход по нейронным слоям слоя
                {
                    for(int f = 0; f < clusters[i-1].filters.GetLength(2); f++)// проход по слоям
фильтров
                    {
                        for (int x = 0; x < clusters[i].inputSize; x++)// проход по ячейкам нейрона
следующего слоя
                        {
                            for(int y = 0; y < clusters[i].inputSize; y++)// проход по ячейкам нейрона
следующего слоя
                            {
                                clusters[i].neurons[x, y, f] = ReLU(Stamping(x, y, i, j, f % 5));
                            }
                        }
                    }
                }
            }

            for (int i = 0; i < clusters[0].neurons.GetLength(2); i++)
            {
                Print(clusters[0].neurons, i);
            }
            for (int i = 0; i < clusters[1].neurons.GetLength(2); i++)
            {
                Print(clusters[1].neurons, i);
```

```
        }
        for (int i = 0; i < clusters[2].neurons.GetLength(2); i++)
        {
            Print(clusters[2].neurons, i);
        }
        return Maximization();
    }

    private double[] Maximization()
    {
        maxi = new double[clusters[2].neurons.GetLength(0)/2, clusters[2].neurons.GetLength(0)
/ 2, clusters[2].neurons.GetLength(2)];
        for(int i = 0; i < maxi.GetLength(2); i++)
        {
            for(int x = 0; x < maxi.GetLength(0); x ++)
            {
                for(int y = 0; y < maxi.GetLength(0); y ++)
                {
                    maxi[x, y, i] = Max(x, y, i);
                }
            }
        }
        for (int i = 0; i < maxi.GetLength(2); i++)
        {
            Print(maxi, i);
        }

        double[] input = new double[maxi.GetLength(0) * maxi.GetLength(1) *
maxi.GetLength(2)];
        for (int i = 0; i < maxi.GetLength(2); i++)
        {
            for (int x = 0; x < maxi.GetLength(0); x ++)
            {
                for (int y = 0; y < maxi.GetLength(0); y ++)
                {
                    input[y + x * maxi.GetLength(0) + i * maxi.GetLength(2)] = maxi[x, y, i];
                }
            }
        }
        return input;
    }

    private double Max(int x, int y, int l)
    {
        double max = -100f;
        for(int i = x; i <= x + 1; i++)
        {
            for(int j = y; j <= y + 1; j++)
            {
                if(max < clusters[2].neurons[i, j, l])
                {
                    max = clusters[2].neurons[i, j, l];
```

```
            }
        }
    }
    return max;
}

public double ReLU(double a)
{
    return a >= 0 ? a : 0f;
}


public double Stamping(int x, int y, int numOfCluster, int mainLayer, int numOfFilters)
{
    int step = (int)Math.Truncate((decimal)clusters[numOfCluster].maskSize / 2);
    int centerX = x + step;
    int centerY = y + step;

    int edgeX = centerX - step;
    int edgeY = centerY - step;

    double sum = 0f;
    for(int k = 0; k < clusters[numOfCluster].maskSize; k++)
    {
        for(int m = 0; m < clusters[numOfCluster].maskSize; m++)
        {
            sum += clusters[numOfCluster - 1].neurons[edgeX + k, edgeY + m, mainLayer] *
clusters[numOfCluster - 1].filters[k, m, numOfFilters] ;
        }
    }
    sum += clusters[numOfCluster - 1].biases[numOfFilters];
    return sum;
}

public double[] FeedForward(double[,] inputs)
{
    double[] input = ClusteringImage(inputs);
    System.Array.Copy(input, 0, layers[0].neurons, 0, input.Length);
    for (int i = 1; i < layers.Length; i++)
    {
        Layer l = layers[i - 1];
        Layer l1 = layers[i];
        for (int j = 0; j < l1.size; j++)
        {
            l1.neurons[j] = 0;

            for (int k = 0; k < l.size; k++)
            {
                l1.neurons[j] += l.neurons[k] * l.weights[k, j];
            }
            l1.neurons[j] += l1.biases[j];
            l1.neurons[j] = Sigmoid(l1.neurons[j]);
```

```
            }
        }

        return layers[layers.Length - 1].neurons;
    }

    private double Sigmoid(double x)
    {
        return 1 / (1 + Math.Exp(-x));
    }

    private double Dsigmoid(double x)
    {
        return x * (1 - x);
    }

    public void BackPropagation(double[] targets)
    {
        double[] errors = new double[layers[layers.Length - 1].size];
        double[,] deltas = new double[layers[layers.Length - 2].size, layers[layers.Length -
1].size];
        for (int i = 0; i < layers[layers.Length - 1].size; i++)
        {
            errors[i] = targets[i] - layers[layers.Length - 1].neurons[i];
        }
        for (int k = layers.Length - 2; k >= 0; k--)
        {
            Layer l = layers[k];
            Layer l1 = layers[k + 1];

            double[] errorsNext = new double[l.size];
            double[] gradients = new double[l1.size];
            for (int i = 0; i < l1.size; i++)
            {
                gradients[i] = errors[i] * Dsigmoid(layers[k + 1].neurons[i]);
                gradients[i] *= learningRate;
            }
            deltas = new double[l1.size, l.size];
            for (int i = 0; i < l1.size; i++)
            {
                for (int j = 0; j < l.size; j++)
                {
                    deltas[i, j] = gradients[i] * l.neurons[j];
                }
            }
            for (int i = 0; i < l.size; i++)
            {
                errorsNext[i] = 0;
                for (int j = 0; j < l1.size; j++)
                {
                    errorsNext[i] += l.weights[i, j] * errors[j];
```

```
            }
        }

        errors = new double[l.size];
        System.Array.Copy(errorsNext, 0, errors, 0, l.size);
        double[,] weightsNew = new double[l.weights.GetLength(0), l.weights.GetLength(1)];
        for (int i = 0; i < l1.size; i++)
        {
            for (int j = 0; j < l.size; j++)
            {
                weightsNew[j, i] = l.weights[j, i] + deltas[i, j];
            }
        }
        l.weights = weightsNew;
        for (int i = 0; i < l1.size; i++)
        {
            l1.biases[i] += gradients[i];
        }
    }
    var error = ReSchaping(errors);
    for(int i = clusters.Length - 2; i >= 0; i--)
    {
        Cluster c = clusters[i];
        Cluster c1 = clusters[i + 1];

        double[,,] nextErrors = new double[clusters[i].neurons.GetLength(1),
clusters[i].neurons.GetLength(1), clusters[i].neurons.GetLength(2)];

        for (int f = 0; f < c.filters.GetLength(2); f++)// проход по слоям фильтров
        {
            for(int x = 0; x < c1.neurons.GetLength(1); x++)
            {
                for (int y = 0; y < c1.neurons.GetLength(1); y++)
                {
                    NewRes(x, y, c, c1);
                }
            }
        }

    }
}

private void NewRes(int x, int y, Cluster c, Cluster c1)
{
    int step = (int)Math.Truncate((decimal)c.filters.GetLength(1) / 2);

    for(int i = x - step; i < c1.neurons.GetLength(1) + step; i++)
    {
        for (int j = y - step; j < c1.neurons.GetLength(1) + step; j++)
        {
            //11;
        }
```

```
    }
}

private double[,,] ReSchaping(double[] errors)
{
    int length = clusters[2].neurons.GetLength(0);
    double[,,] deltas = new double[length, length, clusters[2].layerSize];


    for (int i = 0; i < clusters[2].layerSize; i++)
    {
        for (int x = 0; x < length; x ++)
        {
            for (int y = 0; y < length; y ++)
            {
                deltas[x, y, i] = 0f;
            }
        }
    }
    for (int k = 0; k < maxi.GetLength(2); k++)
    {
        for (int x = 0; x < clusters[2].neurons.GetLength(0); x += 2)
        {
            for (int y = 0; y < clusters[2].neurons.GetLength(0); y += 2)
            {
                deltas[x, y, k] = ReLU(ReMax(x, y, k, errors));
            }
        }
     }
    for (int i = 0; i < clusters[2].layerSize; i++)
    {
        Print(deltas, i);
    }
    return deltas;
}

private double ReMax(int x, int y, int l, double[] errors)
{
    for (int i = x; i <= x + 1; i++)
    {
        for (int j = y; j <= y + 1; j++)
        {
            if (maxi[i/2, j/2, l] == clusters[2].neurons[i, j, l])
            {
                return errors[y + x * clusters[2].neurons.GetLength(0) + l * clusters[2].layerSize];
            }
        }
    }
    return 0;
}

public void SaveGame()
```

```
    {
        BinaryFormatter bf = new BinaryFormatter();
        FileStream file =
File.Create("C:/Users/3030C/AppData/LocalLow/DefaultCompany/NeuralNetwork/SavedData.d
at");
        SaveData[] data = new SaveData[layers.Length];

        for (int i = 0; i < data.Length; i++)
        {
            int nextSize = 0;
            if (i < layers.Length - 1) nextSize = layers[i + 1].size;
            data[i] = new SaveData(layers[i].size, nextSize);
            for (int j = 0; j < layers[i].size; j++)
            {
                data[i].biases[j] = layers[i].biases[j];
                for (int k = 0; k < nextSize; k++)
                {
                    data[i].weights[j, k] = layers[i].weights[j, k];
                }
            }
        }
        bf.Serialize(file, data);
        file.Close();
        Console.WriteLine("Game Saved");
    }

    public void Prints(double[] b)
    {
        for (int i = 0; i < b.Length; i++)
        {
            Console.WriteLine(b[i]);
        }
    }

    public void LoadGame()
    {
        if
(File.Exists("C:/Users/3030C/AppData/LocalLow/DefaultCompany/NeuralNetwork/SavedData.
dat"))
        {
            BinaryFormatter bf = new BinaryFormatter();
            FileStream file =
File.Open("C:/Users/3030C/AppData/LocalLow/DefaultCompany/NeuralNetwork/SavedData.da
t", FileMode.Open);
            SaveData[] data = (SaveData[])bf.Deserialize(file);
            file.Close();

            layers = new Layer[data.Length];
            for (int i = 0; i < layers.Length; i++)
            {
                int nextSize = 0;
                if (i < data.Length - 1) nextSize = data[i + 1].size;
```

```
                layers[i] = new Layer(data[i].size, nextSize);
                for (int j = 0; j < data[i].size; j++)
                {
                    layers[i].biases[j] = data[i].biases[j];
                    for (int k = 0; k < nextSize; k++)
                    {
                        layers[i].weights[j, k] = data[i].weights[j, k];
                    }
                }
            }
            //Prints(layers[1].biases);
            Console.WriteLine("Game Loaded");
        }
    }
}

[Serializable]
public class SaveData
{
    public double[] biases;
    public double[,] weights;
    public int size;
    public SaveData(int size, int nextSize)
    {
        this.size = size;
        biases = new double[size];
        weights = new double[size, nextSize];
    }
}
```

ДОДАТОК В

Код прогрмаи для аналізу табличних даних

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import mpld3 as mpl

#Import models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics

df = pd.read_csv("E:/Downloads/data.csv",header = 0)
df.head()
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(df)
df.diagnosis.unique()
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
df.describe()
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
features_mean=list(df.columns[1:11])
# split dataframe into two based on diagnosis
dfM=df[df['diagnosis'] ==1]
dfB=df[df['diagnosis'] ==0]
traindf, testdf = train_test_split(df, test_size = 0.3)
def classification_model(model, data, predictors, outcome):
  model.fit(data[predictors],data[outcome])

  predictions = model.predict(data[predictors])

  accuracy = metrics.accuracy_score(predictions,data[outcome])
  print("Accuracy : %s" % "{0:.3%}".format(accuracy))

  kf = KFold(data.shape[0], n_folds=5)
  error = []
  for train, test in kf:
    train_predictors = (data[predictors].iloc[train,:])

    train_target = data[outcome].iloc[train]

    model.fit(train_predictors, train_target)
```

```
    error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

    print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))

  model.fit(data[predictors],data[outcome])

predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean']
outcome_var='diagnosis'
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7,
max_features=2)
classification_model(model, traindf,predictor_var,outcome_var)
```

ДОДАТОК Г

Код для аналізу за допомогою TensorFlow

```
import numpy
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
class_num = y_test.shape[1]
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=X_train.shape[1:], padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(256, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(128, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(class_num))
model.add(Activation('softmax'))
```

```
epochs = 25
optimizer = 'adam'
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
print(model.summary())
```