МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Лящинський Петро Борисович

Модуль розпаралелення алгоритмів навчання згорткових нейронних мереж / The software module of learning algorithms parallelization for convolution neural networks

напрям підготовки: 6.050102 - Комп’ютерна інженерія
фахове спрямування - Комп’ютерні системи та мережі
Бакалаврська робота

Виконав студент групи КСМ-
Лящинський Петро Борисович

Науковий керівник: д.т.н.,
Березький О.М..

Тернопіль - 2018

РЕЗЮМЕ

Бакалаврська робота містить 54 сторінки, 23 рисунки, 17 таблиць, 12 додатків. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою бакалаврської роботи є розробка програмного модуля для паралельного навчання згорткових нейронних мереж з використанням графічного процесора.

Розроблено модуль програмно-апаратної системи, що забезпечує:

- вибір/створення моделі згорткової нейронної мережі для навчання;
- отримання результатів навчання мережі;
- класифікацію зображень на основі навченої моделі;
- паралельне навчання моделі нейронної мережі;
- графічний інтерфейс для взаємодії з користувачем;
- можливість задання параметрів навчання мережі.

Для написання програмного коду використовувалася мова програмування Java 8, фреймворк JavaFX та середовище програмування Intellij Idea. Програмний код написаний з використанням архітектурного шаблону Model-View-Controller (MVC), що передбачає розподіл функціоналу на 3 окремі класи: для взаємодії з даними, для взаємодії з графічним інтерфейсом, для керування моделлю та графічним інтерфейсом.

Для експериментів було вибрано два види зображень: цитологічні та гістологічні. Зображення кожного виду були поділені на 5 класів. Кількість цитологічних зображень 80, гістологічних – 91. Показано що при опрацюванні зображень різного розміру графічний процесор дає прискорення приблизно в чотири рази.

Ключові слова: ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, РОЗПАРАЛЕЛЕННЯ, ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ, КЛАСИФІКАЦІЯ, GPU, JAVAFX.

RESUME

The bachelor's thesis contains 54 pages, 23 figures, 17 tables, 12 appendices. Volume of graphic material 2 sheets of A3 format.

The purpose of the bachelor's thesis is to develop a software module for parallel learning of convolutional neural networks using a graphics processor.

A module of software and hardware system has been developed, which provides:

- selection / creation of a model of a convolutional neural network for training;
- obtaining network learning results;
- classification of images based on the trained model;
- parallel learning of the neural network model;
- graphical interface for interaction with the user;
- the ability to set network learning parameters.

Java 8 programming language, JavaFX framework and Intellij Idea programming environment were used to write the program code. The program code is written using the architectural template Model-View-Controller (MVC), which provides for the division of functionality into 3 separate classes: for interaction with data, for interaction with the graphical interface, for model management and graphical interface.

Two types of images were chosen for the experiments: cytological and histological. Images of each species were divided into 5 classes. The number of cytological images is 80, histological - 91. It is shown that when processing images of different sizes, the graphics processor gives an acceleration of about four times.

Keywords: ROLLED NEURAL NETWORK, PARALLELING, IMAGE PROCESSING, CLASSIFICATION, GPU, JAVAFX.

ЗМІСТ

ВСТУП

Зростаюча складність обчислювальних задач вимагає різкого збільшення ресурсів і швидкодії комп’ютерів. Найбільш перспективним напрямком збільшення швидкості розв’язку прикладних завдань є широке впровадження ідей паралелізму в роботу обчислювальних систем. Сьогодні спроектовані і випробувані сотні різних комп'ютерів, що використовують у своїй архітектурі той чи інший вид паралельної обробки даних. Основна складність при проектуванні паралельних програм – це забезпечення правильної послідовності взаємодій між різними обчислювальними процесами, а також координація ресурсів, що розділяються між ними.

Останнім часом дуже великої популярності набуває застосування нейронних мереж у різних сферах суспільства та економіки. Особливої актуальності набули задачі класифікації різного роду інформації. Сучасні класифікатори працюють на основі нейронних мереж, а для класифікації зображень використовують згорткові нейронні мережі.

Основною проблемою при навчанні згорткових нейронних мереж є великий час навчання. Тому, для розв’язку цієї проблеми використовують графічні процесори, які в декілька разів є потужнішими за центральні. У дипломному проекті клас біомедичних зображень – цитологічні та гістологічні зображення, які використовуються для діагностування патологічних станів в онкології, що є актуальною проблемою в медицині.

1 АНАЛІЗ ТЕХНОЛОГІЙ ТА АПАРАТНИХ ЗАСОБІВ РОЗПАРАЛЕЛЕННЯ ІНФОРМАЦІЇ

1.1 Технології розпаралелення інформації

Складність обчислювальних завдань вимагає різкого збільшення ресурсів і швидкодії комп’ютерів. Найбільш перспективним напрямком збільшення швидкості розв’язку прикладних завдань є широке впровадження ідей паралелізму в роботу обчислювальних систем. Сьогодні спроектовано і випробувано сотні різних комп'ютерів, що використовують у своїй архітектурі той чи інший вид паралельної обробки даних. Основна складність при проектуванні паралельних програм – забезпечення правильної послідовності взаємодій між різними обчислювальними процесами, а також координація ресурсів, що розділяються між ними [1].

Більшість сучасних мікропроцесорних архітектур використовують різні методи підвищення продуктивності виконуваних програм за рахунок їх розпаралелення, а саме:

- конвеєризація виконання операцій - розбиття процесу виконання на стадії і одночасне виконання операцій, які перебувають на різних стадіях конвеєра;
- одночасне виконання декількох операцій, що знаходяться на однаковій стадії конвеєрного виконання;
- підтримка багато потокового виконання команд всередині одного процесорного ядра, на декількох процесорних ядрах або в багатопроцесорній системі, що працює на загальній пам’яті;
- застосування одної операції до декількох даних одночасно;
- підтримка такої багатопроцесорної системи, в якій якій процесори можуть працювати одночасно, використовуючи локальну чи загальну пам’ять.

Існують різні способи реалізації паралельних обчислень. Наприклад, кожен обчислювальний процес може бути реалізований у вигляді окремого процесу операційної системи, або обчислювальні процеси можуть являти собою набір потоків виконання всередині одного процесу ОС. Паралельні програми можуть фізично виконуватися або послідовно на єдиному процесорі - по черзі виконувати операції кожного обчислювального процесу, або паралельно - виділяючи кожному обчислювальному процесу один або кілька процесорів (що знаходяться в цій системі або розподілених в комп'ютерну мережу) [8].

Зараз набувають популярності обчислення на графічних процесорах. Наприклад, у [23] описано застосування графічних процесорів для аналізу медичних зображень. Архітектура GPU побудована трохи інакше, ніж CPU. Оскільки графічні процесори спочатку використовувалися тільки для графічних розрахунків, що припускають незалежну паралельну обробку даних, то GPU призначений саме для паралельних обчислень. Він спроектований так, щоб виконувати велику кількість потоків (елементарних паралельних процесів) [1].

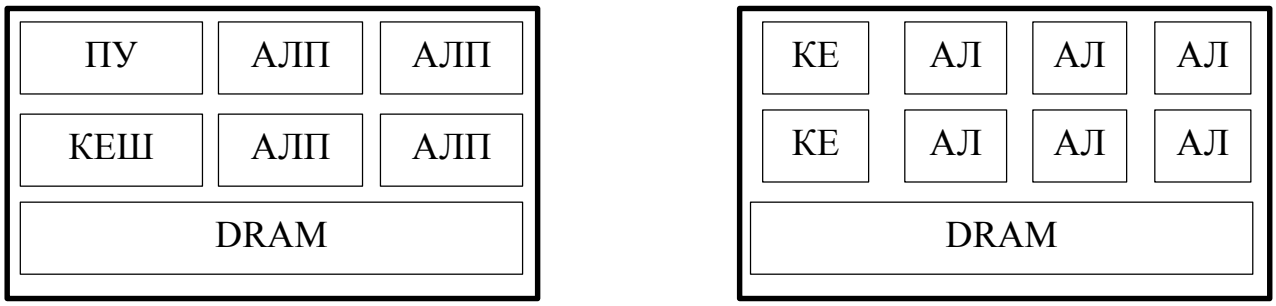

Рисунок 1.1 – Архітектура CPU (зліва) та GPU (справа)

Як видно з рисунка 1.1 – GPU містить дуже багато простих арифметико-логічних пристроїв (АЛП), які об'єднані в декілька груп і мають спільну пам'ять. Це допомагає підвищити продуктивність в обчислювальних завданнях, але трохи ускладнює програмування.

GPU орієнтований на виконання програм з великим обсягом даних та розрахунків і являє собою масив потокових процесорів (Streaming Processor Array), що складається з кластерів текстурних процесорів (Texture Processor

Clusters, TPC). TPC складається з набору мультипроцесорів (SM – Streaming Multi-processor), кожен з яких містить кілька потокових процесорів (SP – Streaming Processors) або ядер (у сучасних графічних процесорах к-сть ядер перевищує 1024). Набір ядер кожного мультипроцесора працює за принципом SIMD (проте, з деякою відмінністю) – реалізація, що дозволяє групі процесорів, які працюють паралельно, працювати з різними даними, але при цьому всі вони в будь-який момент часу повинні виконувати однакову команду [1].

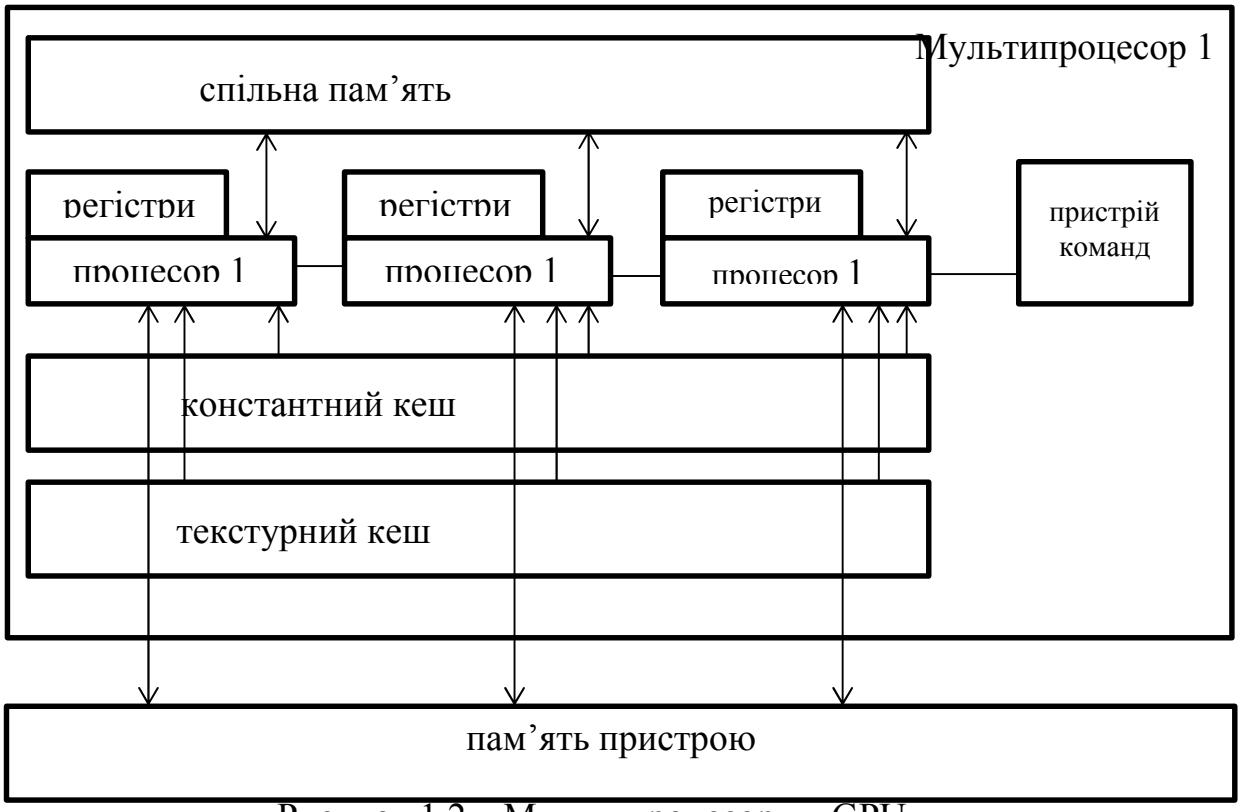

Рисунок 1.2 – Мультипроцесори в GPU

В результаті GPU фактично став пристроєм, що реалізує потокову обчислювальну модель (stream computing model) – є потоки вхідних і вихідних даних, що складаються з однакових елементів, які можуть бути оброблені незалежно один від одного. На рисунку 1.3 зображено потокову модель GPU [1].

Рисунок 1.3 – Потокова модель виконання команд

На даний час існують дві потужні компанії з виготовлення графічних процесорів – AMD та Nvidia. Відповідно, існує і дві технології для розпаралелення інформації використовуючи графічні процесори – AMD FireStream (на OpenCL) та CUDA.

CUDA і OpenCL пропонують два різних інтерфейси для програмування графічних процесорів. OpenCL - це відкритий стандарт, який можна використовувати для програмування CPU, GPU та інших пристроїв різних постачальників, тоді як CUDA - специфічна для NVIDIA GPU. При використанні інструментів компілятора NVIDIA, перетворення ядра CUDA в ядро OpenCL передбачає мінімальні модифікації. Створення такого ядра з інструментами компіляції AMD передбачає більшу кількість модифікацій [13]. Незважаючи на те, що OpenCL підтримує графічні процесори різних виробників – у роботі «A Performance Comparison of CUDA and OpenCL» було продемонстровано, що OpenCL програє CUDA у швидкодії та продуктивності в межах від 13% до 63% [13].

Технологія CUDA використовує дуже багато окремих потоків для розрахунків, що групуються в ієрархію сітка/блок/потік [1].

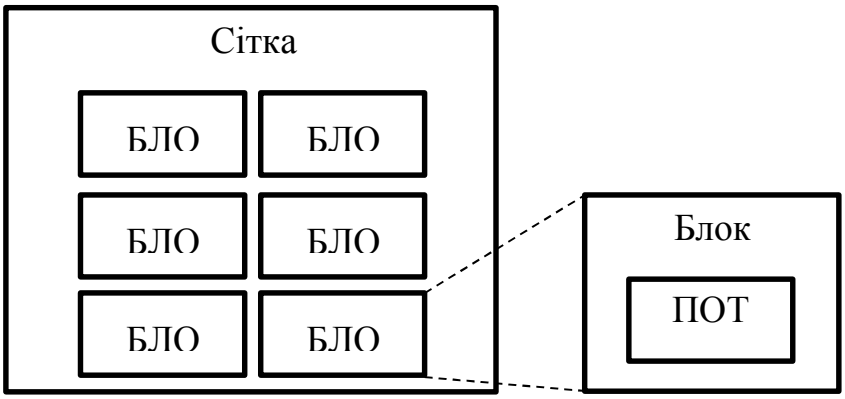

Рисунок 1.4 – Ієрархія потоків

Верхній рівень – grid – відповідає рівню ядра та об'єднує всі потоки, що виконують дане ядро. Grid – одновимірний або двомірний масив блоків (block). Кожен блок (block) являє собою 1 / 2 / 3 - мірний масив потоків (threads). При цьому кожен блок являє собою повністю незалежний набір скоординованих між собою потоків, але потоки з різних блоків не можуть між собою взаємодіяти.

Вище було згадано про відмінність від SIMD архітектури. Існує таке поняття, як warp – група з 32 потоків (залежно від архітектури GPU, але переважно 32). Тільки потоки в межах однієї групи (варпу) можуть фізично виконуватися одночасно. Потоки з різних варпів можуть знаходитися на різних стадіях виконання програми. Такий метод обробки даних називають терміном SIMT (Single Instruction – Multiple Theads). Управління роботою варпів виконується на апаратному рівні [1].

Перевагами технології CUDA на OpenCL є:

- Інтерфейс програмування додатків CUDA (CUDA API) заснований на стандартній мові програмування С з деякими обмеженнями. На думку розробників, це повинно спростити і згладити процес вивчення архітектури CUDA;
- Колективна між потоками пам'ять (shared memory) розміром в 16 Кб може бути використана під організований користувачем кеш з більш широкою смугою пропускання, ніж при вибірці зі звичайних текстур;
- Більш ефективні транзакції між пам'яттю центрального процесора і відеопам'яттю;
- Повна апаратна підтримка цілочисельних і побітових операцій

1.2 Аналіз графічних процесорів

Кожен графічний процесор має так звані обчислювальні можливості – кількісна характеристика швидкості виконання певних операцій на графічному процесорі. Це число показує те, наскільки швидко відеокарта буде виконувати свою роботу.

В таблиці 1.1 наведені деякі відеокарти й відповідні їм аерсії обчислювальних можливостей.

Таблиця 1.1 - Відеокарти Nvidia

Обчислювальні можливості (версія)	Графічний процесор	Відеокарта
1.0	G80, G92, G92b, G94, G94b	GeForce 8800GTX/Ultra, Tesla C/D/S870
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT/GTS, 9400GT, 9600 GSO, 9600GT, 9800GTX/GX2, 9800GT, GTS 250, GT 120/30/40, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50
1.2	GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M
1.3	GT200, GT200b	GeForce GTX 260, GTX 275, GTX 280, GTX 285, GTX 295, Tesla C/M1060, S1070, Quadro CX, FX 3/4/5800
2.0	GF100, GF110	GeForce (GF100) GTX 465, GTX 470, GTX 480, Tesla C2050, C2070, S/M2050/70, Quadro Plex 7000, Quadro 4000, 5000, 6000, GeForce (GF110) GTX 560 TI 448, GTX570, GTX580, GTX590
5.0	GM107, GM108	GeForce GTX 750 Ti, GeForce GTX 750, GeForce GTX 860M,

Наведені графічні процесори відрізняються між собою, перш за все, обчислювальними можливостями. Загалом, на обчислювальні можливості кожного графічного процесора впливає кількість ядер CUDA, пам'ять, її тип та частота і багато інших факторів. Компанія NVIDIA також випускає графічні процесори, які спеціально розробляються для складних розрахунків – сімейство Quadro та Tesla.

У таблиці 1.2 наведено деякі відео карти компанії AMD (ATI).

Таблиця 1.2 – Відеокарти AMD

Найменування	Серія
R7000-R7200	R100
R8500,R9000-R9250	R200
R9500-R9800, X300-X600, X1050	R300
R5 210-230, R7 240-260, R9 270-290	Volcanic Islands
R5 3xx, R7 3xx, R9 3xx, R9 Nano / Fury / Fury X	Pirate Islands, Caribbean Islands
AMD Radeon RX 400 series	Arctic Islands, Polaris
AMD Radeon RX 500 series	Polaris

Для прикладу, порівняємо дві відеокарти середнього цінового сегменту. Розглянемо Radeon R9 270x від компанії AMD, та GeForce GTX 950 – від компанії NVIDIA. Результати порівняння наведені у таблиці 1.3.

Таблиця 1.3 – Порівняння відеокарт

Найменування	Radeon R9 270x	GeForce GTX 950
Ядро	Curacao	GM206
Техпроцес (мкм)	0,028	0,028
Транзисторів (млн)	2800	2940
Частота ядер (МГц)	1050	1024-1188

Продовження таблиці 1.3

Частота пам’яті (МГц)	1400	1650
Шина і тип пам’яті	256-bit GDDR5	128-bit GDDR5
Пропускна здатність (Гб/с)	179,2	105,6
Кількість ядер	1280	768
Обсяг пам’яті (Мб)	2048/4096	2048
DirectX	11.2	12
Інтерфейс	PCI 3.0	PCI 3.0

Як видно з таблиці 1.3, по певних параметрах виграє AMD, а по інших – NVIDIA. Майже в два рази GTX 950 програє конкуренту по кількості ядер. Але це не означає, що відеокарта в якої більше ядер працюватиме швидше. Тут велику роль відіграють частота пам’яті, частота ядра, затримка, архітектура, пропускна здатність та багато інших факторів.

Пропускна здатність – це головне поняття в продуктивності пам’яті відеокарти. Вимірюється в гігабайтах за секунду. Зараз активно використовується відеопам’ять GDDR5, яка має значно більший частотний потенціал, ніж GDDR3, а значить і більшу пропускну здатність. Проте, частота – не найголовніший показник. Другим важливим фактором є розрядність шини пам’яті. Чим вона більша, тим швидше працює пам’ять. Для прикладу, пам’ять з частотою 1000МГц і шиною 256bit в два рази швидша за пам’ять з частотою 1000МГц і шиною 128bit.

У проведеному порівнянні, майже по всіх параметрах виграє відеокарта від компанії AMD. Проте, для складних обчислень в більшості випадків використовують відеокарти NVIDIA, бо технологія CUDA, для програмування цих відеокарт, розроблена самою компанією NVIDIA, а значить добре оптимізована та дає можливість повністю розкрити потенціал графічного процесора.

1.3 Аналіз алгоритмів навчання згорткових нейронних мереж

Загалом, якість роботи нейронної мережі дуже сильно залежить від вхідних даних. Ці дані повинні бути типовими для завдання, яке вирішує мережа.

Дані, що використовуються для навчання нейронної мережі, зазвичай поділені на два класи – для навчання і для тестування. Процес тестування потрібно організувати так, щоб можна було оцінити здатність нейронної мережі узагальнювати отримані знання. Узагальнення в даному випадку – здатність нейронної мережі правильно класифікувати дані, що хоч і є аналогічними до даних, під час процесу навчання, але все ж відрізняються від них.

Навчання нейронної мережі – це процес, в якому параметри нейронної мережі налаштовуються за певним алгоритмом таким чином, щоб мережа на виході давала правильні результати. Тип навчання визначається способом підлаштування параметрів. Розрізняють алгоритми навчання з вчителем (на маркованих даних) і без вчителя [5].

Процес навчання з вчителем передбачає надання мережі вибірки навчальних даних, які вже розподілені на певні класи. Кожен зразок подається на входи мережі, потім проходить обробку всередині структури нейронної мережі, обчислюється вихідний сигнал мережі, який порівнюється з відповідним значенням цільового вектора, що представляє собою необхідний вихід мережі. Після цього, за певним правилом обчислюється помилка та відбувається зміна вагових коефіцієнтів всередині мережі в залежності від обраного алгоритму навчання [5].

При навчанні без вчителя навчальну множину складають лише з вхідних векторів. Навчальний алгоритм налаштовує вагові коефіцієнти мережі так, щоб можна було отримати узгоджені вихідні вектори, тобто щоб надання досить близьких вхідних векторів давало однакові вихідні результати. Процес навчання, виділяє певні властивості навчальної множини і групує подібні вектори в класи. Надання на вхід вектора з даного класу приведе до отримання

певного вихідного вектора, але до процесу навчання неможливо передбачити, який саме вихідний результат дасть певний клас вхідних векторів [5].

В загальному випадку, процес навчання нейронної мережі можна зобразити так, як показано на рисунку 1.5.

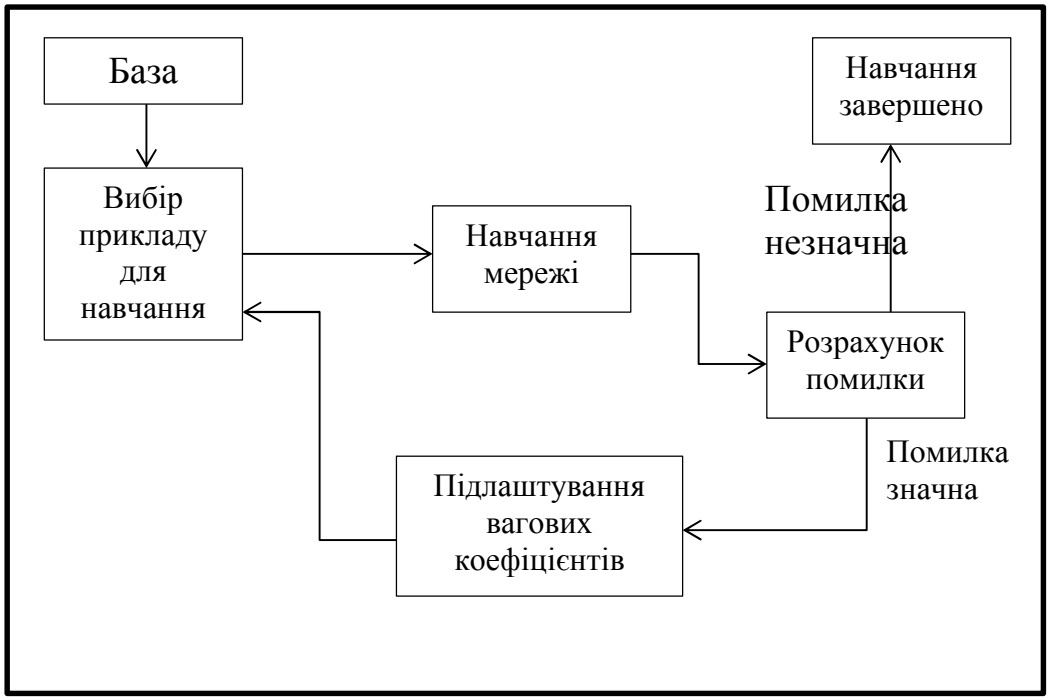

Рисунок 1.5 – Процес навчання нейронної мережі

Для навчання згорткових нейронних мереж застосовують градієнтні алгоритми навчання [6]. В основі цих методів покладений метод градієнтного спуску.

Найбільш популярним з цих алгоритмів є алгоритм зворотного розповсюдження помилки (backpropagation). Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи [6].

Метод зворотного поширення помилки можна розділити на 4 окремі блоки: пряме поширення, функцію втрати (помилки), зворотне поширення і оновлення вагових коефіцієнтів. Під час прямого поширення, береться тренувальне зображення – наприклад, це матриця 28 x 28 x 3 – і проходить

через всю мережу. Оскільки, на початку навчання вагові коефіцієнти ініціалізуються випадковим чином, то на виході буде таке значення, яке не зможе віднести зображення до певного класу. Мережа з такими ваговими коефіцієнтами не може обґрунтовано визначити клас зображення. Це приводить до функції втрат. Ця функція може бути будь-якою, але часто обирають функцію середньоквадратичної помилки. Під час навчання за таким методом, вхідні дані є маркованими, тобто вже розділеними на певні класи. Щоб досягти того, щоб вихідний клас (вихід нейронної мережі) був таким самим як клас (маркер) навчальних даних – потрібно мінімізувати помилку (рисунок 1.6) [11].

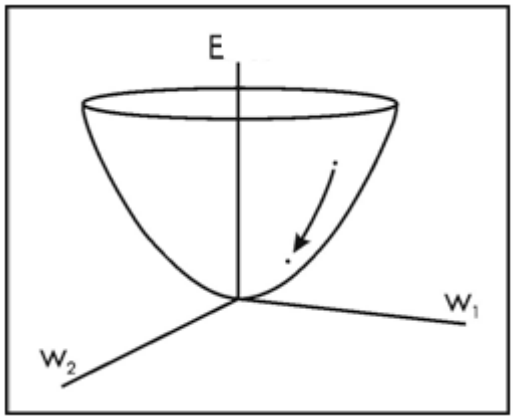

Рисунок 1.6 – Візуалізація функції помилки

На рисунку 1.6, w1 та w2 – незалежні змінні – вагові коефіцієнти, а E – залежна змінна, що позначає помилку. Завдання мінімізації помилки – відрегулювати вагові коефіцієнти так, щоб знизити цю помилку. Візуально, потрібно наблизитися до найнижчої точки графіку. Щоб досягти цього, потрібно знайти похідну втрати (в рамках намальованого графіка - розрахувати кутовий коефіцієнт в кожному напрямку з урахуванням ваг).

Тепер потрібно виконати зворотне поширення через мережу, яке визначить, які вагові коефіцієнти здійснили найбільший вплив на втрати, і знайти способи, як їх налаштувати, щоб зменшити ці втрати. Після цього етапу потрібно оновити відповідні вагові коефіцієнти.

Алгоритм зворотного розповсюдження помилки визначає два потоки в мережі: прямий потік від вхідного шару до вихідного і зворотний потік – від вихідного шару до вхідного. В загальному, метою навчання мережі є пошук таких вагових коефіцієнтів, які мінімізуватимуть помилку. Тобто мережі надається зразок і обчислюється вектор помилок, в результаті чого стає зрозумілим, наскільки потрібно змінити значення ваг; процес повторюється для кожного зразка. Повний цикл проходження всіх зразків називається епохою. Всі зразки подаються на входи мережі, епоха за епохою, поки на протязі однієї епохи всі значення реального виходу мережі не потраплять в допустимі рамки [7].

Також існують генетичні алгоритми навчання – це напрямок в штучному інтелекті, що заснований на моделюванні природного відбору [16].

За аналогією з природною еволюцією, в генетичних алгоритмах кандидати-рішення називаються індивідами або особинами, а безліч кандидатів-рішень – популяцією. Кожен індивід визначає можливе рішення задачі, при цьому, однак, сам по собі він не є вектором рішень, а скоріше кодує його, ґрунтуючись на відповідній структурі кодування рішення. У генетичних алгоритмах ця структура визначається вектором – вектором бітів або вектором дійсних чисел – набором генів, що утворюють хромосоми. Безліч всіх можливих векторів утворює простір індивідів або популяцію [16].

В межах цього методу використовують таку термінологію: коефіцієнти матриці ваг – геном, один коефіцієнт – ген, інвертована функція втрат – ландшафт пристосованості (тут шукаємо локальний максимум, а не мінімум). Цей метод справді дуже простий. Після вибору топології нейронної мережі, потрібно:

1. Проініціалізувати геном випадковими значеннями від -1 до 1. Це потрібно повторити декілька разів, для створення початкової популяції різних, проте, випадкових мереж;
2. Створити певну кількість потомків. Наприклад клонувати батьківський геном, але внести в нього незначні зміни;

3. Оцінити пристосованість кожного потомка, на основі того, як він справляється з прикладами з навчальної вибірки даних. Відсортувати потомків за їх пристосованістю;
4. Залишити лише певну кількість пристосованих потомків та повернутися до пункту 2, повторюючи цикл декілька разів, поки не буде досягнути задовільний результат.

Генетичний метод навчання дуже простий для розуміння. Зараз він використовується в основному для тренування глибоких шарів нейронної мережі. Метод градієнтного спуску і зворотного поширення помилки більш складний, але один з найбільш ефективних і популярних методів навчання [6].

У [14], [19], [21] та [22] описується можливість використання графічних процесорів та нейронних мереж у медицині загалом. А також описуються різні переваги їх використання.

1.4 Аналіз технічного завдання та постановка завдання дипломного проекту

Для правильного функціонування розроблюваного програмного модуля, на комп’ютері користувача мають бути встановлені такі програми:

- операційна система Windows x64(версія 7 або 10);
- середовище Java 8;
- середовище CUDA 8.0;
- бібліотека NVIDIA cuDNN 6.

Комплектуючі комп’ютера повинні відповідати наступним вимогам:

- процесор з тактовою частотою не менш ніж 2,5ГГц (наприклад, Intel Pentium G4400 або i3-4170);
- оперативна пам’ять обсягом 4Гб (мінімум) та поколінням пам’яті DDR4 (наприклад, Kingston DDR4-2400 HyperX Fury);

- відеокарта з підтримкою технології CUDA 8.0 та обсягом пам’яті не менш ніж 2Гб та з поколінням пам’яті GDDR5 (наприклад, Asus PCI-Ex GeForce GTX 950 Strix 2048MB);
- жорсткий диск розміром від 80Гб.

Розроблюваний модуль програмно-апаратної системи, в кінцевому вигляді, має забезпечувати виконання таких завдань:

- завантаження вхідних зображень з директорії (роздільна здатність зображень становить 128x128 або 224x224 пікселі);
- вибір/створення моделі згорткової нейронної мережі для навчання;
- отримання результатів навчання мережі;
- класифікація зображень на основі навченої моделі;
- паралельне навчання моделі нейронної мережі;
- наявність графічного інтерфейсу для взаємодії з користувачем;
- можливість задання параметрів навчання мережі.

Зображення мають бути поділені на класи (директорії). Всі класи мають знаходитися в одному кореневому каталозі (директорії). Приклад структури зображень показано на рисунку 1.7.

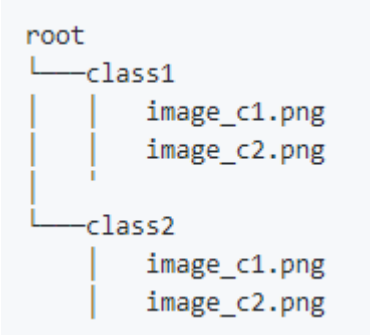

Рисунок 1.7 – Структура директорій

Після вибору директорії із зображеннями, створення або вибору моделі мережі, встановлення параметрів для навчання мережі – навчання повинно відбуватися на графічному процесорі, тобто – паралельно.

Вихідним параметром розроблюваного програмного модуля є точність класифікації, яка має становити не менш ніж 90%.

Для вирішення завдання розпаралелення навчання згорткових нейронних мереж, програмний код буде реалізовуватися на мові програмування Java із використанням бібліотеки Deeplearning4j та технології CUDA.

Графічний інтерфейс програми має бути виконаний у стилі Material Design – принципи дизайну сайтів, програмного забезпечення і застосунків, а також правила дизайну інтерфейсів для операційної системи Android від компанії Google. Для виконання цього завдання можна використати Java бібліотеку JFoenix. Ескіз графічного інтерфейсу зображено на рисунку 1.8.

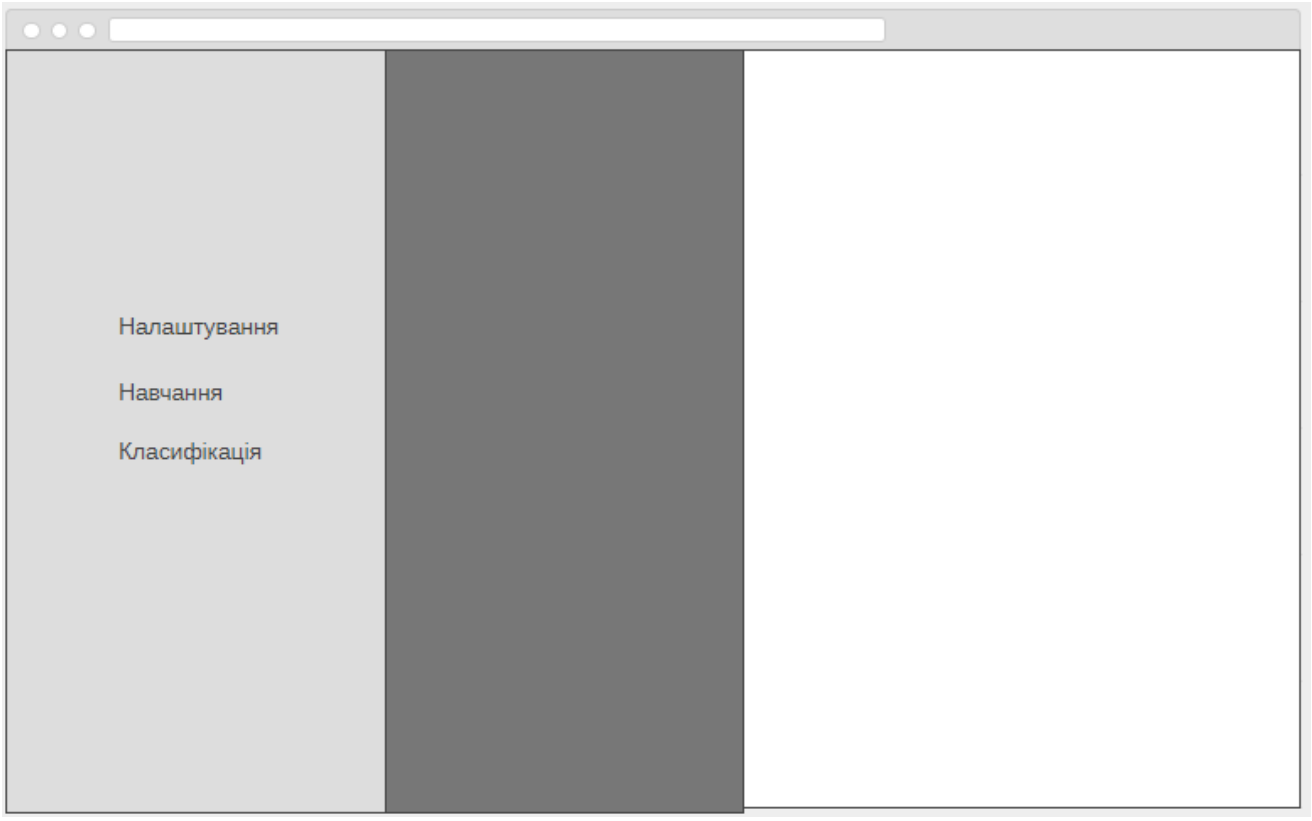

Рисунок 1.8 – Ескіз графічного інтерфейсу

Виходячи з аналізу технічного завдання, метою дипломного проекту є розробка програмного модуля для паралельного навчання згорткових нейронних мереж з використанням графічного процесора. Для досягнення мети потрібно виконати наступні завдання:

- проаналізувати відомі моделі згорткових нейронних мереж;

- проаналізувати алгоритм навчання мережі, а саме, алгоритм зворотного розповсюдження помилки та відомі алгоритми оптимізації;
- проаналізувати бібліотеку Deeplearning4j;
- побудувати таку модель нейромережі, щоб досягти точності класифікації не менше 90%;
- розробити програмний модуль, що має відповідати вищенаведеним вимогам та виконувати описані функції.

Для виконання завдання буде використовуватися комп'ютер з операційною системою Windows 10 та такими комплектуючими:

Таблиця 1.4 – Комплектуючі комп'ютера

Процесор	AMD Athlon 64 x2 Dual Core, 2.51ГГц
Оперативна пам'ять	4ГБ, DDR2
Відеокарта	Asus GeForce GTX 950 Strix, 2Гб, GDDR5
PCI Express	2.0

У даному розділі було проаналізовано сучасні технології та технічні засоби для розпаралелення інформації. Було проаналізовано графічні процесори та проведено їх порівняння. Також було описано та проаналізовано алгоритми навчання згорткових нейронних мереж. Крім того, був проведений аналіз технічного завдання, в якому було прописано чіткі вимоги до розроблюваної прграмно-апаратної системи, та сформовано завдання дипломного проекту.

2 АРХІТЕКТУРА, МОДЕЛІ ТА АЛГОРИТМИ НАВЧАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

2.1 Архітектура і модель згорткової нейронної мережі для класифікації зображень

Згорткова нейронна мережа (ЗНМ) – тип багатошарової нейронної мережі [16], яка свою назву «згорткова мережа» отримала за назвою операції – згортка, вона часто використовується для обробки зображень і може бути описана формулою:

$$(f \times g)[m,n] = \sum_{k,l} f[m-k, n-l] \times g[k,l], \qquad (2.1)$$

де f – вихідна матриця зображення;

g – ядро (матриця) згортки.

Цю операцію можна описати так – вікном ядра g проходимо з заданим кроком (зазвичай 1) все зображення f на кожному кроці поелементно множимо вміст вікна на ядро g, результат сумується і записується в таблицю результату.

Ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів і субдискретизуючих шарів. Така мережа є мережею прямого поширення та багатошаровою. Структура мережі зображена на рисунку 2.1.

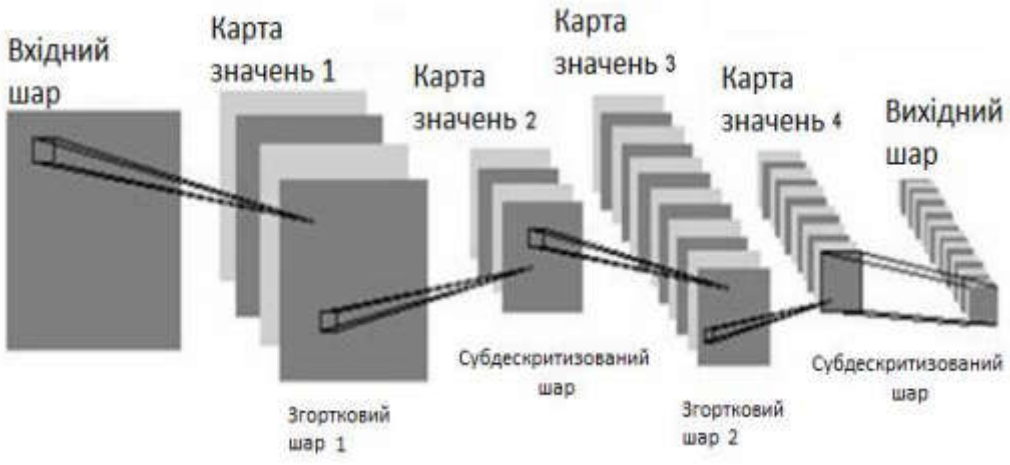

Рисунок 2.1 – Архітектура загорткової нейронної мережі

Модель згорткової мережі складається з трьох типів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling) та повнозв'язні (fully-connected). Архітектура згорткових нейронних мереж реалізує три ідеї, які забезпечують інваріантність мережі до невеликих зрушень, змін масштабу і спотворень:

- кожен нейрон отримує вхідний сигнал від локального рецептивного поля (local receptive fields) у попередньому шарі, що забезпечує локальну двовимірну зв'язність нейронів;
- кожен прихований шар мережі складається з множини карт ознак, на яких всі нейрони мають загальні ваги (shared weights), що забезпечує інваріантність до зміщення і скорочення загальної кількості вагових коефіцієнтів мережі;
- за кожним шаром згортки слідує обчислювальний шар, який здійснює локальне усереднення та підвибірку, що забезпечує зменшення розширення для карт ознак [2].

Загалом, робота згорткової нейронної мережі забезпечується двома основними елементами:

- фільтрами (визначники ознак);

- картами ознак (feature maps).

Фільтр – це невелика матриця, що представляє ознаку, яку необхідно знайти на вихідному зображенні. Результати згортки називаються картами ознак.

Мета процесу згортки – зменшити розмірність карти ознак до такої міри, щоб з повним набором ознак могла працювати мережа прямого поширення. Цю функцію виконує шар субдискретизації.

Згортковий шар реалізує ідею локальних рецептивних полів [9], тобто кожен вихідний нейрон з'єднаний тільки з певною областю вхідної матриці і таким чином моделює деякі особливості людського зору.

Таким чином, повторюючи один за одним кілька шарів згортки і субдискретизації будується згорткова нейронна мережа. На виході мережі часто

встановлюють кілька повнозв’язних шарів, на вхід яких подаються кінцеві карти ознак. У [15], [17] та [18] описуються різні архітектури згорткових нейронних мереж для комп’ютерного розпізнавання, а також проблеми обмеженого набору навчальних даних.

Для проведення експериментів з навчанням мережі використовувалася власна модель загорткової нейронної мережі. Структура мережі наведена на рисунку 2.2.

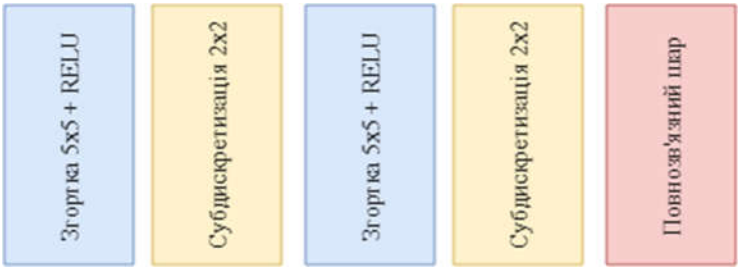

Рисунок 2.2 – Модель ЗНМ для класифікації гістологічних та цитологічних зображень

Модель було навчено для класифікації цитологічних та гістологічних зображень. У таблиці 2.1 наведено параметри навчання мережі для обох видів зображень.

Таблиця 2.1 – Параметри навчання мережі

Параметр	Значення	
	Цитологія	Гістологія
Норма навчання	5e-3	7e-3
L2-регуляризація	5e-7	54e-7
Функція активації	ReLU	ReLU
Градієнтна нормалізація	На кожному шарі	-
Алгоритм оптимізації (мінімізації помилки)	Метод стохастичного градієнту	Метод спряжених градієнтів
Функція активації повнозв’язного шару	ReLU	TANH

Дана мережа містить два згорткових шари з розміром вікна 5x5 пікселів. Перший згорткових шар з кроком 1x1 піксель та відступом 0, другий – з кроком 5x5 пікселів та відступом 1x1 піксель. Також у мережі присутні два субдискретизуючих шари з розміром вікна 2x2 пікселі та один повнозв'язний шар. Функцією активації обрано функцію ReLU, графік якої зображено на рисунку 2.3. У [12] пропонується підхід, що дозволяє створювати нові функції активації комбінуванням декількох вже існуючих.

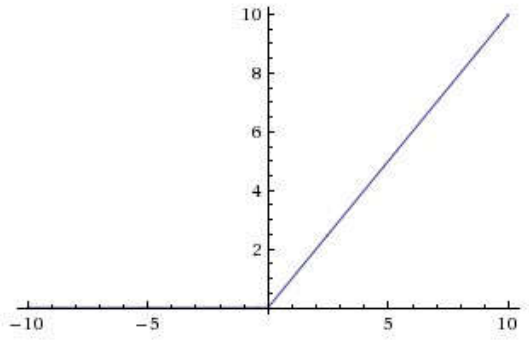

Рисунок 2.3 – Функція активації ReLU

В якості алгоритму оптимізації використовується метод спряжених градієнтів та метод стохастичного градієнту. Навчання відбувалося за методом зворотного поширення помилки, що буде описаний в наступному пункті.

В результаті проведення експериментів, використана мережа показала хорошу якість та точність класифікації (90-100%) для біомедичних зображень розміром 128x128 та 224x224 пікселі.

2.2 Навчання згорткових нейронних мереж методом зворотного поширення помилки

Для навчання згорткових нейронних мереж використовують алгоритм зворотного поширення помилки – ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи багатошарового перцептрону та отримання бажаного результату.

На першому етапі відбувається ініціалізація вагових коефіцієнтів дуже малими випадковими величинами – наприклад, значеннями можуть бути числа

з діапазону -0.2 і +0.2. Навчання передбачається керованим, бо з кожним вхідним прикладом зв'язується цільовий вихідний зразок. Навчання продовжується до тих пір, поки зміни середньої квадратичної помилки не стане меншим деякого мінімального значення при переході між епохами. Наприклад, мінімальне значення помилки 0.001 означає, що середня квадратична помилка сусідніх епох не повинна відрізнятися більше ніж на ±0.001. Якщо в процесі навчання наступає момент, коли помилка мережі потрапляє в допустимі рамки, то кажуть, що спостерігається сходження. Іншим критерієм завершення навчання є те, що настає такий момент, коли вихід для кожного навчального прикладу потрапляє в межі допустимого відхилення від відповідного цільового зразка [7].

Недоліками даного методу є:

- блокування мережі – у процесі навчання мережі значення вагових коефіцієнтів можуть стати дуже великими величинами;
- локальні мінімуми – мережа може потрапити у локальний мінімум (неглибоку впадину), коли поруч є більш глибокі локальні мінімуми, в результаті чого мережа нездатна з нього вибратися;
- розмір кроку (норма навчання) – якщо розмір кроку фіксований і дуже малий, то збіжність надто повільна, якщо ж він фіксований і занадто великий, то може виникнути параліч або постійна нестійкість.

Для того, щоб зменшити імовірність того, що зміна вагових коефіцієнтів набуде осцилюючого характеру, вводиться додатковий інерційний член α, що додається в пропорцію, що відповідає попередній зміні вагового коефіцієнту:

$$\Delta w_{ij}(n+1) = \mu\left(\delta_j o_i\right) + \alpha \Delta w_{ij}(n) \qquad (2.2)$$

Таким чином, зміна ваги на кроці $n+1$ є залежною від зміни ваги на кроці n. Алгоритм зворотного поширення помилки в загальному можна

описати так [7]:

1. Прочитати перший вхідний і відповідний йому вихідний зразок, COVERGE = False;
2. Для вхідного шару встановити сукупний ввід кожного елементу рівним відповідному елементу вхідного вектора. Значення виводу кожного елемента встановити рівним вводу;
3. Для елементів першого прихованого шару порахувати сукупний ввід і вивід:

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij} \tag{2.3}$$

$$o_j = \frac{1}{1 + \exp(-net_j)} \tag{2.4}$$

4. Повторити крок 3 для всіх наступний прихованих шарів;
5. Для елементів вихідного шару порахувати сукупний ввід та вивід:

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij} \tag{2.5}$$

$$o_j = \frac{1}{1 + \exp(-net_j)} \tag{2.6}$$

6. Чи потрапляє різниця між цільовим вихідним зразком і реальним виводом мережі в допустимі межі? ЯКЩО Так ТО COVERGE = True;
7. Для кожного вихідного елемента порахувати його помилку:

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \tag{2.7}$$

8. Для останнього прихованого шару порахувати помилку кожного елемента:

$$\delta_k = o_j(1 - o_j)\sum_k \delta_k w_{kj} \quad (2.8)$$

9. Повторити крок 8 для всіх наступних прихованих шарів;
10. Для всіх шарів оновити значення вагових коефіцієнтів кожного елемента:

$$\Delta w_{ij}(n+1) = \mu(\delta_j o_i) + \alpha \Delta w_{ij}(n) \quad (2.9)$$

11. Останній зразок? Якщо ні, перейти до кроку 12. В іншому випадку – 13.
12. Прочитати наступний вхідний і відповідний йому вихідний зразок. Перейти до кроку 2.
13. Перевірити чи COVERGE == True. Якщо ні, то перейти до кроку 12, якщо так – припинити навчання.

У [19] та [20] пропонується паралельна методика навчання нейронної мережі з підтримкою методу зворотного поширення помилки на графічних процесорах з підтримкою технології CUDA. Запропонована методика використовує декілька графічних процесорів для досягнення максимального прискорення.

2.3 Підбір параметрів навчання згорткових нейронних мереж

Поширеною проблемою при навчанні нейронних мереж є проблема перенавчання. В результаті даної проблеми навчена мережа не може точно класифікувати надані їй зображення.

Створивши таку модель, можна виявити, що вона демонструє майже 100-відсоткову точність на навчальних даних, але значно меншу точність на тестовому наборі даних. Такі результати є гарантованим доказом того, що

модель мережі є перенавчена. Зворотною до перенавченої моделі є недостатньо навчена модель [3].

На рисунках 2.4 та 2.5 зображено модель, яка відповідає контрольним даним і перенавчену модель відповідно.

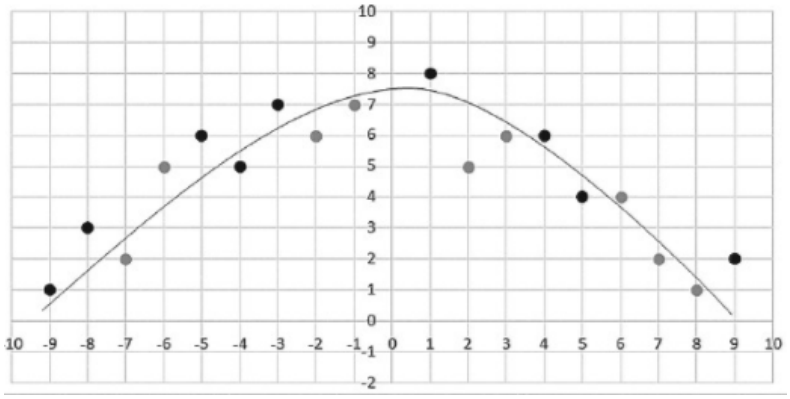

Рисунок 2.4 – Модель, що відповідає контрольним даним

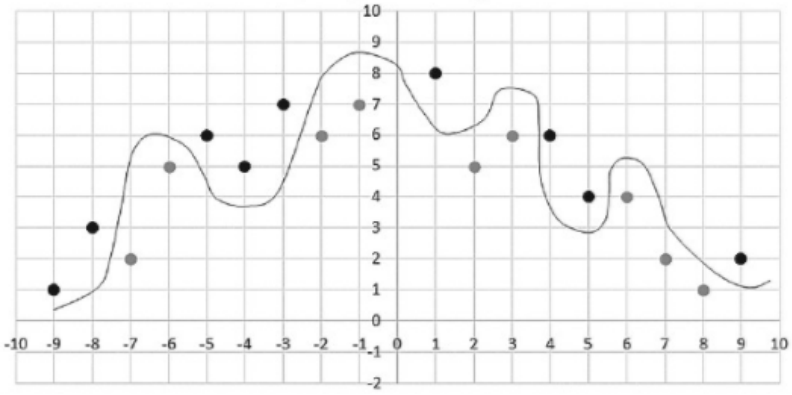

Рисунок 2.5 – Перенавчена модель

Для уникнення згаданих проблем потрібно правильно підібрати параметри навчання мережі. В загальному випадку на навчання мережі впливають такі параметри:

- норма навчання – число, на яке збільшуються значення вагових коефіцієнтів при їх оновленні; в загальному, найбільше впливає на швидкість навчання мережі;
- L2-регуляризація – число, на яке зменшуються значення вагових коефіцієнтів, якщо вони досягають дуже великих значень; в

загальному, запобігає перенавчанню мережі шляхом зменшення вагових коефіцієнтів [10];

- функція активації – функція, від якої залежить вихідний сигнал нейронів мережі; в загальному випадку, перетворює вхідний сигнал у вихідний за певним законом, описує залежність вихідного сигналу нейрона від його вхідного значення;
- алгоритм мінімізації помилки – алгоритм (функція), за допомогою якого відбувається мінімізація помилки (різниця між очікуваним і реальним виходом мережі); в загальному впливає на точність класифікації вхідних даних;
- кількість епох – число, яке збільшується тоді, коли мережа повністю проходить один набір навчальних даних; чим більше епоха, тим краще натренована мережа, а значить і її результат.

У таблиці 2.2 наведено результати навчання мережі описаної у підрозділі 2.2 з різними параметрами навчання. Мережа навчалася на цитологічних зображеннях. Функцією активації було обрано ReLU.

Таблиця 2.2 – Результати навчання мережі

Норма навчання	L2-регуляризація	Кількість епох	Точність класифікації
2e-3	1e-3	20	75%
3e-4	4e-5	25	85%
5e-2	4e-7	25	88%

Продовження таблиці 2.2

54e-3	5e-7	30	90%
54e-6	7e-3	200	99%

На рисунку 2.4 зображено процес навчання мережі (оновлення ітерацій, балів, відображення активації нейронів, загальна інформація про модель).

Рисунок 2.4 – Інформація про модель під час навчання

В описаній вище моделі мережі для мінімізації помилки було використано метод спряжених градієнтів – метод знаходження локального екстремуму функції на основі інформації про її значення і градієнт. Даний алгоритм в загальному можна описати так:

1. Нехай $\overrightarrow{x_0}$ – початкова точка, $\overrightarrow{r_0}$ – напрямок антиградієнту і потрібно знайти мінімум функції $f(\vec{x})$. Покладемо $\overrightarrow{S_0} = \overrightarrow{r_0}$ і знайдемо мінімум в напрямку $\overrightarrow{S_0}$. Позначимо точку мінімуму $\overrightarrow{x_1}$;
2. Нехай на певному кроці ми знаходимося в точці $\overrightarrow{x_k}$, і $\overrightarrow{r_k}$ – напрямок антиградієнту. Покладемо $\overrightarrow{S_k} = \overrightarrow{r_k} + w_k\overrightarrow{S_{k-1}}$, де w_k вибирають або за формулою 2.10 або 2.11. Після чого знаходимо мінімум у напрямку $\overrightarrow{S_k}$ і позначимо очку мінімуму $\overrightarrow{x_{k+1}}$. Якщо в обчисленому напрямку

функція не зменшується, то потрібно «забути» попередній напрямок, поклавши $w_k = 0$ і повторивши крок.

$$\frac{(\overrightarrow{r_k}, \overrightarrow{r_k})}{(\overrightarrow{r_{k-1}}, \overrightarrow{r_{k-1}})} \tag{2.10}$$

$$\max\left(0, \frac{(\overrightarrow{r_k}, \overrightarrow{r_k} - \overrightarrow{r_{k-1}})}{(\overrightarrow{r_{k-1}}, \overrightarrow{r_{k-1}})}\right) \tag{2.11}$$

Обирати параметри нейронної мережі слід дуже ретельно, оскільки при неправильному виборі можна отримати перенавчену або недостатньо навчену модель. В результаті цього модель не зможе досить точно класифікувати вхідні дані.

У даному розділі було проаналізовано загальну структуру згорткової нейронної мережі, алгоритм зворотного поширення помилки для навчання згорткових нейронних мереж, а також запропоновано власну модель такої мережі для класифікації цитологічних та гістологічних зображень.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ

3.1 Системні вимоги

Для правильного функціонування розробленого програмного модуля необхідно дотримуватися таких системних вимог:

- процесор з тактовою частотою не менш ніж 2,5ГГц (наприклад, Intel Pentium G4400 або i3-4170);
- оперативна пам’ять обсягом 4Гб (мінімум) та поколінням пам’яті DDR4 або DDR3 (наприклад, Kingston DDR4-2400 HyperX Fury);
- відеокарта з підтримкою технології CUDA 8.0 та обсягом пам’яті не менш ніж 2Гб та з поколінням пам’яті GDDR5 (наприклад, Asus PCI-Ex GeForce GTX 950 Strix 2048MB);
- жорсткий диск розміром від 80Гб.

Також на комп’ютері користувача має бути встановлене наступне програмне забезпечення:

- операційна система Windows x64(версія 7 або 10);
- середовище Java 8 (версія 1.8.0_151);
- середовище CUDA 8.0;
- бібліотека NVIDIA cuDNN 6.0.

3.2 Програмна реалізація

Для написання програмного коду використовувалася мова програмування Java 8, фреймворк JavaFX та середовище програмування Intellij Idea.

Програмний код написаний з використанням архітектурного шаблону Model-View-Controller (MVC), що передбачає розподіл функціоналу на 3 окремі класи: для взаємодії з даними, для взаємодії з графічним інтерфейсом, для керування моделлю та графічним інтерфейсом.

Весь функціонал розробленого програмного модуля розділений на наступні класи:

- AppLoader (додаток А) – головний клас програми; відповідає за завантаження головної програми та відображення графічного інтерфейсу;
- MainController (додаток Б) – головний контролер; відповідає за обробку користувацьких подій (кліки по кнопках і т.д);
- TrainingController (додаток В) – контролер, що відповідає за навчання моделі;
- ClassificationController (додаток Г) – відповідає за класифікацію зображень з допомогою навченої моделі;
- ChooseModelController (додаток Д) – відповідає за вибір створених та навчених моделей;
- ResultsTableController (додаток Е) – відповідає за відображення результатів навчання у таблиці;
- Utils (додаток К) – клас з допоміжними методами (читання директорії із зображеннями і т.д.);
- Prefs (додаток Л) – відповідає за збереження та завантаження користувацьких налаштувань;
- ClassificationResult (додаток М) – модель даних після навчання нейронної мережі;
- Model (додаток Н) – відповідає за структуру моделі нейронної мережі;
- Layer (додаток П) – відповідає за структуру шарів нейронної мережі;
- JsonModelBuilder (додаток Р) – керує будуванням моделі з файлу.

На рисунку 3.2 зображено UML діаграму класів.

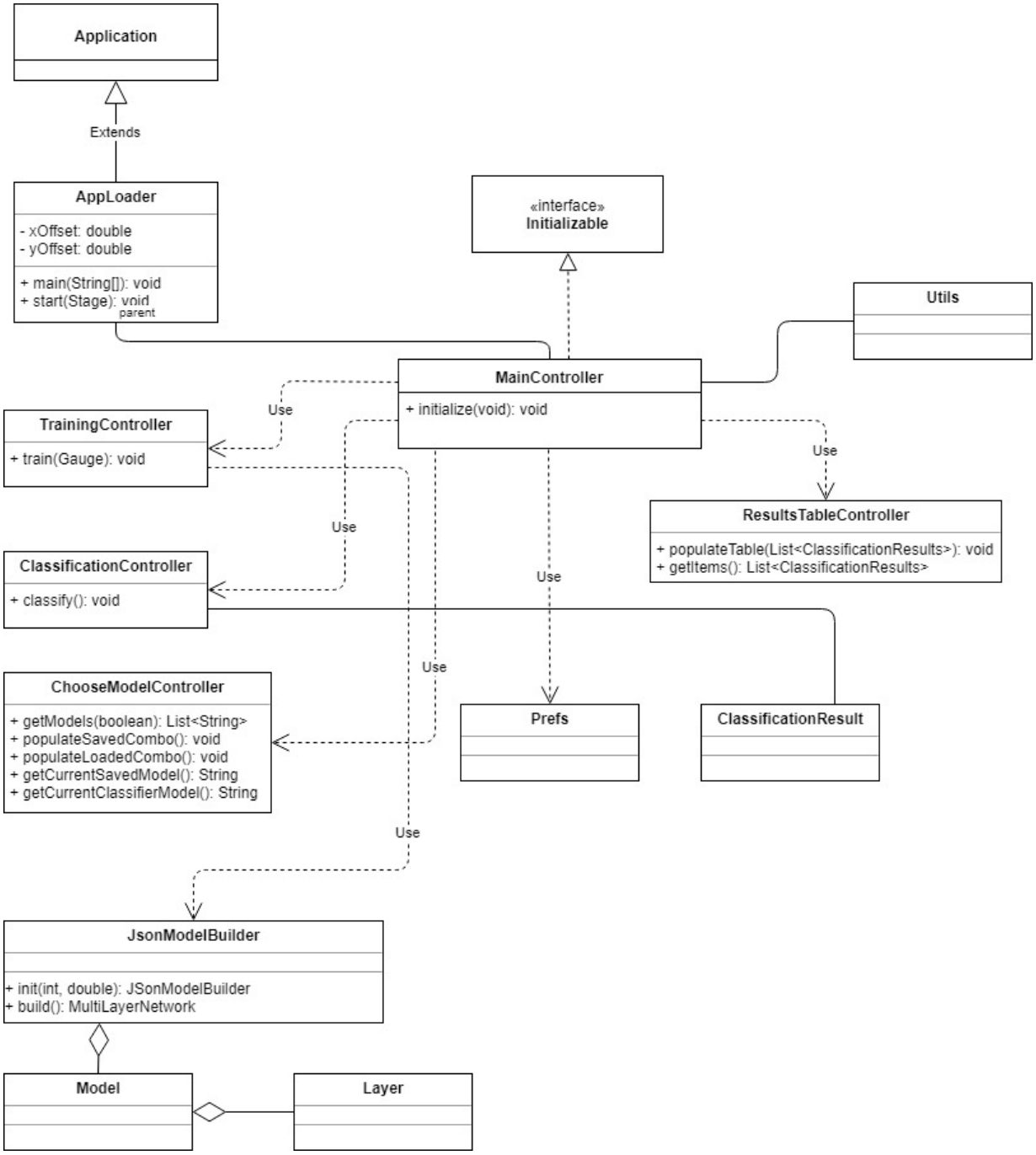

Рисунок 3.2 – UML діаграма класів

3.3 Опис інтерфейсу програмного модуля

Графічний інтерфейс програмного модуля розроблений з використанням бібліотеки JFoenix, яка реалізує Google Material Design за допомогою Java компонентів. Головне вікно програми зображено на рисунку 3.3.

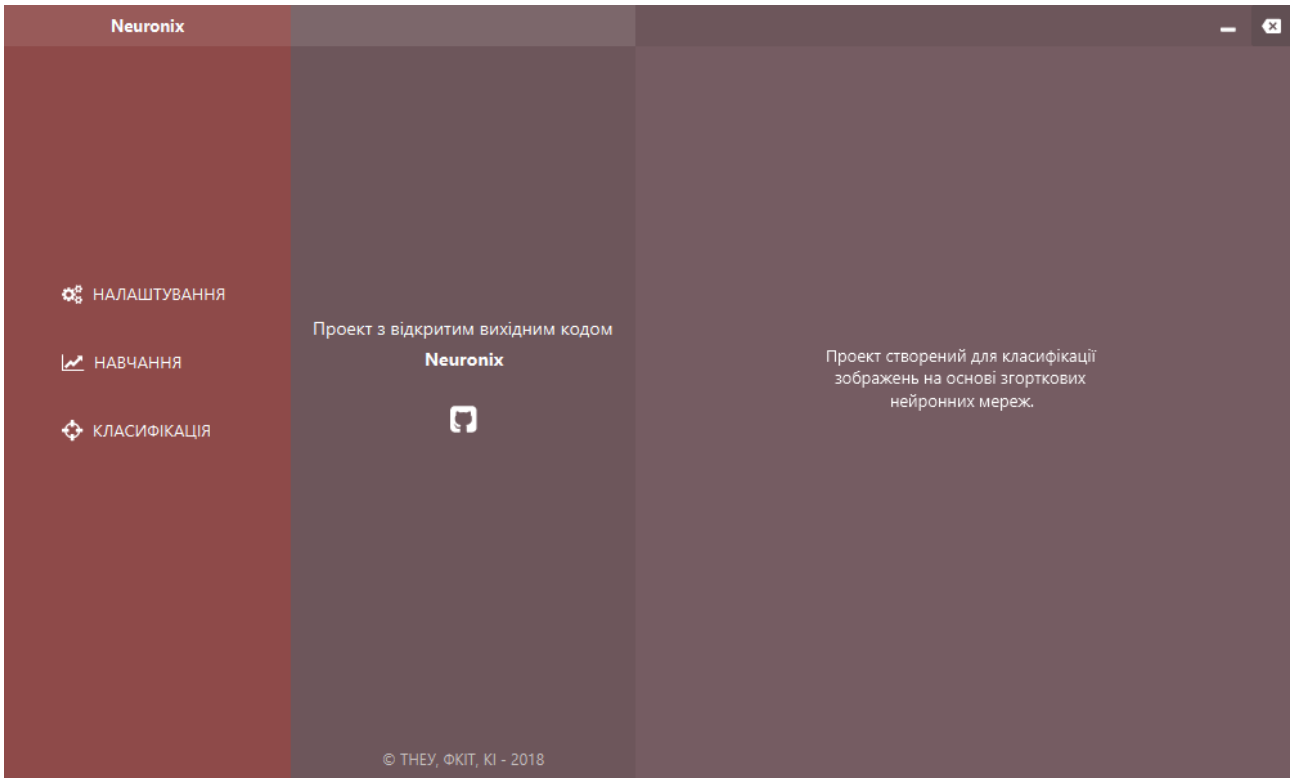

Рисунок 3.3 – Головне вікно програми

Головний екран розділений на три основні частини: меню, основна область, область із додатковою інформацією. Меню містить три кнопки, що дозволяють виконати певну дію, а саме: налаштувати програму, навчати моделі нейронної мережі, класифікувати зображення за допомогою навчених моделей.

Для того, щоб налаштувати програму потрібно натиснути кнопку «НАЛАШТУВАННЯ». Після цього відкриється вікно налаштувань, що зображене на рисунку 3.4.

У вікні налаштувань користувачу доступні такі дії: переглядати поточні налаштування, вказати директорію із навченими моделями у форматі bin, вказати директорію із створеними моделями у форматі json.

Також користувач може обрати чи потрібно йому зберігати модель після навчання, а також встановити Workspace режим у положення SINGLE – обчислення проходять швидше, але потрібно багато пам’яті, або в SEPARATE – обчислення відбуваються повільніше, але використовується менше пам’яті.

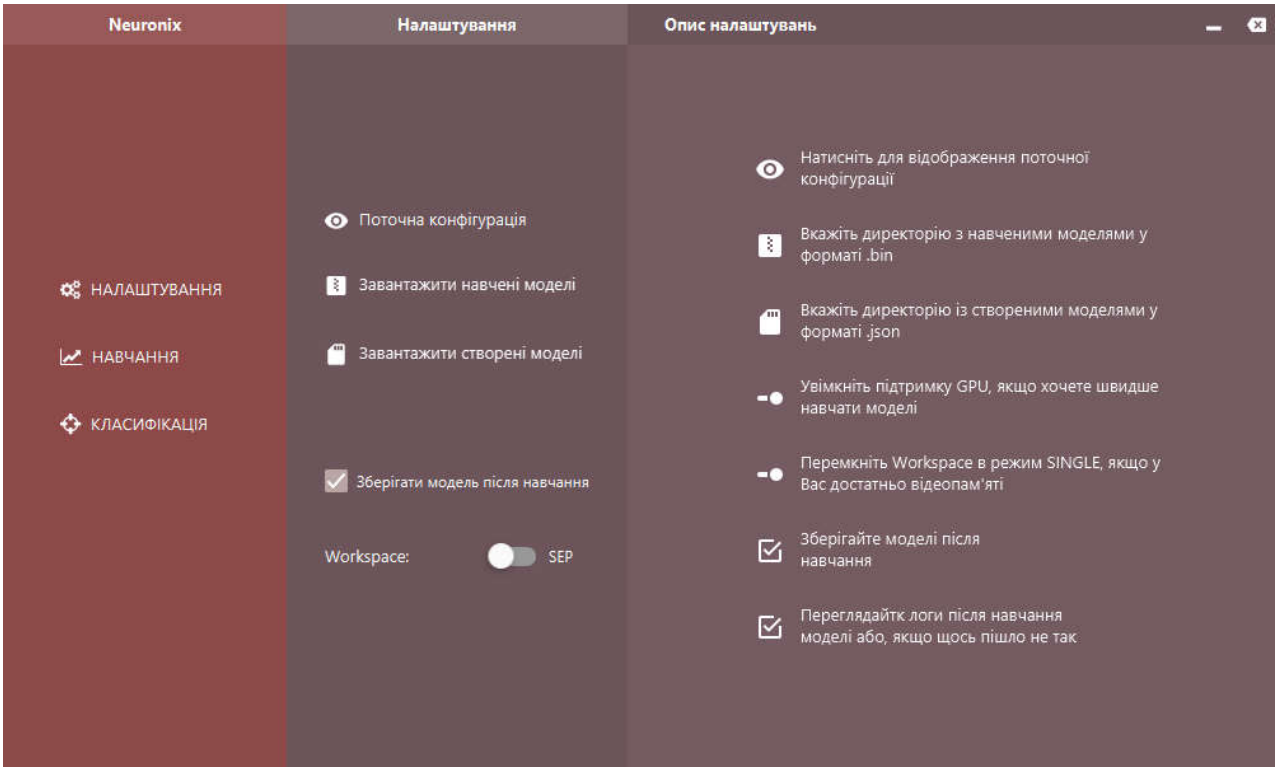

Рисунок 3.4 – Вікно налаштувань програми

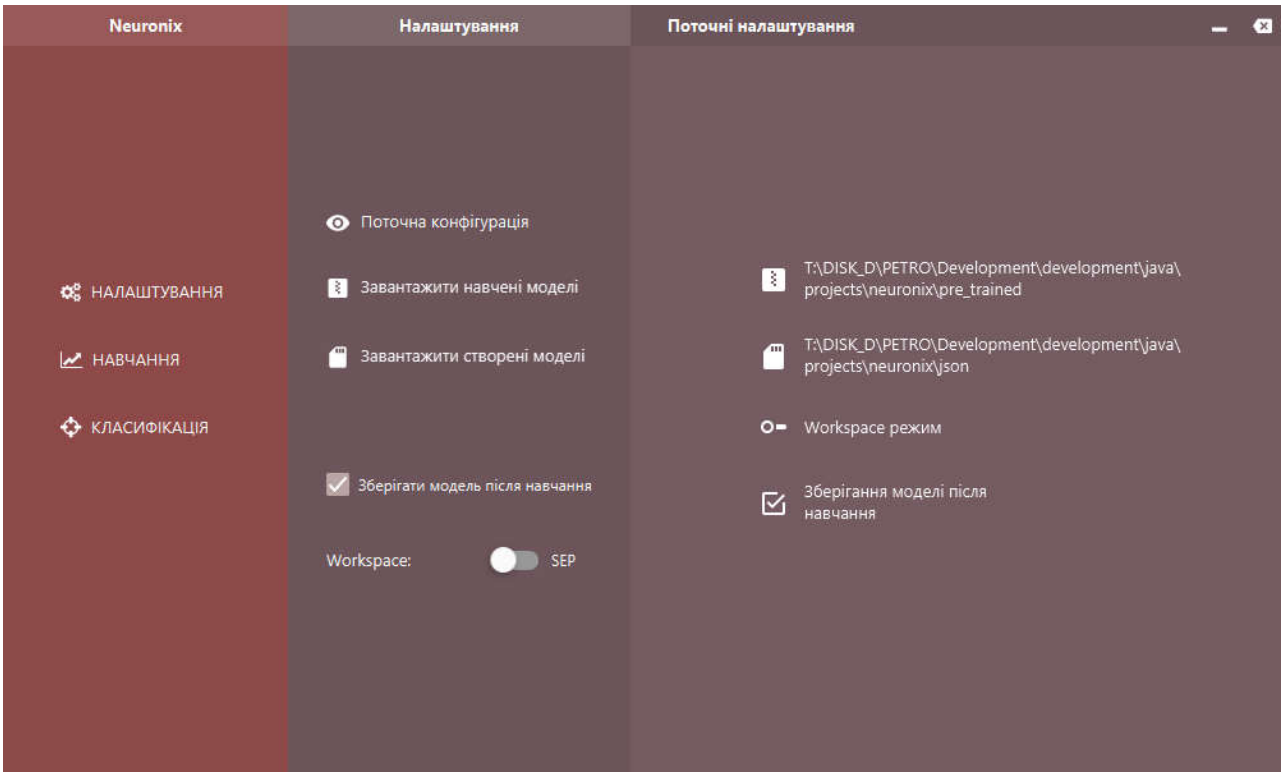

Рисунок 3.5 – Відображення поточних налаштувань

Для того, щоб перейти до навчання нейронної мережі потрібно натиснути кнопку «НАВЧАННЯ», після чого відобразиться наступне вікно:

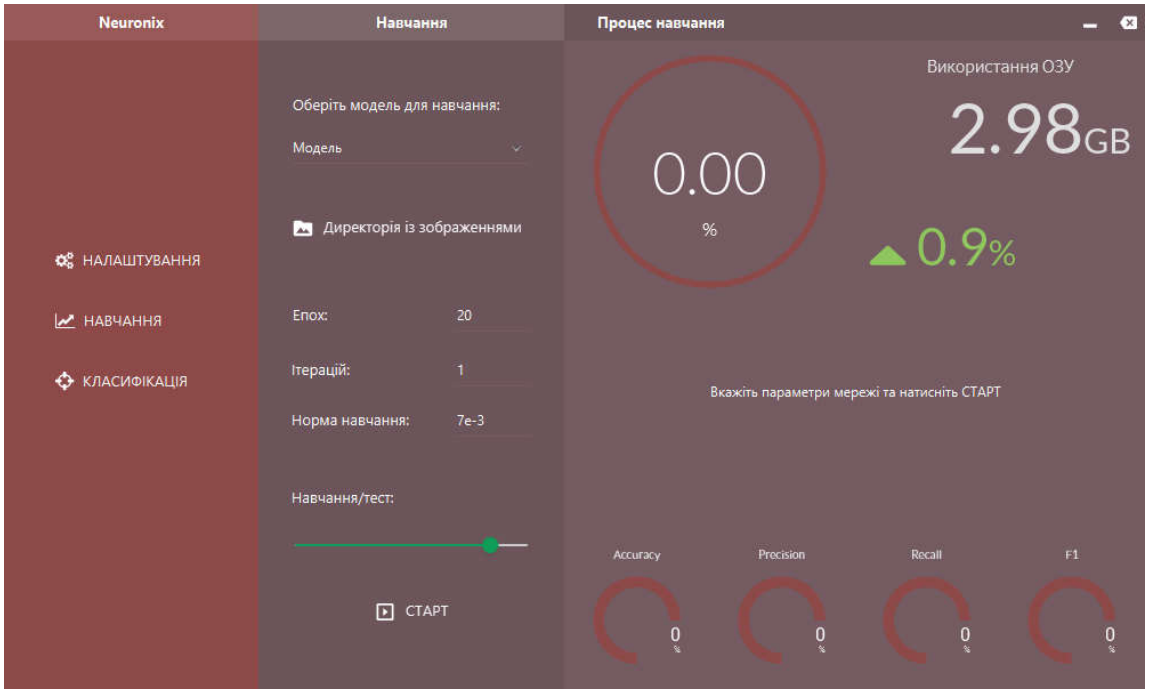

Рисунок 3.6 – Вікно навчання нейронної мережі

Перед тим, як натискати кнопку «СТАРТ» для початку процесу навчання, потрібно вказати параметри навчання нейронної мережі, а саме: обрати модель нейронної мережі у форматі json, вказати директорію із зображеннями у форматі JPG. Додатково користувач може задати кількість епох, ітерації, норму навчання, а також вказати у відсотковому відношенні кількість зображень для навчання та тестування відповідно.

Після натискання кнопки «СТАРТ» буде запущений процес навчання нейронної мережі на основі заданої моделі та з вказаними параметрами. У вікні додаткової інформації буде відображатися прогрес навчання у відсотках, використання оперативної пам’яті, а після завершення навчання – знизу будуть відображені результати навчання мережі у відсотках.

Приклад результатів навчання зображено на рисунку 3.7.

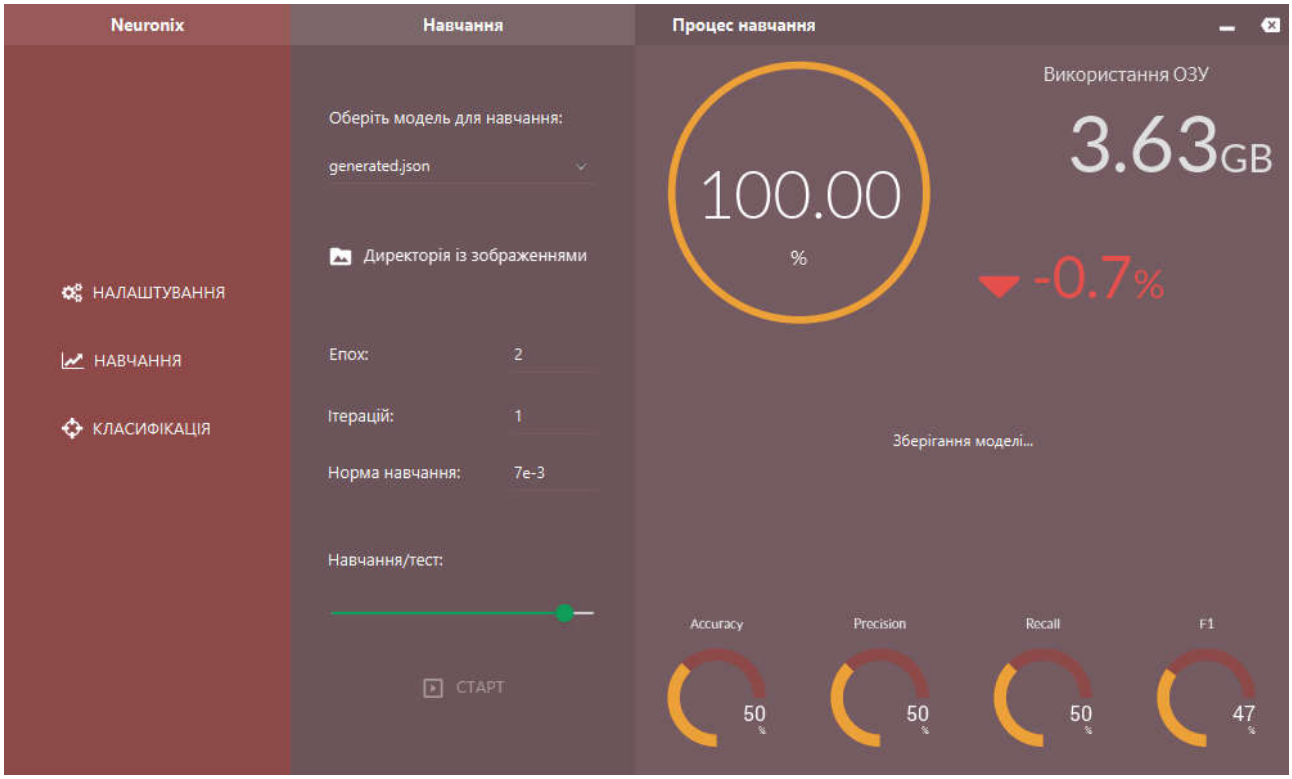

Рисунок 3.7 – Результати навчання обраної моделі нейронної мережі

Створена модель згорткової нейронної мережі зберігається у файлі формату json. Приклад такої моделі наведено у додатку С.

Для переходу у режим класифікації потрібно натиснути кнопку «КЛАСИФІКАЦІЯ», після чого відобразиться наступне вікно:

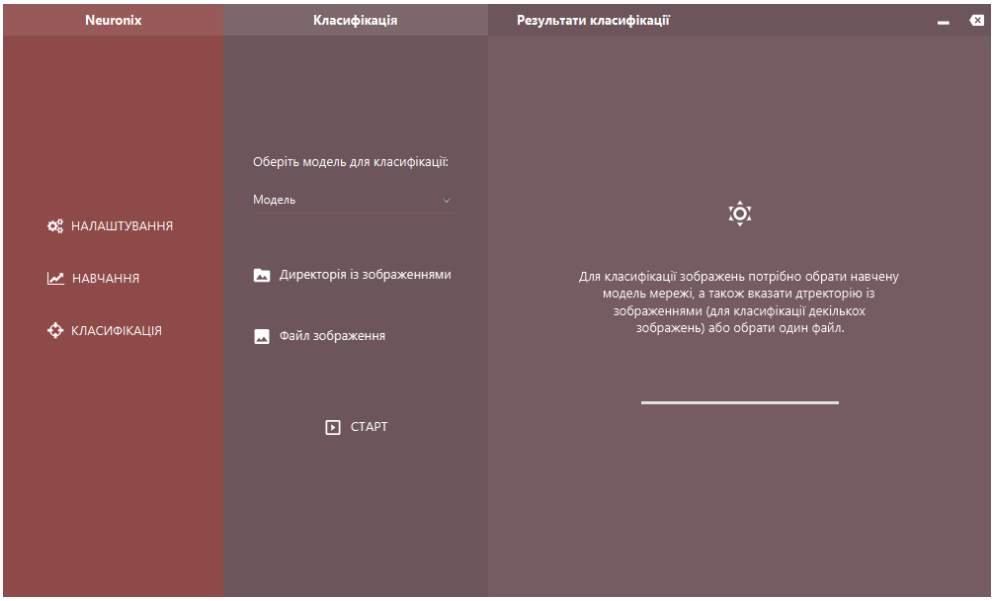

Рисунок 3.8 – Вікно класифікації

Для класифікації зображень потрібно обрати файл навченої мережі у форматі bin, вказати директорію із зображеннями або обрати один файл зображення у форматі JPG та натиснути кнопку «СТАРТ». Зображення повинні бути того ж розміру, що і при навчанні обраної моделі мережі.

Після процесу класифікації у вікні будуть відображені результати:

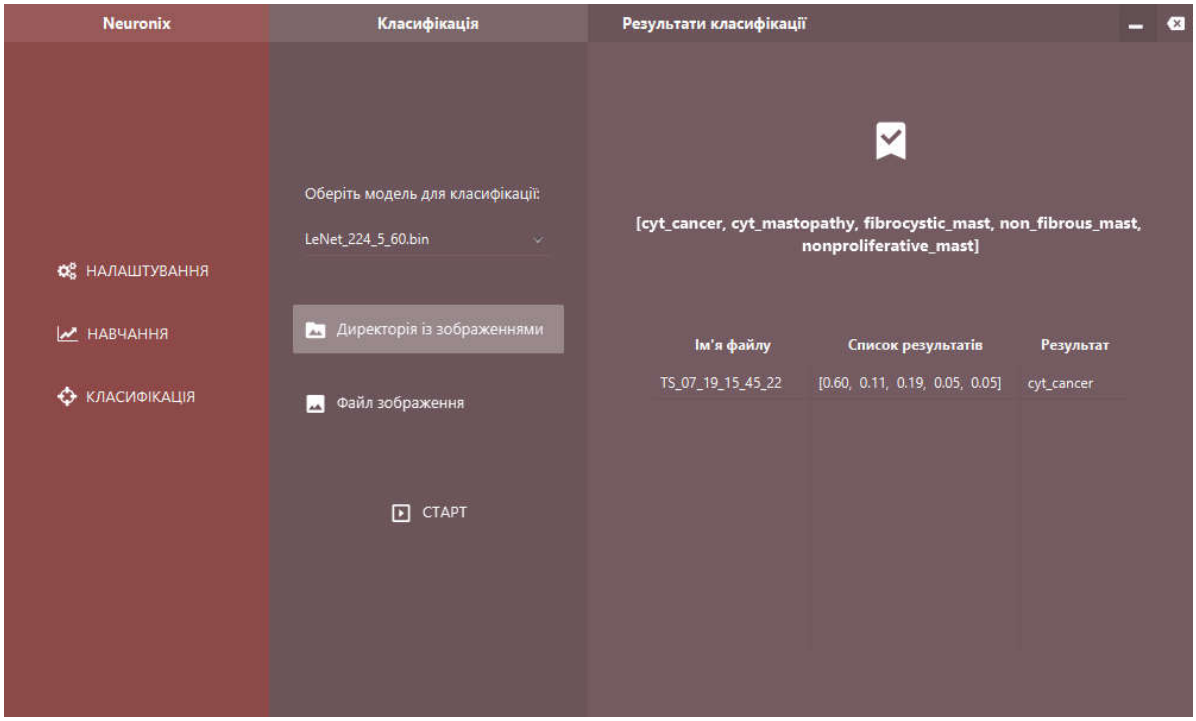

Рисунок 3.9 – Результати класифікації зображення

Як видно з рисунка 3.9, у вікні додаткової інформації відображаються результати класифікації зображення. У верхньому рядку наведено усі класи, які може класифікувати обрана модель. В таблиці записано ім’я файлу зображення, відповідність зображення кожному з наведених класів у відсотках та результат класифікації.

3.4 Тестування програмного модуля

Тестування проводилися на комп’ютері з такою конфігурацією:

Таблиця 3.1 – Конфігурація комп’ютера

Процесор	AMD Athlon 64 x2 Dual Core, 2.51ГГц
Оперативна пам’ять	4ГБ, DDR2
Відеокарта	Nvidia GTX 950 Asus, серія Strix, 2Гб, GDDR5
PCI Express	2.0
Операційна система	Windows 10 Pro

Для експериментів було вибрано два види зображень: цитологічні та гістологічні. Зображення кожного виду були поділені на 5 класів. Кількість цитологічних зображень 80, гістологічних – 91.

Для першого експерименту використовувалися кольорові RGB зображення роздільною здатністю 128x128 пікселів.

Для кожного типу зображень було проведено 3 досліди. Після кожного досліду, в моделі мережі коригувалися деякі параметри для досягнення кращого результату, а саме: норма навчання, алгоритм оптимізації. Результати навчання мережі наведено у таблиці 3.2.

Таблиця 3.2 – Результати навчання (128px)

Вид зображень	Час навчання на GPU (хв)	Час навчання на CPU (хв)	Точність класифікації (%)
Цитологічні	3	15	97
	3	15	93
	2	11	97
Гістологічні	4	18	90
	3	16	93
	2	10	93

Для другого експерименту використовувалися ті ж самі зображення, але з роздільною здатністю 224x224 пікселі. Для цих зображень також було проведено 3 досліди. Результати наведено у таблиці 3.3.

Таблиця 3.3 – Результати навчання (224px)

Вид зображень	Час навчання на GPU (хв)	Час навчання на CPU (хв)	Точність класифікації (%)
Цитологічні	11	47	93
	10	45	98
	9	34	97
Гістологічні	9	36	90
	10	45	90
	10	45	90

Як видно з наведених таблиць, графічний процесор дає прискорення приблизно в чотири рази.

Також, для всіх наборів даних було використано batchSize = 5. Це значення вказує скільки зображень буде використовуватися одночасно при обчисленнях.

Після проведення тестів із зображеннями різної роздільної здатності (від 128 до 3200 пікселів), отримано висновок, що використана відеокарта не дає можливості працювати із зображеннями більшого розміру, ніж наведено в таблицях, оскільки має дуже малий обсяг відеопам’яті.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ ПРОГРАМНОГО ЗАСОБУ

В цьому розділі дипломного проекту проводиться економічне обґрунтування доцільності розробки програмного забезпечення. Зокрема, здійснюється розрахунок витрат на розробку програмного забезпечення, експлуатаційних витрат, ціни споживання проектного рішення. В заключній частині визначаються показники економічної ефективності нового програмного продукту, обґрунтовуються відповідні висновки.

Розроблене програмне забезпечення призначене для візуалізації метричних ультразвукових сигналів і характеризується підвищеною ефективністю виконання алгоритму, що призводить до зменшення часу візуального представлення об’єкту дослідження.

4. 1 Розрахунок витрат на розробку програмного забезпечення

Витрати на розробку і впровадження програмних засобів (К) включають:

$$К = К_1 + К_2, \quad (4.1)$$

де K_1 - витрати на розробку програмних засобів, грн.;

K_2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп’ютері, *грн*.

Витрати на розробку програмних засобів включають:

– витрати на оплату праці розробників ($В_{ОП}$);

– витрати на відрахування у спеціальні державні фонди (Вф);

– витрати на покупні вироби (Пв);

– витрати на придбання спецобладнання для проведення експериментальних робіт (Об), накладні витрати (Н) та інші витрати (Ів).

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП

обчислюється на основі трудомісткості відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти - розробники, а саме: керівник проекту; студент-дипломник; консультант техніко-економічного розділу.

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

№ п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник ДП, професор	5900
2	Консультант техніко-економічного розділу, доцент	3800
3	Студент	1540

Витрати на оплату праці розробників проекту визначаються за формулою:

$$В_{ОП} = \sum_{i=1}^{N}\sum_{j=1}^{M} n_{ij} \times t_{ij} \times C_{ij}, \quad (4.2)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;
t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;
C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн..

Середньо годинна ставка працівника може бути розрахована за формулою:

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_i}, \quad (4.3)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ – місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год. (приймаємо 168 год.).

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 - Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, *год*	Погодинна заробітна плата, *грн/год.*	Витрати на розробку, *грн*
1	Керівник ДП, професор	20,5	35,11	719,75
2	Консультант техніко-економічного розділу, доцент	2	22,61	45,22
3	Студент	144	9,16	1319,04
Разом				2084,01

Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 21,5 % від суми заробітної плати:

$$В_ф = \frac{21,5}{100} \times 2084,01 = 448,06 \text{ грн} \qquad (4.4)$$

У таблиці 4.3 наведений перелік купованих виробів і розраховані витрати на них.

Таблиця 4.3 - Розрахунок витрат на матеріали та комплектуючі

№ п/п	Найменування купованих виробів	Одиниця виміру	Ціна, *грн*	Кількість купованих виробів	Сума, *грн*	Транспортні витрати (10% від суми)	Загальна сума, *грн*
1	Папір (формат А4)	*уп*	80,00	1	80,00	8,0	88,00
2	Ручка кулькова	*шт*	5,00	1	5,00	0,5	5,50
3	Олівець простий	*шт*	1,50	2	3,00	0,3	3,30
4	Диски CD-R	*шт*	4,00	2	8,00	0,8	8,80
5	Зошит, 96 арк	*шт*	3,50	1	3,50	0,35	3,85
6	Тонер для принтера	*уп*	80	1	80	8,0	88,00
Разом							197,45

Витрати на використання комп’ютерної техніки включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. За даними обчислювального центру ТНЕУ для комп’ютера типу IBM PC/ATX вартість години роботи становить 4,5 грн. Середній щоденний час роботи на комп’ютері – 2 години. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4 - Розрахунок витрат на використання комп’ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп’ютер	Час використання комп’ютера, *год.*	Витрати на використання комп’ютера *грн.*

Продовження таблиці 4.4

1	Проведення досліджень та оформлення їх результатів	60	270
2	Оформлення техніко-економічного розділу	8	36
4	Оформлення ДП	12	54
Разом		80	360

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$\text{Н} = 1{,}5 \times 2084{,}01 = 3126{,}01 \text{ грн} \quad (4.5)$$

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$\text{І} = 2084{,}01 \times 0{,}1 = 208{,}40 \text{ грн} \quad (4.6)$$

Витрати на розробку програмного забезпечення складають:

$$\text{К}_1 = \text{В}_{\text{ОП}} + \text{В}_{\text{ф}} + \text{В}_{\text{ПВ}} + \text{Н} + \text{І} \quad (4.7)$$

$$\text{К}_1 = 2084{,}01 + 448{,}06 + 197{,}45 + 3126{,}01 + 208{,}40 = 6063{,}53 \text{ грн} \quad (4.8)$$

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою:

$$\text{К}_2 = S_{\text{м.г.}} \times t_{\text{від}}, \quad (4.9)$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн./год;

$t_{від}$ - комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 30 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 2,5 грн.

$$\text{К}_2 = 2{,}5 \cdot 60 = 150 \text{ грн} \quad (4.10)$$

На основі отриманих даних складаємо кошторис витрат на розробку програмного забезпечення.

Таблиця 4.6 - Кошторис витрат на розробку програмного забезпечення

№ п/п	Найменування витрат	Сума витрат, *грн*.
1	Витрати на оплату праці	2084,01
2	Відрахування у спеціальні державні фонди	448,06
3	Витрати на куповані вироби	197,45
4	Накладні витрати	3126,01
5	Інші витрати	208,40
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту	150,0
Разом		6213,93

Для оцінки економічної ефективності розроблюваного програмного продукту слід порівняти його з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_П = E_{1П} + E_{2П}, \quad (4.11)$$

де $E_п$ - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1п}$ - вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

$E_{2п}$ - вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати $B_{еп}$ визначаються за формулою:

$$B_{ЕП} = E_П \times N_П, \quad (4.12)$$

де $N_п$ - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$\mathrm{E}_{1\Pi} = \sum_{i=1}^{n} n_i t_i c_i, \qquad (4.13)$$

де i - категорії працівників, які приймають участь у підготовці даних (i=1,2,...n);

n_i - кількість працівників і-ої категорії, *осіб*.;

t_i - трудомісткість роботи співробітників і-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника і-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0(1+b)}{m}, \qquad (4.14)$$

де c_i^0 - основна місячна заробітна плата працівника і-ої категорії, *грн.*;

b - коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57;

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: с° = *1200 грн*. Тоді:

$$c_1 = \frac{1200(1+0{,}57)}{22 \times 8} = 10{,}7\,\frac{\text{грн}}{\text{год}} \qquad (4.15)$$

Трудомісткість підготовки даних для проектного рішення складає 1 год., для аналога 1,5 год.

Таблиця 4.7 - Розрахунок витрат на підготовку даних та реалізацію проектного рішення на комп'ютері

№	Час роботи співробітників, год.	Середньогодинна заробітна плата, грн./год.	Витрати , грн.
	Проектне рішення		
1	1	10,7	10,7
	Аналог		
1	1,5	10,7	16,05

Витрати на експлуатацію комп'ютера визначається за формулою:

$$\text{Е}_{2\text{П}} = t \times S_{\text{МГ}}, \quad (4.16)$$

де t - витрати машинного часу для реалізації проектного рішення (аналогу), год.;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн./год.

$E_{2п} = 1*2,5 = 2,5$ грн.; $E_{2а}=1,5 \cdot 2,5 = 3.75$ грн.

$E_п=10,7+2,5=13,2$ грн.; $E_а=16,0+3,75 = 19,75$ грн.

$B_{еп}= 13,2*252=3326,4$ грн.; $B_{еа}=19,75*252=4977$ грн.

Ціна споживання – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$\text{Ц}_{\text{С(П)}} = \text{Ц}_{\text{П}} + \text{В}_{\text{(Е)}NPV}, \quad (4.17)$$

де $Ц_п$ - ціна придбання проектного рішення, грн.:

$$\text{Ц}_{\text{П}} = \text{К}\left(1 + \frac{\Pi_{\text{р}}}{100}\right) + \text{К}_0 + \text{К}_{\text{к}}, \quad (4.18)$$

де К - кошторисна вартість;

Пр - рентабельність;

$К_о$ - витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, *грн*.;

$К_к$ - витрати на доукомплектування технічних засобів на об'єкті, *грн*.;

$$Ц_Д = 3605{,}88 \times (1 + 0{,}3) = 4687{,}64 \text{ грн} \quad (4.19)$$

Вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$В_{впрv} = \sum_{t=0}^{T} \frac{В_{еп}}{(1+R)^t}, \quad (4.20)$$

де $В_{еп}$ - річні експлуатаційні витрати, грн.;

Т - строк служби проектного рішення, років;

R - річна ставка проценту банку.

$$В_{впрv} = \sum_{t=1}^{5} \frac{3326{,}4}{(1+0{,}08)^t} = 13272{,}3 \text{ грн} \quad (4.21)$$

$$В_{впрv} = \sum_{t=1}^{5} \frac{4977}{(1+0{,}08)^t} = 19858{,}2 \text{ грн} \quad (4.22)$$

Тоді ціна споживання проектного рішення дорівнюватиме:

$$Ц_{СП} = 4687{,}64 + 13272{,}3 = 17959{,}24 \text{ грн} \quad (4.23)$$

Аналогічно визначається ціна споживання для аналогу:

$$Ц_{СА} = 3500{,}0 + 19858{,}2 = 23358{,}2 \text{ грн} \quad (4.24)$$

Економічний ефект в сфері проектування рішення:

$$Е_{ПР} = Ц_{П} - Ц_{А} \quad (4.25)$$

$$Е_{ПР} = 4687,64 - 3500,0 = 1187,64 \text{ грн} \quad (4.26)$$

Річний економічний ефект в сфері експлуатації:

$$Е_{КС} = В_{ЕА} - В_{ЕП} \quad (4.27)$$

$$Е_{КС} = 4977 - 3326,4 = 1650,6 \text{ грн} \quad (4.28)$$

Додатковий економічний ефект у сфері експлуатації:

$$\Delta Е_{вкс} = \sum_{t=1}^{T} Е_{вкс} \times (1 + R)^{T-t} \quad (4.29)$$

$$\Delta Е_{вкс} = \sum_{t=1}^{5} 1650,6 \times (1 + 0,08)^{5-t} = 9656,01 \text{ грн} \quad (4.30)$$

Сумарний ефект складає:

$$Е = Е_{ПР} + \Delta Е_{вкс} = 1187,64 + 9656,01 = 10843,65 \text{ грн} \quad (4.31)$$

Таблиця 4. 8 - Показники економічної ефективності проектного рішення

№	Найменування	Одиниці вимірювання	Значення показників	
			Базовий варіант	Новий варіант
1	Капітальні вкладення	*грн.*	-	3350,44
2	Ціна придбання	*грн.*	3500,0	4687,64
3	Річні експлуатаційні витрати	*грн.*	19858,2	13272,3
4	Ціна споживання	*грн.*	23358,2	17959,24
5	Економічний ефект в сфері проектування	*грн.*	-	1187,64
6	Економічний ефект в сфері експлуатації	*грн.*	-	1650,6
7	Додатковий ефект в сфері експлуатації	*грн.*	-	9656,01
8	Сумарний ефект	*грн.*	10843,65	

В даному розділі проведено розрахунок витрат на розробку проектного рішення. Здійснено порівняння з існуючим аналогом, і цим показано, що дане проектне рішення має переваги в порівнянні з аналогами, зокрема: надійність, простота використання, гнучкість, зручність. Згідно проведеного економічного обґрунтування дане проектне рішення є конкурентноздатним. Крім того, отримано економічний ефект у розмірі 10511,58 грн. і тому розробка і впровадження цього проектного рішення є економічно доцільними.

ВИСНОВКИ

В результаті виконання бакалаврської роботи було отримано наступні результати. Розроблено модуль програмно-апаратної системи, що забезпечує:

- завантаження вхідних зображень з директорії (роздільна здатність зображень становить 128x128 або 224x224 пікселі);
- вибір/створення моделі згорткової нейронної мережі для навчання;
- отримання результатів навчання мережі;
- класифікацію зображень на основі навченої моделі;
- паралельне навчання моделі нейронної мережі;
- графічний інтерфейс для взаємодії з користувачем;
- можливість задання параметрів навчання мережі.

Для написання програмного коду використовувалася мова програмування Java 8, фреймворк JavaFX та середовище програмування Intellij Idea. Програмний код написаний з використанням архітектурного шаблону Model-View-Controller (MVC), що передбачає розподіл функціоналу на 3 окремі класи: для взаємодії з даними, для взаємодії з графічним інтерфейсом, для керування моделлю та графічним інтерфейсом.

Графічний інтерфейс програмного модуля розроблений з використанням бібліотеки JFoenix, яка реалізує Google Material Design за допомогою Java компонентів. В роботі наведено зображення всіх вікон інтерфейсу.

Для експериментів було вибрано два види зображень: цитологічні та гістологічні. Зображення кожного виду були поділені на 5 класів. Кількість цитологічних зображень 80, гістологічних – 91. Показано що при опрацюванні зображень різного розміру графічний процесор дає прискорення приблизно в чотири рази.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Liashchynskyi P. Знайомство з технологією CUDA [Електронний ресурс] / Petro Liashchynskyi // Codeguida. – 2018. – Режим доступу до ресурсу: https://codeguida.com/post/1150.
2. Класифікатор зображень на основі згорткової мережі [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://mechanoid.kiev.ua/ml-lenet.html.
3. Машинное «забывание». Достоинства несовершенных моделей [Електронний ресурс] // DataReview. – 2014. – Режим доступу до ресурсу: http://datareview.info/article/mashinnoe-zabyivanie-dostoinstva-nesovershennyih-modeley/.
4. Dropout — метод решения проблемы переобучения в нейронных сетях [Електронний ресурс] // Habrahabr. – 2017. – Режим доступу до ресурсу: https://habrahabr.ru/company/wunderfund/blog/330814/.
5. Обучение нейронной сети [Електронний ресурс] // Портал исскуственного интеллекта. – 2014. – Режим доступу до ресурсу: http://neuronus.com/theory/240-algoritmy-obucheniya-iskusstvennykh-nejronnykh-setej.
6. О методах обучения многослойных нейронных сетей прямого распространения [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://mechanoid.kiev.ua/neural-net-backprop.html.
7. Каллан Р. Основные концепции нейронных сетей / Роберт Каллан. – Киев: Вильямс, 2001. – 286 с.
8. Параллельные вычисления [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Параллельные_вычисления.
9. Применение нейросетей в распознавании изображений [Електронний ресурс] // Geektimes. – 2009. – Режим доступу до ресурсу: https://geektimes.ru/post/74326/.

10. Маккафри Д. L1- и L2-регуляризация в машинном обучении [Електронний ресурс] / Джеймс Маккафри // Microsoft. – 2015. – Режим доступу до ресурсу: https://msdn.microsoft.com/uk-ua/magazine/dn904675.aspx.
11. Давлеткалиев Р. Что такое свёрточная нейронная сеть [Електронний ресурс] / Рахим Давлеткалиев // Habrahabr. – 2016. – Режим доступу до ресурсу: https://habrahabr.ru/post/309508/.
12. Qian S. Adaptive activation functions in convolutional neural networks / S. Qian, H. Liu, C. Liu, S. Wu, H. San Wong // Neurocomputing. – 2018. – Vol. 272. – P. 204-212.
13. Karimi K. A Performance Comparison of CUDA and OpenCL [Електронний ресурс] / K. Karimi, N. G. Dickson, F. Hamze // Computing Research Repository - CORR. – 2010. – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf.
14. Li S. A fast and memory saved GPU acceleration algorithm of convolutional neural networks for target detection / S. Li, Y. Dou, X. Niu, Q. Lv, Q. Wang // Neurocomputing. – 2017. – Vol. 230. – P. 48-59.
15. Shin H.-C. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning / H.-C. Shin, R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, R. M. Summers // IEEE Transactions on Medical Imaging. – 2016. – Vol. 35, No. 5. – P. 1285-1298.
16. Sineglazov V. Deep Neural Networks for Solution Problems of Recognition and Classification of Images [Електронний ресурс] / V. Sineglazov, O. Chumachenko // ITCM-2017. – 2017. – Режим доступу до ресурсу: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf.
17. Shaikhina T. Handling limited datasets with neural networks in medical applications: A small-data approach / T. Shaikhina, N. A. Khovanova // Artificial Intelligence in Medicine. – 2017. – Vol. 75. – P. 51-63.

18. Zou Q. Machine learning and graph analytics in computational biomedicine / Q. Zou, L. Chen, T. Huang, Z. Zhang, Y. Xu // Artificial Intelligence in Medicine. – 2017. – Vol. 83. – P. 1.
19. Qayyum A. Medical Image Retrieval using Deep Convolutional Neural Network / A. Qayyum, S. M. Anwar, M. Awais, M. Majid // Neurocomputing. – 2017. – Vol. 266, No. C. – P. 8-20.
20. Zhang S. Parallel back-propagation neural network training technique using CUDA on multiple GPUs / S. Zhang, P. Gunupudi, Q.-J. Zhang // 2015 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO) – 2015.
21. Singh S. Parallelization of digit recognition system using Deep Convolutional Neural Network on CUDA / S. Singh, A. Paul, M. Arun // 2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS) – 2017. –P. 379-383.
22. Akizur Rahman M. Review of GPU implementation to process of RNA sequence on cancer / M. Akizur Rahman, R. Chandren Muniyandi // Informatics in Medicine Unlocked. – 2018. – Vol. 10. – P. 17-26.
23. Kalaiselvi T. Survey of using GPU CUDA programming model in medical image analysis / T. Kalaiselvi, P. Sriramakrishnan, K. Somasundaram // Informatics in Medicine Unlocked. – 2017. – Vol. 9. – P. 133-144.
24. Li S. A two-channel convolutional neural network for image super-resolution / S. Li, R. Fan, G. Lei, G. Yue, C. Hou // Neurocomputing. – 2018. – Vol. 275. – P. 267-277.
25. Basu S. Deep neural networks for texture classification – A theoretical analysis / S. Basu, S. Mukhopadhyaya, M. Karki, R. DiBiano, S. Ganguly, R. Nemani, S. Gayaka // Neural Networks. – 2018. – Vol. 97. – P. 173-182.
26. Pratondo A. Integrating machine learning with region-based active contour models in medical image segmentation / A. Pratondo, C.-K. Chui, S.-H. Ong // Journal of Visual Communication and Image Representation. – 2017. – Vol. 43. – P. 1-9.

27. Biswas N. A parameter independent fuzzy weighted k-Nearest neighbor classifier / N. Biswas, S. Chakraborty, S. Subhra Mullick, S. Das // Pattern Recognition Letters. – 2018. – Vol. 101. – P. 80-87.
28. Yuan X. A regularized ensemble framework of deep learning for cancer detection from multi-class, imbalanced training data / X. Yuan, L. Xie, M. Abouelenien // Pattern Recognition. – 2018. – Vol. 77. – P. 160-172.
29. Danaee P. A Deep learning approach for cancer detection and relevant gene identification / P. Danaee, R. Ghaeini, David A. Hendrix // Biocomputing. – 2017. – P. 219-229.
30. Song T. The method for breast cancer grade prediction and pathway analysis based on improved multiple kernel learning / T. Song, Y. Wang, W. Du, S. Cao, Y. Tian, Y. Liang // Journal of Bioinformatics and Computational Biology. – 2017. – Vol. 15, No. 1.
31. Srivastava N. Dropout: A Simple Way to Prevent Neural Networks from overfitting / N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov // Journal of Machine Learning Research. – 2014. – Vol. 15.
32. Huang Y. Recognition of convolutional neural network based on CUDA Technology / Y. Huang, K. Li, G. Wang, M. Cao, P. Li, Y. Zhang // CoRR. – 2015.
33. Strigl D. Performance and Scalability of GPU-Based Convolutional Neural Networks / D. Strigl, K. Kofler, S. Podlipnig // 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing – 2010. –P. 317-324.
34. Zlateski A. ZNN – A Fast and Scalable Algorithm for Training 3D Convolutional Networks on Multi-core and Many-Core Shared Memory Machines / A. Zlateski, K. Lee, H. Sebastian Seung // 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) – 2016. –P. 801-811.
35. Luebke D. CUDA: Scalable parallel programming for high-performance scientific computing / D. Luebke // 2010 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro – 2012.

36. Pendlebury J. Artificial Neural Network Simulation on CUDA / J. Pendlebury // 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications – 2012.