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

Лящинський Петро Борисович

Синтез гістологічних зображень на основі згорткових нейронних мереж / Synthesis of histological images based on convolution neural networks

спеціальність:123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Випускна кваліфікаційна робота

Виконав студент групи КІм-21
П. Б. Лящинський
_____

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

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Випускна кваліфікаційна робота на тему "Синтез гістологічних зображень на основі згорткових нейронних мереж ” на здобуття освітньо-кваліфікаційного рівня “Магістр” зі спеціальності “Комп’ютерна інженерія” написана обсягом 87 сторінок і містить 39 ілюстрацій, 4 таблиць, 8 додатків та 66 джерел за переліком посилань.

Метою випускної кваліфікаційної роботи є розроблення алгоритмів синтезу біомедичних зображень на основі генеративно-змагальних мереж для збільшення потужності навчальних вибірок.

У випускній кваліфікаційній роботі на основі аналізу методів і засобів синтезу зображень розроблено алгоритми синтезу гістологічних зображень. Для реалізації алгоритмів синтезу використано GAN мережі.

Суть розробленого програмного продукту полягає в можливості розширення потужності навчальних вибірок для навчання згорткових нейронних мереж.

Ключові слова: АЛГОРИТМ, СИНТЕЗ, АНАЛІЗ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ.

RESUME

The final qualification work on the topic "Synthesis of histological images based on convolutional neural networks" for obtaining the Master's degree in Computer Engineering specialty is written with a volume of 87 pages and contains 39 illustrations, 4 tables, 8 appendices and 66 sources for a list of links.

The purpose of the final qualification work is to develop algorithms for the synthesis of biomedical images on the basis of generative-competitive networks to increase the power of training samples.

In the final qualification work, based on the analysis of methods and means of image synthesis, algorithms for synthesis of histological images were developed. GAN networks were used to implement the synthesis algorithms.

The essence of the developed software product is the possibility of expanding the power of training samples for training convolutional neural networks.

Keywords: ALGORITHM, SYNTHESIS, ANALYSIS, CONVOLUTIONAL NEURAL NETWORKS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ЗНМ (CNN) – згорткова нейронна мережа;

ГЗМ (GAN) – генеративно-змагальна мережа;

ШНМ – штучна нейронна мережа;

GPU – графічний процесор;

API – прикладний програмний інтерфейс;

CUDA – програмно-апаратна архітектура паралельних обчислень;

DDR – подвійна швидкість передачі даних;

GDDR – графічна подвійна швидкість передачі даних;

AWS – Amazon Web Services;

GCP – Google Cloud Platform.

ВСТУП

Актуальність теми. Випускна кваліфікаційна робота виконана згідно вимог [1, 2] і присвячена дослідженню питань, пов’язаних із синтезом біомедичних зображень.

Сучасні класифікатори зображень будуються на основі нейронних мереж. Особливої популярності набули згорткові нейронні мережі, тому що на їх вхід поступають зображення, а на виході отримують мітку класу.

Однією із основних проблем при навчанні згорткових нейронних мереж є великий час навчання. Для подолання цієї проблеми використовують графічні процесори. Другою проблемою є обмежена навчальна вибірка, особливо для біомедичних зображень.

Для діагностування передракових і ракових станів органів людини використовуються цитологічні і гістологічні зображення. Отримання цих зображень є трудомістким процесом. Більше того, наявні бази даних цитологічних і гістологічних зображень, які є у вільному доступі, є уже обмеженими. Тому збільшення кількості цих зображень є архіважливою проблемою.

Мета і завдання дослідження. Метою роботи є розроблення алгоритмів синтезу гістологічних зображень на основі згорткових нейронних мереж для збільшення кількості навчальних вибірок.

Об’єкт дослідження – процес синтезу біомедичних зображень.

Предмет дослідження – методи та алгоритми синтезу зображень.

Для досягнення поставленої мети потрібно розв’язати такі задачі:

- проаналізувати методи та алгоритми синтезу біомедичних зображень;
- проаналізувати методи синтезу зображень;
- проаналізувати архітектури генеративно-змагальних мереж;
- програмно реалізувати синтез гістологічних зображень на основі ЗНМ;

− провести тестування реалізованих алгоритмів.

Наукова новизна отриманих результатів. Розроблено алгоритм та модель нейронної мережі для синтезу гістологічних зображень на основі згорткових нейронних мереж, що дозволило синтезувати дані зображення в потрібній кількості.

Практичне значення отриманих результатів. Розроблено програмне забезпечення для синтезу гістологічних зображень на мові програмування Python, а також розроблено бібліотеку на мові програмування Java, що дозволило використовувати розроблені алгоритми у системі автоматизованої мікроскопії.

Публікації та апробація випускної кваліфікаційної роботи. Отримані результати апробовані в межах конференції «Інтелектуальні комп'ютерні системи та мережі» Тернопільського національного економічного університет, де опубліковано тези доповіді [3]. Також за результатами роботи опубліковано статті в журналі [4, 5, 6].

Впровадження результатів ВКР. Результати роботи планується використати в роботі науково-дослідного інституту інтелектуальних комп'ютерних систем (додаток А).

Випускна кваліфікаційна робота складається із трьох розділів, висновків, списку використаної літератури та додатків. У першому розділі проаналізовано біомедичні зображення [7, 8, 9], проведено огляд алгоритмів для їх синтезу та огляд архітектур згорткових нейронних мереж.

В другому розділі проаналізовано алгоритми синтезу зображень на основі афінних перетворень, архітектури генеративно-змагальних мереж та розроблено алгоритм синтезу гістологічних зображень на основі ЗНМ [10, 11].

У третьому розділі описано програмну реалізацію розроблених алгоритмів, а також проведено тестування.

1 МЕТОДИ ТА АЛГОРИТМИ СИНТЕЗУ БІОМЕДИЧНИХ ЗОБРАЖЕНЬ

1.1 Аналіз гістологічних зображень

Термін «гістологія» був запропонований німецьким вченим Р.Майєром у 1819 році. Під цим терміном розумілася наука про тканини багатоклітинних тварин та людини [12]. Тканиною називають групу клітин, схожих формою, розмірам і функціям і по продуктах своєї життєдіяльності. У всіх рослин і тварин, за винятком найпримітивніших, тіло складається з тканин, причому у вищих рослин і у високоорганізованих тварин тканини відрізняються великою різноманітністю структури і складністю своїх продуктів; поєднуючись один з одним, різні тканини утворюють окремі органи тіла.

Гістологію інколи називають мікроскопічною анатомією, оскільки вона вивчає будову (морфологію) організму на мікроскопічному рівні (об'єктом гістологічного дослідження служать дуже тонкі тканинні зрізи і окремі клітини).

Наука гістологія вивчає не тільки тканини, але й клітини, з яких вони складаються, будову органів і систем організму в цілому. Згідно з цим існують такі розділи гістології: цитологія (наука про клітину), загальна гістологія (вивчає тканини), спеціальна гістологія (вивчає будову органів і їх систем). Предмет гістології людини охоплює вивчення тонкої (мікроскопічної) та ультратонкої (субмікросокопічної) будови структур людського організму, їх розвитку і змін у різноманітних умовах життєдіяльності [12].

Гістологія характеризує структурну організацію макромолекул у ті чи інші мікрооб’єкти. Гістологічне зображення – це зображення гістологічного препарату, яке зроблене за допомогою цифрового мікроскопу. До складу об’єкту дослідження (гістологічні зображення раку молочної залози) входять зображення таких видів раку (класів): непроліферативна мастопатія (рисунок 1.1), проліферативна мастопатія (рисунок 1.2), фіброаденома 12 (рисунок 1.3),

листовидна фіброаденома (рисунок 1.4), фіброзно-кістозна мастопатія (1.5). Нижче наведений якісний опис кожного з цих класів.

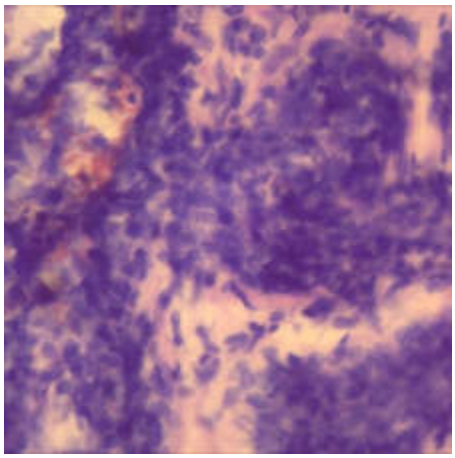

Рисунок 1.1 – Непроліферативна мастопатія раку молочної залози. Гістологія

Мастопатія з переважанням кістозного компонента проявляється кістами, чітко відмежовані від навколишніх тканин залози, утвореними з атрофованих часточок і розширених проток залоз з фіброзними змінами інтерстиціальної тканини.

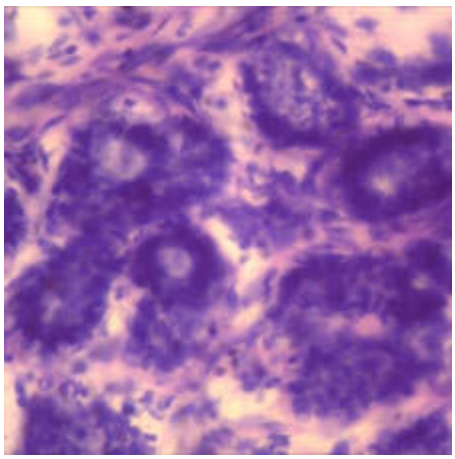

Рисунок 1.2 – Проліферативна мастопатія

При проліферативній мастопатії міоепітеліальні клітини зазнають змін. В залежності від свого функціонального стану вони поодинокі або багаточисельні, темні і видовжені, світлі і які містять міхурцеві включення

[13]. Ці зміни зазнають клітини, які знаходяться між базальною мембраною та секретуючим епітелієм альвеол і дрібних протоків. Фіброзуючий аденоз є виключенням. Базальна мембрана зникає і проліферуючі міоепітеліальні клітини проникають в навколишню внутрішньодолькову сполучну тканину, де стають подібними на гладком'язеві елементи. Виникають мікроскопічні фокуси, які складаються із скупчень видовжених міоепітеліальних клітин, серед яких спостерігаються епітеліальні трубочки. Мікроскопічні фокуси мають неправильні контури або округлу форму. В останньому випадку вони подібні на збільшені або змінені дольки. Між ними з'являються колагенові волокна і міоепітеліальний проліферат склерозується [13].

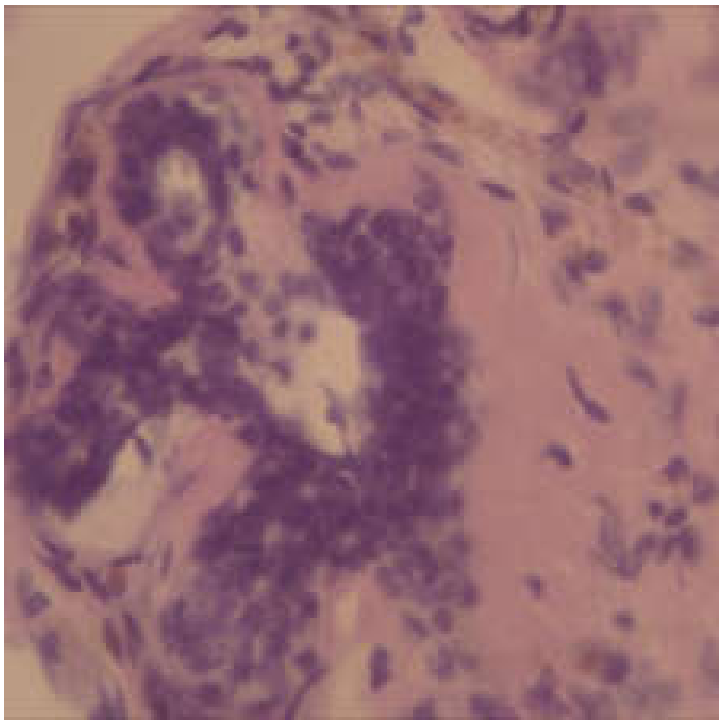

Рисунок 1.3 – Фіброаденома

Фіброаденома має вид інкапсульованого вузла щільної консистенції, волокнистої будови. Фіброаденома є найбільш частою доброякісною пухлиною, має вид інкапсульованого вузла щільної консистенції, волокнистої будови. Вона складається із фіброзної та залозистої тканини. Мікроскопічно фіброаденома виявляється проліферацією альвеол і внутрішньодолькових

протоків з розростанням внутрішньодольковоїр сполучної тканини. Якщо вона оточує внутрішньодолькові протоки, то це свідчить про периканалікулярну фіброаденому [13]. При вростанні сполучної тканини в стінку протоків виникають їх химерні риси. Така пухлина має назву інтраканалікулярна фіброаденома.

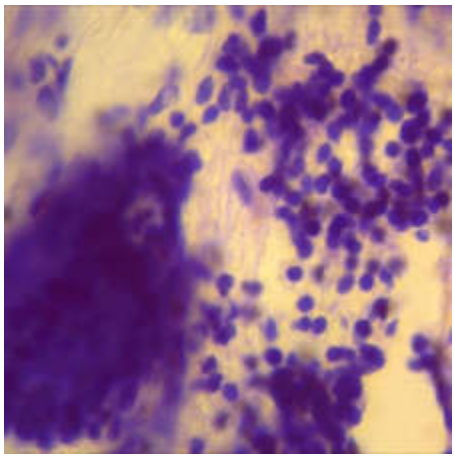

Рисунок 1.4 – Листовидна фіброаденома

Основними критеріями діагностики листовидної фіброаденоми є структурні зміни апокринового епітелію, клітини якого переважно збільшені у розмірах з інтенсивно вираженими гіперхромними ядрами та вузьким обідком цитоплазми. Клітини зазвичай формують сосочкоподібні структури.

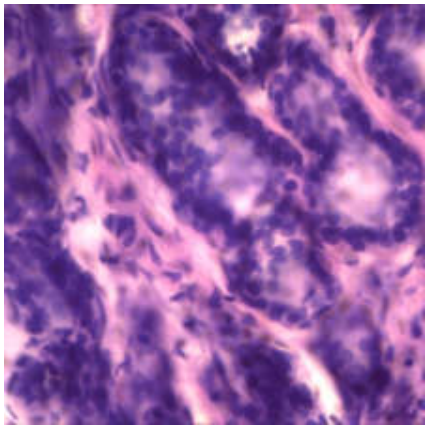

Рисунок 1.5 – Фіброзно-кістозна мастопатія

Крім загальної гістології існує така наука як кількісна гістологія. Кількісна гістологія - це наука, що вивчає закономірності розвитку та функціонування тканин, використовуючи при цьому кількісні параметри та строгі методи перевірки гіпотез [14]. Кількісну гістологію більш коректно розглядати не як окрему наукову дисципліну, а як перехідний стан власне гістології, на шляху її розвитку від описової до точної науки [15].

Впровадження елементів кількісного аналізу в практику гістологічного дослідження робить більш ефективним пошук залежностей між структурою і функцією, підвищує точність одержуваних оцінок, знижує вплив суб'єктивного фактора на результати аналізу гістологічного зображення, а також допомагає автоматизувати процедури дослідження і діагностики [15].

Якщо головним інструментом гістології є мікроскоп, то у кількісній гістології цю роль відіграє система аналізу гістологічних зображень, яку спрощено можна уявити собі як мікроскоп, з'єднаний з комп'ютером. Програмна частина таких систем включає десятки алгоритмів, націлених на аналіз різноманітних параметрів тканин та/або клітин. Деякі системи аналізу зображень навіть спроможні приймати діагностичні рішення, що відносить їх до категорії так званих експертних систем [14].

Гістологічні зображення отримуються різними методами: ультрафіолетова мікроскопія, світлооптичний метод, люмінесцентна мікроскопія, фазовоконтрастна мікроскопія, поляризаційна мікроскопія та ін. [15]. Однак найбільшого поширення набув метод світлової мікроскопії, який базується на використанні світлового мікроскопа.

Сучасні застосування кількісної гістології спрямовані на розробку нових методів, що дозволяють позбутися високої варіабельності оцінок, отриманих різними дослідниками; знаходження нових критеріїв оцінки функції тканин, а також маркерів патологічних процесів [16]; розробку алгоритмів для комп'ютеризованих систем, спроможних допомогти лікарю у постановці діагнозу(програмна частина таких систем може включати елементи штучного інтелекту) та ін.

В гістологічному дослідженні кількісна оцінка мікроструктур є необхідною умовою отримання об'єктивних даних про їх стан. Головними кількісними показниками мікроструктур є морфометричні (число структур і їх геометричні параметри) і денситометричні, що відображають концентрацію (оптичну щільність) хімічних речовин в мікроструктурах [18]. Для визначення цих параметрів застосовують морфометричні і спектрофотометричні методи. Морфометрія включає сукупність прийомів і методів визначення геометричних характеристик дослідних об'єктів – гістологічних препаратів (зрізів, мазків, відбитків і т.п.) і мікрофотографій. Вимірювання числа структур, їх площин, діаметрів, периметрів і інших показників здійснюють, наприклад, за допомогою спеціальних морфометричних сіток або окуляр-мікрометра, які вставляють в окуляр світлового мікроскопа [17].

Отже, серед кількісних характеристик гістологічного зображення виділяють:

- число структур;
- кількість об’єктів;
- геометричні параметри структур;
- розмір структур;
- форма структур;
- розподіл структур та об’єктів у просторі (топографія);
- орієнтація у просторі;
- приналежність до групи (кластерний аналіз);
- топологічні властивості.

Гістологічне дослідження – ведучий, вирішальний і завершальний етап діагностики онкологічних захворювань [18]. Його роль безперервно зростає у зв'язку з прагненням до ранньої діагностики пухлинного процесу і виявленню передпухлинних змін, фонових станів і ранніх стадій розвитку злоякісних пухлин. Вдосконалення ендоскопічної і оперативної техніки, прагнення до виконання органозберігаючих оперативних втручань обумовлюють

необхідність ширшого використання морфологічної діагностики пухлин і принципово нових підходів до її проведення.

Гістологічне дослідження включає три основні етапи. Перший і найважливіший етап — встановлення морфологічного діагнозу перед початком лікування (хірургічне, хіміотерапія, променева терапія, гормональна дія). Не менш важлива роль морфолога на другому етапі — етапі проведення термінового гістологічного дослідження під час операції [18]. Нарешті, найбільшу інформацію про характер патологічного процесу можна отримати на завершальному етапі — при плановому морфологічному дослідженні операційного матеріалу.

1.2 Аналіз методів та алгоритмів синтезу зображень

Аналіз методів і алгоритмів синтезу зображень розглянемо на прикладі текстурних зображень. Текстурне зображення зазвичай включає впорядковані узори, що складаються з певних елементів. Більшість дослідників погоджуються, що зображення текстури повинне бути просторово гомогенне (однорідне), і зазвичай містить повторювані структури, часто з деякими випадковими змінами (наприклад, зміною позиції, орієнтації або кольору).

Текстури за просторовим розміщенням повторюваних елементів поділяються на регулярні, близькорегулярні, нерегулярні, близькостохастичні та стохастичні [19].

Текстура є відображенням фізичних властивостей поверхні об’єкту і відповідно до психології сприйняття володіє такими властивостями як зернистість, контраст, спрямованість, лінійна подібність і шершавість. Текстурне зображення визначимо як функцію просторової зміни значень пікселів.

Для синтезу текстурних зображень виділимо два підходи [20] (рисунок 1.6).

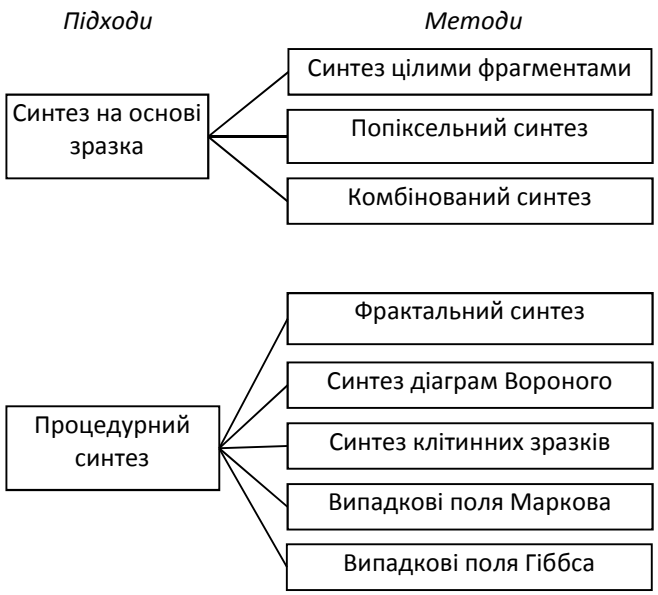

Рисунок 1.6 – Класифікація методів текстурного синтезу

Перший полягає у використанні фрагмента готової текстури, яким заповнюють вихідне зображення більше за розміром, тобто створюють одне текстурне зображення на основі одного або комбінації інших.

Другий підхід називають процедурним, який полягає у використанні певної параметричної моделі для побудови зображення. Модель реалізують у вигляді функції чи алгоритму.

Методи, які реалізують перший підхід, можна розділити за способом вибірки пікселів із вхідного зразку текстури на фрагмент-базовані та попіксельні.

Методи, що використовують фрагмент-базований синтез, дозволяють зберегти глобальну структуру створюючи текстуру, копіюючи і стикуючи разом фрагменти з різними зміщеннями [21]. Автори пропонують алгоритм, який вирівнює границі фрагментів, з обмеженням на накладання і виконує обтинання границі всередині зони накладання. Це дозволило зменшити

візуальні артефакти при накладанні фрагментів. Щоб усунути артефакти автори застосовують альфа-змішування зон накладання та різні типи структур даних, що дозволило досягти синтезу в реальному часі. Для збереження локальних ознак зразку в роботах [22] пропонують алгоритми з мозаїчним розміщенням фрагментів. Замість копіювання одного пікселя за один раз із вхідного зразку копіюють цілі фрагменти. Фрагменти вхідної текстури випадково розміщуються на вихідній сітці, результуючі границі між ними згладжуються фільтрацією.

Алгоритми, що реалізують попіксельний синтез запропоновано у роботах [23]. Вони синтезують текстуру по рядку, шукаючи і копіюючи пікселі із подібними локальними сусідами. Ці методи використовуються для заповнення зображення. Оскільки, повний перебір для найкращого піксела повільний, тоді використовують метод апроксимації найближчих сусідніх пікселів. Автори [24] удосконалили алгоритм за допомогою піраміди синтезу, що дозволило зменшити кількість сусідніх точок і підвищити якість зображення. Удосконалення алгоритму дозволило значно скоротити простір пошуку і підвищити швидкодію до рівня інтерактивного редагування .

У роботі [19] вхідний зразок розглядають як геометрично деформовану регулярну текстуру. Це дозволило визначити поле деформації, яке можна накласти на іншу вхідну текстуру. Методи, базовані на зразках, найкраще оперують з регулярними, близькорегулярними та стохастичними текстурами. Ці алгоритми можуть бути ефективнішими і швидшими, ніж методи базовані на попіксельному синтезі текстури.

Метод комбінованого синтезу текстури [25], поєднує переваги обох підходів: із вхідного зразка адаптивно вибирається фрагмент і копіюється у результуючу текстуру, наступні фрагменти вибираються таким чином, щоб мінімізувати кількість помилкових пікселів на границі фрагментів. Для безшовного з’єднання результуючих фрагментів використовується розмиття границь.

Розглянемо моделі, що визначають різні методи процедурного синтезу.

Алгоритм із роботи [26] використовує фрактальну модель. При створенні зображень фрактальних поверхонь виділено два етапи:

1) побудова поверхні $z = f(x, y)$, де x, y – координати,

2) зафарбовування поверхні $w = (g(z))$ та її візуалізація $w_{x,y} = g(f(x, y))$.

Для синтезу поверхні використовується метод випадкового серединного зсуву, який передбачає лінійну інтерполяцію точок поверхні. Але, застосувавши згладжування кубічним поліномом, можна досягнути кращих результатів синтезу.

У роботі [27] запропонована модель, яка подібна до фрактальної, але для побудови поверхні використовується згладжена псевдовипадкова послідовність. Для генерації поверхні застосовується генератор псевдовипадкових чисел, вибір параметрів якого впливає на результуюче зображення. На відміну від фрактальної моделі значення точок поверхні інтерполюються не лінійно, а сплайном. Цей метод показує добрі результати при синтезі близько-стохастичних та стохастичних текстур.

Випадкові марківські поля використані у роботі [28] для візуалізації векторного поля, особливістю візуалізації яких є створення інформативних і виразних зображень. На відміну від звичайних методів візуалізації, де використовуються графічні елементи (лінії або стрілки), у цій роботі синтезуються близько-стохастичні текстури. В обробленні біомедичних зображень синтез текстур застосовується для моделювання і візуалізації хірургічних операцій [29].

Синтез на основі клітинних зразків полягає в побудові масиву випадкових опорних точок у прямокутнику, визначення функції в кожній точці зображення, як найближчої відстані від неї до масиву опорних точок [27].

Діаграма Вороного – це набір скінченної множини точок на площині, який представляє таке розбиття площини, при якому кожна область розбиття утворює множину точок, найближчих до одного з елементів множини. При візуалізації діаграм Вороного крім параметрів опорних точок, які

використовуються в алгоритмі клітинних зразків, застосовуються додаткові параметри: колір опорної точки, спектральна складова і т. п.

Ці алгоритми використовуються для синтезу нерегулярних текстур для моделювання таких об'єктів реального світу, як луска рептилій, галька, плитка.

Інший метод текстурного синтезу базується на випадкових полях Гіббса.

Гіббсівські випадкові поля другого порядку є досить виразним засобом для моделювання складних форм [30]. З їх допомогою можна моделювати як, просторові відношення між сегментами так і прості форми. Модель дозволяє охопити ці дві ознаки одночасно. Це відкриває можливість для представлення складних форм, таких як просторові композиції простих частин.

Синтез текстури в межах статистичного підходу представляється як проблема вибірки по функції розподілу ймовірностей. Проте при моделюванні текстури як марківського випадкового поля та використання вибірки з розподілом Гіббса дослідники вказують на малу швидкодію алгоритмів .

1.3 Аналіз архітектур згорткових нейронних мереж

Згорткові нейронні мережі дуже подібні до звичайних штучних нейронних мереж: вони також складаються з нейронів, які мають певні ваги і зміщення [31]. Кожен нейрон отримує деякі вхідні дані, обчислює скалярний добуток та в деяких випадках вносить нелінійність шляхом застосування функції активації. Всю мережу можна описати, як єдину диференційовану функцію оцінки: від неопрацьованих пікселів зображення на вході до мітки класу на виході. Такі нейронні мережі також мають функцію втрат (наприклад, Sigmoid / Softmax / Tanh) на останньому (повнозв’язному) шарі.

Нейронні мережі отримують на вхід вектор і перетворюють його шляхом передавання через набір прихованих шарів. Кожен прихований шар складається з набору нейронів, де кожен нейрон повністю з'єднаний з усіма нейронами

попереднього шару, і де нейрони в одному шарі функціонують повністю незалежно і не мають жодних з'єднань. Останній повноз’язний шар називається вихідним шаром, а для завдання класифікації він представляє імовірності потрібних класів [35].

Звичайні нейронні мережі погано масштабуються у випадку з зображеннями великих розмірів. Наприклад, в датасеті комп'ютерного зору CIFAR-10, картинки мають роздільну здатність [32 × 32 × 3] (32 - ширина, 32 - висота, 3 - колірні канали RGB), тому один повністю підключений нейрон в першому прихованому шарі звичайної нейронної мережі має вагу 3072 (32 × 32×3). Здається, що це значення можна змінювати, але повнозв’язна структура не підходить для великих зображень. Картинка з більшою роздільною здатністю, наприклад, [224 × 224 × 3], призведе до того, що повністю підключений нейрон буде важити 150528. Крім того, для досягнення хороших результатів нейромережа повинна мати більше нейронів, а відповідно і шарів, що призведе до зростання параметрів. Величезна кількість параметрів може швидко привести до перенавчання [35].

Згорткові нейронні мережі (ЗНМ) стали такими популярними завдяки тому, що вхідні дані складаються із зображень, і сама побудова мережі організована більш розумним способом. На відміну від звичайної нейронної мережі, шари згорткової нейронної мережі складаються з нейронів, розташованих в 3-х вимірах: ширині, висоті і глибині (ці виміри формують вхідний об’єм). Наприклад, зображення на вході CIFAR-10 є вхідними активаційними об’ємами, які формують вихідний об’єм [32 × 32 × 3]. Крім того, результуючий вихідний шар для даного датасету складе [1×1×10] (10 – кількість класів), оскільки останній шар ЗНМ перетворить повне зображення в єдиний вектор оцінок класу. Нижче наводиться візуалізація описаного процесу. У [32], [33] та [34] описуються різні архітектури згорткових нейронних мереж для комп’ютерного розпізнавання, а також проблеми обмеженого набору навчальних даних.

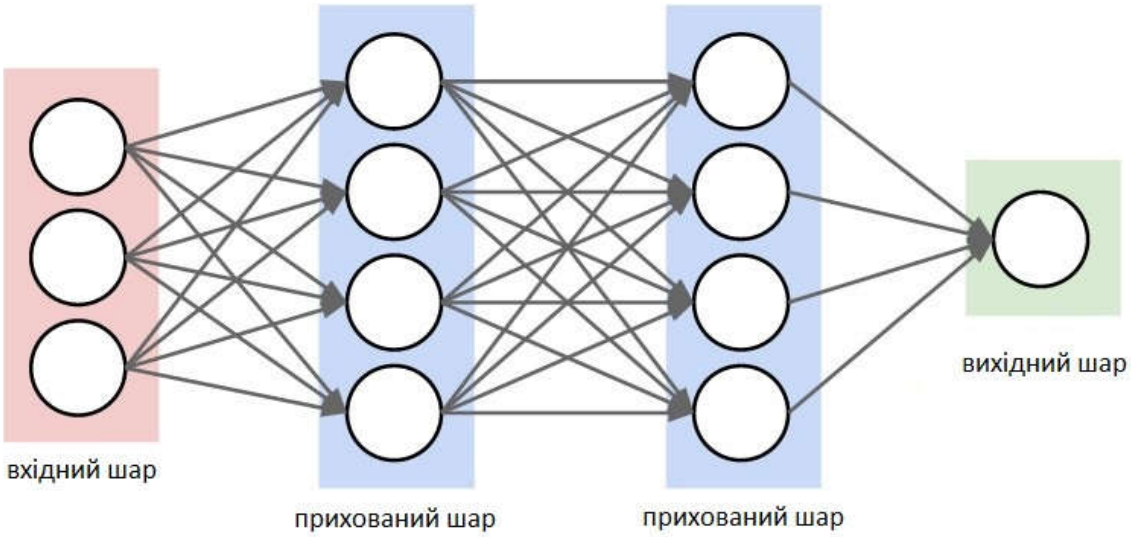

Рисунок 1.7 – Звичайна нейронна мережа

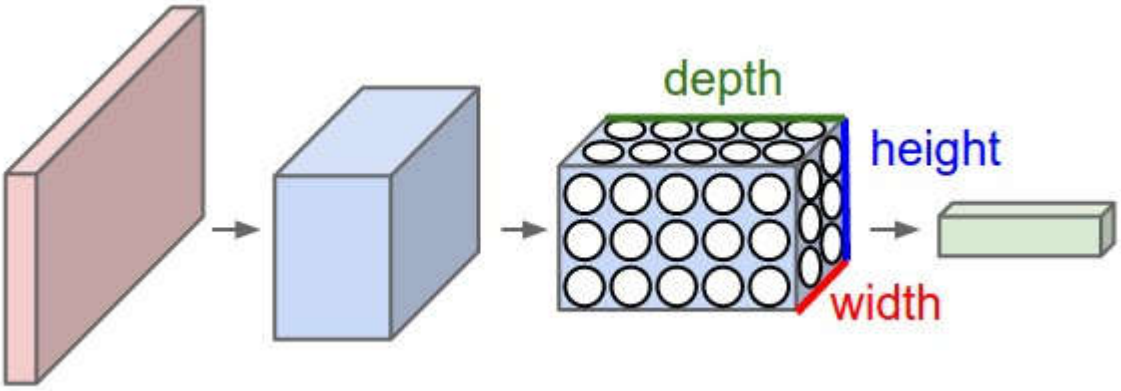

Рисунок 1.8 – Згорткова нейронна мережа

Згорткова нейронна мережі складається з трьох типів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling) та повнозв’язні (fully-connected) [35]. Загальна структура такої мережі представлена на рисунку 1.9.

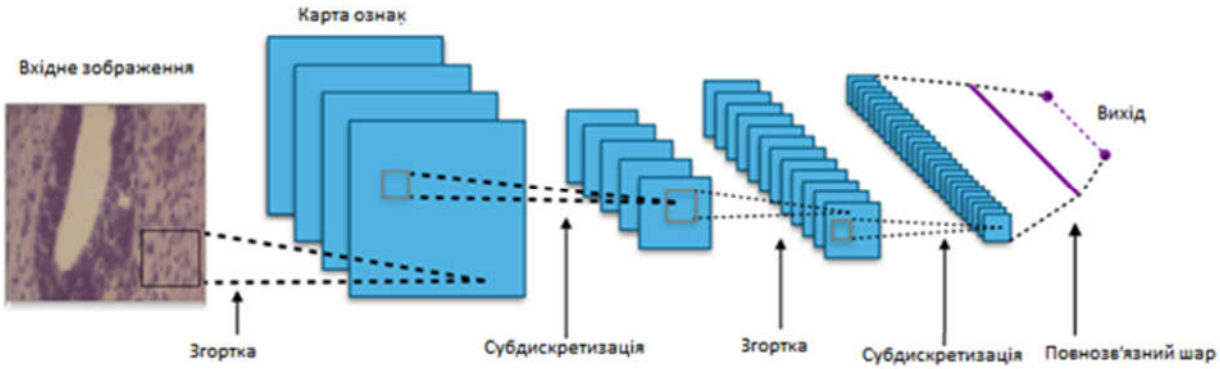

Рисунок 1.9 – Загальна структура ЗНМ

ЗНМ представляє собою певну послідовність із трьох типів шарів. Розглянемо детальніше кожен із них на прикладі датасету CIFAR-10.

У вхідному шарі INPUT [32×32×3] міститься інформація про вхідні зображення (в даному випадку 32 – ширина, 32 – висота, 3 – кольорові канали RGB);

Шар згортки CONV поелементно перемножує значення ядра на вихідні значення пікселів зображення, після чого всі множники сумуються. Після загорткового шару вихідний об’єм може зрости до [32×32×12], якщо використати 12 фільтрів. Отже, згортковий шар працює за принципом згортки, що часто використовується для обробки зображень:

$$(f \times g)[m,n] = \sum_{k,l} f[m-k, n-l] \times g[k,l], \tag{1.1}$$

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, що називається картою ознак [36].

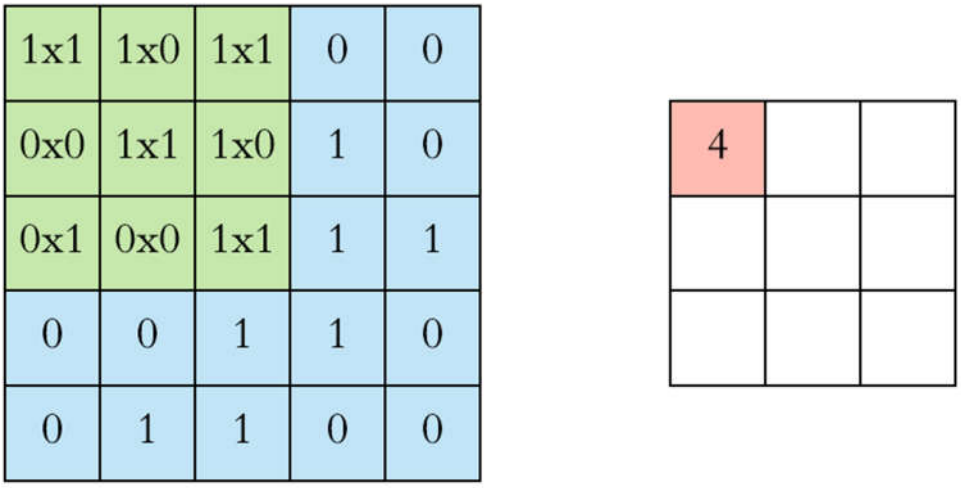

Рисунок 1.10 – Приклад згортки

Шар активації, додає нелінійність до даних шляхом застосування функції активації. Найчастіше застосовують функцію RELU, що визначається як max(0, x) і має порогове значення в нулі.

Шар пулінгу POOL або субдискретизації застосовується для зменшення розмірності вхідного об’єму, тому вхідний об’єм може зменшитися до [16×16×12]. На даному етапі відбувається ущільнення карт ознак. Логіка роботи така: якщо на попередній операції згортки вже були виявлені деякі властивості (ознаки) на зображенні, то для подальшої обробки такий детальний образ вже непотрібний, і він «ущільнюється» до менш детальної картинки. Існує декілька типів пулінгу: максимальний, середній, глобальний [37].

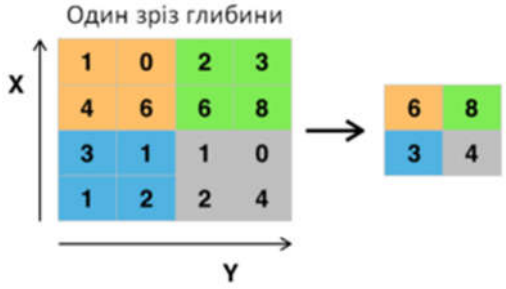

Рисунок 1.11 – Приклад максимального пулінгу

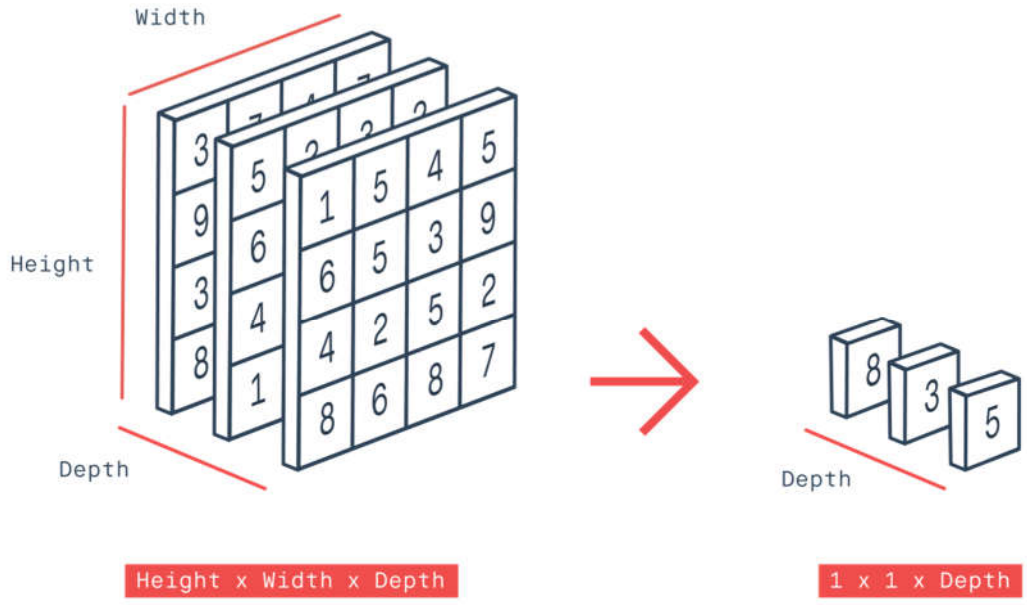

Рисунок 1.12 – Приклад глобального пулінгу

Повнозв’язний або вихідний шар виводить N-мірний вектор (N – кількість класів) для визначення потрібного класу.

При роботі з багатомірними вхідними даними, такими як зображення, зв'язувати нейрони з усіма нейронами в попередньому об’ємі недоцільно. Замість цього ми з'єднуємо кожний нейрон лише з локальною областю вхідного об’єму. Просторова протяжність цього зв'язку є гіперпараметром, що називається рецептивним полем нейрона (еквівалентно розміру ядра).

Ступінь зв'язку по осі глибини завжди дорівнює глибині вхідного об’єму. Важливо ще раз підкреслити цю асиметрію в тому, як ми розглядаємо просторові розміри (ширину і висоту), а як – глибину: з’єднання є локальними по ширині і по вистоті, але завжди проходять через всю глибину вхідного об’єму.

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.

Згортковий шар має 3 гіперпараметри, які контролюють розмір вихідної інформації: глибина, крок і доповнення нулями.

По-перше, глибина вихідного об’єму є гіперпараметром: вона відповідає кількості фільтрів, які ми хотіли б використовувати, кожен з яких навчається знаходити різні закономірності у вхідних даних. Наприклад, якщо перший згортковий шар приймає в якості вхідної інформації зображення, то різні нейрони по розмірності глибини можуть активуватися за наявності, наприклад, згустків певного кольору.

По-друге, ми повинні вказати крок, з яким ми пересуваємо фільтр по зображенні. Коли крок дорівнює 1, ми переміщуємо фільтри по одному пікселю за один раз. Коли крок дорівнює 2 (або незвичайно 3 або більше, хоча це практично рідко трапляється на практиці), фільтри одночасно перескакують по 2 пікселі, коли ми переміщуємо їх. Це призводить до просторового зменшення розмірності, тому втрачається певна інформація.

Іноді буває зручно оточувати кордон вхідного зображення нулями. Розмір цього доповнення нулями і є гіперпараметром. Ключовою особливістю доповнення нулями є те, що воно дозволяє контролювати просторовий розмір вихідних об’ємів. Найчастіше цю техніку використовують з метою збереження вхідних та вихідних значень висоти і ширини зображення однаковими [37].

Просторовий розмір вихідного об'єму можна обчислити як функцію від вхідного об’єму (W), розміру рецептивного поля нейронів згорткового шару (F), кроку, з яким вони переміщуються (S), і кількості заповнення нулями на кордоні вхідного зображення (P). Для підрахунку кількості «потрібних» нейронів є правильною формула $(W - F + 2P) / S + 1$. Наприклад, для вхідного об’єму [7×7] і фільтра [3×3] з кроком 1 і доповненням 0, на виході ми отримаємо об’єм [5×5]. З кроком 2 вихідне значення було б рівне [3×3].

Таким чином, повторюючи один за одним кілька шарів згортки і субдискретизації будується згорткова нейронна мережа. На виході мережі часто додатково встановлюють кілька шарів повнозв’язної нейронної мережі, на вхід якого подаються кінцеві карти ознак [37].

На даний час існує багато архітектур згорткових нейронних мереж. Найпопулярнішими можна назвати такі:

– LeNet. Перше успішне застосування згорткової нейронної мережі вдалося розробити Яну Лекуну в 1990-і роки. Архітектура LeNet застосовувалася для зчитування поштових індексів, цифр і т. д [39].

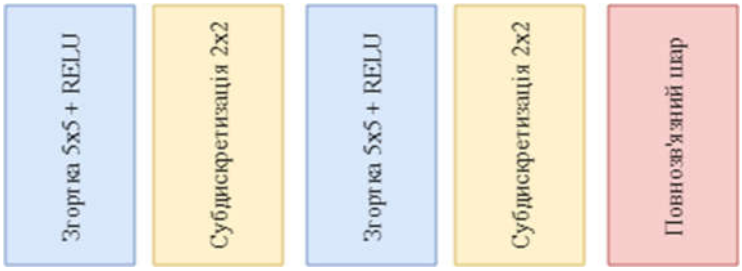

Рисунок 1.13 – Архітектура LeNet

– AlexNet. Це робота Алекса Крижевського, Іллі Суцкевера і Джеффа Хинтона, яка зіграла істотну роль в популяризації ЗНМ в області комп'ютерного зору. Архітектура AlexNet була представлена на ImageNet ILSVRC Challenge в 2012 році і обійшла всі роботи конкурентів (16% помилок проти 26% у архітектури, яка зайняла друге місце) [40].

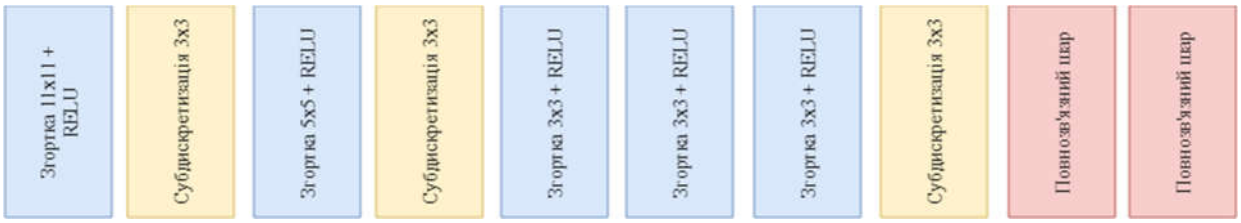

Рисунок 1.14 – Архітектура AlexNet

– ZF Net. А ось переможцем ILSVRC 2013 стала згорткова нейронна мережа Метью Зеллера і Роба Фергюса, яка відома як ZF Net (абревіатура від Zeiler і Fergus). Дана архітектура була поліпшеною версією AlexNex: тут збільшили розміри середніх згорткових шарів і зменшили крок і розмір фільтра на першому шарі [41].

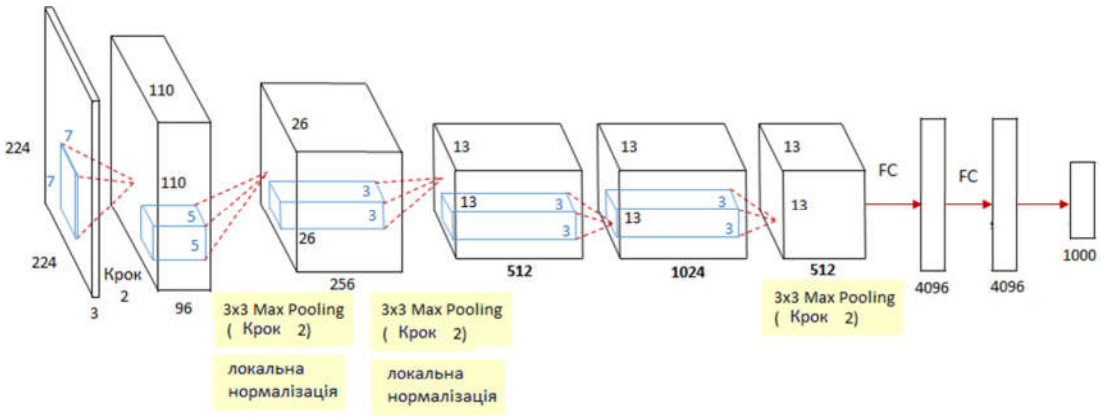

Рисунок 1.15 – Архітектура ZF Net

– GoogLeNet. У 2014 році вищезгаданий конкурс виграла ЗНМ розробки Шегеда і інших - співробітників корпорації Google. Основна заслуга даної архітектури полягає в розробці та впровадженні вхідного модуля (Inception Module), що дозволило різко скоротити число параметрів до 4 млн з 60 млн. Скорочення параметрів відбувається також завдяки заміні повнозв'язних шарів у верхній частині мережі шарами середнього пулінгу [42].

– VGGNet. Відразу за GoogLeNet на ILSVRC 2014 перемогла мережа Карен Симонян і Ендрю Ціссермана, яка стала відома як VGGNet. Розробникам вдалося наочно продемонструвати, що глибина є ключовим фактором для

продуктивності. Їх мережа містить 16 згорткових і повнозв'язних шарів і має надзвичайно однорідну архітектуру, яка виконує згортку 3x3 і пулінг 2x2 від початку до кінця. Вихідна модель доступна в режимі Plug and Play у фреймворку для глибокого навчання Caffe. Недоліком VGGNet є те, що потрібно оцінювати і використовувати набагато більше пам'яті і параметрів (140M). Більшість цих параметрів знаходяться в першому повнозв'язному шарі, і з тих пір було виявлено, що ці FC-шари можуть бути видалені без зниження продуктивності, значно зменшуючи кількість необхідних параметрів [43].

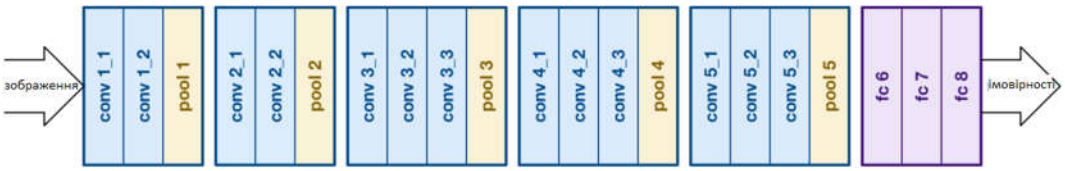

Рисунок 1.16 – Архітектура VGGNet

– ResNet. Залишкова мережа (Residual Network), розроблена Каймінгом Хе і іншими, стала переможцем ILSVRC 2015. Ключові особливості - інтенсивне використання пакетної нормалізації і спеціальні скіп-з'єднання. В кінці архітектури відсутні повнозв'язні шари. ResNet станом на сьогоднішній день є справжнім витвором мистецтва в світі згорткових нейронних мереж і використовується найчастіше [44].

Найбільші перешкоди при організації ЗНМ може чинити пам'ять. Сучасні графічні процесори мають ліміт пам'яті в 3, 4 або 6 Гб, в той час як кращі GPU мають цілих 12 Гб. Є 3 основних джерела пам'яті, за якими обов'язково потрібно стежити:

– кількість активацій на кожному шарі нейронної мережі;

– кількість параметрів нейронної мережі;

– будь-яка реалізація згорткової нейронної мережі повинна підтримувати різнорідну пам'ять: пакет даних зображень, можливо, їх вдосконалені реалізації.

Як тільки ви отримали приблизний розмір загальної кількості параметрів (для активацій, градієнтів і іншого), результат необхідно перетворити в Гб. Візьміть кількість значень, помножте його на 4, щоб отримати число байтів, а потім розділіть на 1024 кілька разів, щоб отримати пам'ять спочатку в кілобайтах, потім мегабайтах і, нарешті, гігабайтах. Якщо ваша мережа «невідповідна», тоді «підігнати» її можна шляхом зменшення пакетного розміру, оскільки більшість пам'яті зазвичай використовують активації.

1.4 Аналіз завдання магістерської роботи та постановка задач дослідження

Метою магістерської роботи є програмна реалізація синтезу гістологічних зображень на основі ЗНМ.

Для досягнення поставленої мети необхідно:

- проаналізувати алгоритми та методи синтезу біомедичних зображень;
- проаналізувати методи синтезу зображень;
- проаналізувати архітектури генеративно-змагальних мереж;
- програмно реалізувати синтез гістологічних зображень на основі ЗНМ;
- провести тестування реалізованих алгоритмів.

У першому розділі було проведено аналіз гістологічних зображень. Також було проаналізовано існуючі алгоритми та методи синтезу зображень. Крім того, було проведено аналіз згорткових нейронних мереж.

Також, у першому розділі було проаналізовано статті, в яких описані існуючі проблеми синтезу зображень та способи їх вирішення.

2 АЛГОРИТМИ СИНТЕЗУ ЗОБРАЖЕНЬ

2.1 Алгоритм синтезу на основі афінних перетворень

Афінним перетворенням простору R_n називають лінійне перетворення з наступним перетворенням зсуву [45]. В матричній формі афінне перетворення T простору R_n представляється таким чином $T(x) = Ax + a, x \in R_n$.

Афінні перетворення використовуються для розміщення початкового зображення на площині із зміною його розмірів. В даному алгоритмі використано принцип багаторазового копіювання і зміщення стартового зображення за допомогою афінних перетворень, що використовується в системах ітерованих функцій зі згущенням.

Нехай E – початкова множина. Тоді відображення множин E_1 і E_2 (рисунок 2.1) можна задати за допомогою системи двох афінних перетворень T_1 і T_2, що діють на початкову множину E:

$$\begin{cases} E_1 = T_1(E) \\ E_2 = T_2(E) \end{cases}. \tag{2.1}$$

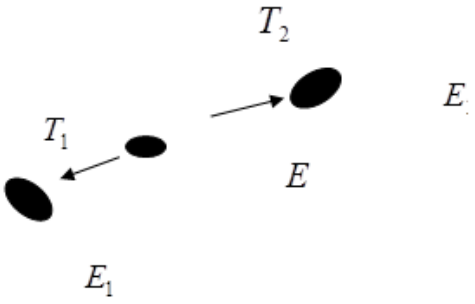

Рисунок 2.1 – Афінні перетворення над стартовою множиною

Перетворення T початкової множини представляється у вигляді шести коефіцієнтів (коефіцієнтів афінних перетворень), які задаються у матричному вигляді наступним чином:

$$E = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}, \tag{2.2}$$

де a, b, c, d, e, f – коефіцієнти афінного перетворення,

x, y – координати точок стартової (початкової) площини.

Геометричний зміст даних коефіцієнтів:

a – визначає зміну масштабу початкової множини по координаті x;

b – визначає перетворення початкової множини по координаті x;

c – визначає перетворення початкової множини по координаті y;

d – визначає зміну масштабу початкової множини по координаті y;

e – визначає зсув відображення початкової множини по осі OX;

f – визначає зсув відображення початкової множини по осі OY.

В такому випадку система (2.1), із врахуванням (2.2) перепишеться у вигляді:

$$\begin{cases} E_1 = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_1 \\ f_1 \end{bmatrix} \\ E_2 = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_2 \\ f_2 \end{bmatrix} \end{cases}. \tag{2.3}$$

При обробці початкового зображення кожна точка з координатами (x, y) перевіряється на належність стартовій множині. Якщо вона належить, то згідно (2.3) проходить обчислення координат точок (x_1, y_1) і (x_2, y_2), які відносяться відповідно до результуючих відображень E_1 і E_2.

Для визначення точок початкової множини у кольоровому зображенні проходить перевірка „неналежності” кожної точки початкового зображення до початкової множини. При цьому із стартового зображення відкидаються всі точки, що мають колір ідентичний до кольору фону, що задається користувачем. Отримане зображення (початкова множина) є стартовою для отримання бажаного розміщення цієї множини на площині [46].

Алгоритм генерування параметрів афінних перетворень. Для отримання одного відображення початкового зображення ядра клітини за допомогою афінних перетворень необхідно визначити шість коефіцієнтів a, b, c, d, e, f .

Алгоритмічно реалізовано можливість автоматичного генерування коефіцієнтів афінних перетворень для двох типів розміщення ядер клітин на площині:

– тісне розміщення. На результуючому зображенні ядра клітин розміщуються щільною групою, з невеликими відстанями одна від одної (в межах до одного діаметра ядра);

– розсіяне розміщення. На згенерованому зображенні ядра клітин розміщуються на відстанях в межах від одного до двох діаметрів ядра.

Значення коефіцієнтів *a* і *d* рівні 0.9, що задає зміну розміру вихідного відображення у сторону зменшення.

Значення коефіцієнтів *b* і *c* задаються як випадкове число в межах [0, 0.5]. Це дає змогу деформувати форму вихідного зображення.

Значення коефіцієнтів *e* і *f* обчислюються за формулами:

$$e = L + \gamma_1,$$

$$f = P + \gamma_2,$$

де *L* – попередньо визначене значення, що задає певний зсув відображення по осі *OX*;

P – попередньо визначене значення, що задає певний зсув відображення по осі OY;

γ_1 і γ_2 – випадкові числа.

Для i-го відображення значення L і P вибираються із масивів $\{L_1,\ldots,L_i,\ldots,L_n\}$ і $\{P_1,\ldots,P_i,\ldots,P_n\}$, що містять значення, які визначають необхідний для розміщення в певній області зображення зсув відносно початкового зображення, яке знаходиться в центрі екрану.

Для отримання багатоваріантного розміщення зображення ядра клітини на площині в межах вибраного типу, введені випадкові величини γ_1 і γ_2, що генеруються за рівномірним законом розподілу. Для тісного розміщення випадкові числа генеруються в межах [0, 0.5], а для розсіяного в межах [0, 1].

2.2 Архітектури генеративно-змагальних мереж

Генеративно-змагальні мережі (GAN) були запропоновані Яном Гудфеллоу у 2014 році [47]. Вони складаються із двох нейронних мереж, генератора G та дискримінатора D. Генератор намагається створити фейкові але водночас правдоподібні зображення, в той час як дискримінатор намагається відрізнити фейкові зображення (створені генератором) від реальних даних. Формально, генератор G ставить у відповідність вектору шуму z, в певному латентному просторі, вхідне зображення: $G(z) \to x$, а дискримінатор визначається як $D(x) \to [0,1]$, що класифікує зображення як реальне (близько до 1) або фейкове (близько до 0).

Порівняно з іншими генеративними моделями, такими як автоенкодери [46, 47], зображення, згенеровані за допомогою GAN зазвичай є менш розмитими і більш реалістичними. Також, теоретично доведено, що існує

оптимальна модель GAN в якій генератор створює зображення, що співпадають із розподілом реальних даних і дискримінатор завжди видає імовірність 0.5 [46]. Проте, на практиці, навчання GAN мереж є складним з декількох причин: мережам складно досягти рівноваги Неша (дискримінатор завжди видає імовірність 0.5), моделі GAN дуже часто потрапляють у так званий колапс (mode collapse) в якому генератор створює однакові зображення незалежно від вхідного вектора шуму z [49, 50, 51].

Розглянемо деякі найпопулярніші архітектури генеративно-змагальних мереж:

– Fully Connected GANs. Найпершою найбільш широко використовуваною архітектурою для генератора та дискримінатора була повнозв’язна нейронна мережа. Цю архітектуру можна застосовувати до відносно простих дата сетів, наприклад, MNIST або CIFAR-10 [53].

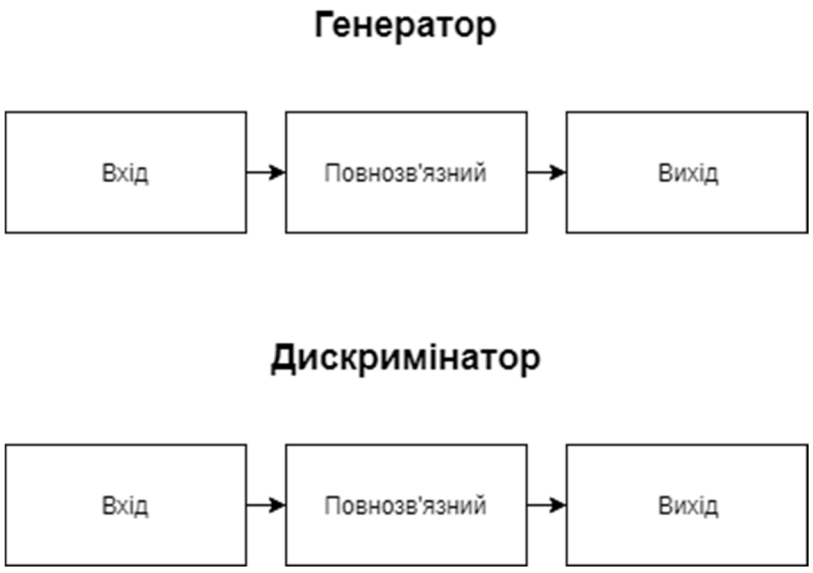

Рисунок 2.2 – FCGAN

– Deep Convolutional GANs – DCGAN. Дана архітектура була запропонована Метцом та Редфордом у 2015 році. В своїй роботі розробники пропонують використання глибокої згорткової нейронної мережі для

генератора та дискримінатора. Архітектурними особливостями є те, що автори пропонують:

– замінити всі субдискретизуючі шари на згорткові шари та регулювати зменшення розмірності гіперпараметром крок – stride;

– використовувати пакетну нормалізацію;

– не використовувати приховані повнозв'язні шари;

– використовувати функцію Tanh в моделі генератора та RELU для решти шарів;

– використовувати функцію LeakyRELU для дискримінатора.

DCGAN є більш стабільною під час навчання та дозволяє генерувати більш реалістичні зображення, проте також може потрапити у колапс [52, 53].

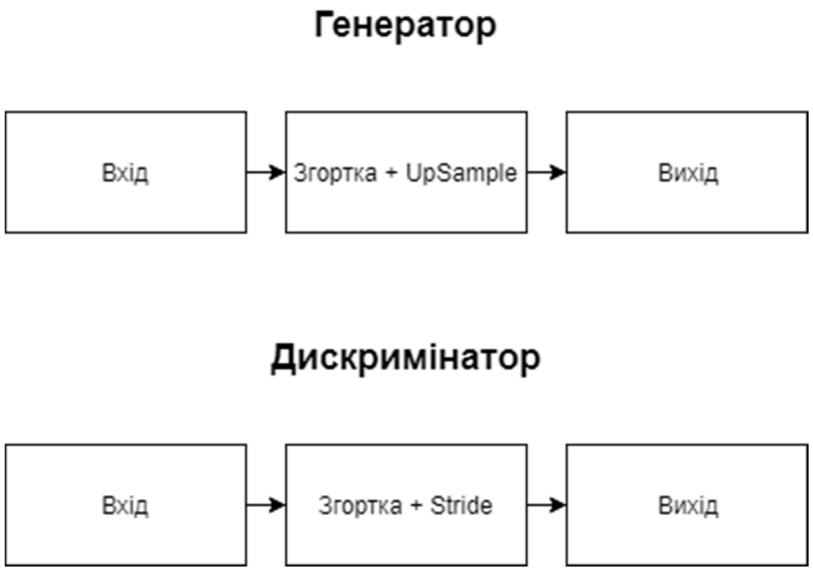

Рисунок 2.3 – DCGAN

– Conditional GAN – cGAN. Дану архітектуру представили у 2015 році два науковці – Мірза та Осіндеро. Автори пропонують додавати до обох моделей генератора та дискримінатора мітку класу, що дозволяє однією моделлю генерувати зображення, що відноситься до певного класу із навчальної вибірки. В архітектурі cGAN на вхід генератора подається вектор шуму z, що об'єднаний із міткою класу c. Також мітка класу подається і на вхід

дискримінатора разом із відповідними фейковими чи реальними даними. Даний підхід дозволяє покращити стабільність навчання та генерування вихідних зображень [52, 54, 55].

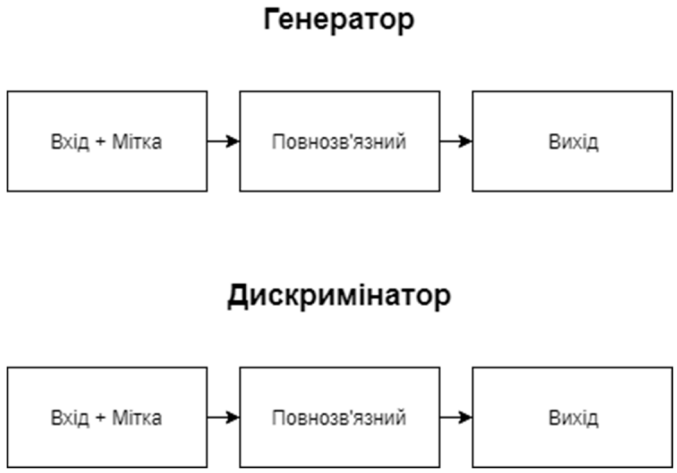

Рисунок 2.4 – CGAN

– LaplacianGAN – LAPGAN. Дана архітектура будується на основі каскаду простих GAN, що складають піраміду Лапласа з K рівнями. На найвищому рівні натренована GAN генерує зображення високої чіткості. На кожному рівні, за винятком найвищого, навчається GAN мережа, яка приймає вихідне зображення із найвищого рівня, як умовну змінну для генерування залишкового зображення на цьому рівні [57].

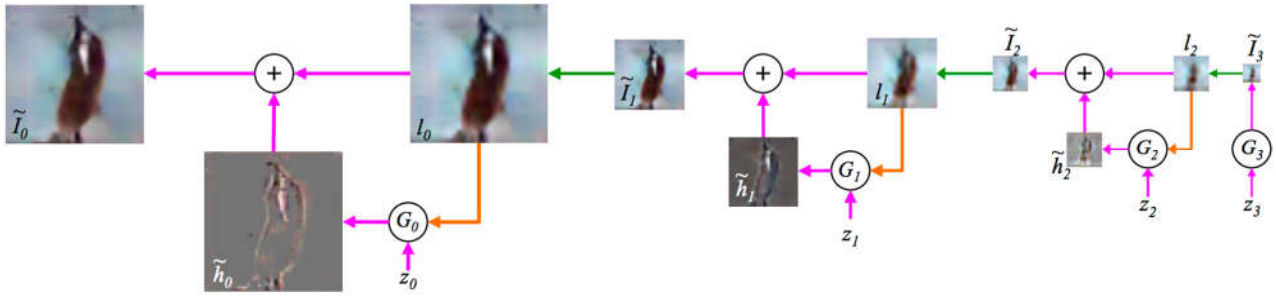

Рисунок 2.5 – LAPGAN

– Wasserstein GAN – WGAN. В оригінальній архітектурі GAN, розподіл згенерованих та реальних даних порівнювався за домопогою дивергенції Дженсена-Шеннона (JS Divergence). Застосування цієї метрики може зробити оптимізацію неможливою та призвести до затухання градієнтів, що в свою чергу приводить до колапсу та нестабільності. Тому використання іншої метрики може бути хорошим рішенням. У 2017 році Аржовський запропонував архітектуру WGAN, що використовує відстань Васерштейна замість дивергенції Шеннона [58]. На додачу, обидві нейромережі (генератор та дискримінатор) відповідають DCGAN архітектурі. WGAN надає надійну модель за допомогою більш осмисленої процедури навчання, яка здатна знайти більш глибокі залежності між розподілами даних. Незважаючи на ці теоретичні переваги, на практиці, застосування WGAN призводить до повільного процесу оптимізації. Пізніше, в цьому ж році, автори опублікували другу статтю з описом покращень для процесу навчання WGAN, які дозволяють позбутися проблеми колапсу.

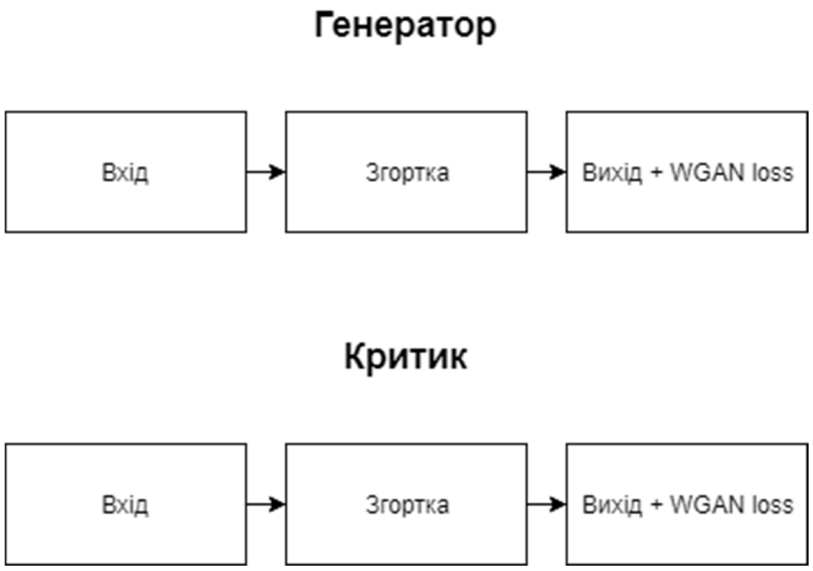

Рисунок 2.6 – WGAN

На даний час існує дуже багато архітектур GAN мереж, які застосовується для різних завдань: генерування зображень, перенесення стилів

одного зображення на інше, створення відео і т.д.. Але базовою для всіх архітектур є проста генеративно-змагальна мережа.

2.3 Алгоритми навчання генеративно-змагальних мереж

Навчання GAN моделі передбачає пошук параметрів дискримінатора, що максимізує точність його класифікації, та пошук параметрів генератора, щоб максимально «обманути» дискримінатора.

Для оцінки якості навчання використовується функція втрат $V(G, D)$, що залежить від генератора та дискримінатора [46, 58].

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))], \quad (2.1)$$

де G – генератор,

D – дискримінатор,

E – математичне сподівання.

Під час навчання, параметри однієї моделі оновлюються в той час, як іншої залишаються фіксованими. Ян Гудфеллоу показав, що для фіксованого генератора існує унікальний оптимальний дискримінатор, $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$ [40]. Це також показує, що генератор G є оптимальним, коли $p_g(x) = p_{data}(x)$, що є еквівалентним оптимальному дискримінатору, що видає імовірність 0.5 для всіх прикладів, створених з x. Тобто, генератор є оптимальним тоді, коли дискримінатор є максимально «заплутаним» і не може відрізнити реальних і фейкових зображень. Цей процес проілюстровано на рисунку нижче [47].

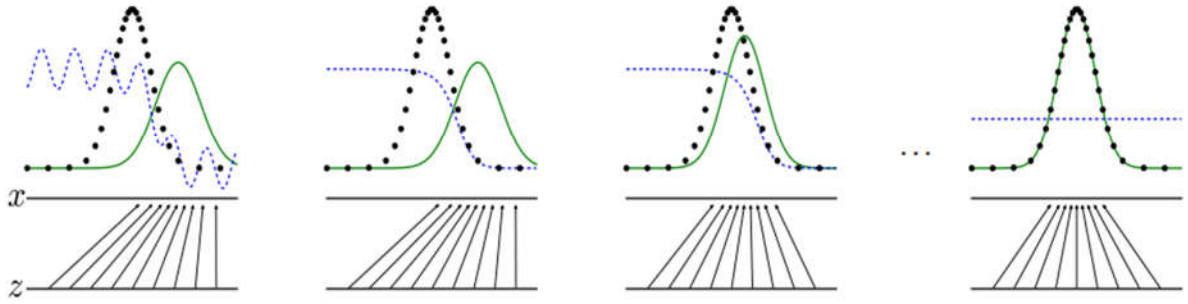

Рисунок 2.2 – Відповідність між згенерованими та реальними даними

Чорна точкова крива позначає реальний розподіл даних $P(X)$, зелена – розподіл генератора $P_g(X)$, синя – розподіл імовірності дискримінатора передбачити мітку реального об'єкта, нижня та верхня прямі – множина всіх Z та множина всіх X. Як бачимо, на рисунку 2.2 (зліва направо) розподіли даних досить різні і генератор невпевнено відрізняє їх один від одного. Далі, дискримінатор навчався протягом k кроків і вже відрізняє їх більш впевнено. Наступний крок демонструє те, як генератор починає створювати розподіл, більш подібний до розподілу реальних даних. На останньому кроці продемонстровано, як в результаті повторення попередніх кроків, два розподіли стали ідентичними – генератор навчився створювати дані, ідентичні до реальних, а дискримінатор більше не здатен відрізняти один розподіл від іншого та видає імовірність 0.5.

В ідеальному випадку, дискримінатор навчається до оптимального значення, відповідно для конкретного генератора, тоді оновлюються параметри генератора [59]. Проте на практиці, дискримінатор може навчатися не до оптимального значення, а протягом деякої кількості ітерацій, тоді генератор оновлюється разом із дискримінатором. Також, як альтернативний критерій для генератора, використовується рівняння $\max_G \log D(G(z))$ на відміну від $\min_G \log(1 - D(G(z)))$.

Не беручи до уваги теоретичне існування унікальних рішень, навчання GAN мереж є дуже складним і часто нестійким із декількох причин [54, 60, 61]. Одним із підходів до покращення навчання GAN мереж є оцінка емпіричних «симптомів», що можуть виникати під час навчання. До них відносять:

– труднощі під час сходження пари моделей G та D [57];

– генеративна модель колапсує та генерує однакові дані для різних вхідних векторів [54];

– функція втрат дискримінатора дуже швидко сходиться до нуля, відбувається затухання градієнтів.

Перші спроби пояснити чому навчання GAN мереж є нестійким зробив Гудфеллоу та Саліманс, які дійшли висновку, що градієнтні методи оптимізації, які є типовими для оновлення параметрів генератора та дискримінатора є неприйнятними, коли рішення задачі оптимізації, під час навчання GAN, фактично є сідловою точкою [60]. Саліманс навів приклад, який показує це. Однак стохастичний градієнтний спуск дуже часто використовується для навчання нейронних мереж. Також існують дуже добре розвинені середовища програмування для машинного навчання, які дозволяють легко створювати та навчати нейронні мережі, використовуючи градієнтний спуск для оновлення ваг. Наприклад, Tensorflow, Keras, PyTorch.

Мартін Аржовський у показав, що $p_g(x)$ та $p_{data}(x)$ знаходяться у нижчому за розмірністю просторі, ніж відповідний X [54]. Наслідком цього є те, що $p_g(x)$ та $p_{data}(x)$ можуть не перекриватися, і тому існує майже тривіальний дискримінатор, здатний відрізнити реальні зразки, $x \sim p_{data}(x)$, від фейкових даних, $x \sim p_g(x)$, з точністю до 100%. В цьому випадку відбувається швидке сходження помилки дискримінатора до нуля. Параметри генератора можуть бути оновлені тільки через дискримінатор, тому, коли це стається, градієнти, що використовуються для оновлення параметрів генератора також сходяться до нуля і більше не можуть бути корисними при оновленні генератора. Відбувається затухання градієнтів.

Яе Гудфеллоу у також показав, що коли дискримінатор є оптимальним, навчання генератора є еквівалентним до мінімізації дивергенції Дженсена-Шеннона між $p_g(x)$ та $p_{data}(x)$ [47]. Якщо дискримінатор не є оптимальним, оновлення може бути не таким значущим або взагалі неточним. Це теоретичне припущення дало поштовх дослідженню функцій втрат на основі альтернативних дивергенцій.

На рисунку нижче наведено алгоритм навчання GAN із оригінальної статті 2014 року [47].

Алгоритм 1 Стохастичний градієнтний спуск для генеративно-змагальних мереж. Кількість кроків k, застосованих до дискримінатора, є гіперпараметром. Ми використовуємо $k = 1$.

for кількість ітерацій **do**

for k кроків **do**

- Вибрати міні-батч розміром m (шум) $\{z^{(1)}, ..., z^{(m)}\}$ із розподілу $p_g(z)$
- Вибрати міні-батч розміром m прикладів $\{x^{(1)}, ..., x^{(m)}\}$ із початкового розподілу $p_{data}(x)$
- Оновити параметри дискримінатора збільшуючи його стохастичний градієнт:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(x^{(i)}) + \log \left(1 - D(G(z^{(i)}))\right) \Big]$$

end for

- Вибрати міні-батч розміром m (шум) $\{z^{(1)}, ..., z^{(m)}\}$ із розподілу $p_g(z)$
- Оновити параметри генератора зменшуючи його стохастичний градієнт:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(z^{(i)}))\right)$$

end for

Рисунок 2.3 – Алгоритм навчання GAN

2.4 Архітектура та алгоритм навчання генеративно-змагальних мереж для синтезу гістологічних зображень

Класичні генеративні моделі складаються з двох різних нейронних мереж: дискримінатора $D(x)$ та генератора $G(z)$, які змагаються між собою. Дискримінатор вчиться відрізняти оригінальні зображення від згенерованих, а генератор синтезує зображення, що максимально подібні до оригінальних зображень із навчальної вибірки [47].

На вхід генератора $G(z)$ поступає екземпляр z з певного розподілу ймовірністей $p_z(z)$ (рисунок 2.2). Далі генератор створює власний екземпляр, що поступає на вхід дискримінатора $D(x)$. Дискримінатор приймає на вхід екземпляр x з розподілу $p_{data}(x)$. На виході $D(x)$ отримуємо ймовірність того, що вхідний екземпляр оригінальний. Відповідно, якщо вихід $D(x)$ близький до 0, то дані згенеровані, якщо до 1 – оригінальні, або дуже подібні до них) .

Обидві нейромережі змагаються між собою. Коли дискримінатор стає кращим (краще відрізняє згенеровані зображення від оригінальних), генератор також має стати кращим (генерувати зображення, більш схожі до оригінальних), в іншому випадку він ніколи не обдурить дискримінатора. І навпаки, коли генератор стає кращим, дискримінатор також повинен стати кращим, інакше він втратить здатність відрізнити підробку від оригінальних даних.

Головною ціллю навчання дискримінатора є:

– максимізувати вихід $D(x)$ для кожного екземпляру x із оригінального розподілу $p_{data}(x)$;

– мінімізувати вихід $D(x)$ для кожного екземпляру x що не належить до оригінального розподілу $p_{data}(x)$, тобто належить до p_z.

Ціллю навчання генератора $G(z)$ є створення прикладів, які дискримінатор буде вважати за оригінальні. Оскільки на виході генератора отримуємо зображення, то його можна подати на вхід дискримінатора. Тому, генератор намагається максимізувати $D(G(z))$ або еквівалентно мінімізувати $1 - D(G(z))$. Схематично GAN мережу можна зобразити так:

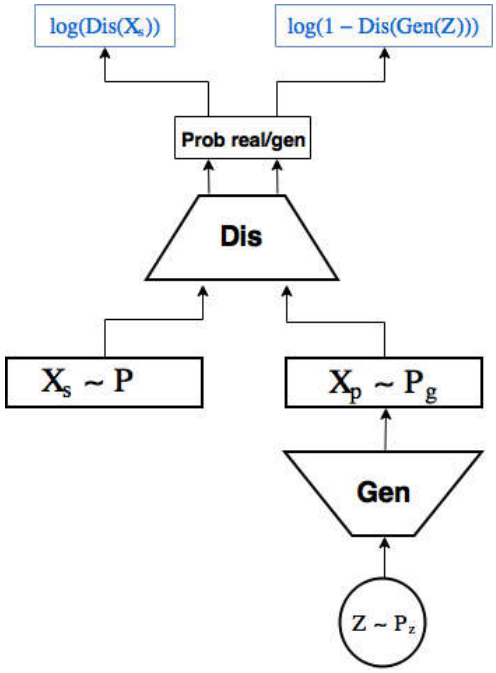

Рисунок 2.4 – Схема GAN мережі

Розглянемо детальніше процес навчання GAN мережі. Генератор та дискримінатор навчаються окремо, але в рамках однієї мережі. Робимо k кроків навчання дискримінатора: за крок навчання дискримінатора параметри θ_d оновлюються в напрямку зменшення функції втрат (в даному випадку, кросс-ентропії) [62]:

$$\theta_d = \theta_d - \nabla\theta_d(\log(D(X_S))) + \log(1 - D(G(Z))). \quad (2.2)$$

Наступним кроком оновлюється генератор: оновлюємо параметри генератора θ_g в напрямку збільшення імовірності дискримінатору присвоїти синтезованому зображенню мітку реального.

$$\theta_g = \theta_g + \nabla\theta_g \log(1 - D(G(Z))). \tag{2.3}$$

Схему навчання зображено на рисунку нижче.

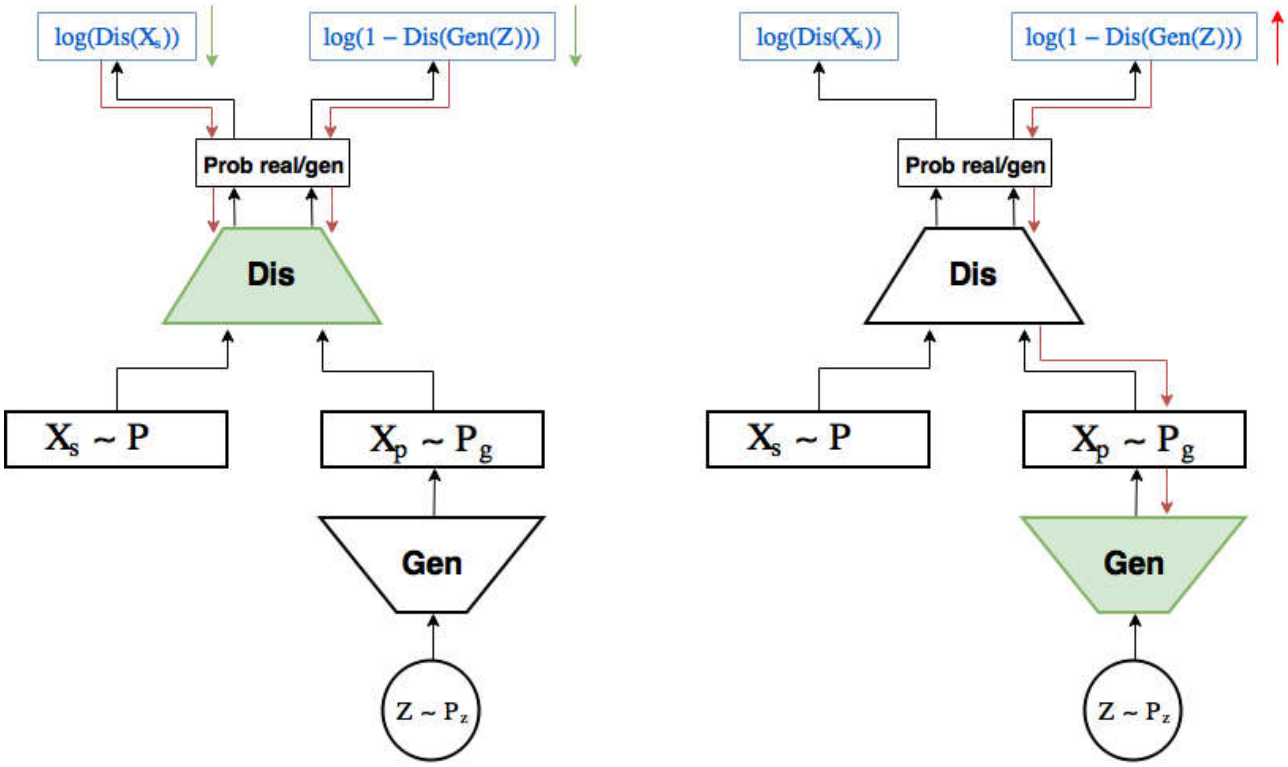

Рисунок 2.5 – Схема навчання дискримінатора (зліва) та генератора (справа)

Як видно з рисунка 2.5, під час навчання дискримінатора, градієнт (червоні стрілки) прямує від функції втрат тільки до дискримінатора, де оновлюються тільки параметри дискримінатора θ_d (зелені стрілки) в напрямку зменшення функції втрат. Коли навчається генератор, градієнт від правої частини функції втрат (помилка ідентифікації синтезованого об’єкта) прямує тільки до генератора, при цьому оновлюються тільки його параметри θ_g (зелені стрілки) в напрямку збільшення імовірності дискримінатора помилитися.

Як бачимо, генератор може оновити свої параметри тільки опираючись на функцію втрат дискримінатора, тобто генератора знаходиться у зворотному зв'язку із дискримінатором.

Класичні моделі GAN мають ряд недоліків про які було згадано у попередньому пункті. Проте основним недоліком таких моделей залишається колапс моделі – незалежно від вхідних даних, на виході отримаємо одне і те ж зображення. Так виникла WGAN архітектура, яка дозволяє частково позбавитися цієї проблеми [58].

Дана архітектура пропонує використання нової функції втрат, що використовує відстань Васерштейна, яка має «плавніший» градієнт. Проте рівняння цієї відстані є складним для розв'язання [52]. Використовуючи дуальність Канторовича-Рубінштейна дане рівняння можна переписати так:

$$W(P_r, P_\theta) = \sup_{\|f\|_{L\le 1}} E_{x\sim P_r}[f(x)] - E_{x\sim P_\theta}[f(x)], \tag{2.4}$$

де супремум обчислюється по всіх функціях Ліпшиця $f : X \to R$, які мають наступне обмеження:

$$|f(x_1) - f(x_2)| \le K\,|x_1 - x_2|. \tag{2.5}$$

Якщо замінити супремум по всіх функціях 1-Ліпшиця $\|f\|_L \le 1$ на супремум по функціях K-Ліпшиця $\|f\|_L \le K$, тоді супремум стане $K \times W(P_r, P_\theta)$ Припускаючи, що ми маємо параметризований набір функцій $\{f_w\}_{w\in W}$, де w є ваговими коефіцієнтами, а W – набором можливих ваг. Далі, припустимо, що всі ці функції є функціями K-Ліпшиця для деяких K [56]. Тоді маємо:

$$\begin{aligned} &\max_{w\in W} E_{x\sim P_r}[f_w(x)] - E_{x\sim P_\theta}[f_w(x)] \le \\ &\le \sup_{\|f\|\le K} E_{x\sim P_r}[f(x)] - E_{x\sim P_\theta}[f(x)] = K \times W(P_r, P_\theta). \end{aligned} \tag{2.6}$$

Отже, ми хочемо навчити $P_\theta = g_\theta(Z)$ генерувати розподіл, рівний P_r. Інтуїтивно, при заданому g_θ ми можемо обчислити оптимальну f_w для відстані Васерштейна. Далі, можемо виконати зворотне поширення помилки через $W(P_r, g_\theta(Z))$ для обчислення градієнту для θ.

$$\begin{aligned} &\nabla_\theta W(P_r, P_\theta) = \nabla_\theta (E_{x \sim P_r}[f_w(x)] - E_{z \sim Z}[f_w(g_\theta(z))]) = \\ &= -E_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]. \end{aligned} \tag{2.7}$$

Процес навчання поділений на 3 кроки:

– для фіксованого θ обчислити апроксимацію $W(P_r, P_\theta)$ шляхом навчання f_w до сходження;

– якщо знайшли оптимальну f_w, обчислюємо градієнт θ за формулою $-E_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]$ шляхом генерування декількох $z \sim Z$;

– оновлюємо θ та повторюємо процес [62].

Таким чином, дискримінатор не намагається класифікувати зображення як реальні чи фейкові, а дає оцінку того, наскільки згенероване зображення відрізняється від реального. Дискримінатор вчиться обчислювати відстань Васерштейна між розподілами. Коли функція втрат зменшується під час навчання, відстань Васерштейна стає меншою і генератор створює зображення, що наближаються до реального розподілу даних.

Ці формули працюють тільки тоді, коли набір функцій $\{f_w\}_{w \in W}$ є Ліпшицевими K-функціями [55]. Для того, щоб все працювало, потрібно після кожного оновлення градієнта, обмежувати ваги маленькими значеннями, наприклад, [-0.01, 0.01]. На рисунку 2.6 наведено алгоритм навчання WGAN із оригінальної статті [58].

Алгоритм 2 WGAN, пропонований алгоритм. Всі експерименти у статті використовували стандартні значення $\alpha = 0.00005, c = 0.01, m = 64, n_{critic} = 5$

Вимоги: α, норма навчання. c, параметр обмеження. m, розмір батчу. n_{critic}, кількість ітерацій критика перед навчанням генератора.

Вимоги: w_0, початкові параметри критика. θ_0, початкові параметри генератора.

while θ не сходиться **do**
  for $t = 0, ..., n_{critic}$ **do**
    Вибрати $\{x^{(i)}{}_{i=1}^{m}\} \sim \mathbf{P_r}$ батч з реальних даних.
    Вибрати $\{z^{(i)}{}_{i=1}^{m}\} \sim p(z)$ батч із згенерованих даних.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \right]$
    $w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$
    $w \leftarrow \mathrm{clip}(w, -c, c)$
  end for
  Вибрати $\{z^{(i)}{}_{i=1}^{m}\} \sim p(z)$ батч із згенерованих даних.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \mathrm{RMSProp}(\theta, g_\theta)$
end while

Рисунок 2.6 – Алгоритм навчання WGAN

Але і ця архітектура має свої недоліки. Як повідомляють автори оригінальної статті, обмеження ваг є жахливим способом для дотримання обмеження Ліпшицевого відображення. Якщо параметр дуже великий, то може знадобитися дуже багато часу для того, щоб всі ваги досягли своєї межі, що в свою чергу робить складним навчання критика (дискримінатора). Якщо параметр надто малий, це може призвести до затухання градієнтів, коли кількість шарів є дуже великою або не застосовується пакетна нормалізація. Автори повідомляють: «Ми застрягли на обмеженні ваг завдяки своїй простоті та вже хорошій продуктивності» [58]. Тому автори пропонують нову архітектуру WGAN-GP, що є покращеною версією попередньої [64].

В новій архітектурі пропонується новий спосіб для забезпечення обмеження Ліпшицевого відображення – застосування обмеження градієнту.

Диференційована функція f є функцією Ліпшиця тоді і тільки тоді, коли вона має градієнти, норма яких не є більшою за 1. Тобто замість застосування обмеження ваг, архітектура WGAN-GP штрафує модель, якщо норма градієнта перевищує одиницю [64].

$$L = \underset{\tilde{x}\sim P_g}{E}[D(\tilde{x})] - \underset{x\sim P_r}{E}[D(x)] + \lambda \underset{\hat{x}\sim P_{\hat{x}}}{E}[(\left\|\nabla_{\hat{x}} D(\hat{x})\right\|_2 - 1)^2]. \quad (2.8)$$

Також автори не рекомендують [63] використовувати пакетну нормалізацію для критика, бо вона створює певну кореляцію між даними одного пакета. Це впливає на ефективність застосування штрафу для градієнту, як повідомляють автори [66]. Алгоритм навчання WGAN-GP із оригінальної статті наведено на рисунку нижче.

Алгоритм 3 WGAN із обмеженням градієнту. Ми використовували стандартні значення $\lambda = 10, n_{critic} = 5, \alpha = 0.0001, \beta_1 = 0, \beta_2 = 0.9$

Вимоги: λ, коефіцієнт обмеження градієнту. n_{critic}, кількість ітерацій критика перед навчанням генератора. m, розмір батчу. α, β_1, β_2, гіперпараметри методу оптимізації Adam.

Вимоги: w_0, початкові параметри критика. θ_0, початкові параметри генератора.

while θ не сходиться **do**
  for $t = 1, ..., n_{critic}$ **do**
    for $i = 1, ..., m$ **do**
      Вибрати приклад $x \sim \mathbf{P_r}$ із реальних даних, латентний вектор $z \sim p(z)$, випадкове число $\epsilon \sim U[0, 1]$
      $\tilde{x} \leftarrow G_\theta(z)$
      $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
      $L^{(i)} \leftarrow D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Вибрати приклад із латентних змінних $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2)$
end while

Рисунок 2.7 – Алгоритм навчання WGAN-GP

Для синтезу гістологічних зображень буде використовуватися архітектура DCGAN + WGAN-GP + CGAN (Deep Convolutional Wasserstein Conditional Generative Adversarial Network). Дану архітектуру обрано тому, що вона є більш стабільною під час навчання та уникає колапсу завдяки обмеженню градієнта.

У другому розділі проаналізовано алгоритми синтезу зображень на основі афінних перетворень, архітектури генеративно-змагальних мереж. Також проаналізовано алгоритми навчання цих мереж та обрано архітектуру мережі для синтезу гістологічних зображень.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАВДАННЯ

3.1 Системні вимоги

Для правильного функціонування розробленого програмного модуля на персональному комп’ютері, він повинен відповідати наступним мінімальним системним вимогам:

– процесор з тактовою частотою не менш ніж 2,2ГГц;

– оперативна пам’ять обсягом 8Гб та поколінням пам’яті DDR4 або DDR3 (наприклад, Kingston DDR4-2400 HyperX Fury);

– відеокарта з підтримкою технології CUDA та обсягом пам’яті не менш ніж 4Гб та з поколінням пам’яті не нижче GDDR5 (наприклад, GeForce GTX 1070 8GB);

– жорсткий диск розміром 50Гб.

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

– операційна система Windows 10 x64 або Ubuntu x64;

– середовище Python 3.6;

– середовище Anaconda із встановленим Tensorflow ≥ 1.13;

– остання версія драйверів для відеокарти.

3.2 Підготовка віртуальної машини та програмного середовища

Перед початком реалізації проекту мій домашній комп’ютер було протестовано шляхом навчання класифікатора (згорткової нейронної мережі) для кольорових гістологічних зображень розміром 128x128 пікселів. В ході тестування було отримано помилки типу Out of memory error. Ці помилки виникають, коли на комп’ютері недостатньо пам’яті, щоб виділити її під

завантажені вхідні дані та модель. Отже, домашній комп'ютер не підходить для реалізації проекту.

Таблиця 3.1 – Конфігурація домашнього 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
Операційна система	Windows 10 Pro

Існує три можливі варіанти вирішення проблеми.

Перший варіант полягає у створенні віртуальної машини у сервісі Google Cloud Platform – запропонований компанією Google набір хмарних служб, які виконуються на тій же самій інфраструктурі, яку Google використовує для своїх продуктів призначених для кінцевих споживачів, таких як Google Search та YouTube. Окрім інструментів для керування, також надається ряд модульних хмарних служб, таких як обчислення, зберігання даних, аналіз даних та машинне навчання. Для реєстрації потрібно мати банківську карту або банківський рахунок.

Для того, щоб створити віртуальну машину потрібно перейти в Google Cloud Console, створити новий проект та обрати пункт Compute Engine у меню. Після чого відкриється вікно, зображене на рисунку 3.1, де можна обрати характеристики віртуальної машини.

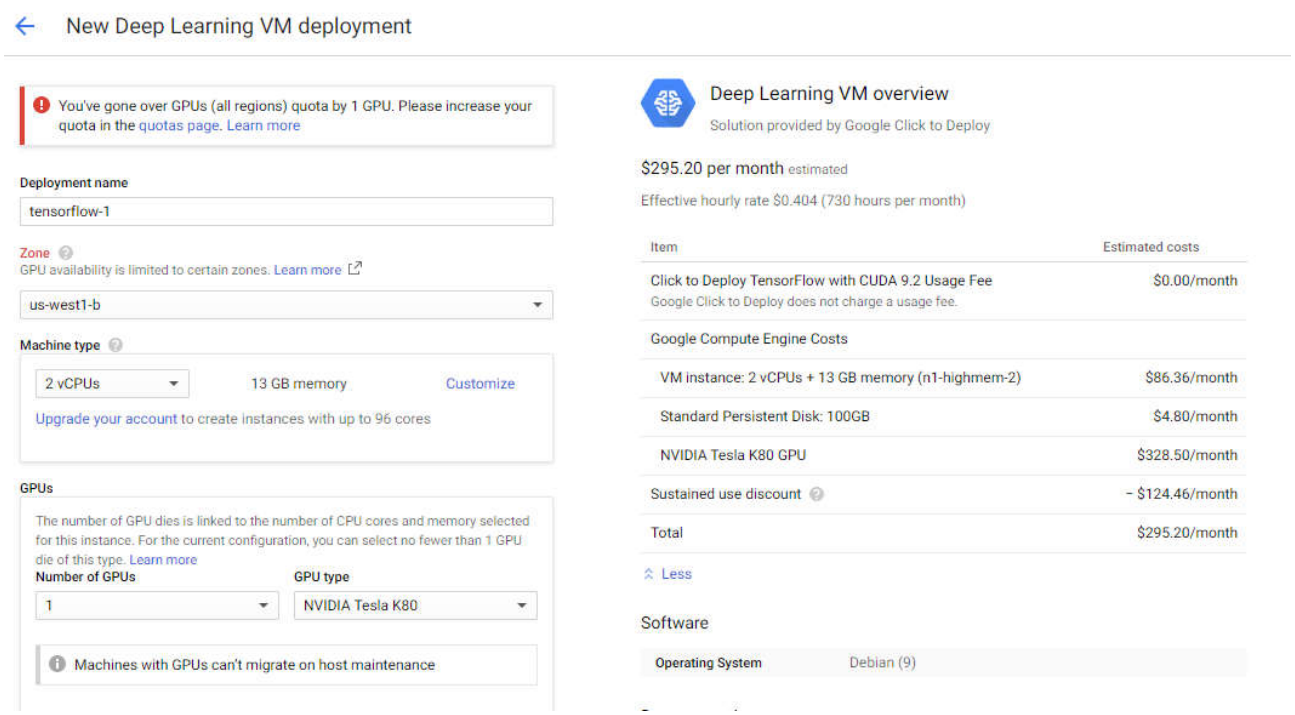

Рисунок 3.1 – Створення віртуальної машини в Google Cloud

Також у користувача є можливість збільшити або зменшити обсяг оперативної пам’яті та кількості ядер.

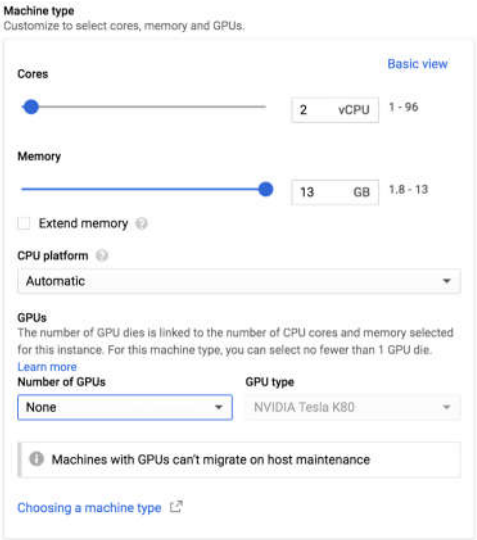

Рисунок 3.2 – Конфігурація VM

Підсумуємо характеристики віртуальної машини, яка буде оптимальним варіантом для реалізація поставленого завдання, у таблиці 3.2.

Таблиця 3.2 – Характеристики віртуальної машини

Процесор	2 ядра + 13Гб пам’яті
Диск	100Гб
Графічний процесор	Nvidia Tesla K80, 12Гб
Ціна за місяць	296 $
Ціна за годину	0.5 $

Як бачимо з таблиці 3.2, віртуальна машина із наведеною конфігурацією буде обходитися нам у майже 300 американських доларів. Під ціною за місяць мається на увазі, що віртуальна машина (VM) буде працювати 7 днів на тиждень та 24 години на добу, тобто буде працювати неперервно. Для того, щоб уникнути зайвих витрат не потрібно забувати, що VM потрібно вимикати, коли немає необхідності у її використанні. Ціна за одну годину роботи становить 0,5 долара.

Для того, щоб користувачі мали змогу порахувати ціну використання віртуальної машини, команда Google пропонує використовувати онлайн Google Cloud Pricing Calculator, де користувачу необхідно вказати бажані характеристики VM, після чого можна переглянути ціну обраної віртуальної машини.

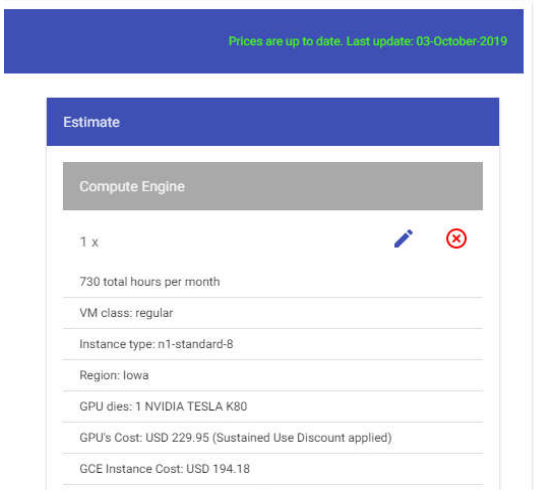

Рисунок 3.3 – Google Cloud Pricing Calculator

Другий варіант також полягає у створенні віртуальної машини, але вже у сервісі AWS (Amazon Web Services) – надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки. Існує і безкоштовна підписка, яка доступна протягом перших 12 місяців. Технологія дозволяє абонентам мати у своєму розпорядженні повноцінний віртуальний кластер комп'ютерів, який завжди доступний через Інтернет. Віртуальні комп'ютери AWS мають більшість атрибутів реального комп'ютера, включаючи апаратні пристрої (процесор, відеокарту, локальну та оперативну пам'ять, жорсткий диск або SSD-накопичувач); операційну систему на вибір; мережу; і попередньо встановлені прикладні програми, такі як веб-сервер, база даних, CRM і т. д. Кожна система AWS також віртуалізує консольний ввід/вивід (клавіатура, дисплей і миша), що дозволяє користувачам AWS підключитися до своєї системи AWS за допомогою браузера. Браузер виступає як вікно у віртуальний комп'ютер, дозволяючи користувачу входити в систему, налаштовувати та використовувати свої віртуальні системи так само, як справжній, фізичний комп'ютер. Це дозволяє їм налаштувати систему так, щоб надавати інтернет-орієнтовані сервіси та послуги своїм клієнтам.

Під час першої реєстрації у сервісі AWS у користувача є можливість "безкоштовного" використання деяких сервісів протягом 12 місяців. Ця послуга називається Free Tier і накладає певні обмеження на використання тих чи інших сервісів. Наприклад, для сервісу EC2 (віртуальні машини) стоїть обмеження у 750 годин роботи. Тобто за 750 годин використання віртуальної машини користувач не буде платити. Але є один важливий нюанс. Якщо створювати VM типу t3 або t2, то в них автоматично вмикається послуга, яка називається unlimited feature.

Кожна віртуальна машина має певний поріг базового використання процесора – baseline. Уявимо, що користувач виконує якісь складні завдання і процесор віртуальної машини завантажений набагато більше, ніж це передбачає baseline, а може навіть і на всі 100 відсотків. В такому випадку, якщо у

користувача увімкнена послуга unlimited feature, то він буде змушений платити додаткові гроші за використання процесора.

Але якщо у користувача увімкнено standart feature – процесор не буде навантежений більше, ніж це передбачає baseline. Тобто, якщо частота підніметься вище baseline, вона одразу ж буде знижена.

Для реалізації завдання потрібно обрати віртуальну машину із досить хорошими характеристиками, а основне із потужним відеоадаптером.

Віртуальна машина на AWS, що є схожою за конфігурацією до VM (таблиця 3.3) на Google Cloud обійдеться у 0,9 долара за годину, що майже у два рази дорожче. Проте, варто врахувати, що у цій VM процесор має 4 ядра, а не два, та кількість оперативної пам'яті становить 61Гб, а не 13.

Таблиця 3.3 – Характеристики віртуальних машин AWS

Назва	GPU	vCPU	RAM (GB)	Ціна/год.
p2.xlarge	1	4	61	0.9 $
p2.8xlarge	8	32	488	7.2 $
p2.16xlarge	16	64	732	14.4 $

Третій варіант полягає у використанні сервісу Google Colaboratory, який надає змогу користувачеві запускати Jupyter Notebook («ноутбук», інтерактивне виконання коду на мові Python) та можливість використовувати графічний процесор Nvidia Tesla K80 безкоштовно. Проте, є один нюанс – середовище перезапускається кожних 12 годин. Тобто всі дані, які користувач завантажить у свій «ноутбук» зникнуть через 12 годин а всі виконувані процеси (в тому числі і навчання нейромережі, якщо користувач використовує «ноутбук» для цього) припиняться. Даний варіант підійде тим, хто не має бажання витрачати кошти на дорогі графічні процесори, встановлювати драйвери та налаштовувати середовище програмування самостійно.

Технічні характеристики такого «ноутбука» наступні:

Таблиця 3.4 – Технічні характеристики Google Colab

Процесор	Intel Xeon 2,3ГГц, 1 ядро
Диск	320Гб
Пам’ять	12,6Гб
Графічний процесор	Nvidia Tesla K80, 12Гб
Кожні 12 годин дані із диска, оперативної пам’яті, кешу процесора тощо, які будуть розміщені на виділеній віртуальній машині, будуть стерті	

Плюсом даного сервісу є те, що користувачу непотрібно самостійно встановлювати популярні бібліотеки для машинного навчання, такі як Tensorflow, PyTorch – вони встановлені за замовчуванням. Користувач зосереджений тільки на написанні коду. Також є можливість запускати не тільки Jupyter Notebook, а й звичайні файли коду Python.

Переваги Google Colaboratory:

– безкоштовна підтримка GPU;

– Google Colab дозволяє розробникам використовувати та обмінюватися ноутбуками Jupyter між собою;

– підтримує команди Bash;

– усі основні бібліотеки Python, такі як TensorFlow, Scikit-learn, Matplotlib тощо, попередньо встановлені;

– побудований на основі Jupyter.

Замість використання Jupyter, Google Colab надає користувачам хмарну систему обчислень, щоб вони могли віддалено ділитися файлами з іншими розробниками.

Крім того, це значно зменшує час на налаштування середовища, бо у сервісі є багато встановлених бібліотек та залежностей. Але головна перевага Google Colab – це її безкоштовний сервіс GPU.

Зараз кожен може безкоштовно навчати свої моделі глибокого навчання з будь-якої точки світу. Скористаємося цим варіантом.

3.3 Засоби та платформа реалізації

Проаналізувавши поставлену задачу, зрозуміло, що перевага має надаватися тій мові програмування, яка найбільш пристосована до роботи із машинним навчанням та великими даними. По суті, машинне навчання – це технологія, яка допомагає додаткам на основі штучного інтелекту навчатися і видавати результати автоматично, без людського втручання. Спеціаліст з машинного навчання повинен збирати, систематизувати і аналізувати дані, а потім на основі отриманої інформації створювати алгоритми для штучного інтелекту. Python найкраще підходить для виконання таких завдань, тому що він досить зрозумілий в порівнянні з іншими мовами. Більш того, у нього відмінна продуктивність при обробці даних.

Одна з основних причин, чому Python використовується для машинного навчання полягає в тому, що у нього є безліч фреймворків, які спрощують процес написання коду і скорочують час на розробку. До таких фреймворків входять Tensorflow, Keras, PyTorch, Theano, Caffe та багато інших. Плюсом є те, що зараз кожна така бібліотека підтримує обчислення на графічних процесорах із коробки. Користувачу залишається тільки встановити програмний пакет CUDA та найсвіжіші драйвери для своєї відеокарти.

Для реалізації поставленого завдання обрано фреймворк Tensorflow версії 1.14 – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Її зараз застосовують для досліджень та розробки власних продуктів у Google.

В якості віртуальної машини обрано сервіс Google Colaboratory та мову програмування Python 3.6.

3.4 Вхідні дані та архітектура мережі

Оскільки завдання полягає у генеруванні гістологічних зображень, на рисунку нижче наведено всю навчальну вибірку цих зображень. Дані у навчальній вибірці є кольоровими RGB зображеннями розміром 128x128 пікселів.

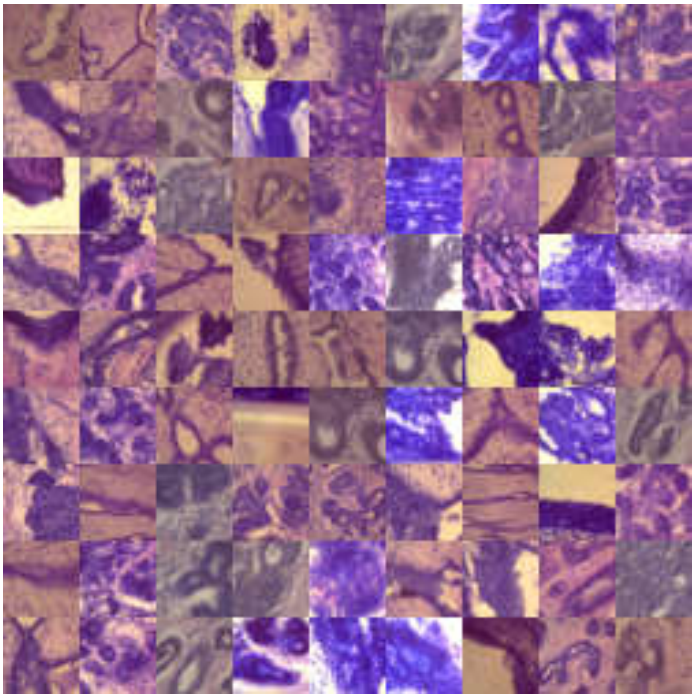

Рисунок 3.4 – Оригінальна вибірка гістологічних зображень

Оригінальні зображення поділені на 5 класів. Розмір оригінальної вибірки для гістології 91 зображення. Очевидно, що це дуже мало, тому навчальну вибірку було розширено приблизно до 1000 зображень шляхом афінних перетворень (масштабування, поворот, збільшення і т.д.) з використанням утиліти Rudi.

Приклад афінних перетворень над зображеннями наведено на рисунку 3.5.

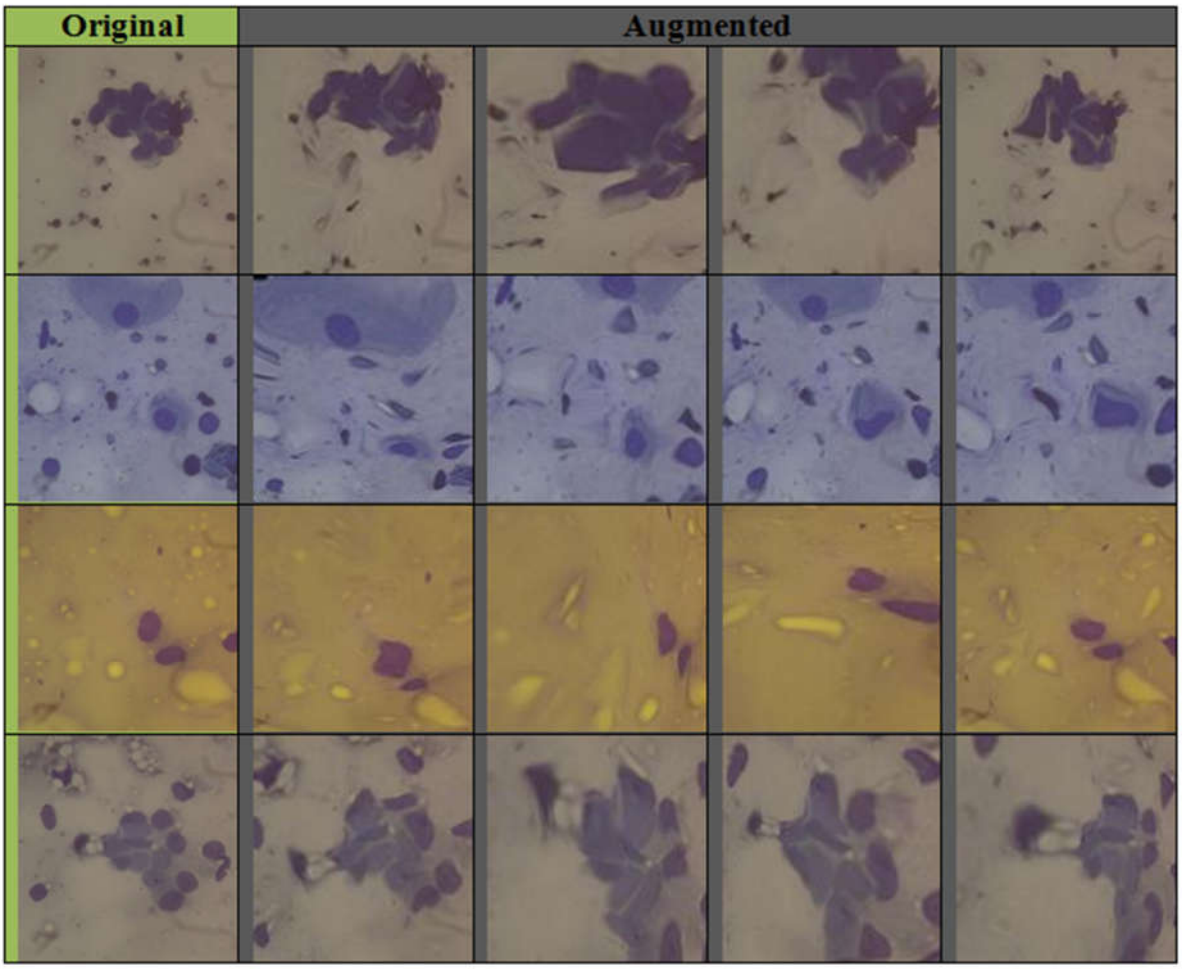

Рисунок 3.5 – Приклад афінних перетворень над зображеннями

На рисунку 3.6 наведено розширену вибірку після застосування афінних перетворень до оригінальних зображень.

Для генерування таких зображень я пропоную побудувати нейрону мережу за архітектурою WGAN – Wasserstein Generative Adversarial Network, яка була описана у другому розділі. Також пропоную модифікувати дану архітектуру для розширення її можливостей – додати блок, що буде відповідати за генерування зображень певного класу. В кінцевому випадку отримаємо Conditional Wasserstein Generative Adversarial Network.

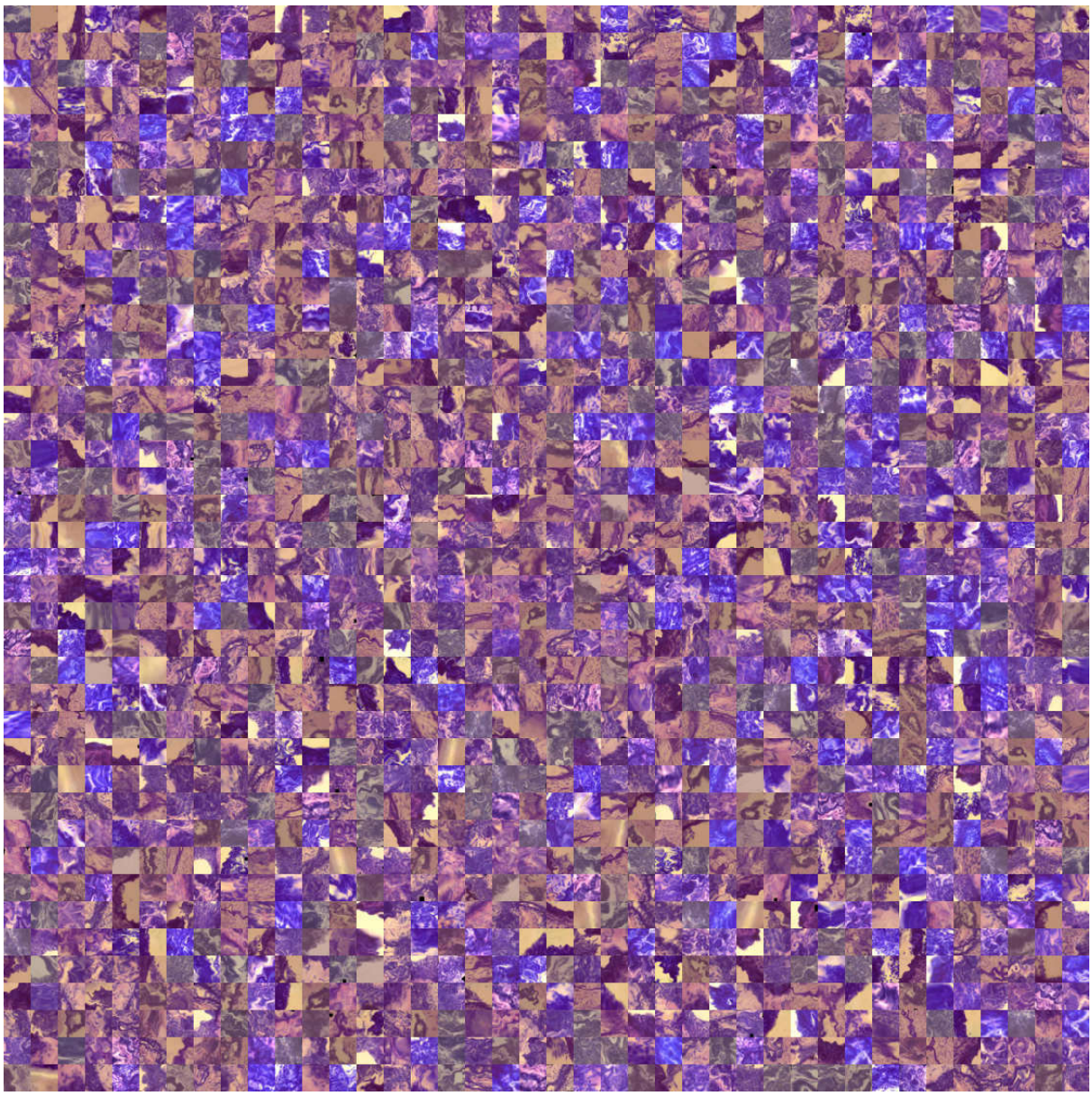

Рисунок 3.6 – Розширена навчальна вибірка

Дискримінатором є глибока згорткова нейромережа, на вхід якої поступають кольорові зображення розміром 128x128 пікселів, а на виході значення 0 або 1 (вхідне зображення фейкове або реальне). Дана мережа складається з 6 згорткових шарів із розміром вікна 5 пікселів. Після кожного шару згортки застосовано шар dropout. В перших 5 згорткових шарах використовується крок рівний 2 для зменшення розмірності зображення. Згорткові шари використовують функцію Leaky ReLU, $f(x) = max(x, leak \times x)$, в якості функції активації. Останнім шаром іде повнозв'язний шар з одним виходом та функцією активації – сигмоїда.

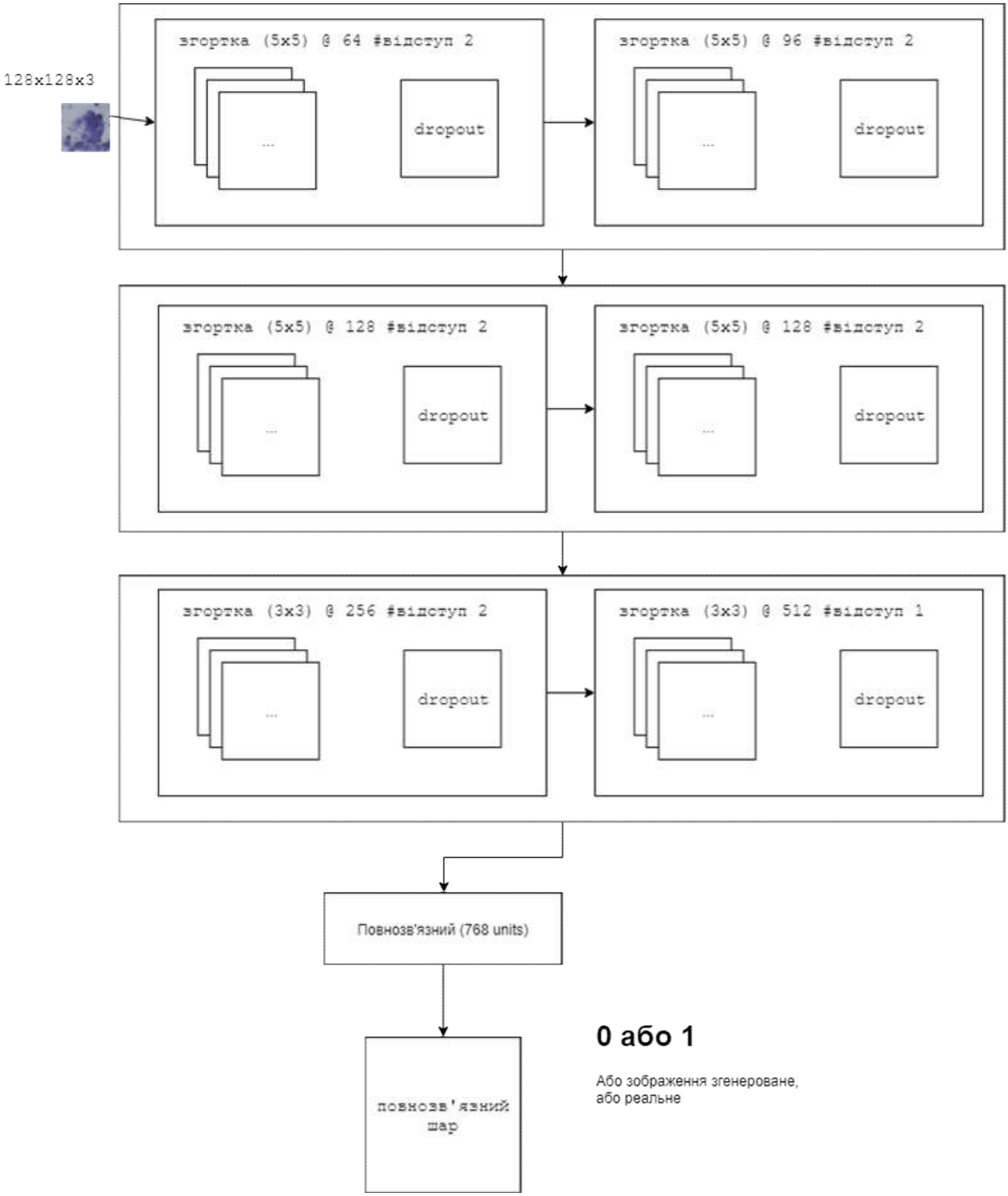

Рисунок 3.7 – Архітектура дискримінатора

На вхід генератора поступає вектор, що містить 64 випадкові числа з розподілом Гауса із середнім 0 та стандартним відхиленням 1. Дана нейромережа складається з повнозв’язного шару та 7 деконволюційних шарів з розміром вікна 5x5 пікселів для збільшення зображення. Після кожного такого

шару застосовано шар dropout та batch-normalization. На всіх шарах застосовано функцію активації Leaky ReLU. Останній шар використовує гіперболічний тангенс, як функцію активації.

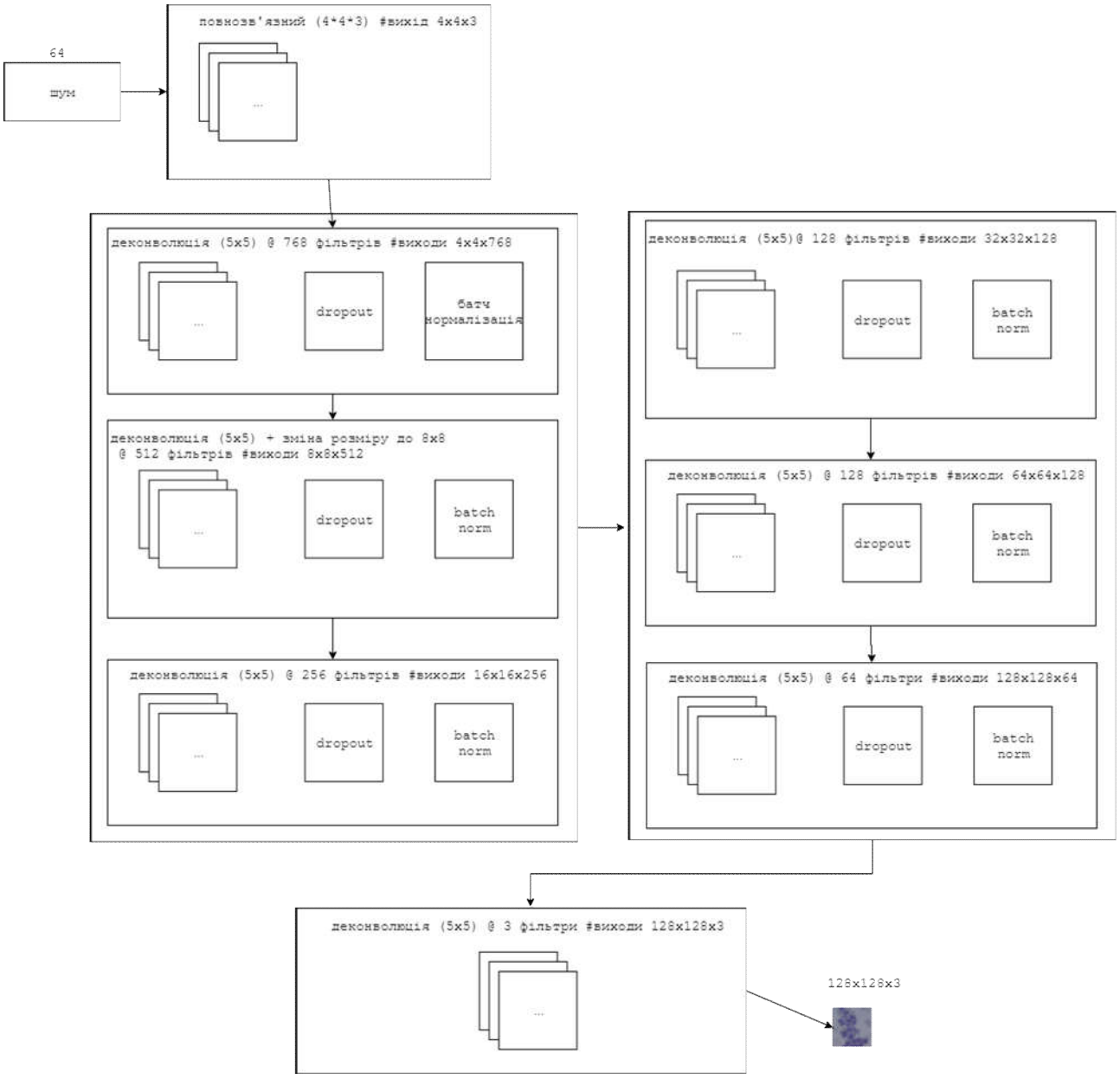

Рисунок 3.8 – Архітектура генератора

3.5 Параметри навчання та отримані результати

Обидві частини мережі (дискримінатор та генератор) скомпільовані з використанням оптимізатора Adam. Швидкість (норма) навчання дискримінатора та генератора становить $1e-3$. До обох мереж застосовано ℓ_2 регуляризацію з параметром регуляризації $1e-6$. У функції Leaky ReLU параметр $leak$ встановлений у значення 0.2. Мережа навчалася протягом 3000 епох. Час навчання склав приблизно 9 годин.

На рисунку нижче наведено приклад згенерованих гістологічних зображень.

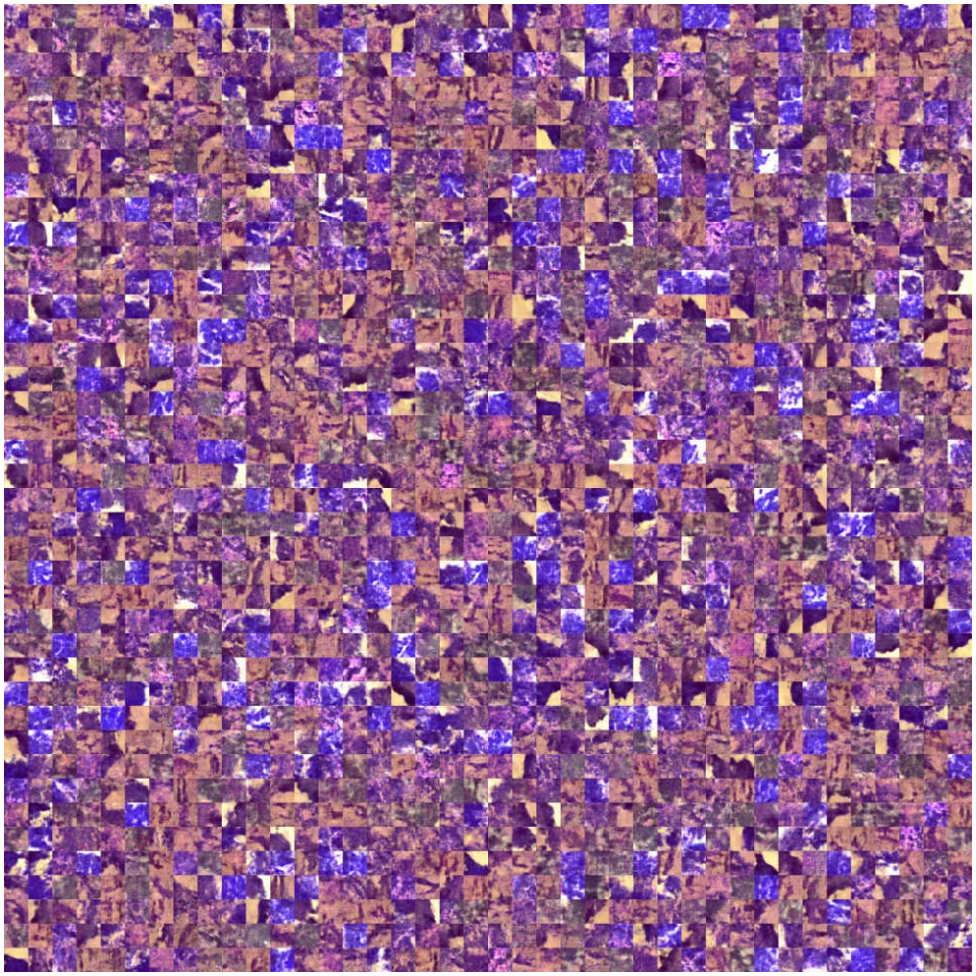

Рисунок 3.9 – Приклад згенерованих зображень

3.6 Тестування згенерованих зображень

Після навчання GAN мережі було згенеровано приблизно 1600 гістологічних (приблизно 320 для кожного класу) зображень. Час генерування та збереження на диск становив приблизно 27хв.

Після чого оригінальні зображення було подано в якості навчальної вибірки для класифікатора. Навчання класифікатора на оригінальній вибірці дало показник точності приблизно 84% для гістологічних зображень. Після навчання класифікатора на вибірці із згенерованих зображень вихідна точність склала приблизно 96,5%. Отже приріст точності становить 12,5%.

На рисунку нижче наведено ROC-криву для класифікатора (гістологія, оригінальна вибірка).

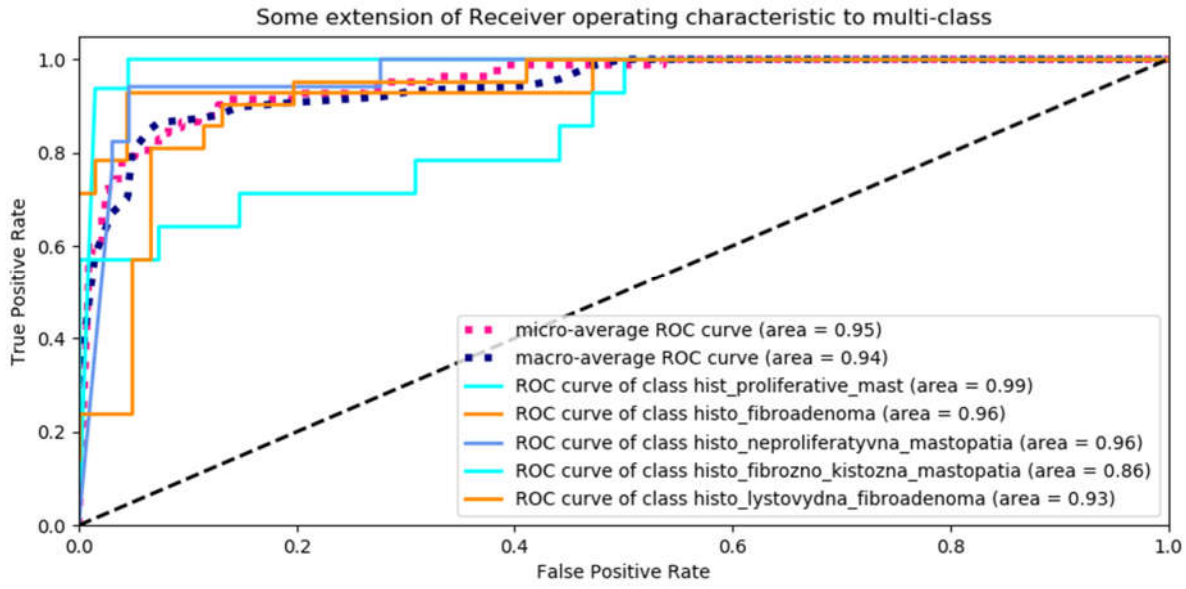

Рисунок 3.10 – ROC-крива класифікатора (оригінальна вибірка)

На малюнку нижче наведено ROC-криву для класифікатора (гістологія, згенеровані зображення).

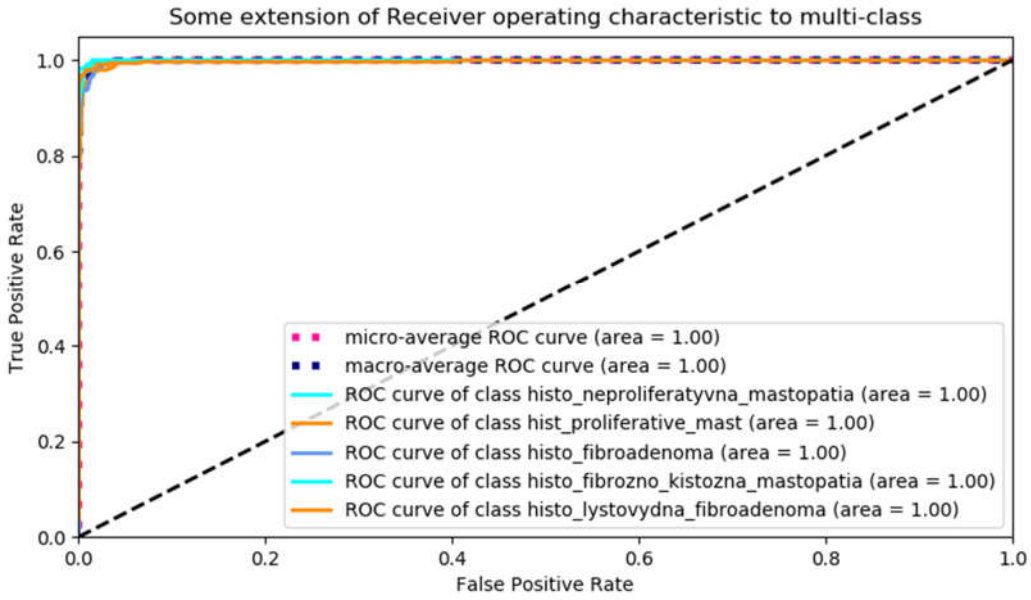

Рисунок 3.11 – ROC-крива класифікатора (згенеровані зображення)

Як видно з рисунків, згенеровані зображення подібні до оригінальної вибірки. А класифікатор дає правильну мітку зображенням із кожного класу.

3.7 Бібліотека для мови програмування Java

Для використання розробленого програмного забезпечення у системі автоматизованої мікроскопії, яка написана на мові Java, потрібно створити бібліотеку з програмним кодом для цієї мови. Використання розробленого програмного забазпечення передбачається наступним чином:

– користувач має в наявності збережену та натреновану модель;

– за допомогою бібліотеки він завантажує дану модель та може генерувати зображення у вказаній кількості.

Оскільки завадання реалізовано на мові програмування Python із використанням бібліотеки Tensorflow, яка також реалізована і для мови програмування Java, створити бібліотеку JAR стає не таким складним

завданням. Потрібно лише мати збережену модель та її сигнатуру (імена входів та виходів). На рисунку 3.12 наведено приклад використання розробленої бібліотеки.

```
import org.liashchynskyi.gan.Generator;
import org.liashchynskyi.gan.GeneratorBuilder;

...

    public static String[] labels = new String[]{"label1", "label2", "label3", "label4", "label5"}; //
    public static final String MODEL_PATH = "path/to/model/dir";

    Generator generator = new GeneratorBuilder().labels(labels).modelPath(MODEL_PATH).num(5).build();
    generator.loadAndPredict("output/folder");

...
```

Рисунок 3.12 – Приклад використання JAR-бібліотеки

Як видно з рисунка, користувачу потрібно вказати шлях до моделі та імена міток класів (для гістології це 5 класів). Після чого, користувач вказує директорію, де мають бути збережені згенеровані зображення та вказує їх кількість (кратну кількості класів).

ВИСНОВКИ

1. Здійснено аналіз біомедичних зображень, що показало актуальність проблеми малих вибірок і дозволило поставити задачі дослідження.

2. Проведено аналіз методів і алгоритмів синтезу зображень, що дало можливість виділити перспективний напрямок синтезу на основі генеративно-змагальних нейронних мереж.

3. Розроблено алгоритм синтезу зображень на основі генеративно-змагальних нейронних мереж, що дало змогу генерувати навчальні вибірки в потрібній кількості.

4. Розроблено модель нейронної мережі, що дало змогу генерувати гістологічні зображення в майбутньому.

5. Розроблено бібліотеку програмного модуля для мови програмування Java, що дало змогу використовувати його у системі автоматизованої мікроскопії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Березький О.М. Методичні рекомендації до виконання магістерської роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма - Комп’ютерна інженерія" / О.М. Березький, Л.О. Дубчак, Г.М. Мельник /Під ред. О.М. Березького – Тернопіль: ТНЕУ, 2018.– 41 с.

2. Гураль І.В. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. - Тернопіль: ТНЕУ, 2019. – 33 с.

3. Лящинський П. Б., Сухович А.Р. Синтез біомедичних зображень на основі згорткових нейронних мереж: зб. матеріалів доп. учасн. наук.-практ. конф. Інтелектуальні комп'ютерні системи та мережі. Тернопіль: ТНЕУ, 2019. С. 55.

4. Liashchynskyi P. Rudi: Lightweight image converter and dataset augmentor. Zenodo. 2019.

5. Березький О. М., Піцун О.Й., Лящинський П.Б., Лящинський П.Б., Мельник Г.М. Інтелектуальна система автоматизованої мікроскопії аналізу гістологічних та цитологічних зображень. Штучний інтелект, Київ. 2017. Вип. 2 (76). С. 128-140.

6. Березький О.М., Піцун О.Й., Боднар А.Р., Долинюк Т.М. Класифікація гістологічних та цитологічних зображень на основі згорткових нейронних мереж. Штучний інтелект, Київ. 2017. Вип. 1 (75). С. 33-42.

7. Свідоцтво про реєстрацію авторського права на твір №75360. Комп’ютерна програма «Інтелектуальна система діагностування передракових станів молочної залози на основі аналізу гістологічних та цитологічних зображень "HIAMS"». / О.М. Березький, О.Й. Піцун, Г.М. Мельник, П.Б. Лящинський, П.Б. Лящинський. Дата реєстрації 14.12.2017 р.

8. Berezsky O., Pitsun O., Dubchak L., Liashchynskyi P., Liashchynskyi P. GPU-based biomedical image processing. 14th International Conference on Perspective Technologies and Methods in MEMS Design, MEMSTECH 2018. Proceedings. 96-99 pp.

9. Березький О.М., Лящинський П.Б., Лящинський П.Б, Сухович А.Р., Долинюк Т.М. Синтез біомедичних зображень на основі генеративно-змагальних мереж. УЖІТ 2019. Вип. 1. (прийнято до публікації).

10. Березький О. М., Батько Ю.М., Березька К.М., Вербовий С.О., Дацко Т.В., Дубчак Л.О., Ігнатєв І.В., Мельник Г.М., Николюк В.Д., Піцун О.Й. Методи, алгоритми і програмні засоби опрацювання біомедичних зображень. Тернопіль: Економічна думка, ТНЕУ. 2017. 330 с.

11. Березький О.М., Піцун О.Й. Засоби класифікації біомедичних зображень на основі нейронних мереж. Науковий вісник НЛТУ України, 2018. т. 28, Вип. 9.

12. Гістологія людини / О.Д. Луцик та ін. Київ, «Книга Плюс», 2003. С.593.

13. Методи, алгоритми та програмні засоби опрацювання біомедичних зображень: монографія / О. М. Березький та ін; під наук. ред. Березький О. М. – Тернопіль: ТНЕУ, 2017. 330 с.

14. Кількісна гістологія: веб-сайт. URL: https://uk.wikipedia.org /wiki/Кількісна_гістологія(дата звернення: 12.10.2019).

15. Кількісна гістологія: веб-сайт. URL: http://quantitative.histology.org.ua (дата звернення: 12.10.2019).

16. Bourzac K. Software: The computer will see you now. Nature, 2013. Vol. 502, No.7473. P. 92-94.

17. Гістологічні методи дослідження: веб-сайт. URL: http://ukrbukva.net/ page,3,94380-Gistologicheskie-metody-issledovaniya.html(дата звернення: 13.09.2019).

18. Предмет гістології: веб-сайт. URL: http://studfiles.net /preview/5258227(дата звернення: 10.10.2019).

19. Методы гистологического анали за: веб-сайт. URL: http://worldofscience.ru/biologija/6481-metody-gistologicheskogo-analiza.html(дата звернення: 13.09.2019).

20. Автандилов Г.Г. Основы количественной паталлогической анатомии. Медицина, 2002. 240 с.

21. Liu Y., Liu Y., Lin W., Hays J. Near-regular Texture Analysis and Manipulation. ACM Transactions on Graphics (SIGGRAPH 2004), 2004. P. 368−376.

22. Романюк О.Н., Курінний М.С. Методи та засоби антиаліайзингу контурів об’єктів у системах комп’ютерної графіки: моногр. Вінниця: УНІВЕРСУМ-Вінниця, 2006. 163 с.

23. Efros A. A., Efros A., Freeman W. Image Quilting for Texture Synthesis and Transfer. Proceedings of SIGGRAPH 2001, 2001. P. 341–346.

24. Yingqing X., Song-Chun Z., Baining G., Heung-Yeung S. Asymptotically Admissible Texture Synthesis. In International Workshop on Statistical and Computational Theories of Vision, 2001. P. 22.

25. Efros A. Texture Synthesis by Non-Parametric Sampling. IEEE International Conference on Computer Vision, 1999. No.2. P. 1033.

26. Wei L.Y., Levoy M. Fast texture synthesis using tree-structured vector quantization . Proceedings of the 27th annual conference on Computer graphics and interactive techniques 2000, 2000. P. 479–488.

27. Nealen A., Nealen A., Alexa M. Hybrid Texture Synthesis, 2003. P. 68.

28. Musgrave F. K. The Synthesis and Rendering of Eroded Fractal. Computer graphics and interactive techniques, 1989. P. 41−50.

29. Perlin K., Perlin K., Ebert D.S., Musgrave F.K., Peachey D., Worley S. Texturing and Modeling: A Procedural Approach. 3rd edition. San Francisco,USA: Morgan Kaufmann, 2003. 712 p.

30. Taponecco F. Vector field visualization using Markov Random Field texture synthesis. VISSYM 03: Proceedings of the symposium on Data visualisation 2003, 2003. P. 195–202.

31. Neyret F. Realistic Rendering of an Organ Surface in Real-Time for Laparoscopic Surgery Simulation. Visual Computing. 2002. Vol. 3, No.18. P. 135−149.

32. Flach B., Schlesinger D. Modelling composite shapes by Gibbs random fields. The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 2011. P. 2177-2182.

33. Что такое свёрточная нейронная сеть: веб-сайт. URL: https://habrahabr.ru/post/309508/(дата звернення: 01.09.2019).

34. Zlateski A., Lee K., Seung H.S. ZNN – A Fast and Scalable Algorithm for Training 3D Convolutional Networks on Multi-core and Many-Core Shared Memory Machines. 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016. P. 801-811.

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

36. Qian S., Liu H., Liu C., Wu S., San Wong H. Adaptive activation functions in convolutional neural networks. Neurocomputing. 2018. Vol. 272. P. 204-212.

37. CS231n Convolutional Neural Networks for Visual Recognition: веб-сайт. URL: http://cs231n.github.io/convolutional-networks/(дата звернення: 01.09.2019).

38. Наглядно объясняем операцию свертки в моделях глубокого обучения: веб-сайт. URL: https://proglib.io/p/convolution/(дата звернення: 02.09.2019).

39. Wu H., Gu X. Max-Pooling Dropout for Regularization of Convolutional Neural Networks. Neural Information Processing. 2015. No.9489.

40. A guide to convolution arithmetic for deep learning: веб-сайт. URL: https://arxiv.org/pdf/1603.07285.pdf(дата звернення: 11.10.2019).

41. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998.

42. ImageNet Classification with Deep Convolutional Neural Networks: веб-сайт. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf(дата звернення: 11.10.2019).

43. Visualizing and Understanding Convolutional Networks: веб-сайт. URL: https://arxiv.org/pdf/1311.2901.pdf(дата звернення: 11.10.2019).

44. Going Deeper with Convolutions: веб-сайт. URL: https://arxiv.org/pdf/1409.4842.pdf(дата звернення: 12.10.2019).

45. Very Deep Convolutional Networks for Large-Scale Image Recognition: веб-сайт. URL: https://arxiv.org/pdf/1409.1556.pdf(дата звернення: 12.10.2019).

46. Deep Residual Learning for Image Recognition: веб-сайт. URL: https://arxiv.org/pdf/1512.03385.pdf(дата звернення: 12.10.2019).

47. Кроновер Р.М. Фракталы и хаос в динамических системахю Основы теории. Москва:Постмаркет, 2000. 352 с.

48. Афінне перетворення: веб-сайт. URL: https://uk.wikipedia.org/wiki/Афінне_перетворення(дата звернення: 14.10.2019).

49. Goodfellow J., Pouget-Abadie M., Mirza B., Xu D., Warde-Farley S., Ozair A., Courville Y., Bengio Y. Generative adversarial nets. Advances in neural information processing systems, 2014. P. 2672–2680.

50. Baldi P. Autoencoders, Unsupervised Learning, and Deep. Workshop on Unsupervised and Transfer Learning, 2012.

51. Recent Advances in Autoencoder-Based Representation Learning: веб-сайт. URL: https://arxiv.org/pdf/1812.05069(дата звернення: 12.10.2019).

52. GAN — What is Generative Adversary Networks GAN?: веб-сайт. URL: https://medium.com/@jonathan_hui/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09(дата звернення: 12.10.2019).

53. What is wrong with the GAN cost function?: веб-сайт. URL: https://medium.com/@jonathan_hui/gan-what-is-wrong-with-the-gan-cost-function-6f594162ce01(дата звернення: 14.10.2019).

54. From GAN to WGAN: веб-сайт. URL: https://arxiv.org/pdf/1904.08994.pdf(дата звернення: 14.10.2019).

55. GANs for Medical Image Analysis: веб-сайт. URL: https://arxiv.org/pdf/1809.06222.pdf(дата зверненняЖ 12.10.2019).

56. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks: веб-сайт. URL: https://arxiv.org/pdf/1511.06434.pdf(дата звернення: 15.10.2019).

57. Conditional Generative Adversarial Nets: веб-сайт. URL: https://arxiv.org/pdf/1411.1784.pdf(дата звернення: 14.10.2019).

58. High-Resolution Deep Convolutional Generative Adversarial Networks: веб-сайт. URL: https://arxiv.org/pdf/1711.06491.pdf(дата звернення: 15.10.2019).

59. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks: веб-сайт. URL: https://arxiv.org/pdf/1506.05751.pdf(дата звернення: 16.10.2019).

60. Arjovsky M. Wasserstein GAN: веб-сайт. URL: https://arxiv.org/pdf/1701.07875.pdf(дата звернення: 16.10.2019).

61. Generative Adversarial Networks: An Overview: веб-сайт. URL: https://arxiv.org/abs/1710.07035(дата звернення: 12.10.2019).

62. Salimans T., Goodfellow I., Zaremba W., Cheung V. Improved techniques for training gans. Advances in Neural Information Processing Systems, 2016. P. 2226–2234.

63. Arjovsky M., Bottou L. Towards principled methods for training generative adversarial networks. NIPS 2016 Workshop on Adversarial Training, 2016.

64. Автоэнкодеры в Keras: веб-сайт. URL: https://habr.com/ru/post/332000/(дата звернення: 14.10.2019).

65. Read-through: WGAN: веб-сайт. URL: https://www.alexirpan.com/2017/02/22/wasserstein-gan.html(дата звернення: 14.10.2019).

66. Improved Training of Wasserstein GANs: веб-сайт. URL: https://arxiv.org/abs/1704.00028(дата звернення: 14.10.2019).