Вінницький національний технічний університет
Факультет комп'ютерних систем та автоматики
Кафедра автоматизації та інтелектуальних інформаційних технологій

Пояснювальна записка

до бакалаврської дипломної роботи

бакалавр
(освітньо-кваліфікаційний рівень)

на тему: Розробка програмного забезпечення для автоматичного детектування пневмотораксу із використанням нейронних мереж

Виконав: студент 4 курсу, групи 1СІ-16б
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Півошенко В. В.

Керівник: к.т.н., доцент каф. АІІТ
Іванов Ю. Ю.

Рецензент: _____

Вінниця ВНТУ – 2020 року

Вінницький національний технічний університет

Факультет комп'ютерних систем і автоматики

Кафедра автоматизації та інтелектуальних інформаційних технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 151 – «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
Завідувач кафедрою АІІТ
д.т.н., проф. Квєтний Р. Н.
«___»_____ 202_ р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Півошенко Володимиру Володимировичу

1. Тема роботи: Розробка програмного забезпечення для автоматичного детектування пневмотораксу із використанням нейронних мереж
   Керівник роботи: к.т.н., доцент каф. АІІТ Іванов Ю. Ю.
   Затверджені наказом ВНТУ від «30»____08___ 2019 року № 1.
2. Строк подання студентом роботи: «20» ___06_____ 2020 р.
3. Вихідні дані до роботи: згорткова нейронна мережа, автокодер, вибірка даних (метадані пацієнтів - вік та стать, проекції знімків), мова програмування *Python*.
4. Зміст розрахунково-пояснювальної записки: огляд предметної області; розробка математичного апарату нейронної мережі; розробка програмного забезпечення та експериментальні дослідження.
5. Перелік графічного матеріалу: архітектура мережі; структура для роботи з програмним забезпеченням; приклад вихідних даних; лістинг програмного забезпечення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
1-3	Іванов Ю. Ю., к.т.н., доцент каф. АІІТ		

7. Дата видачі завдання: «10»____09____ 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд предметної області	до 10.11.2019	
2	Розробка математичного апарату нейронної мережі	до 10.02.2020	
3	Розробка програмного забезпечення та експериментальні дослідження	до 10.05.2020	
4	Оформлення пояснювальної записки і графічного матеріалу	до 30.05.2020	
5	Попередній захист роботи	до 04.06.2020	
6	Остаточний захист роботи	до 20.06.2020	

Студент _____ (підпис) Півошенко В. В. (прізвище та ініціали)

Керівник роботи _____ (підпис) Іванов Ю. Ю. (прізвище та ініціали)

АНОТАЦІЯ

У даній роботі розглядається задача детектування пневмотораксу легень на медичних рентгенівських зображеннях. Для її розв'язання використано ансамбль комбінаційних моделей автокодера та згорткової нейронної мережі. Розроблено алгоритмічне та програмне забезпечення, яке дозволяє виконати низку експериментальних досліджень.

ABSTRACT

In this work has been considered the task of lung pneumothorax detecting on medical X-ray images. To solve it, an ensemble of combination models of the autoencoder and the convolutional neural network was used. The algorithms and software, allows us to perform some experimental researches, have been developed.

ЗМІСТ

ВСТУП

Актуальність. Нині процес медичної діагностики різних захворювань, що реєструються за допомогою апаратури, яка візуалізує досліджувані органи пацієнта в статичному (зображення) або динамічному (відео) вигляді, все більше змінюється в напрямку автоматизації рутинних дій медичного фахівця та застосування засобів підтримки прийняття рішень у ході постановки діагнозу. Дані тенденції проявляються завдяки розвитку комп'ютерних систем медичної діагностики, які дозволяють виявити і локалізувати патологію на ранніх стадіях розвитку захворювання, а також оцінити її при постановці діагнозу [1].

Однак, оскільки дана галузь є досить новою, а відповідні методи дослідження тільки розвиваються, то існуючі на даний момент рішення в сфері комп'ютерної діагностики захворювань на медичних зображеннях і відеоданих все ще не забезпечують необхідну для медичної практики точність. Тому завдання розробки ефективної високоточної моделі розпізнавання патологій для таких систем, стійкої до змін даних, є актуальним.

В останні роки безліч наукових груп по всьому світу займаються завданням розробки нових і модифікації існуючих підходів до ранньої комп'ютерної діагностики захворювань на медичних зображеннях і відеоданих. Основні зусилля дослідників спрямовані на поліпшення процесу детектування патологій шляхом розширення простору ознак для виділених областей інтересу, що ймовірно є патологіями, підвищення точності розрахунку значень ознак областей інтересу, а також застосування найбільш сучасних і ефективних моделей класифікації патологій, які властиві певним захворюванням [2]. У більшості сучасних наукових робіт для здійснення одного з ключових етапів комп'ютерної діагностики, пов'язаного з класифікацією патологій, використовуються згорткові нейронні мережі (ЗНМ), як одні з найбільш ефективних моделей розпізнавання образів складних об'єктів на зображенні [3].

Складність застосування такого підходу полягає у відсутності формалізованого підходу до формування архітектури глибоких ЗНМ. В існуючих дослідницьких роботах, наприклад A. Blumenfeld [4], R. Golan [5], B. Golosio [6], залишаються практично не розглянутими питання вибору найбільш ефективних параметрів архітектури ЗНМ, які в значній мірі можуть впливати на точність класифікації патологій і якість роботи системи діагностики в цілому. Отже, дослідження даного питання є актуальними.

Мета і задача дослідження. Метою роботи є підвищення ефективності комп'ютерної діагностики пневмотораксу легень за рахунок розробленої архітектури нейронної мережі.

Для досягнення мети необхідно розв'язати наступні *задачі*:

– проаналізувати поставлену задачу та можливості застосування нейронних мереж для її розв'язання;

– розробити математичну модель глибокої нейронної мережі для класифікації зображень;

– розробити програмні засоби та оцінити ефективність класифікаційної моделі, інтерпретувати отримані результати.

Об'єктом дослідження є процеси оброблення даних, розпізнавання та детектування пневмотораксу на зображеннях.

Предметом дослідження є моделі та інструментальні засоби для розв'язання задачі детектування пневмотораксу.

Методи дослідження. При виконанні роботи використовувалися такі методи досліджень: методи цифрової обробки зображень, комп'ютерного зору та машинного навчання для розробки архітектури ансамблю класифікаційних моделей на основі ЗНМ; експериментальні дослідження для перевірки достовірності отриманих теоретичних положень; візуалізація та інтерпретація отриманих результатів. Основним засобом для розробки програмного забезпечення та оброблення даних є мова програмування *Python*.

Науково-технічний результат роботи полягає у тому, що запропоновано адаптивну модель класифікатора, особливістю якої є використання ансамблю моделей автокодерів та ЗНМ, що дозволило підвищити точність розпізнавання пневмотораксу легень на зображеннях.

Практична цінність роботи полягає у тому, що розроблено архітектуру нейронної мережі для детектування пневмотораксу та наведено низку експериментальних досліджень.

Апробація результатів та публікації. За результатами даної роботи опубліковано тези доповіді [7] на XLIX науково-технічній конференції Факультету комп'ютерних систем і автоматики (м. Вінниця, 2020) та [42] на V міжнародній науковій конференції "Вимірювання, контроль та діагностика в технічних системах" (м. Вінниця, 2019).

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Нині комп'ютерні системи медичної діагностики, які використовуються для детектування патологій на медичних зображеннях і відеоданих, починають (в експериментальному режимі) застосовуватися у роботі різних спеціалізованих медичних установ та організацій. У багатьох випадках подібні системи дозволяють підвищити точність діагностики і здійснити допомогу в ухваленні рішення із фахівцем, але якість детектування і можливість їхнього застосування для широкого кола задач залишається на недостатньо високому рівні [5, 8]. А у діагностиці найбільш поширених захворювань точність визначення патології відіграє ключову роль.

Захворювання легень є одними з найпоширеніших і найскладніших патологій у світі. Вони виступають причиною кожної шостої людської смерті. Найбільш поширеними і небезпечними є інфекції нижніх дихальних шляхів, хронічна обструктивна хвороба легень, рак, тромбоемболія легеневої артерії, туберкульоз, а також пневмоторакс легень [6, 9, 10]. Оскільки дані захворювання значно розрізняються по медичній семіотиці, то і методи їх діагностики також є різними. Основними і найбільш точними методами діагностики перерахованих захворювань є променеві методи, які включають в себе аналіз рентгенографічних знімків і даних комп'ютерної томографії. Даний тип досліджень вимагає великої уваги та досвіду медичного фахівця, тому часто через складність роботи виникають різного роду помилки. Крім того, існуючі на сьогоднішній день системи діагностики здатні визначити далеко не всі види існуючих захворювань легень. У даному розділі увагу приділено особливостям детектування тромбів в артеріях легень, солітарних вузлів та пневмотораксу легень.

1.1 Виявлення тромбів у легенях

Метод ангіопульмонографії із застосуванням комп'ютерної томографії (КТ) з ангіографією гілок легеневої артерії дозволяє виявляти тромби в судинах легень, які зазвичай властиві для тромбоемболії легеневої артерії [6] (рисунок 1.1).

Рисунок 1.1 – Знімок КТ-ангіопульмонографії:
стрілками вказані тромби в просвіті правої і лівої легеневих артерій

Однією з перших значних робіт по розробці методів комп'ютерної медичної діагностики для детектування тромбів є праця [11]. У даній роботі процес детектування проводиться у кілька стадій: сегментація легень на знімках комп'ютерної томографії; визначення областей інтересу (областей, що ймовірно містять патології, в даному випадку – емболії) шляхом контрастного виділення областей різної інтенсивності за допомогою алгоритму Тоббогана; класифікація виділених областей інтересу. Розроблена комп'ютерна система діагностики забезпечує значення *True Positive Rate* (TPR – метрика, яка показує кількість правильно класифікованих об'єктів із позитивною міткою) 80% при значенні *False Positive Rate* (FPR – метрика, яка показує кількість правильно класифікованих об'єктів із негативною міткою) на КТ-знімку – 4%.

Цікавим напрямком розвитку даних досліджень є робота [12], в якій проводиться додаткова сегментація судин легень та використовується повнозв'язна штучна нейронна мережа для зменшення кількості невірно виділених областей інтересу (класифікатор на основі генетичного алгоритму). Значення TPR – 62,5% при значенні FPR = 17,1% на наборі КТ-даних пацієнта.

Шляхом поліпшень точності алгоритмів сегментації судин у тривимірному просторі і більш точного виділення ознак для областей інтересу, отриманих після сегментації, у дослідженні [13] вдалося досягти значення TPR = 95,1% при значенні FPR на наборі КТ-даних пацієнта 17,1%. Даний результат є одним з найбільш точних із існуючих на сьогоднішній день.

1.2 Виявлення солітарних легеневих вузлів

Солітарні легеневі вузли на рентгенологічних знімках, які мають округлу форму розміром від 10 мм до 3 см, можуть бути властиві низці захворювань, серед яких кіста, абсцес, аневризми, туберкульоз та рак легень. Більшість сучасних комп'ютерних систем медичної діагностики використовують набір КТ-даних пацієнта для детектування патологій такого виду. Приклад КТ-знімка із зазначенням місця розташування патології наведено на рисунку 1.2.

Рисунок 1.2 – Знімок КТ: стрілкою вказано на солітарний легеневий вузол

Одним з перших підходів до створення системи детектування солітарних вузлів є робота [6], в якій за допомогою застосування декількох порогів інтенсивності, при яких структура вузла може бути ідентифікована на КТ-знімку, проводиться тріангуляція області інтересу, яка може бути вузлом. Потім з використанням ознак об'єму, площі поверхні, округлості і щільності проводиться класифікація областей інтересу за допомогою штучної нейронної мережі. Запропонований метод детектування вузлів, розмір яких складає більше 2,9 мм, забезпечує значення TPR – 71% при значенні FPR = 4% на всьому наборі КТ-даних пацієнта.

Розвитком цих досліджень стала робота [5], в якій сегментація солітарних вузлів, обчислення характерних ознак сегментів та класифікація виділених областей інтересу здійснюється за допомогою штучної нейронної мережі і методу опорних векторів. Значення TPR, отримане за допомогою найкращої моделі при детектуванні вузлів, розмір яких становить понад 2,9 мм, складає 87,5%, а FPR = 4% на всьому наборі КТ-даних пацієнта.

Застосування в якості класифікатора ЗНМ у роботі [14] дозволило досягти значення TPR = 88 %. Даний результат є найкращим з існуючих на сьогоднішній день для систем детектування солітарних легеневих вузлів.

1.3 Виявлення пневмотораксу легень

Пневмоторакс – це патологія легень, яка пов'язана з аномальним збиранням повітря в плевральному просторі між легенею та грудною стінкою (рисунок 1.3). Це може бути наслідком різних етіологій, включаючи травми грудної клітини, легеневі захворювання тощо. Пневмоторакс може бути небезпечним для життя і вважається випадком для надання екстреної допомоги в реанімації, що вимагає оперативного розпізнавання та втручання.

Рисунок 1.3 – Рентген легень: стрілкою вказано на пневмоторакс

Оскільки детектування пневмотораксу методами машинного навчання є досить новою галуззю в медицині та комп'ютерних науках, то кількість опублікованих досліджень є обмеженою.

У роботі [15] для детектування пневмотораксу використовувався підхід дотренування ЗНМ ResNet-50, яка була попередньо натренована на даних, що відображають 14 різних легеневих патологій. Для вирішення поставленої задачі було змінено вихідний повнозв'язний шар із функцією активації для мультикласифікації на функцію активації для бінарної класифікації. Даний підхід дозволив досягти значення *Area Under Curve* (AUC – метрика, яка використовується у задачах класифікації; чим більше її значення, тим краще) на рівні 0,92.

Розвитком попереднього дослідження стала робота [4], у якій покращено результат шляхом застосування ансамблю моделей на основі дотренованих ЗНМ ResNet-50, але зі зміненим методом оптимізації мережі (замість методу Adam використовувся RMSProp [16]). Це дозволило збільшити показник AUC до 0,96, що є одним із найкращих результатів на сьогодні.

Проте, більшість досліджень пов'язаних із детектуванням пневмотораксу дають лише ймовірність його наявності на медичному зображенні, а не виділяють ймовірну область локалізації, що значно полегшило б постановку діагнозу медичним фахівцем.

1.4 Висновки до розділу

У даному розділі були описані основні методи, які застосовуються для детектування різних патологій легень при розробці комп'ютерних систем медичної діагностики. Точність, що забезпечується більшістю з них, все ще є недостатньо високою для використання у медичній практиці. Слід зазначити, що підхід до детектування патологій є схожим для кожної з розглянутих систем діагностики:

а) виділення областей інтересу (патологій). Методи, які застосовуються на даному етапі, є індивідуальними для кожного типу патології;

б) класифікація областей інтересу, заснована на значеннях їх основних ознак. Методи даного етапу є схожими для різних задач діагностики.

Виходячи з цього, можна зробити висновок, що в якості класифікаторів, які застосовуються на другому етапі детектування патологій, слід застосовувати ЗНМ, оскільки вони дозволяють ефективно розпізнавати образи на зображеннях.

Однак у наукових роботах часто розглядається модель з архітектурою, яка вже успішно застосована для інших задач розпізнавання, або із випадково підібраними параметрами, що є неефективним для різних типів завдань. Питання визначення значень параметрів архітектури ЗНМ практично не розглядається, але вона має безліч параметрів, які впливають на точність розпізнавання. Саме тому дана робота присвячена питанню формування найбільш ефективної архітектури класифікатора на ЗНМ для комп'ютерних систем медичної діагностики.

2 РОЗРОБКА МАТЕМАТИЧНОГО АПАРАТУ НЕЙРОННОЇ МЕРЕЖІ

2.1 Аналіз компонентів та архітектур згорткової нейронної мережі

Згорткова нейронна мережа (*Convolutional Neural Network*) – це особлива архітектура штучної нейронної мережі, яка імітує особливості зорової кори головного мозку, складається з декількох багатовимірних шарів і призначена для ефективного розпізнавання складних зображень [17]. У даний час ЗНМ і машинне навчання є одними з головних напрямків розвитку більшості галузей сучасних інформаційних технологій. Однак базові принципи побудови архітектури таких мереж мало змінилися [18].

Для задач розпізнавання об'єктів на зображеннях вхідний шар найчастіше представляється у вигляді тривимірної сітки, розміри якої залежать від вхідного зображення, що можна представити у вигляді

$$I = W \cdot H \cdot D, \tag{2.1}$$

де I – розмірність вхідного шару мережі; W та H – ширина та висота вхідного зображення; D – глибина або кількість колірних каналів зображення.

Згортковий шар (ЗШ) нейронної мережі призначений для виділення ознак зображення та їх перетворень, які, в свою чергу, на більш глибоких шарах мережі, використовуються для отримання більш складних ознак, а у підсумку визначають клас об'єкта. Основною характеристикою даного шару є так звані фільтри – багатовимірні (зазвичай двовимірні або тривимірні) матриці ваг зв'язків нейронів попереднього шару з нейронами ЗШ. Фільтрами вони називаються тому, що операція згортки, тобто отримання вихідного сигналу нейрона ЗШ, дуже схожа на

операцію фільтрації зображення [19]. Процес формування вихідних значення ЗШ показаний на рисунку 2.1.

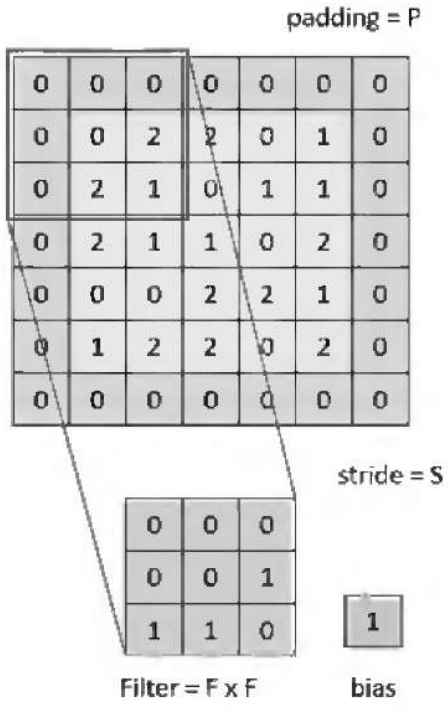

Рисунок 2.1 – Формування ЗШ

Розмірність ЗШ визначається його параметрами і розмірами попереднього шару мережі, тобто

$$W_{\text{ЗШ}} = \frac{(W_p - F + 2P)}{S} + 1, \tag{2.2}$$

$$H_{\text{ЗШ}} = \frac{(H_p - F + 2P)}{S} + 1, \tag{2.3}$$

де $W_{\text{ЗШ}}$ і $H_{\text{ЗШ}}$ – ширина і висота ЗШ відповідно; W_p і H_p – ширина і висота попереднього шару відповідно; F – розмір квадратного фільтру ЗШ мережі; P – кількість доданих нулів по краях попереднього шару; S – зміщення фільтра.

Для формування вихідного сигналу нейронів ЗШ в ЗНМ використовуються функції активації нейрона, які за певними правилами перетворюють сигнал *x*. У ЗНМ найчастіше використовуються такі види функцій активації: порогова або передавальна функція випрямляч (*relu*), сигмоїдна (*sigmoid*), нормована експоненційна (*softmax*), гіперболічна (*tanh*) функції.

Зворотно-згортковий шар (ЗЗШ) є протилежним шаром до ЗШ, оскільки він використовується для збільшення розмірності попереднього шару та відновлення виділених попередніми шарами ознак. Саме тому даний тип шарів широко застосовується у автокодерах та ЗНМ, які використовуються для сегментації та детектування об'єктів на зображеннях [22]. Розмірність ЗЗШ визначається його параметрами і розмірами попереднього шару мережі, а отримані вихідні значення ЗЗШ подаються, як аргументи, у функції активації, тобто

$$W = \left(W_p - 1\right) \times S + F - 2 \times P, \tag{2.4}$$

$$H = \left(H_p - 1\right) \times S + F - 2 \times P, \tag{2.5}$$

де W і H – ширина і висота ЗЗШ відповідно; W_p і H_p – ширина і висота попереднього шару відповідно; F – розмір квадратного фільтру ЗЗШ мережі; P – значення доповнення; S – зміщення фільтра.

Повнозв'язний шар є одновимірним, у ньому кожен нейрон пов'язаний з кожним нейроном попереднього шару на всіх рівнях, якщо у попередньому шарі присутній параметр глибини. Даний шар також може використовуватися в якості останнього (вихідного) шару ЗНМ, результатом якого є ймовірність приналежності вхідного зображення до певного класу. Єдиним параметром даного шару є кількість нейронів D_n. Зазвичай вона вибирається, виходячи з розміру попереднього шару і кількості класів (необхідної кількості виходів) мережі [23]. Схема розглянутого шару представлена на рисунку 2.2.

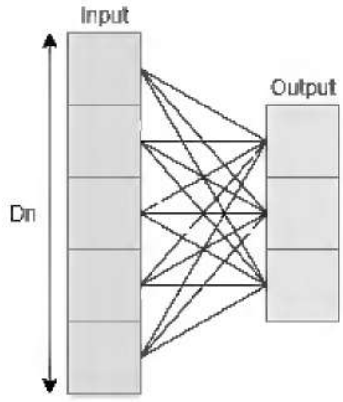

Рисунок 2.2 – Два повнозв'язних шари ЗНМ

Шар виключення використовується для регуляризації даних, що допомагає запобігти перенавчанню мережі шляхом випадкового виключення нейронів (вузлів) у певному шарі на кожній епосі навчання (рисунок 2.3). Таким чином, отримується n умовних моделей, результат яких усереднюється [24].

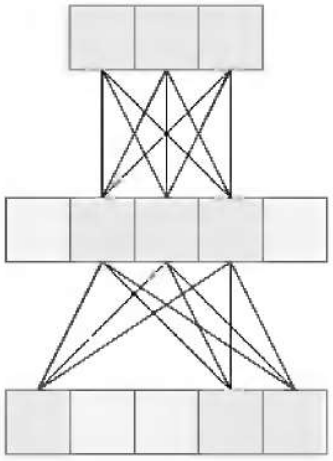

Рисунок 2.3 – Формування шару виключення

Шар субдискретизації ЗНМ використовується для скорочення розмірності даних з метою зменшення ймовірності швидкого перенавчання, а також для зниження обчислювальної складності і витрат пам'яті. В даному шарі використовується вікно певного розміру (U) для виділення областей нейронів

попереднього шару, які пов'язані з нейроном шару субдискретизації. Потім, за певними правилами вибирається і розраховується один сигнал нейрона шару субдискретизації. Найбільш типові способи формування сигналу даного шару – це знаходження максимального значення серед сигналів попереднього шару (*max*), які знаходяться в межах вікна субдискретизації, або розрахунок середнього значення (*avg*) [25]. Процес формування сигналів шару субдискретизації показаний на рисунку 2.4.

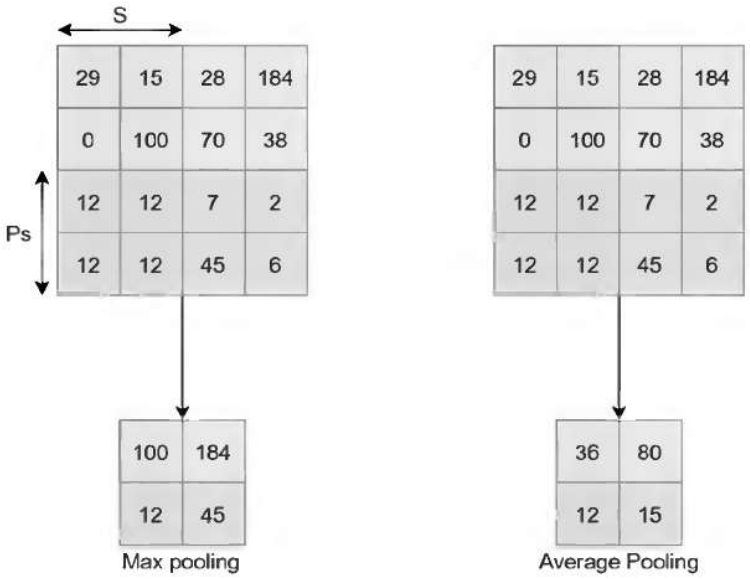

Рисунок 2.4 – Формування шару субдискретизації за допомогою операції *max*

Розмірність шару субдискретизації визначається його параметрами і розмірами попереднього шару мережі, відповідно до формул

$$W = \frac{(W_p - P_s)}{S} + 1, \tag{2.6}$$

$$H = \frac{(H_p - P_s)}{S} + 1, \tag{2.7}$$

де W і H – ширина і висота шару субдискретизації відповідно; W_p і H_p – ширина і висота попереднього шару відповідно; P_s – розмір квадратного вікна шару субдискретизації мережі; S – зміщення вікна шару субдискретизації.

Першою запропонованою ЗНМ була модель Яна Лекуна – *LeNet*. Вона містила два ЗШ, шари субдискретизації, які чергуються, а також три повнозв'язних шари. Дана архітектура з параметрами, наведеними на рисунку 2.5, вважається класичною.

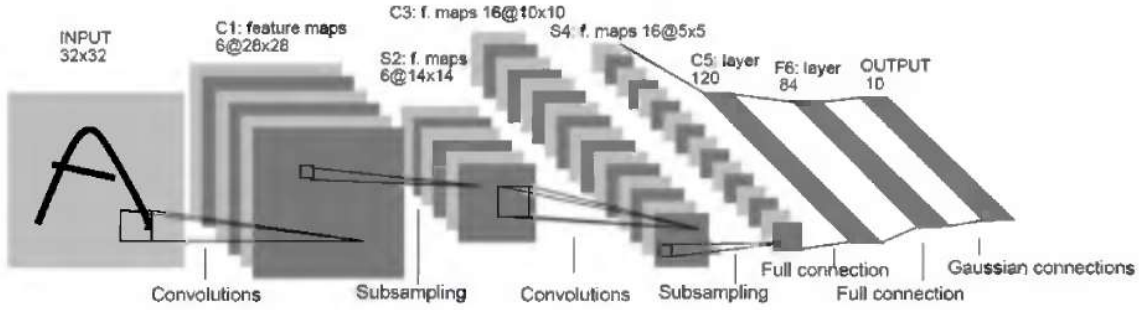

Рисунок 2.5 – Схема архітектури ЗНМ *LeNet*

Наступною значною ЗНМ була мережа *AlexNet*, запропонована Алексом Крижевським (рисунок 2.6) [26]. Вона була дуже схожою на мережу *LeNet*, однак відрізнялася від неї більш масивною і складною архітектурою. У її складі був лише один згортковий шар і кілька шарів субдискретизації з операцією вибору максимального значення, 96 фільтрів. Мережа *AlexNet* призначена для розпізнавання об'єктів будь-якої складності на великих зображеннях розміром 224*224. У 2012 році вона стала переможцем у конкурсі *ImageNet*, значно обігнавши своїх конкурентів.

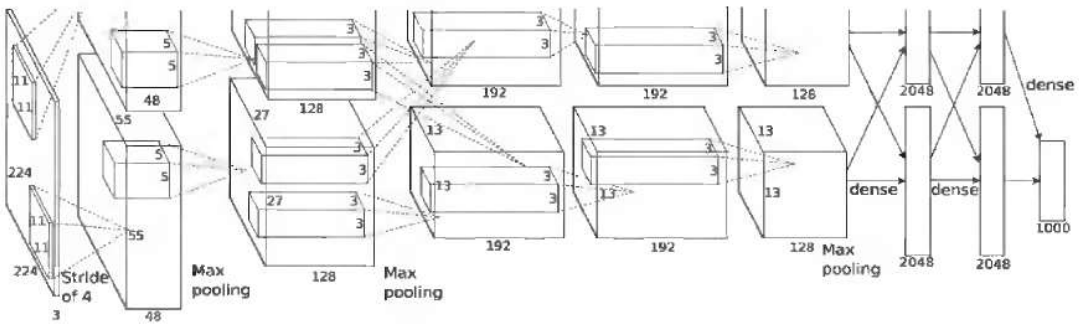

Рисунок 2.6 – Схема архітектури ЗНМ *AlexNet*

У 2014 році компанія *Google* почала розробляти свою архітектуру ЗНМ і представила *GoogLeNet* (рисунок 2.7). Дана ЗНМ є дуже глибокою – до 22 шарів. Але, незважаючи на це, має в 10 разів менше параметрів, ніж мережа *AlexNet*, що позитивно позначається на продуктивності роботи і витратах пам'яті. Також у ній використовуються малі розміри фільтрів, а шар субдискретизації реалізований за принципом вибору середнього значення [27].

Рисунок 2.7 – Схема архітектури ЗНМ *GoogLeNet*

Наступний значний внесок в розвиток архітектур ЗНМ внесла мережа *VGGNet* [28]. Дана ЗНМ складається з великої кількості шарів, які чергуються і мають малі розміри (3x3 – розмір фільтрів ЗШ, 2x2 – розмір шару субдискретизації). Негативною стороною є те, що вона зберігає до 140 мільйонів параметрів, а це робить її дуже громіздкою і низькопродуктивною (рисунок 2.8).

Однією з найбільш сучасних та ефективних ЗНМ на сьогоднішній день є мережа *ResNet* (рисунок 2.9) [29], яка стала переможцем у конкурсі *ILSVRC* 2015. Архітектура даної мережі передбачає велику кількість ЗШ, які містять багато фільтрів (до 512) малого розміру (3*3). Глибина мережі може досягати 152 шарів. На її прикладі було встановлено, що ЗНМ може використовувати тільки згорткові шари, а якість розпізнавання значно збільшується при збільшенні глибини мережі.

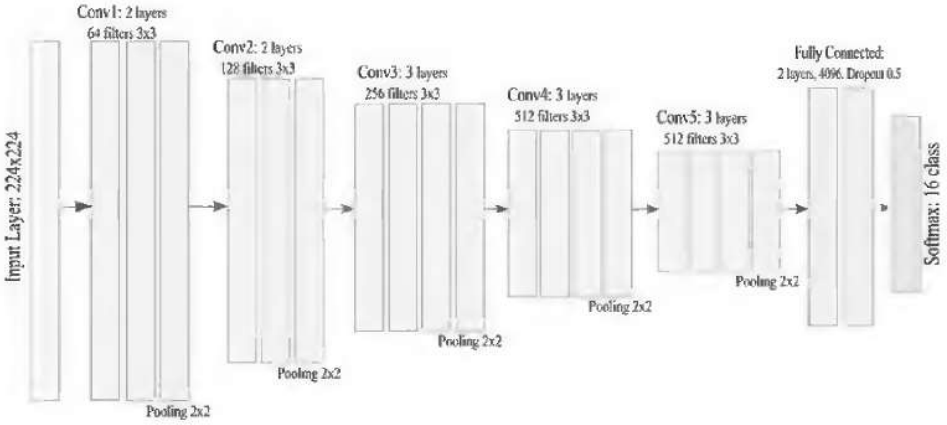

Рисунок 2.8 – Архітектура мережі *VGGNet*

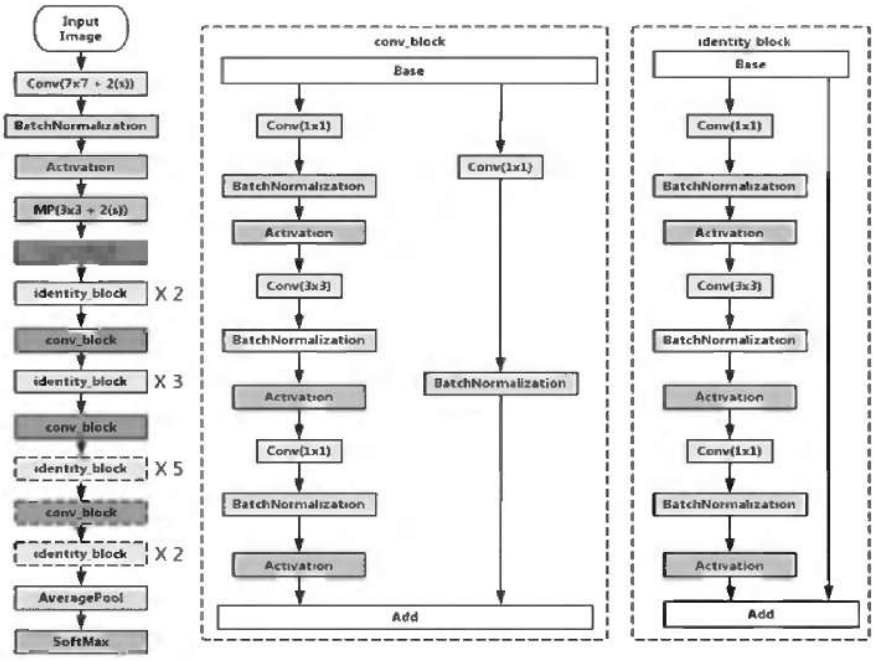

Рисунок 2.9 – Архітектура мережі *ResNet*

Проведений аналіз архітектур найбільш ефективних ЗНМ, існуючих на сьогоднішній день, дозволив виділити особливості побудови ЗНМ, які забезпечують найкращу якість їхньої роботи. Виділені особливості дозволяють сформувати рекомендації, які дадуть змогу розробити найбільш ефективну архітектуру ЗНМ.

2.2 Навчання згорткової нейронної мережі

Навчанням штучної нейронної мережі називається процес налаштування значень ваг зв'язків між нейронами мережі. У ЗНМ використовується навчання з учителем, яке вимагає використання навчальної вибірки для порівняння вихідного сигналу мережі з еталонним значенням навчальної вибірки, розрахунок помилки і налаштування ваг зв'язків мережі з метою зменшення значення цієї помилки [5].

У ЗНМ, як алгоритм навчання, використовується алгоритм зворотного поширення помилки на базі стохастичного градієнтного спуску (СГС). Для повнозв'язних і згорткових шарів значення помилки вихідного сигналу мережі розраховується за загальною формулою [30]

$$E(w) = \frac{1}{2}\sum_{i,k}(f_{ik} - y_{ik})^2, \tag{2.8}$$

де $E(w)$ – функція помилки мережі; f_{ik} та y_{ik} – розраховане та очікуване значення вихідного сигналу k-го нейрона мережі при подачі i-го зразка навчальної вибірки.

Навчання мережі направлене на мінімізацію функції помилки. Значення даної функції для шару мережі визначається за допомогою виразу

$$\delta_i^{(q)} = (f_{ik}^{(q)}(S))' \sum_j w_{ij}\delta_j^{(q+1)}, \tag{2.9}$$

де $\delta_i^{(q)}$ – помилка i-го нейрона в шарі q; $(f_{ik}^{(q)}(S))'$ – похідна функції активації i-го нейрона шару q для k-го зразка навчальної вибірки; S – сигнал, який подається на вхід i-го нейрона мережі; w_{ij} – вага зв'язку, що з'єднує i-ий нейрон шару q і j-ий нейрон шару $(q + 1)$; $\delta_j^{(q+1)}$ – помилка j-го нейрона в шарі $(q + 1)$.

На основі розрахованої помилки нейрона визначається зміна ваги зв'язку цього нейрона за формулою

$$\Delta w_{ij}^{(q)} = -\eta \delta_j x_i, \tag{2.10}$$

де $\Delta w_{ij}^{(q)}$ – значення зміни ваги зв'язку, що з'єднує i-ий нейрон шару q і j-ий нейрон шару $(q + 1)$; η – значення, яке визначає швидкість навчання; δ_j – значення помилки j-го нейрона в шарі q; x_i – значення i-го вхідного сигналу.

Мінімізація втрат цільової функції відбувається на кожній ітерації t з налаштуванням матриці ваг у вигляді

$$w_{t+1} = w_t + \Delta w_{ij}^{(q)}. \tag{2.11}$$

Слід зазначити, що класичний градієнтний спуск є застарілим і на складних даних забезпечує погану оптимізацію функції втрат. В останні роки розроблено ряд модифікацій даного методу: *Adam*, *Adagrad*, *Momentum*, *Adadelta*, *RMSProp*. Кожен із них має свої переваги і недоліки [31], але найбільшою ефективністю вирізняється оптимізатор *Adam* (*Adaptive Moment Estimation*).

Adam – це метод адаптивної оптимізації параметрів навчання мережі [32]. Його робота заснована на формулах

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.13}$$

де g_t – градієнт; m_t – перший момент; v_t – другий момент; $m_0 = v_0 = 0$; $\beta_1 = 0{,}9$ та $\beta_2 = 0{,}999$ – коефіцієнти.

Основною відмінністю від інших оптимізаторів є початкове калібрування параметрів m_t та v_t. На перших кроках m_t та v_t штучно збільшують за правилом калібрування

$$\begin{cases} \hat{m}_t = \dfrac{m_t}{1-\beta_1^t}, \\ \hat{v}_t = \dfrac{v_t}{1-\beta_2^t}. \end{cases} \tag{2.14}$$

Враховуючи вираз (2.11) та правило оновлення ваг для градієнтного методу, маємо

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t, \tag{2.15}$$

де $\epsilon = 10^{-8}$ – параметр згладжування (необхідний, щоб запобігти діленню на 0).

У ЗШ дані формули використовуються для налаштування ваг зв'язків для кожного фільтра. Для шарів субдискретизації розрахунок помилки не проводиться, оскільки вони не беруть участі в навчанні мережі.

2.3 Аналіз архітектур та навчання автокодерів

Автокодер (*autoencoder*) – особливий вид архітектури штучних нейронних мереж, який дозволяє вирішувати завдання “навчання без вчителя” із використанням методу зворотного поширення помилки (*backpropagation algorithm*). Перевагою автокодерів є те, що їх можна проектувати, використовуючи будь-які існуючі шари штучних нейронних мереж. Розрізняють автокодер *vanilla* (*vanilla autoencoder*), розріджений автокодер (*sparse autoencoder*), знешумлювальний автокодер (*denoising autoencoder*), варіаційний автокодер (*variational autoencoder*).

Автокодер *vanilla* – це найпростіший тип автокодерів, що представляє собою мережу прямого поширення без зворотного зв'язку, яка найбільш схожа на

перцептрон (містить вхідний шар, проміжний шар і вихідний шар) [33]. Проте, на відміну від перцептрону, вихідний шар автокодера *vanilla* повинен містити стільки ж нейронів (X), скільки і вхідний шар (Y) (рисунок 2.10).

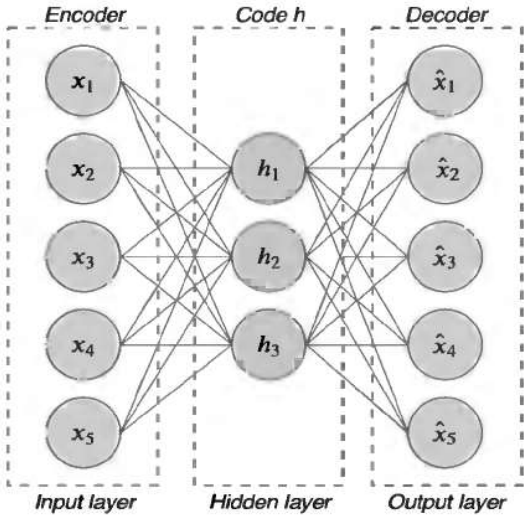

Рисунок 2.10 – Автокодер *vanilla*

Даний тип автокодерів складається із двох частин: кодувальника $\phi: \chi \to \mathcal{F}$ (*encoder*) та декодувальника $\psi: \mathcal{F} \to \chi$ (*decoder*). У найпростішому випадку, коли є лише один прихований шар, автокодер приймає вхід $X \in \mathbb{R}^d = \chi$ і відображає його на $h \in \mathbb{R}^d = \mathcal{F}$, тобто

$$h = \sigma(WX + b), \tag{2.16}$$

де h – код або латентне представлення; σ – функція активації; W – матриця ваг; b – вектор зміщення.

Матриця ваг та вектор зміщення задаються випадково та ітеративно оновлюються під час навчання, завдяки методу зворотного поширення помилки. На етапі декодування код h відображається на реконструйований вектор X', такої ж розмірності як і вхідний вектор X, тобто

$$X' = \sigma'(W'h + b'), \tag{2.17}$$

де X' – реконструйований вектор; σ' – функція активації декодувальника; W'–матриця ваг декодувальника; b'– вектор зміщення декодувальника.

Автокодери *vanilla* тренують, мінімізуючи похибки побудови (*loss*), що можна представити у вигляді

$$\mathcal{L}(X, X') = |X - X'|^2 = \left|X - \sigma'\left(W'\left(\sigma(WX + b)\right) + b'\right)\right|^2. \tag{2.18}$$

Розріджений автокодер є розвитком автокодеру *vanilla*, але він може включати більше прихованих шарів та вузлів у них. Слід зазначити, що лише певній частині із них дозволено бути активними на кожній епосі тренування [34]. Розріджений автокодер під час тренування включає в себе штраф за розрідженість $\Omega(h)$ у коді h, тобто

$$\mathcal{L}(X, X') = \mathcal{L}(X, X') + \Omega(h). \tag{2.19}$$

Це заохочує модель активувати ($\widehat{x_n} \to 1$) певні ділянки мережі на основі вхідних даних, змушуючи інші вузли бути неактивними ($\hat{x}_n \to 0$). Штраф можна визначити за допомогою дивергенції Кульбака-Лейблера [35], L1 (*lasso regression*) або L2 (*ridge regression*, *Tikhonov regularization*) регуляризацій [35, 36].

Знешумлювальний автокодер отримує частково спотворені дані і тренується відновлювати початкові неспотворені значення на вході [37]. Вхідний вектор X відповідає вектору із шумом $\tilde{X}$, який формується шляхом стохастичного відображення виду $\tilde{X} \sim q_D(\tilde{X}|X)$. Далі вектор $\tilde{X}$ подається на прихований шар (як і у звичайних автокодерах), із виходу якого модель відновлює вхідний вектор [38].

Моделі варіаційних автокодерів успадковують архітектуру автокодування, але роблять припущення щодо розподілу латентних змінних. Для навчання

використовується стохастично-градієнтний варіаційний Баєсівський метод (*Stochastic gradient variational Bayes method*), що призводить до додаткової складової втрат [38].

2.4 Формування ансамблю моделей

Для побудови адаптивної моделі автокодерів на базі ЗНМ використано здатність СГС уникати “сідлових” точок та локальних мінімумів, досягаючи глобального мінімуму у конвексних функціях. Проте, встановлено, що кількість можливих локальних мінімумів зростає експоненціально зі збільшенням параметрів, кількість яких у сучасних ЗНМ може бути величезною [39]. Це явище сприяє тому, що СГС не досягає глобального екстремуму, а потрапляє у певний локальний мінімум. Тому дві моделі з однаковою архітектурою, оптимізовані різними ініціалізаціями СГС, збігаються до різних локальних оптимумів (рисунок 2.11).

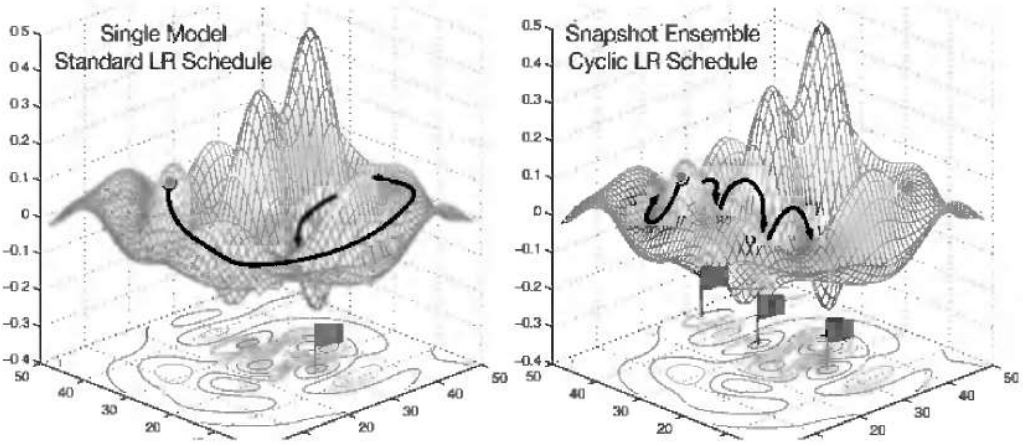

Рисунок 2.11 – Зліва: типовий процес оптимізації СГС;
справа: використання локальних мінімумів для створення ансамблю моделей

Такий підхід дозволяє створити ансамблі, в яких навчаються декілька моделей з визначеною архітектурою, але з різною ініціалізацією СГС та усередненням виходів, що призводить до зниження частоти помилок [7, 40].

Застосовуємо СГС M разів для пошуку локальних мінімумів, зберігаючи ваги моделі кожного разу після того, як процес конвергується. Для руху до локальних мінімумів використано циклічну зміну КШН [7], який зменшується під час руху до першого локального екстремуму, а потім різко збільшується, що і виводить модель з локального оптимуму (рисунок 2.12). Отже, процес навчання поділено на M циклів, кожен з яких починається з великого значення КШН, яке поступово зменшується, наприклад, за формулою

$$\eta(t) = \frac{\eta_{start}}{2}\left(\cos\left(\frac{\pi \bmod\left(t-1,\left\lceil\frac{T}{M}\right\rceil\right)}{\left\lceil\frac{T}{M}\right\rceil}\right)+1\right), \tag{2.20}$$

де η_{start} – початкове значення КШН.

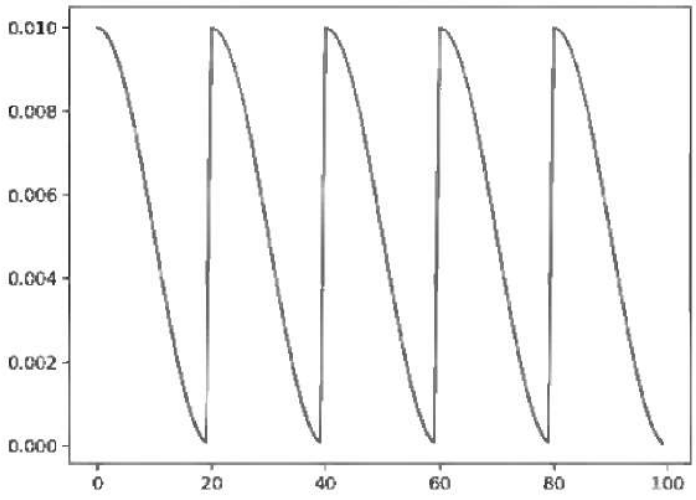

Рисунок 2.12 – Графік циклічно-косинусоїдної зміни КШН

На початку наступного циклу значення ваг коригується за методом стохастичного усереднення ваг за формулою

$$w_{SWA} = \frac{w_{SWA} \times M + w}{M+1}, \tag{2.21}$$

де w_{SWA}, w – коригована та поточна матриці ваг мережі.

З отриманого набору моделей, створених відповідно до запропонованого підходу, формується ансамбль моделей, в якому вихідний сигнал формується шляхом усереднення результатів, отриманих від кожної моделі окремо:

$$Y_{ensemble} = \frac{1}{B} \sum_{i=1}^{B} y_i(x), \tag{2.22}$$

де $Y_{ensemble}$ – вихідний сигнал ансамблю; B – кількість моделей; $y_i(x)$ – вихідний сигнал *i*-ої моделі, отриманий при подачі вхідного сигналу x.

Для оцінювання ансамблю використовують метрику Жаккара *IoU* (*Jaccard similarity, intersection over union*), яку можна представити у вигляді

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, \tag{2.23}$$

де $J(A, B) \in [0, 1]$ (причому, якщо множини пусті, то $J(A, B) = 1$); $\cap$ та $\cup$ – перетин та об'єднання множин; $|\ .\ |$ – потужність множини.

2.5 Висновки до розділу

У даному розділі проведено аналіз ключових компонентів та архітектур ЗНМ та автокодерів. Представлено математичні моделі їх навчання, включаючи алгоритм зворотного поширення помилки, ADAM та відображення у автокодерах. Розроблено ансамбль автокодера та ЗНМ з використанням циклічно-косинусоїдної зміни КШН, що дозволяє краще рухатися до глобального оптимуму функції втрат. Архітектура мережі наведена у додатках.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Вибір мови програмування

Програми, написані на мові *Python*, можуть бути виконані на будь-якому персональному комп'ютері під керуванням десктопних операційних систем *Windows*, *MacOS*, *Linux*. Крім того, відсутні механізми, які потенційно призводять до помилок (наприклад, перетворення типів з втратою точності) та присутній обов'язковий контроль виняткових ситуацій. Функціонування програми повністю визначається виконавчим середовищем. Відсутні покажчики та інші механізми для безпосередньої роботи з фізичною пам'яттю й іншим апаратним забезпеченням комп'ютера. Зручним є механізм автоматичного генерування документації на основі коду. Багато допоміжних задач, які зустрічаються при розробці програмного забезпечення, вже вирішені в рамках стандартних та користувацьких бібліотек. Існує безліч платформ для реалізації ЗНМ, наприклад *Caffe*, *TensorFlow*, *Theano*, *PyTorch* та інші. На нашу думку, найкращим рішенням для реалізації експериментального програмного стенда і оцінювання ефективності запропонованого класифікатора є бібліотека *TensorFlow*, яка розповсюджується за відкритою ліцензією [41].

Отже, мова програмування *Python* досить зручна, надійна та потужна, що і обумовило необхідність її застосування у даному проекті.

3.2 Експериментальні дослідження

В якості вхідних даних для проведення експерименту по оцінці ефективності моделі були взяті дані прикладної задачі медичної діагностики пневмотораксу, які

опубліковані на сторінках платформи *Kaggle* організацією "*Society for Imaging Informatics in Medicine*". Структура даних: 11582 рентгенівських зображень легень (рисунок 3.1).

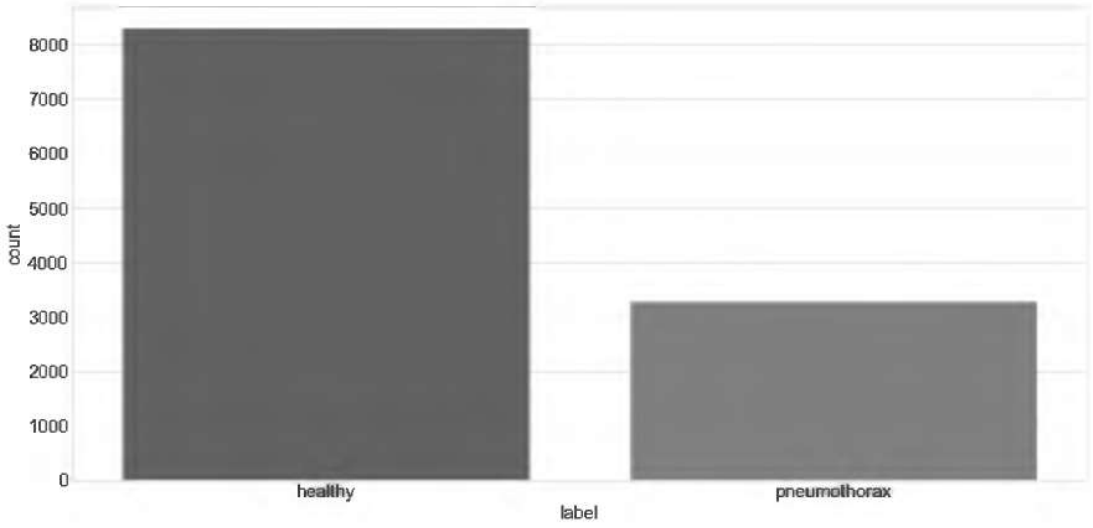

Рисунок 3.1 – Розподіл зображень

Приклад візуалізації областей інтересу наведено на рисунку 3.2, де PA / AP – задньо-передня та передньо-задня проекції відповідно

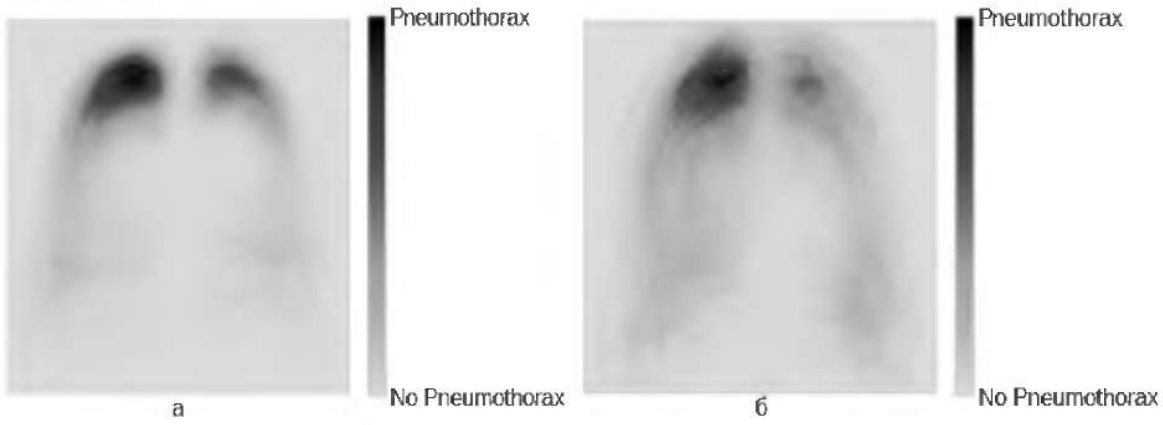

Рисунок 3.2 – Візуалізація появи пневмотораксу на *AP* (а) та *PA* (б) проекціях

Розподіли анонімізованих метаданих пацієнтів – вік та стать наведені на рисунку 3.3.

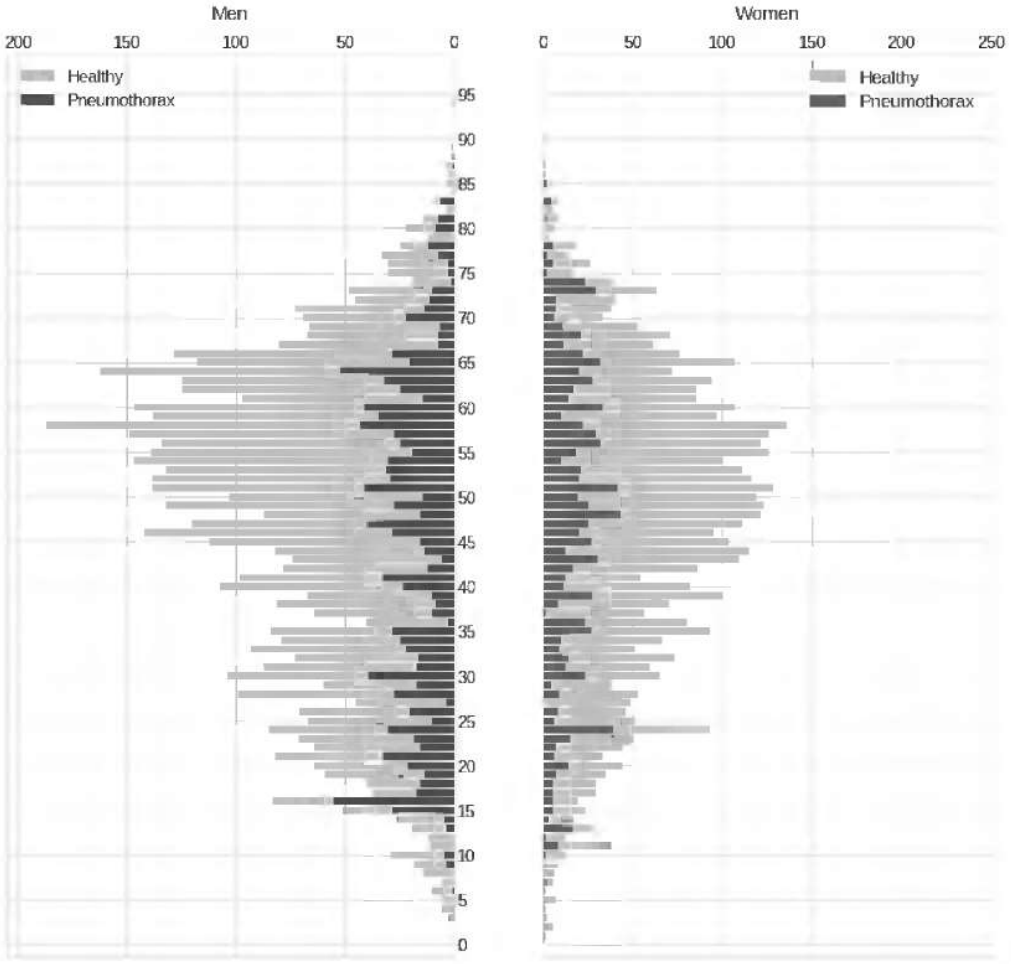

Рисунок 3.3 – Розподіл появи пневмотораксу залежно від віку та статі пацієнта

Проаналізувавши вхідні дані, помітно, що розподіл появи пневмотораксу не є збалансованим, тому для підвищення ефективності та генералізуючих властивостей моделі було вирішено при створені тренувального, валідаційного та тестувального наборів даних зберегти пропорції розподілу (рисунок 3.4).

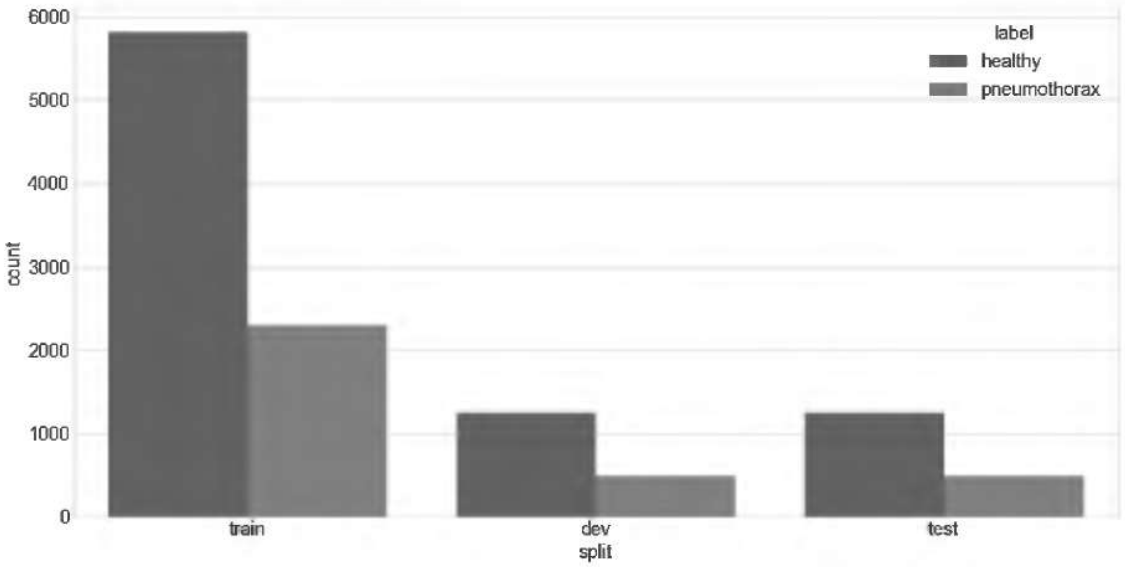

Рисунок 3.4 – Розподіл даних у тренувальному, валідаційному та тестовому наборах

Для проведення експериментів по оцінці ефективності моделі, необхідно використовувати бібліотеки, які дозволяють задати необхідні параметри архітектури, навчити мережу на вибірці даних, а також отримати результати класифікації зразків валідаційного та тестового наборів даних. Найбільш підходящим рішенням для реалізації програми і оцінки ефективності запропонованих в даній роботі моделей класифікації є бібліотека *TensorFlow*. Для підвищення генералізації моделі, було використано різні методи аугментації даних, але лише для навчальної вибірки: горизонтальне обертання зображення; еластичні трансформації; викривлення сітки; оптичне викривлення; випадкова зміна контрасту, гами, яскравості [42]. Також було застосовано бінарну перехресну ентропію L_{CE} з коефіцієнтом Соренсена-Дайса L_{SD}:

$$\mathcal{L}(y,\hat{y}) = L_{CE} + L_{SD} = \left(-\frac{1}{N}\sum_{i=1}^{N}(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i))\right) + \frac{2|y_i \cap \hat{y}_i|}{|y_i|+|\hat{y}_i|}, \quad (3.1)$$

де y та $\hat{y}$ – вхідний набір та передбачений (вихідний) відповідно; y_i – бажане значення на виході мережі із набору y; $\hat{y}_i$ – вихідні значення нейронної мережі; N – розмір вибірки; $\cap$ – перетин множин; $|\,.\,|$ – потужність множини.

Додавання коефіцієнту Соренсена-Дайса дозволяє нейромережі збільшувати значення метрики Жаккара без перенавчання (рисунок 3.5).

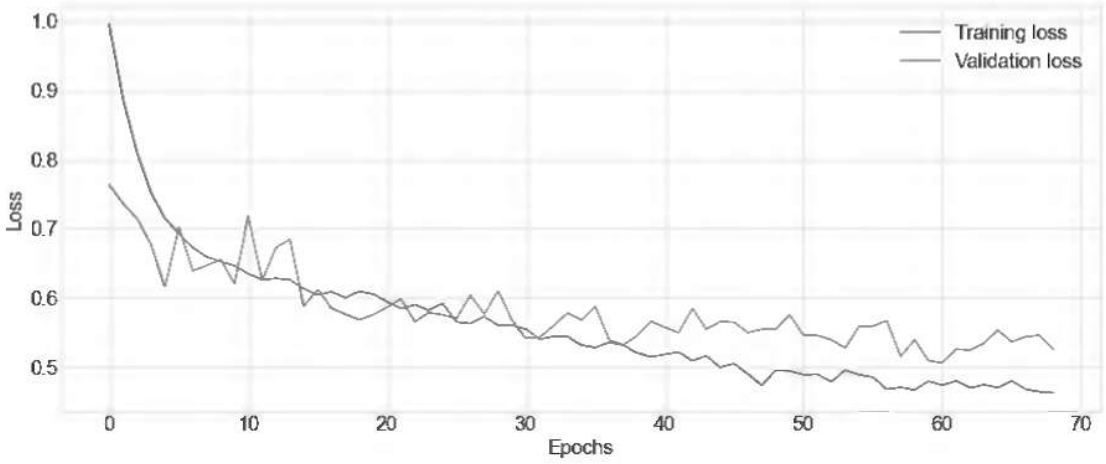

Рисунок 3.5 – Крива функції втрат

Після навчання ансамблю впродовж 70 епох важливо обрати поріг чутливості (вказуватиме на наявність пневмотораксу), щоб зменшити значення FN (*False Negative*) і FP (*False Positive*), тобто неправильно розпізнаних зображень. Для цього потрібно знайти максимум функції залежності встановленого порогу від середнього значення коефіцієнту Жаккара на валідаційній вибірці. У нашому випадку дане значення порогу дорівнює 0,775. Після налаштування параметрів та навчання ансамблю було отримано значення коефіцієнту Жаккара *J*, які наведено у таблиці 3.1 та на рисунку 3.6.

Таблиця 3.1 – Оцінка ефективності моделі

Набір даних	*J*, %
Тренувальний	77,9 %
Валідаційний	77,6 %
Тестувальний	82,4 %

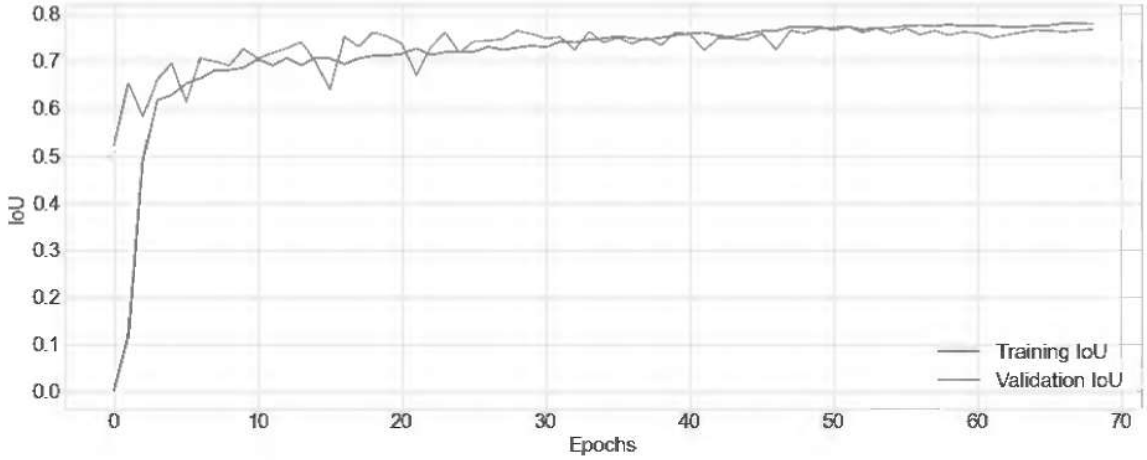

Рисунок 3.6 – Криві навчання та валідації

3.3 Висновки до розділу

У даному розділі оцінено ефективність розробленої моделі на реальних даних прикладної задачі діагностики пневмотораксу. Результати показали, що отримана модель забезпечує необхідну якість класифікації, а особливості побудови ансамблю, які регламентуються запропонованим підходом, дозволяють адаптуватися до зміни вхідних даних.

Отже, можна зробити висновок, що запропонований підхід до формування архітектури моделі на основі ансамблю автокодерів та ЗНМ дозволив підвищити точність детектування патологій для систем медичної діагностики на зображеннях.

ВИСНОВКИ

У даній роботі розглядаються питання, пов’язані з детектуванням пневмотораксу легень на рентгенівських зображеннях для застосування у комп’ютерній медичній діагностиці. Для цього використано ансамблювання автокодерів та ЗНМ.

У першому розділі розглянуто поширені складні хвороби легень та методи їх виявлення. Визначено, що виділяється підхід із застосуванням нейронних мереж для оброблення медичних знімків. У другому та третьому розділах представлено математичне, алгоритмічне та програмне забезпечення, а також розроблено оптимальну архітектуру мережі, яка ефективно розв’язує поставлену задачу. Основними результатами роботи є:

а) алгоритм для формування найбільш ефективної архітектури ЗНМ, заснований на наведених рекомендаціях до побудови архітектури високоточних класифікаторів;

б) підхід до створення адаптивного ансамблю на основі автокодерів та ЗНМ.

Одним із напрямків застосування результатів проведеного дослідження може бути розробка або модифікація комп'ютерних систем медичної діагностики.

Отримані результати дозволять збільшити точність детектування патологій і забезпечать адаптивність системи діагностики до зміни вхідних даних, що, в свою чергу, підвищить імовірність успішного лікування пацієнта. Дана робота робить внесок в розвиток досліджень, пов'язаних з комп'ютерним зором і глибоким машинним навчанням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання методів машинного навчання в медичній діагностиці [Електронний ресурс]. – Режим доступу: http://elartu.tntu.edu.ua/bitstream /lib/28445/2/SNT_2019_Dorofey_V_Using_of_machine_learning_methods_28.pdf.

2. Методи інтелектуального аналізу даних для прийняття рішень щодо діагностування пацієнта [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/23990/1/Malashenko_magistr.pdf.

3. Doi K. Computer-Aided Diagnosis in Medical Imaging: Historical Review, Current Status and Future Potential // Computerized Medical Imaging and Graphics. 2007. Vol. 31. № 4-5. P. 198–211.

4. Blumenfeld A., Greenspan H., Konen E. Pneumothorax Detection in Chest Radiographs Using Convolutional Neural Networks // Medical Imaging 2018: Computer-Aided Diagnosis. 2018.

5. Golan R., Jacob C., Denzinger J. Lung Nodule Detection in CT Images Using Deep Convolutional Neural Networks // 2016 International Joint Conference on Neural Networks (IJCNN). 2016.

6. Golosio B. et al. A Novel Multithreshold Method for Nodule Detection in Lung CT // Medical Physics. 2009. Vol. 36. № 8. P. 3607–3618.

7. Півошенко В.В., Здітовецький Ю.С., Іванов Ю.Ю., Кривогубченко С.Г. Оптимізація глибокої нейронної мережі на основі використання циклічно-косинусоїдної зміни коефіцієнта швидкості навчання та ансамблювання моделей [Електронний ресурс]: матер. XLIX науково-технічної конференції факультету комп'ютерних систем і автоматики. Вінниця: ВНТУ, 11-13 березня, 2020. – Режим доступу:https://conferences.vntu.edu.ua/index.php/allfksa/allfksa2020/paper/view/8719.

8. Искусственный интеллект в медицине: как машинное обучение ставит диагнозы [Электронный ресурс]. – Режим обращения: https://aiconference.com.ua /ru/news/iskusstvenniy-intellekt-v-meditsine-kak-mashinnoe-obuchenie-stavit-diagnozi-97639.

9. The Global Impact of Respiratory Disease [Web resource]. – Access mode: https://www.who.int/gard/publication/The_Global_Impact_of_Respiratory_Disease.pdf.

10. Травма органів грудної клітки – пневмоторакс [Електронний ресурс]. Режим доступу: https://urgent.com.ua/ua-issue-article-644.

11. Liang J., Bi J. Computer Aided Detection of Pulmonary Embolism with Tobogganing and Mutiple Instance Classification in CT Pulmonary Angiography // Lecture Notes in Computer Science Information Processing in Medical Imaging. P. 630–641.

12. Wittenberg R. et al. Computer-Assisted Detection of Pulmonary Embolism: Evaluation of Pulmonary CT Angiograms Performed in On-call Setting // European Radiology. 2009. Vol. 20, № 4. P. 801–806.

13. Özkan H. et al. A Novel Method for Pulmonary Embolism Detection in CTA Images // Computer Methods and Programs in Biomedicine. 2014. Vol. 113, № 3. P. 757–766.

14. Nibali A., He Z., Wollersheim D. Pulmonary Nodule Classification with Deep Residual Networks // International Journal of Computer Assisted Radiology and Surgery. 2017. Vol. 12, № 10. P. 1799–1808.

15. Choudhary P., Hazra A. Chest Disease Radiography in Twofold: Using Convolutional Neural Networks and Transfer Learning // Evolving Systems. 2019.

16. Generating Sequences with Recurrent Neural Networks [Web resources]. – Access mode: https://arxiv.org/abs/1308.0850.

17. Lecun Y. et al. Gradient-Based Learning Applied to Document Recognition // Proceedings of the IEEE. 1998. Vol. 86, № 11. P. 2278–2324.

18. Нейронні мережі - шлях до глибинного навчання [Електронний ресурс]. – Режим доступу: https://codeguida.com/post/739/.

19. Becherer N. et al. Improving Optimization of Convolutional Neural Networks Through Parameter Fine-Tuning // Neural Computing and Applications. 2017. Vol. 31, № 8. P. 3469–3479.

20. Venkatesan R., Li B. Convolutional Neural Networks // Convolutional Neural Networks in Visual Computing. 2017. P. 89–116.

21. Функции активации нейросети [Электронный ресурс]. – Режим обращения: https://neurohive.io/ru/osnovy-data-science/activation-functions/.

22. Wilusz T. Neural Networks – Comprehensive Foundation // Neurocomputing. 1995. Vol. 8, № 3. P. 359–360.

23. Hertz J., Krogh A., Palmer R.G. Formal Statistical Mechanics of Neural Networks // Introduction to the Theory of Neural Computation. 2018. P. 251–274.

24. Wu H., Gu X. Max-Pooling Dropout for Regularization of Convolutional Neural Networks // Neural Information Processing Lecture Notes in Computer Science. 2015. P. 46–54.

25. Poernomo A., Kang D.-K. Biased Dropout and Crossmap Dropout: Learning Towards Effective Dropout Regularization in Convolutional Neural Network // Neural Networks. 2018. Vol. 104. P. 60–67.

26. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks // Communications of the ACM. 2017. № 6. P. 84–90.

27. Going Deeper with Convolutions - arXiv [Web resource]. – Access mode: https://arxiv.org/pdf/1409.4842.pdf.

28. Large Scale Visual Recognition Challenge (ILSVRC) - ImageNet [Web resource]. – Access mode: http://image-net.org/challenges/LSVRC/.

29. Deep Residual Learning for Image Recognition [Web resource]. – Access mode: https://arxiv.org/pdf/1512.03385.pdf.

30. Bottou L. Large-Scale Machine Learning with Stochastic Gradient Descent // Proceedings of COMPSTAT'2010. 2010. P. 177–186.

31. He S., Wu Q.H., Saunders J.R. A Group Search Optimizer for Neural Network Training // Computational Science and Its Applications - ICCSA 2006 Lecture Notes in Computer Science. 2006. P. 934–943.

32. Adam: A Method for Stochastic Optimization - Open Access. [Web resource]. – Access mode: http://www.oalib.com/paper/4068193.

33. Обучение без учителя: автокодировщики, ограниченные машины Больцмана, разверточные сети [Электронный ресурс]. – Режим обращения: http://hpc-education.unn.ru/files/courses/intel-neon-course/Rus/Lectures/Presentations/6_UnsupervisedLearning.pdf.

34. Huang G., Reichel L., Yin F. On the Choice of Solution Subspace for Nonstationary Iterated Tikhonov Regularization // Numerical Algorithms. 2015. Vol. 72, № 4. P. 1043–1063.

35. Байесовская регуляризация рекуррентных нейронных сетей [Электронный ресурс]. – Режим обращения: http://www.machinelearning.ru/wiki/images/7/79/2018_617_ChirkovaNA.pdf.

36. Meng L., Ding S., Xue Y. Research on Denoising Sparse Autoencoder // International Journal of Machine Learning and Cybernetics. 2016. № 5. P. 1719–1729.

37. Вариационные автокодировщики [Электронный ресурс]. – Режим обращения: https://habr.com/ru/post/429276/.

38. An Introduction to Variational Autoencoders [Web resource]. – Access mode: https://arxiv.org/abs/1906.02691.

39. Rozložník M. Solution Approaches for Saddle-Point Problems // Nečas Center Series Saddle-Point Problems and Their Iterative Solution. 2018. P. 33–39.

40. Kawaguchi K., Huang J., Kaelbling L.P. Effect of Depth and Width on Local Minima in Deep Learning // Neural Computation. 2019. Vol. 31, № 7. P. 1462–1498.

41. TensorFlow. Open Source Software Library for Machine Intelligence [Web resource]. – Access mode: https://www.tensorflow.org.

42. Півошенко В.В., Іванов Ю.Ю., Кривогубченко С.Г., Іванов І.Ю. Особливості розв'язання задачі мультикласифікації епітеліом шкіри з використанням нейронних мереж: матер. V міжнародної наукової конференції "Вимірювання, контроль та діагностика в технічних системах" (ВКДТС – 2019). Збірник тез доповідей. Вінниця: ВНТУ, 29-31 жовтня, 2019. С. 62. – Режим доступу: http://mpa.vntu.edu.ua/wp-content/uploads/Abstracts_2019.pdf.

ДОДАТКИ

					08-02.БДР.010.00.000 ПД		
					Розробка програмного забезпечення для автоматичного детектування пневмотораксу із використанням нейронних мереж.	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Півошенко В.В.					
Перевір.		Іванов Ю. Ю.				Арк.	Аркушів 1
Т. Контр.		Іванов Ю. Ю.			*Архітектура нейронної мережі*		
Реценз.						*ВНТУ, гр. 1СІ-16б*	
Н. Контр.		Іванов Ю. Ю.					
Затверд.		Квєтний Р.Н.					

Додаток А

(обов'язковий)

Архітектура нейронної мережі

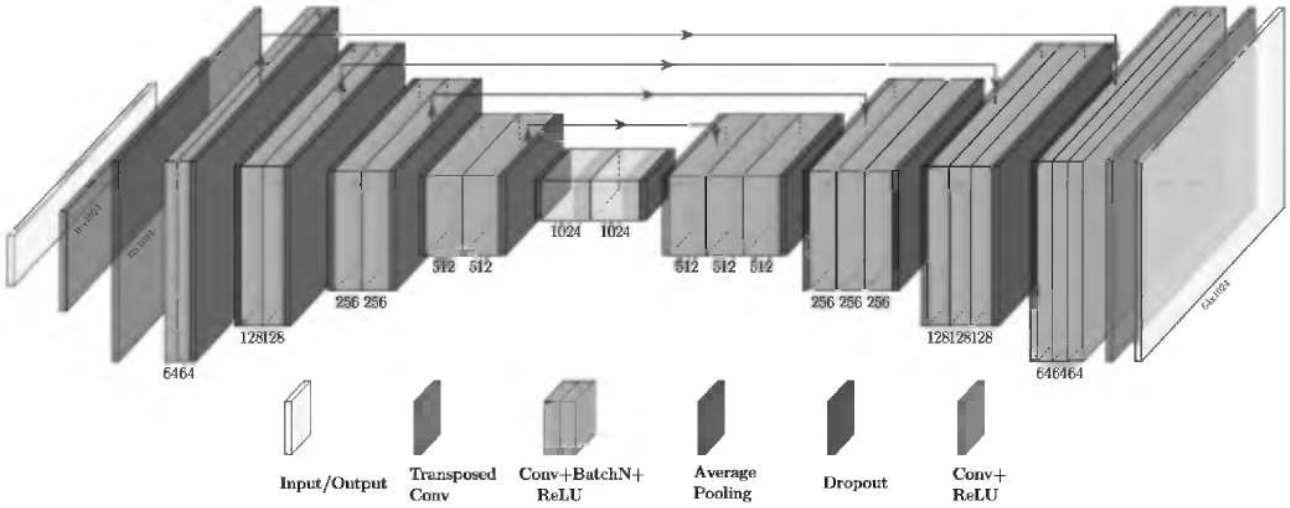

Рисунок А.1 – Архітектура нейронної мережі

					08-02.БДР.010.00.000 ПД		
					Розробка програмного забезпечення для автоматичного детектування пневмотораксу із використанням нейронних мереж. Структура для роботи з програмним забезпеченням	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Півошенко В.В.				Арк.	Аркушів 1	
Перевір.		Іванов Ю. Ю.						
Т. Контр.		Іванов Ю. Ю.						
Реценз.						*ВНТУ, гр. 1СІ-16б*		
Н. Контр.		Іванов Ю. Ю.						
Затверд.		Квєтний Р.Н.						

Додаток Б

(обов'язковий)

Структура для роботи з програмним забезпеченням

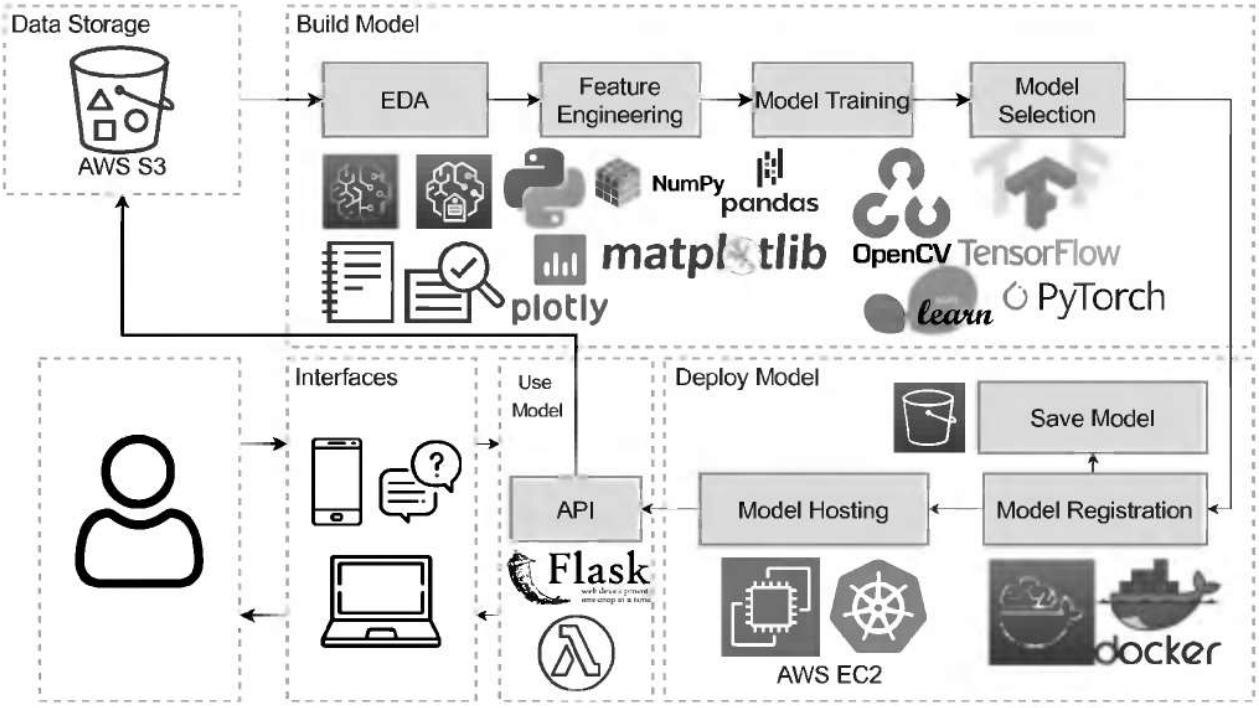

Рисунок Б.1 – Структура для роботи з програмним забезпеченням

					08-02.БДР.010.00.000 ПД		
					Розробка програмного забезпечення для автоматичного детектування пневмотораксу із використанням нейронних мереж.	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Півошенко В.В.					
Перевір.		Іванов Ю. Ю.					
Т. Контр.		Іванов Ю. Ю.			*Приклад вихідних даних*	Арк.	Аркушів 1
Реценз.							
Н. Контр.		Іванов Ю. Ю.				*ВНТУ, гр. 1СІ-16б*	
Затверд.		Квєтний Р.Н.					

Додаток В

(обов'язковий)

Приклад вихідних даних

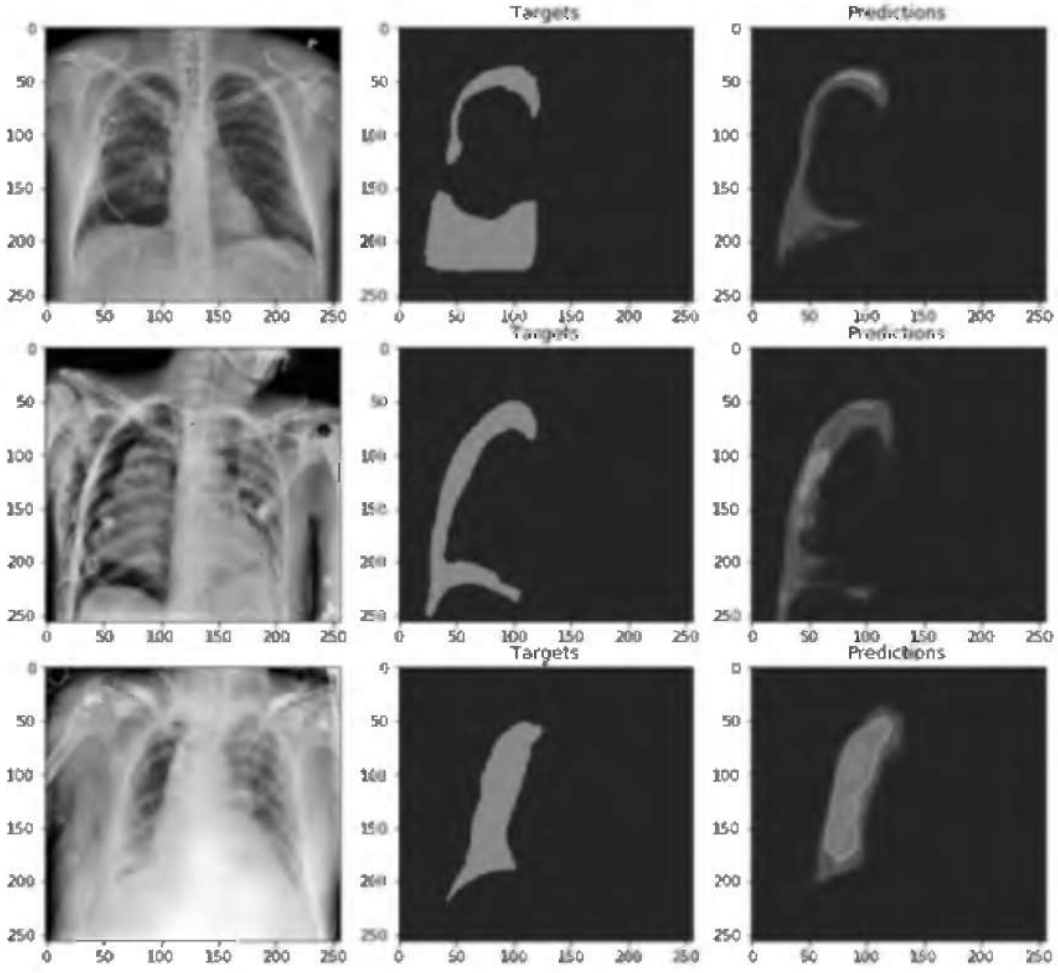

Рисунок В.1 – Приклад вихідних даних

Додаток Г

(обов'язковий)

Лістинг програмного забезпечення

```
import os
import random

import numpy as np
import tensorflow as tf

__author__ = 'Volodymyr Pivohsenko'

class Model:

    @staticmethod
    def model_architecture(input_shape=(None, None, 3),
                           dropout_rate: float = 0.1) -> tf.keras.Model:
        """
        Build model architecture

        Parameters
        ----------
        input_shape : Tuple[int, int, int]
            Input shape of network
        dropout_rate : float
            Rate of dropouts
        Returns
        -------
        tf.keras.Model
            Model
        """

        def convolution_block(previous_layer: tf.keras.layers.Layer,
                              filters: int,
                              size: Tuple[int, int],
                              strides: Tuple[int, int] = (1, 1),
                              padding: str = 'same',
                              activation: bool = True) -> tf.keras.layers.Layer:
            """
            Convolution block

            Parameters
            ----------
            previous_layer : tf.keras.layers.Layer
                Previous layer
            filters : int
                Amount of filters
            size : Tuple[int, int]
                Size of filters
            strides : Tuple[int, int]
                Stride
            padding : str
                Padding
            activation : bool
```

```
        Whether to use activation
    Returns
    -------
    tf.keras.layers.Layer
        Convolution block
    """
    convolution_block_layer = Conv2D(
        filters, size,
        strides=strides,
        padding=padding)(previous_layer)

    convolution_block_layer = BatchNormalization()(convolution_block_layer)

    if activation:
        convolution_block_layer = LeakyReLU(alpha=0.1)(convolution_block_layer)

    return convolution_block_layer

def residual_block(previous_layer: tf.keras.layers.Layer,
                   filters: int = 16) -> tf.keras.layers.Layer:
    """
    Residual block

    Parameters
    ----------
    previous_layer : tf.keras.layers.Layer
        Previous layers
    filters : int
        Filters
    Returns
    -------
    tf.keras.layers.Layer
        Residual block
    """
    layer = LeakyReLU(
        alpha=0.1)(previous_layer)

    layer = BatchNormalization()(layer)

    previous_layer = BatchNormalization()(previous_layer)

    layer = convolution_block(
        layer, filters, (3, 3))

    layer = convolution_block(
        layer, filters, (3, 3), activation=False)

    layer = Add()([layer, previous_layer])

    return layer

backbone = EfficientNetB4(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape)

input_layer = backbone.input
```

```
convolution_layer_0 = backbone.layers[342].output

convolution_layer_0 = LeakyReLU(
    alpha=0.1)(convolution_layer_0)

pool_layer_0 = MaxPooling2D(
    (2, 2))(convolution_layer_0)

pool_layer_0 = Dropout(
    dropout_rate)(pool_layer_0)

convolution_layer_1 = Conv2D(
    (8 * 32), (3, 3),
    activation=None,
    padding='same')(pool_layer_0)

convolution_layer_1 = residual_block(
    convolution_layer_1, (8 * 32))

convolution_layer_1 = residual_block(
    convolution_layer_1, (8 * 32))

convolution_layer_1 = LeakyReLU(
    alpha=0.1)(convolution_layer_1)

convolution_transpose_layer_0 = Conv2DTranspose(
    (8 * 16), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_layer_1)

convolution_transpose_layer_1 = Conv2DTranspose(
    (8 * 16), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_0)

convolution_transpose_layer_2 = Conv2DTranspose(
    (8 * 16), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_1)

convolution_transpose_layer_3 = Conv2DTranspose(
    (8 * 16), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_2)

convolution_layer_2 = concatenate([
    convolution_transpose_layer_0,
    convolution_layer_0])

convolution_layer_2 = Dropout(
    dropout_rate)(convolution_layer_2)

convolution_layer_2 = Conv2D(
    (8 * 16), (3, 3),
    activation=None,
    padding='same')(convolution_layer_2)
```

```
convolution_layer_2 = residual_block(
    convolution_layer_2, (8 * 16))

convolution_layer_2 = LeakyReLU(
    alpha=0.1)(convolution_layer_2)

convolution_transpose_layer_4 = Conv2DTranspose(
    (8 * 8), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_layer_2)

convolution_transpose_layer_5 = Conv2DTranspose(
    (8 * 8), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_4)

convolution_transpose_layer_6 = Conv2DTranspose(
    (8 * 8), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_5)

convolution_layer_3 = backbone.layers[154].output

convolution_layer_3 = concatenate([
    convolution_transpose_layer_4,
    convolution_transpose_layer_1,
    convolution_layer_3])

convolution_layer_3 = Dropout(
    dropout_rate)(convolution_layer_3)

convolution_layer_3 = Conv2D(
    (8 * 8), (3, 3),
    activation=None,
    padding='same')(convolution_layer_3)

convolution_layer_3 = residual_block(
    convolution_layer_3, (8 * 8))

convolution_layer_3 = LeakyReLU(
    alpha=0.1)(convolution_layer_3)

convolution_transpose_layer_7 = Conv2DTranspose(
    (8 * 4), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_layer_3)

convolution_transpose_layer_8 = Conv2DTranspose(
    (8 * 4), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_transpose_layer_7)

convolution_layer_4 = backbone.layers[92].output

convolution_layer_5 = concatenate([
    convolution_transpose_layer_7,
```

```
    convolution_transpose_layer_5,
    convolution_transpose_layer_2,
    convolution_layer_4])

convolution_layer_5 = Dropout(0.1)(convolution_layer_5)

convolution_layer_5 = Conv2D(
    (8 * 4), (3, 3),
    activation=None,
    padding='same')(convolution_layer_5)

convolution_layer_5 = residual_block(
    convolution_layer_5, (8 * 4))

convolution_layer_5 = LeakyReLU(
    alpha=0.1)(convolution_layer_5)

convolution_transpose_layer_9 = Conv2DTranspose(
    (8 * 2), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_layer_5)

convolution_layer_6 = backbone.layers[30].output

convolution_layer_7 = concatenate([
    convolution_transpose_layer_9,
    convolution_transpose_layer_8,
    convolution_transpose_layer_6,
    convolution_transpose_layer_3,
    convolution_layer_6])

convolution_layer_7 = Dropout(0.1)(convolution_layer_7)

convolution_layer_7 = Conv2D(
    (8 * 2), (3, 3),
    activation=None,
    padding='same')(convolution_layer_7)

convolution_layer_7 = residual_block(
    convolution_layer_7, (8 * 2))

convolution_layer_7 = LeakyReLU(
    alpha=0.1)(convolution_layer_7)

convolution_transpose_layer_10 = Conv2DTranspose(
    (8 * 1), (3, 3),
    strides=(2, 2),
    padding='same')(convolution_layer_7)

convolution_transpose_layer_10 = Dropout(0.1)(convolution_transpose_layer_10)

convolution_transpose_layer_10 = Conv2D(
    (8 * 1), (3, 3),
    activation=None,
    padding='same')(convolution_transpose_layer_10)

convolution_transpose_layer_10 = residual_block(
```

```
            convolution_transpose_layer_10, (8 * 1))

        convolution_transpose_layer_10 = LeakyReLU(
            alpha=0.1)(convolution_transpose_layer_10)

        convolution_transpose_layer_10 = Dropout(dropout_rate / 2)(convolution_transpose_l
ayer_10)

        output_layer = Conv2D(
            1, (1, 1),
            padding='same',
            activation='sigmoid')(convolution_transpose_layer_10)

        model = Model(input_layer, output_layer)

        return model
```