НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено:
В.о. завідувач кафедри
_____ Оксана ТИМОЩУК
(підпис)
“___”_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи та методи штучного інтелекту»
спеціальності 122 «Комп’ютерні науки та інформаційні технології»
на тему: «Система розпізнавання раку шкіри з використанням нейронних мереж»

Виконала: студентка 4 курсу, групи КА-66 _____
Житанська Дар’я Сергіївна

Керівник: к.ф.-м.н., доцент Шубенкова Ірина Анатоліївна _____

Консультант з економічних питань: д.е.н., доцент, _____
Шевчук Олена Анатоліївна

Консультант з нормоконтролю: к.т.н., доцент,
Коваленко Анатолій Єпіфанович _____

Рецензент: к.т.н., доцент Гіоргізова-Гай Вікторія Шалвівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
«Інститут прикладного системного аналізу»
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Оксана ТИМОЩУК
«25» травня 2020 р.

ЗАВДАННЯ
на дипломну роботу студенту
Житанській Дар'ї Сергіївні

1. Тема роботи «Система розпізнавання раку шкіри за допомогою нейронних мереж», керівник роботи Шубенкова Ірина Анатоліївна, к.ф.-м.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1143-с

2. Термін подання студентом роботи 8.06.2020

3. Вихідні дані до роботи: визначення підвиду хвороби раку шкіри та поставлення діагнозу з конкретною точністю.

4. Зміст роботи: повний аналіз зображень у тренувальній вибірці та визначення певної закономірності у кожному з класів хвороби; спроба навчитись виявляти підвид хвороби за допомогою фото ураженої ділянки шкіри, прогнозування наявності раку шкіри.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Ключові поняття в теорії нейронних мереж, навчання нейронних мереж: види, способи, методи. Ключові архітектури нейронних мереж та сучасні моделі.

Порівняння моделей, дослідження результатів навчання, створення власної моделі та порівняння кінцевих результатів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	13.04.2020	
2	Підготовка першого розділу	22.04.2020	
3	Підготовка другого розділу	1.05.2020	
4	Розробка програмного продукту	15.05.2020	
5	Підготовка третього розділу	18.05.2020	
6	Підготовка економічної частини 30.05.2020	30.05.2020	
7	Оформлення розділів відповідно до нормоконтролю	2.06.2020	
8	Оформлення дипломної роботи	4.06.2020	
9	Підготовка презентації доповіді	7.06.2020	

Студент Дар’я ЖИТАНСЬКА

Керівник роботи Ірина ШУБЕНКОВА

РЕФЕРАТ

Дипломна робота: 118 с., 40 рис., 10 табл., 3 додатки, 26 джерел.

НЕЙРОННІ МЕРЕЖІ, МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ, ПРОГНОЗ, КЛАСИФІКАЦІЯ, ДІГНОСТИКА РАКУ ШКІРИ.

Об’єктом дослідження є можливість розпізнавання та передбачення хвороби раку шкіри.

Предметом дослідження є аналіз методів та моделей оптимальних варіантів рішення заданої задачі.

Мета дослідження:

1) розробка програмного продукту призначеного для автоматизації розрахунків та полегшення аналізу;

2) дослідити існуючі методи для розпізнавання та класифікації конкретних захворювань та застосувати їх до нашої задачі .

Теоретичною та методологічною основою дослідження є праці вітчизняних і зарубіжних вчених в галузі штучного інтелекту та математичних апаратів.

Результатом дипломної роботи є створення програмного продукту для передбачення хвороби та визначення діагнозу. Програмний продукт реалізовано за допомогою мови програмування Python.

ABSTRACT

Diploma Thesis (Bachelor's Thesis): 118 p., 40 fig., 10 tabl., 3 annexes, 26 sources.

NEURAL NETWORKS, METHODS OF ARTIFICIAL INTELLIGENCE, FORECAST, CLASSIFICATION, DIAGNOSIS OF SKIN CANCER.

The object of research is the possibility of recognizing and predicting skin cancer.

The subject of research is the analysis of methods and models of optimal solutions to a given problem.

The aim of the study:

1) development of a software product designed to automate calculations and facilitate analysis;

2) to explore existing methods for the recognition and classification of specific diseases and apply them to our task.

The theoretical and methodological basis of the study are the works of domestic and foreign scientists in the field of artificial intelligence and mathematical apparatus.

The result of the thesis is the creation of a software product for disease prediction and diagnosis. The software product is implemented using the Python programming language.

Зміст

ВСТУП

Нейронні мережі – одна з найзручніших, коли-небудь створених, парадигм програмування. Зазвичай, при будь-якому іншому підході ми розповідаємо комп'ютеру, що робити, розбиваючи великі проблеми на безліч дрібних, які система може легко виконати. Нейронна мережа ж самостійно навчається на основі даних із раніше проведених спостережень та знаходить власне вирішення проблеми. Вона є оцифрованою моделлю біологічного мозку і може виявляти складні нелінійні зв’язки між залежними, а також незалежними змінними в даних, коли людський мозок може не виявити.

У 2005-2006 роках групи дослідників під керівництвом Джеффрі Хінтона (Geoffrey Hinton) в університеті Торонто і Йошуа Бенджі (Yoshua Bengio) в університеті Монреаля навчилися тренувати глибокі нейронні мережі. І це перевернуло весь світ машинного навчання! Тепер в найрізноманітніших предметних областях кращі результати виходять за допомогою глибоких нейронних мереж. Одним з перших гідних успіхів стало розпізнавання мови: розроблені групою Хінтона мережі дуже швидко поліпшили результати розпізнавання мови класичними підходами. Зараз сучасні голосові помічники на зразок Apple Siri і Google Now, працюють виключно на глибоких нейронних мережах. Люди змогли удосконалити та навчити нейронні мережі до такого рівня, що тепер вони можуть вирішувати абсолютно різні завдання: від розпізнавання осіб до водіння автомобілів і гри в го.

Одне з найефективніших рішень за історію створення нейронних мереж було запропоновано групою Хінтона в середині 2000-х років. Воно полягало у навчанні без вчителя, коли мережа спочатку тренується на великому наборі даних без розмітки, а потім вже тестується на розмічених даних, використовуючи наближення. Виявилося, що при цьому кінцевий результат стає набагато краще, а хороші і цікаві ознаки мережа починає виділяти ще на етапі навчання без вчителя.

Розвивалися і загальні методи навчання нейронних мереж, і класичні архітектури нейронних мереж, згорткові мережі та рекурентні мережі. Але з'являлися і абсолютно нові архітектури: нейронні мережі в навчанні з підкріпленням привели до небачених раніше проривів, нейробайесівські методи з'єднали нейронні мережі і класичний імовірнісний висновок за допомогою варіаційних наближень, а потреби конкретних програм привели до розробки таких нових архітектур, як мережі з увагою і мережі з пам'яттю, які вже знаходять і інші застосування [7, с.8-9].

В даний час нейронні мережі широко використовуються в різних галузях медицини, особливо в кардіології. Також вони часто застосовуються в діагностиці, електронному аналізі сигналів, рентгенології та медичному аналізі зображення. Останнє дуже сильно допомагає у виявленні шкірних захворювань, таких як дерматит, екзема, псоріаз та навіть рак. Дане дослідження ґрунтується на аналізі зображень шкірних ділянок та розпізнаванні меланоми, за допомогою нейронних мереж.

Рак шкіри є однією з найбільш поширених форм раку у світі. Точне розпізнавання цього захворювання на ранній стадії може значно підвищити відсоток виживання пацієнтів. Однак ручне виявлення раку шкіри має чималу потребу у висококваліфікованих фахівцях. Крім того, за рахунок великої кількості типів раку шкіри та широкого діапазону його ознак, не завжди поставлений діагноз є правильним. Саме тому варто розробити надійну автоматичну систему розпізнавання, яка підвищує точність виявлення та класифікації патологів.

Одним із сучасних та дієвих методів виявлення раку є дерматоскопія. Це неінвазійна методика візуалізації шкіри, що дозволяє отримати збільшене і освітлене зображення конкретної області, для підвищення чіткості плям. Тим самим такий метод посилює візуальний ефект ураження шкіри шляхом усунення поверхневого відображення. Проте, автоматичне розпізнавання раку шкіри з дерматоскопії зображень, як і раніше, є складним завданням та має кілька проблем:

- низький контраст між місцем ураження шкіри і її здоровими областями ускладнює сегментну точність областей поразки;
- меланомні і немеланомні ураження можуть мати високу ступінь візуальної схожості, що призводить до утруднення виявлення різниці між цими областями шкіри;
- унікальність стану шкіри, призводить до збільшення варіантів того, як можуть виглядати меланомні ураження, наприклад, колір шкіри, її природний волосяний покров або вени впливають на вид, колір та структуру меланоми.

Тому, наступним та не менш важливим кроком дослідження хвороби є сегментація. Це допомагає більш точно класифікувати зображення та визначити діапазон конкретних властивостей(колір, структура). Широкі дослідження для отримання гідних результатів сегментації поразки проводяться до сьогоднішнього дня. Наприклад, David Delgado Gómez і співавт. запропонував алгоритм, названий Independent Histogram Pursuit (IHP) [3]. Алгоритм був протестований на п'яти різних дерматологічних наборах даних і досяг конкурентоспроможної точності, близькою до 97%. Rahil Garnavi і співавт. запропонував автоматизований підхід для сегментації ділянки поразки, з використанням оптимальних колірних каналів і гібридної технології визначення порогу. Le Yu використовував метод глибокого навчання, тобто повністю згорткову залишкову мережу (FCRN), та досяг високого результату на наборі даних ISIC 2016 [4].

Крім того, саме етап сегментації дає змогу визначити певні ознаки вручну. Наприклад, M. Emre Celebi та співавт. після безпосередньої сегментації ураженої ділянки, зміг виділити деякі додаткові властивості самостійно, включаючи колір, текстуру і форму. Guido Schäfer використовував метод автоматичного визначення кордону області пошкодження, а потім збирав витягнуті елементи для розпізнавання. Також в деяких дослідження була зроблена спроба використовувати визначені вручну ознаки, для розпізнавання меланоми без етапу сегментації [5-6].

Незабаром, через прориви, досягнуті глибоким навчанням та збільшенням кількості завдань з обробки медичних зображень, у деяких дослідженнях розпізнавання раку шкіри почали застосовувати метод глибокого навчання. Зазвичай, такий підхід використовує ієрархічну структуру для автоматичного вилучення ознак. Noel Codella та ін. запропонував гібридний підхід, що об'єднує згорткову нейронну мережу (CNN), розріджене кодування та support vector machine (SVM) для виявлення патологів. У недавньому його дослідженні було створено систему, що поєднує останні розробки в підходах до глибокого навчання, машинного навчання для сегментації та класифікації ураження шкіри.

Незважаючи на те, що було запропоновано доволі багато вдалих робіт для виявлення раку шкіри, багато науковців намагаються домогтися покращення точності результатів як для сегментації, так і для класифікації ураження шкіри. Але не тільки науковці можуть брати участь у вирішенні даної задачі.

Наприклад, міжнародне співробітництво в області візуалізації шкіри (ISIC)- це постійний конкурс для розвитку інструментів аналізу автоматичної сегментації та діагностики уражень шкіри, з метою точного виявлення меланоми по дерматоскопічним зображенням. Завдання розбите на такі три етапи:

1. Сегментація поразки,
2. Визначення ознак ураження
3. Класифікація захворювання

Підхід, прийнятий в цій роботі, полягає в використанні традиційних методів класифікатора зі створеними вручну ознаками. Для сегментації поразки використовують метод, що обробляє зображення в просторі RGB (Red Green Blue). Підхід починається з розробки класифікатора кольорів, щоб відрізнити уражену тканину від нормальної шкірної тканини на основі виключно RGB кольорових векторів. Метод класифікації діагностики ураження в даному дослідженні виконується за допомогою класифікатора SVM з 200 рукотворними функціями. Характеристики обчислюються із зображення RGB разом із маскою сегментації

ураження, отриманою за допомогою методу сегментації. Усі ознаки обчислюються для кожного з трьох кольорових каналів і з'єднуються, а потім подаються в класифікатор SVM.

Отже, одним із найвдаліших методів рішення задачі по виявленню рака шкіри – є глибоке навчання. Тому, опираючись на вже існуючі дослідження, можемо розбити нашу подальшу роботу на такі етапи:

1. Знаходження даних, які вже мають коректний діагноз, для тренування та знаходження кореляції між даними.
2. Підготовка зображень(даних), які були зроблені за допомогою дерматоскопії. Цей крок вимагає мінімальної обробки, так як зображення робиться, зазвичай, спеціальним апаратом та дає гідні результати для подальшої роботи.
3. Сегментація. На цьому кроці ми повинні визначити межі ураженої ділянки. Тут ми працюємо із ознаками раку шкіри, його властивостями та намагаємось витягти як найбільше інформації з кожного зображення. Потрібно домогтися автоматичного виявлення усіх ознак.
4. Класифікація. Цей етап вимагає великої точності усіх попередніх. Так як на основі дослідження ознак уражених ділянок та діагнозів, ми повинні зробити певні висновки та знайти взаємозв’язок між ними. Це допоможе ставити діагнози у подальшій роботі над задачею. [2, с.1-3]

Робота складається з п’яти розділів.

У першому розділі відбувається ознайомлення із нейронними мережами, їх головними структурами та основними методами реалізації.

Другий розділ присвячений детальному огляду способів навчання нейронних мереж (корегування ваг) та їх удосконалених аналогів. Крім того, у розділі розглядаються деякі з основних архітектур, що використовуються у подальшій роботі із мережами

В третьому розділі детально розглядаються сучасні нейронні моделі та відбувається їх порівняння. Це робиться для визначення оптимальної моделі, яка зможе дати точний результат, за доволі не великий проміжок часу та при малих витратах ресурсів.

Четвертий розділ присвячений опису розробленого програмного продукту. Розглянуто його функціонал. Приведена детальна інструкція користувача.

В п’ятому розділі проводиться економічно-вартісний аналіз розробленого продукту.

1 ТЕОРІЯ НЕЙРОННИХ МЕРЕЖ: ОСОБЛИВОСТІ НАВЧАННЯ, ГОЛОВНІ СТРУКТУРНІ ЕЛЕМЕНТИ

1.1 Біологічні нейронні мережі

Штучні нейронні мережі є представленням дуже спрощених моделей функцій мозку. Тому спочатку розглянемо аналогії між штучними нейронними мережами Бейлера-Джонса, Гупта, Сінга та біологічними. Звичайно, останні є значно складнішими, і більшість моделей нейронних мереж на них зовсім схожі.

Мозок людини складається з мільярдів взаємопов'язаних нейронів. Це клітини, які мають спеціалізовані мембрани, що дозволяють передавати сигнали сусіднім нейронам. На рисунку 1.1 показана структура нейрона. Один аксон простягається від основного тіла клітини. Це вихід з нейрона і, як правило, він ділиться на багато гілок до закінчення синапсу. Електричні імпульси передаються по аксону до синапсів передачею іонів Na+. Імпульс, що виникає внаслідок будь-якої напруги мозку, стимулює вивільнення нейромедіаційних хімікатів через синаптичну щілину до постсинаптичної клітини, яка є приймаючою частиною наступного нейрона. Ця постсинаптична клітина передає сигнал через дендрит до основної частини тіла нейрона. Потім входи різних дендритів поєднуються для отримання вихідного сигналу, який передається по аксону, і таким чином триває процес передачі інформації. Інформація передається в нейрон від інших нейронів через дендрити зліва. Давши відповідні вхідні подразники, клітина посилає вихідний сигнал по аксону до синапсів, де сигнал передається іншим нейронам.

Однак сигнал в аксоні є вихідним лише в тому випадку, якщо є достатньо входів певної сили для подолання деякого порогового значення. Тобто, можна зробити висновок, що вихід є деякою нелінійною функцією вхідних подразників.

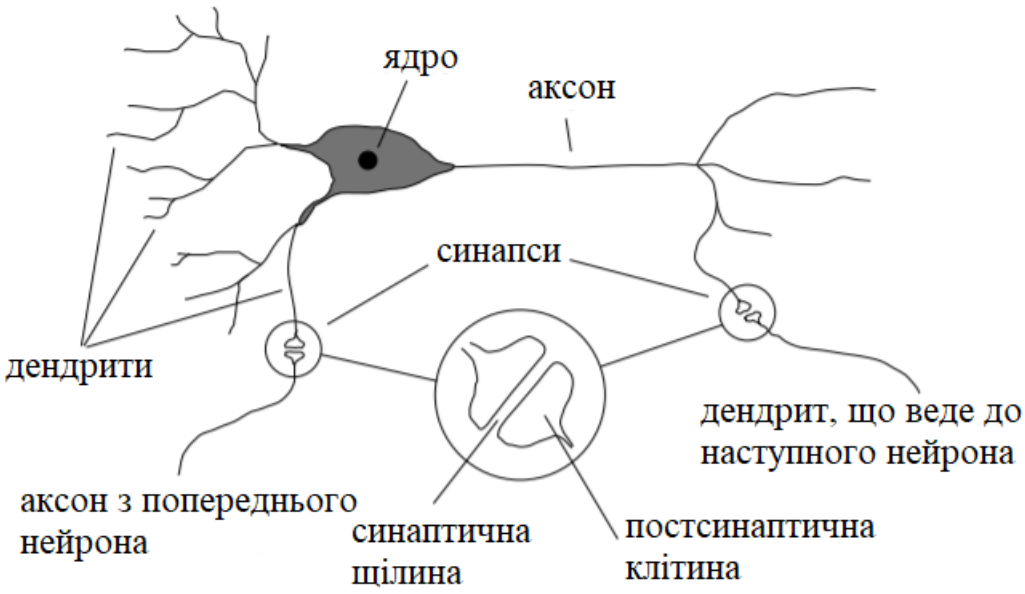


Рисунок 1.1 – Будова єдиного нейрона в мозку

Мозок людини складається з приблизно 10^{11} нейронів, і кожен нейрон має від декількох до кількох тисяч синапсів на своїх дендритах, що дає в цілому близько 10^{14} синапсів (з'єднань) в мозку. «Сила» синаптичного зв’язку між нейронами може хімічно змінити мозок у відповідь на сприятливі та несприятливі подразники таким чином, щоб адаптувати організм до оптимального функціонування в його середовищі.

Таким чином, вважається, що синапси є ключем до вивчення біологічних систем [8, с. 1-2].

1.2 Штучні нейронні мережі

Штучні нейронні мережі - це достатньо широкий термін, що описує архітектуру математичних моделей, які складаються з вузлів обробки (аналогічних нейронам) та з'єднаннями між ними (аналогічними дендритам та аксонам).

Ці з'єднання, як правило, мають параметри, за допомогою яких можуть змінювати сигнали, що проходять уздовж них. Існують численні типи штучних нейронних мереж для вирішення багатьох різних типів проблем: моделювання пам'яті, виконання розпізнавання шаблонів та прогнозування еволюції динамічних систем. Більшість мереж здійснюють моделювання даних, і зазвичай розділені на два широкі класи: навчання з учителем і без. Перший відноситься до мереж, які намагаються дізнатися співвідношення між даними та областю заданого параметра. Тоді як другий тип стосується мереж, що використовуються для пошуку "природних" угруповань із набором даних незалежно від зовнішніх обмежень.

Але спільним між ними є ідея дізнатися про проблему за допомогою взаємозв'язків, що присутні у даних, а не через набір заздалегідь визначених правил. Вступ до декількох типів нейронних мереж (але далеко не всіх) дають Герц, Крог та Палмер (1991). Менш математичний огляд надає Білл і Джексон (1990).

Також потрібно підкреслити, що одним із найважливіших типів керованих нейронних мереж є прямий багатошаровий перцептрон. Термін перцептрон є історичним і відноситься до функції, яку виконують вузли. Прямий зв'язок у такій моделі означає, що існує певне введення, виведення даних і рухаються вони в одному напрямку. Це можна порівняти з рекурентними нейронними мережами, в яких дані передаються в циклі: така мережа є оптимальною, коли час грає важливу роль в рішенні проблеми.

На рисунку 1.2 показана модель нейронної мережі з декількома входами, двома прихованими шарами та кількома виходами. Цей приклад містить не явно показані вузли зміщення: кожен шар має додатковий вузол, який утримує постійне значення (і не має входів) і надає зміщення наступним шарам. Вони необхідні для належного моделювання функцій мережі.

Кожен вузол вхідного шару містить значення, x_i. Кожен з вхідних вузлів з'єднується з кожним вузлом y наступному шарі вузлів, першим «прихованим» шаром, і кожен з цих з'єднань має вагу $w_{i,j}$, пов’язану з ним.

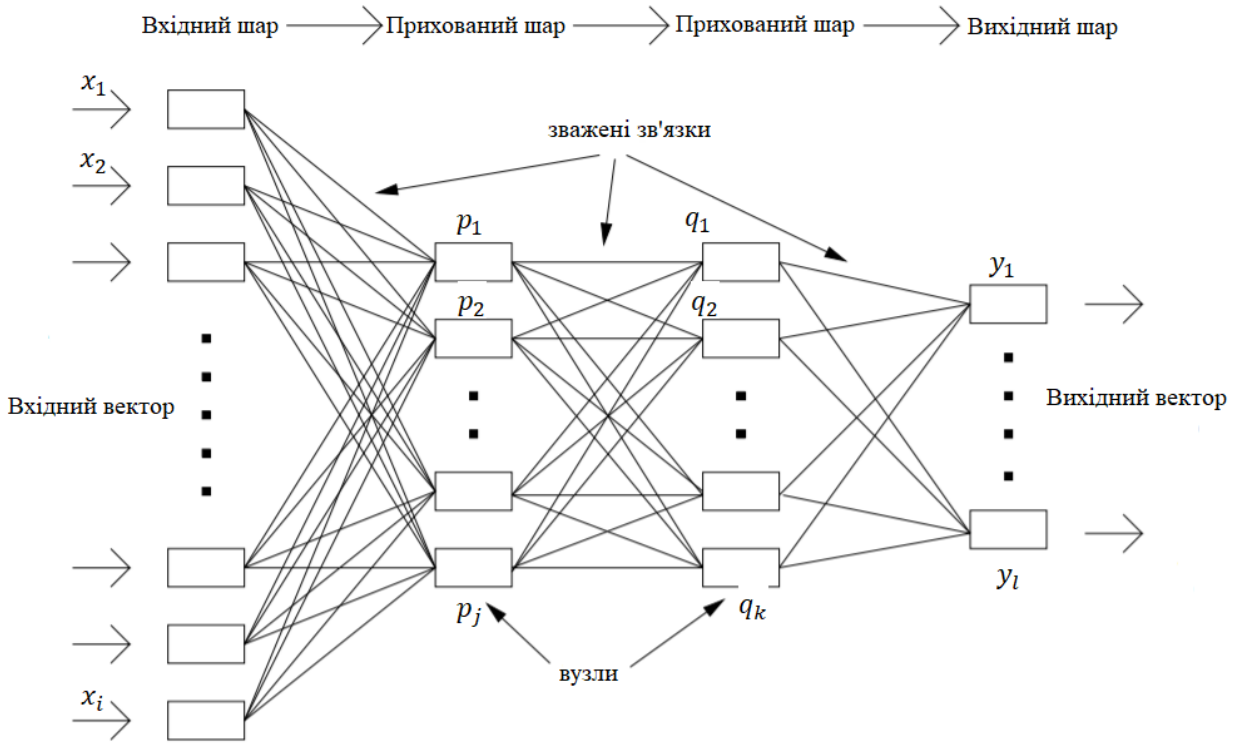


Рисунок 1.2 – Архітектура штучної прямої нейронної мережі

Вузол у прихованому шарі утворює зважену суму своїх входів і передає її через нелінійну функцію активації, так що вихід з j-го прихованого вузла дорівнює:

$$p_j = \tanh\left(\sum_i w_{i,j} x_i\right), \tag{1.1}$$

Ці значення передаються другому прихованому шару, який виконує аналогічну обробку, вихід з цього шару - вектор ***q:***

$$q_k = tanh\left(\sum_j w_{j,k} p_j\right), \tag{1.2}$$

Потім вихідний шар виконує просту суму шару, що входить в нього , так що вихід мережі - вектор $\boldsymbol{y}$:

$$y_l = \sum_k w_{k,l} q_k, \tag{1.3}$$

Функція tanh у прихованих шарах забезпечує нелінійну здатність мережі. Можливі й інші нелінійні функції. Наприклад, часто використовується сигмоїдна функція $\frac{1}{1+e^{-\sum wx}}$. Обидві функції відображають нескінченний можливий діапазон вводу в кінцевий діапазон виходу, у випадку tanh від 1 до +1. Це є імітацією функції передачі нейронів [8, с 2-3].

1.3 Навчання штучної нейронної мережі

Ваги $\boldsymbol{w}$, що знаходяться в рівняннях (1.1) – (1.3), є вільними параметрами мережі.

Зрозуміло, для того, щоб мережа дала відповідні виходи для заданих входів, ваги повинні бути встановлені на конкретні значення. Це робиться шляхом навчання мережі на наборі вхідних векторів, для яких уже відомі ідеальні виходи (або цілі). Це навчання з учителем. Ідея полягає в тому, що мережа інкапсулює взаємозв'язок. У нашому прикладі, це взаємозв’язок між зображенням раку та його вже відомим діагнозом. Мережа не розробляє нову систему класифікації, а скоріше інтерполює навчальні дані, щоб дати узагальнення взаємозв'язку між хворобою та її класифікацією . Потім можна використовувати цю навчену мережу для отримання надійних класифікацій для некласифікованих спектрів. Навчання з учителем триває шляхом мінімізації функції помилок стосовно всіх ваг мережі.

Зазвичай, функція помилок - це помилка суми квадратів, яка для одного вхідного вектора $\boldsymbol{n}$, є:

$$e^{\{n\}} = \frac{1}{2}\sum_l \beta_l \left(y_l^{\{n\}} - T_l^{\{n\}}\right)^2, \tag{1.4}$$

де $\boldsymbol{T_l}$ – цільове значення виходу для l-го вихідного вузла.

Умови $\boldsymbol{\beta_l}$ дозволяють присвоювати різні ваги різним результатам, і тим самим надавати більше пріоритету правильному визначенню певних результатів.

Найпопулярніший метод тренування мережі - це алгоритм зворотного розповсюдження, в якому визначається градієнт функції помилок для кожного з ваг у мережі. Спочатку мережу ініціалізують, встановивши ваги на невеликі випадкові значення. Потім передається один з навчальних векторів через мережу і оцінюються виходи $\boldsymbol{y_l}$, а також усі проміжні значення вузла в мережі (вектори p і q). Диференційоване рівняння (1.4) по відношенню до ваги, $\boldsymbol{w_{k,l}}$, в кінцевому шарі ваг і заміни на $\boldsymbol{y_l}$ з рівняння (1.3) дає

$$\frac{\partial e}{\partial w_{k,l}} = \beta_l(y_l - T_l)q_k, \tag{1.5}$$

при тому, що ваги причинно не залежать один від одного і від значень вузла.

Визначимо градієнт $\boldsymbol{e}$ відносно ваги $\boldsymbol{w_{j,k}}$, у попередньому шарі ваг таким же чином, але тепер $\boldsymbol{q_k}$ є функцією $\boldsymbol{w_{j,k}}$, маємо:

$$\frac{\partial e}{\partial w_{j,k}} = \sum_l \beta_l(y_l - T_l)\frac{\partial y_l}{\partial w_{j,k}} = \frac{\partial q_k}{\partial w_{j,k}}\sum_l \beta_l(y_l - T_l)w_{k,l}, \tag{1.6}$$

Решту часткової похідної можна потім оцінити з рівняння (1.2). Таким чином, бачимо, що градієнт помилок щодо будь-якої ваги в мережі можна оцінити, поширюючи залежність помилок назад через мережу на цю вагу. Це можна продовжити для будь-якої кількості шарів, аж до вхідного шару, що дає нам повний вектор градієнта помилок $\frac{\partial e}{\partial w}$, де $\boldsymbol{w}$ - набір усіх ваг мережі.

Оцінивши вектор градієнта помилок, розглянемо ряд способів, за допомогою яких його можна використовувати для тренування мережі. Найпоширенішим є метод спуску градієнта, при якому регулюється ваговий вектор у напрямку протилежному градієнтному вектору, тобто

$$\Delta w = -\mu \frac{\partial e}{\partial w}, \tag{1.7}$$

Коефіцієнт $\boldsymbol{\mu}$ визначає, наскільки великий крок зроблений, і, як правило, має значення від 0,1 до 1,0. Потім градієнт перераховується за допомогою нових значень ваг, і протягом кожної ітерацій ваги наближаються до такого значення, щоб зменшити помилку $\boldsymbol{e}$. Щоб тренування було успішним (тобто значення помилки досягалося точного мінімуму), $\boldsymbol{\mu}$ потрібно кожної ітерації знижувати до нуля, хоча на практиці досить часто може бути досягнута невелика помилка і без цього.

Звичайно, потрібно, щоб мережа узагальнила усі вхідні та вихідні зображення для цілого ряду типів об'єктів (наприклад, різних видів раку шкіри), тому будемо використовувати вибірку об'єктів, які напряму стосуються нашої задачі.

Тому застосуємо описаний вище алгоритм тренувань послідовно до кожного вектора в навчальній вибірці. Далі можна або оновлювати ваги після передачі кожного вектора, або можемо тримати ваги фіксованими та оновлюватись лише після того, як помилки будуть обчислені для всіх векторів у навчальному наборі. У

останньому випадку, відомому як «пакетне» (batch) навчання, ваги оновлюються, використовуючи середню помилку (формула 1.8).

$$E = \frac{1}{N}\sum_{n=1}^{n=N} e^{\{n\}}, \tag{1.8}$$

і відповідний середній вектор градієнта похибки.

Навчання мережі - це нелінійний процес мінімізації у розмірах $\boldsymbol{w}$, де $\boldsymbol{w}$ - кількість ваг у мережі. Оскільки $\boldsymbol{w}$ зазвичай велике, це може призвести до різних ускладнень. Однією з найважливіших є проблема локальних мінімумів. Очевидно, що навчання з використанням методики градієнтного спуску зупиняється, досягнувши мінімальної межі, але це може бути лише локальний мінімум, а не глобальний мінімум, і йому може відповідати набагато більша помилка. Щоб уникнути локальних мінімумів, у рівнянні (1.7), до оновлення ваги іноді додається термін імпульсу, в якому до поточного значення додається частка $\boldsymbol{v}$ попереднього оновлення ваги. Це забезпечує певну здатність «вирватися» з невеликих локальних мінімумів: чим більшими є $\boldsymbol{v}$, тим більша кількість мінімумів, які можна ігнорувати. Підхід для уникнення локальних мінімумів полягає в тому, щоб кілька разів перевчити мережу з різних початкових випадкових ваг і побачити, чи сходиться кожен випадок до одного рішення. Якщо більшість результатів будуть вдалими - це може вважатися найкращим рішенням. Крім того, можна використовувати всі ці мережі разом у системі, в якій результати з даних мереж усереднюють.

Загальна складність полягає в тому, що потрібно знати, коли припинити навчання, оскільки справжня конвергенція у всіх (навіть у дуже простих) випадках досягається доволі рідко. Без пошуку всіх можливих ваг неможливо дізнатися, чи є досягнутий мінімум глобальним чи просто локальним, а якщо локальним - наскільки кращим є глобальний мінімум. Крім того, мінімум може мати довге, майже горизонтальне дно, тому доволі часто буде виникати потреба припиняти

тренування, коли градієнт впаде нижче якогось невеликого значення. Встановлення занадто малого градієнта конвергенції сповільнить цей процес; якщо ж задати занадто великий, то пошук може закінчитися передчасно.

Отже, спуск градієнта використовує лише знання локального градієнта, який не буде одразу вказувати на глобальний мінімум, тому для досягнення цього мінімуму потрібно багато кроків. Для подолання цієї проблеми існує безліч інших методів оптимізації. Один з них - метод спряжених градієнтів, в якому більш ефективний градієнт визначається за допомогою похідних другого порядку похибки щодо ваг. Кілька інших методів також безпосередньо використовують другі похідні (або матрицю Гессе) для отримання більш швидкої конвергенції.

Відображення, створене нейронною мережею, є інтерполяцією навчальних даних. Кожного разу, коли дані інтерполюють, потрібно здійснювати певний контроль складності, щоб не допустити перенавантаження даних. Наприклад, якщо уявити, як встановити криву через п'ять точок даних у двовимірному просторі, то - якщо дані не є лінійними - поліноміальна модель четвертого порядку дасть кращу відповідність (фактично ідеальну), ніж пряма лінія. Це навряд чи буде доречно, оскільки дані містять шум. Отже, існує компроміс між отриманням належного результату і шумом даних. Він може бути досягнутий за допомогою техніки регуляризації. Дуже базовий, але дієвий підхід - розділити навчальну вибірку на дві частини та протягом тренувань на одній половині - стежити за загальною помилкою на другій. Іншими словами – тестувати нашу модель на кожному етапі навчання. Але, потрібно пам’ятати, що помилка тесту може почати зростати після деякої кількості ітерацій, що свідчить про те, що мережа почала переповнюватися даними. Навчання можна припинити безпосередньо перед тим, як помилка почне наростати. Однак досвід показує, що в деяких випадках помилка тренувань ніколи не підвищується, навіть після того, як відбулася дуже велика кількість повторень і була досягнута невелика помилка.

Інший підхід до регуляризації - це зниження ваг. Підбір даних високого порядку, характеризується великою кривизною функції відображення, що в свою чергу відповідає великій вазі. Можемо штрафувати великі ваги, додаючи наступний термін до тренувальної помилки у рівняння (1.8):

$$\alpha \frac{1}{2} \sum_i w_i^2, \tag{1.9}$$

коли сума перевищує всі ваги в мережі, а α - деяка константа.

Більшість хто має справу із нейронними мережами використовує градієнтний спуск для навчання, настільки часто, що деякі люди вважають спуск градієнта та нейронні мережі синонімами. Це не так: прямі нейронні мережі - це загальний засіб подання нелінійного зв'язку між двома змінними, з особливістю, що градієнт помилок легко обчислюється з точки зору вільних параметрів (ваг) моделі. Яка техніка оптимізації потім використовується для визначення найкращого рішення для ваг - це справді окреме питання. Градієнтний спуск часто використовується, тому що його легко кодувати, але це може бути не найкращим вибором [8, с. 5-8].

1.4 Баєсівський метод

Інший підхід до інтерполяції даних із інформаційною мережею надається у байєсівській імовірнісній системі. Якщо повернутися до перших принципів, то побачимо, що проблема, на яку потрібно вирішити за допомогою нейронної мережі, є принципово імовірнісною: враховуючи набір навчальних даних $\boldsymbol{D}$, деякі припущення $\boldsymbol{H}$ та вектор вихідних даних $\boldsymbol{y*}$, для деякого заданого вектора входу $\boldsymbol{x}$. Якщо дивитись на задачу таким чином, можна помітити, що нас не дуже цікавлять мережеві ваги, і, крім того, «оптимальний» набір ваг мережі насправді не потрібен. Наприклад, якщо мережа навчена і знайдена гарна модель, її можна перевчити та

знайти інший набір ваг, який дасть модель, з таким же гарним результатом. Таким чином, хоча модель і дасть майже той же результат, що і раніше, ваги можуть бути дуже різними. В оптимізаційному підході ігноруються всі рішення, крім одного, тоді як при байєсівському підході поєднуються всі такі рішення шляхом маргіналізації. Зокрема, якщо вихід мережі з вагами ***w*** і вхідним вектором ***x*** записується ***y(x, w),*** то

$$y^* = \int y(x,w)P(w|D,H)dw, \tag{1.10}$$

Іншими словами, зважимо кожен можливий результат на ймовірність $P(w|D,H)$, що дані тренувань створюють ці ваги. Цю наступну ймовірність для ваг дає теорема Байєса:

$$P(w|D,H) \propto P(D|w,H)\,P(w,H), \tag{1.11}$$

Перший член правої частини цього рівняння - це ймовірність. Ймовірно, що мережа з заданим набором ваг виконує завдання у навчальних даних ***D***, коли ці дані подаються на вхід. Приймемо модель помилок Гауса для розбіжностей між результатами та цілями, тоді маємо

$$P(w|D,H) \propto \exp -E, \tag{1.12}$$

де E - сума помилок квадратів з рівнянь (1.8) і (1.4).

Додаток у рівнянні (1.11) є попередньою ймовірністю щодо ваг.

З рівнянь (1.11) і (1.12) оцінимо всі множники інтеграла у рівнянні (1.10). Однак цей інтеграл зазвичай слід оцінювати чисельним методом Монте-Карло. Це передбачає заміну інтеграла на підсумовування по ***y(x,w)*** для великої кількості значень ***w***, отриманих з розподілу $P(w|D,H)$. У цьому байєсівському підході оптимізація ваг замінюється інтеграцією за всіма можливими значеннями ваг, тому

жоден «оптимальний» ваговий вектор не обчислюється. Хоча цей підхід теоретично більш практичний, ніж оптимізація, це забирає багато часу, оскільки маргіналізація повинна здійснюватися для кожного нового вхідного вектора, який треба класифікувати. Тим не менш, у байєсівському підході є кілька переваг, такі як уникання перенавчання та природний спосіб визначення невизначеності у кожному виході мережі [8, с. 8-9].

1.5 Впровадження нейронних мереж

Тепер, коли відомі основні принципи нейронних мереж, потрібно детальніше розглянути її складові.

1.5.1 Вхідні дані

Вхідні дані - деякий вимірюваний вектор функцій. У випадку класифікації раку шкіри це буде спектр кольорових каналів зображень, зроблених за допомогою дерматоскопії. Усі вхідні дані не повинні бути суміжними або в якомусь конкретному порядку. Якщо можна зменшити кількість вхідних вузлів, то зменшуються ваги і, отже, розмірність функції відображення. Це призводить до більш швидкого навчання і може зменшити шанс застрягти в локальних мінімумах. Це також збільшує співвідношення даних до ваги та щільність даних, забезпечуючи тим самим більш надійне відображення. Існує ряд підходів до зменшення розмірності. Найпростіше - видалити будь-які матеріали, які можуть бути пошкодженими, щоб мати незначний або взагалі не впливати на проблему. Більш складний підхід - аналіз основних компонентів (PCA), який виявляє найбільш дискримінаційні лінійні комбінації вхідних даних.

1.5.2 Тренувальний та тестовий набір даних

Перше, що має бути визначено, це набір даних навчання, що складається з попередньо класифікованих зображень. Також слід переконатися, що приклади в наборі даних для тренування охоплюють увесь спектр типів, до яких треба застосувати навчену мережу.

Після того, як мережа пройшла навчання, потрібно оцінити її ефективність, і це потрібно зробити за допомогою окремого набору даних. Як було сказано раніше, мережа може підлаштовуватись під шум і таким чином запам'ятовувати конкретний набір навчальних даних, а не узагальнювати його. Це може скласти особливу проблему, якщо дані шумні або якщо мережа дуже складна. Це можна перевірити, навчивши мережу лише на підмножині наявних даних, а потім оцінити їх на решті. Але потрібно пам’ятати, що обидва набори даних повинні мати однаковий розподіл об'єктів та включати в себе весь спектр можливих класів [8, с. 10].

1.5.3 Приховані шари та вузли

Визначення відповідної кількості прихованих вузлів і шарів багато в чому залежить від досвіду. З багатьма проблемами можна отримати достатню точність з одним або двома прихованими шарами та 5–10 прихованими вузлами в цих шарах. Існує теорема, яка говорить, що мережа з одним прихованим шаром може наближати будь-яку безперервну функцію до довільної точності, за умови, що вона має достатню кількість прихованих вузлів (Hornick, Stinchcombe and White 1990). На практиці може знадобитися така велика кількість вузлів, що ефективніше перейти на другий прихований шар. Помилка може досягати невеликого значення навіть для мережі лише з одним прихованим вузлом, що еквівалентно мережі без

прихованого шару. Додавання декількох прихованих вузлів покращує помилку, але коли є 5–10 вузлів, помилка не зменшується далі [8, с. 11].

1.6 Висновки до розділу 1

Отже, у даному розділі були розглянуті основні складові нейронних мереж та деякі найпоширеніші методи, що використовуються для їх навчання. З розділу, можна зробити висновок, що одним з найважливіших етапів роботи з нейронними мережами є їх навчання. Для кожної конкретної задачі потрібно вибрати свій метод навчання, що буде оптимальним у даному випадку. Правильний вибір методу є запорукою швидко знайденого та гідного результату. Детальніше ми розглянемо способи навчання у наступному розділі.

2 ВПРОВАДЖЕННЯ МЕТОДІВ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ. ОСНОВНІ ВИДИ АРХІТЕКТУР

2.1 Методи навчання нейронних мереж

Навчання нейронної мережі - це процес визначення ваг з'єднань між нейронами таким чином, щоб мережа наближала необхідну функцію із заданою точністю. Існує три підходи до навчання нейронних мереж: навчання з учителем (supervised learning), навчання без вчителя (unsupervised learning) і навчання з підкріпленням (reinforcement learning). При навчанні з учителем на вхід мережі подаються набори вхідних сигналів (об'єктів), для яких заздалегідь відома правильна відповідь (навчальна множина). Ваги змінюються за певними правилами в залежності від того, чи правильний вихідний сигнал видала мережу. При навчанні без учителя на вхід мережі подаються об'єкти, для яких правильний вихідний сигнал заздалегідь не відомий. Навчання з підкріпленням передбачає наявність зовнішнього середовища, з яким взаємодіє мережа. Навчання відбувається на підставі сигналів, отриманих від цього середовища.

2.1.1 Модель Маккалока-Піттса

Найперша та найпростіша нейронна мережа Маккалока-Піттса лягла в основу теорії логічних мереж та кінцевих автоматів і активно використовувалася психологами і нейрофізіологи при моделюванні деяких локальних процесів нервової діяльності. В силу своєї дискретності вона цілком узгоджується з комп'ютерної парадигмою і, більш того, служить її «нейронних фундаментом». Ця модель не піддається навчанню. Ваги для всіх входів нейронів мали бути задані заздалегідь. На рисунку 2.1 можна побачити модель даної мережі.

Її робота виглядає наступним чином:

Нехай є $\boldsymbol{n}$ вхідних величин x_1, ..., x_n бінарних ознак, що описують об'єкт $\boldsymbol{x}$. Значення цих ознак будемо трактувати як величини імпульсів, що надходять на вхід нейрона через $\boldsymbol{n}$ вхідних синапсів. Будемо вважати, що, потрапляючи в нейрон, імпульси складаються з вагами w_1, ..., w_n.

Якщо значення ваги w_i позитивне, то відповідний синапс збужений, якщо негативне, то гальмуючий. Якщо сумарний імпульс перевищує заданий поріг активації w_0, то нейрон збуджується і видає на виході 1, інакше видається 0. Таким чином, нейрон обчислює $\boldsymbol{n}$-арну булеву функцію:

$$a(x) = \varphi\left(\sum_{j=1}^{n} w_j x^j - w_0\right), \tag{2.1}$$

де $\varphi(z) = [z > 0]$ – ступінчаста функція Хевісайда.

В теорії нейронних мереж функцію φ, що перетворює значення сумарного імпульсу в вихідне значення нейрона, прийнято називати *функцією активації*. Таким чином, модель Маккалока-Піттса (рисунок 2.1) еквівалентна *пороговому лінійному класифікатору* [9].

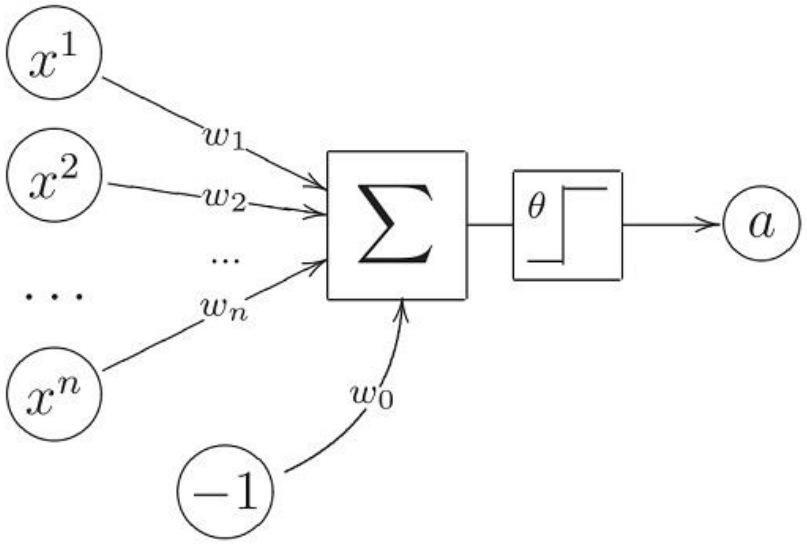


Рисунок 2.1 – Модель Маккалока-Піттса

2.1.2 Метод Хебба

Вперше ідею навчання нейронних мереж запропонував Дональд Хебб (Donald Hebb) в 1949 році. Згідно досліджень Хебба, зв'язки нейронів, які активуються разом, повинні посилюватися, а зв'язки нейронів, які спрацьовують окремо один від одного, повинні слабшати. Хебб запропонував правила зміни ваги вхідних сигналів нейронів відповідно до того, правильна відповідь видавала мережу, чи ні (навчання з учителем). А.В.Новиков довів збіжність запропонованого методу навчання нейрона на основі правил Хебба, за умови, що вибірка об'єктів лінійно роздільна.

Правила коригування ваг запропоновані Хеббом мають наступний сенс:

Перше правило Хебба: Якщо вихідний сигнал даної моделі невірний і дорівнює нулю, то необхідно збільшити ваги тих входів, на які була подана одиниця.

Друге правило Хебба: Якщо вихідний сигнал даної моделі невірний і дорівнює одиниці, то необхідно зменшити ваги тих входів, на які була подана одиниця.

Крім цих правил Хебб запропонував одне загальне, в якому були поєднані усі минулі дослідження: якщо j-та нервова клітина з вихідним сигналом y_j пов'язана з і-ою клітиною, що має вихідний сигнал y_i, зв'язком з вагою w_{ij} , то на силу зв'язку цих клітин впливають значення вихідних сигналів y_j та y_i.

Маємо, що вага w_{ij} нейрона змінюються пропорційно добутку його вхідного і вихідного сигналів:

$$\Delta w_{ij} = \mu y_j y_i, \tag{2.2}$$

де μ - це коефіцієнт навчання, значення якого вибирається в інтервалі (0,1).

Це правило Хебба може застосовуватися для нейронних мереж різних типів з різноманітними функціями активації моделей окремих нейронів.

Навчання нейрона за правилом Хебба може проводитися як з учителем, так і без нього. У другому випадку в правилі Хебба використовується фактичне значення y_i вихідного сигналу нейрона. При навчанні з учителем замість значення вихідного сигналу y_i використовується очікувана від цього нейрона реакція d_i. У цьому випадку правило Хебба записується у вигляді:

$$\Delta w_{ij} = \mu y_j d_i, \tag{2.3}$$

Правило Хебба характеризується тим, що в результаті його застосування ваги можуть приймати довільно великі значення, оскільки в кожному циклі навчання відбувається підсумовування поточного значення ваги і його приросту Δw_{ij}, маємо:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}, \tag{2.4}$$

Один із способів стабілізації процесу навчання за правилом Хебба полягає в обліку для корегування ваги останнього значення w_{ij} зменшеного на коефіцієнт забування γ. При цьому правило Хебба представляється у вигляді:

$$w_{ij}(t+1) = w_{ij}(t)(1-\gamma) + \Delta w_{ij}, \tag{2.5}$$

Значення коефіцієнта забування γ вибирається, як правило, з інтервалу (0,1) і найчастіше становить певний відсоток від коефіцієнта навчання μ. Застосування великих значень γ призводить до того, що нейрон забуває значну частину того, чого

він навчився в минулому. Рекомендовані значення коефіцієнта забування $\gamma < 0{,}1$, при яких нейрон зберігає більшу частину інформації, накопиченої в процесі навчання, і отримує можливість стабілізувати значення ваг на певному рівні.

Але, потрібно пам'ятати, що при навчанні лінійного нейрона за правилом Хебба стабілізація не відбувається навіть при введенні коефіцієнта забування. Вихідний сигнал нейрона визначається виразом:

$$y = \sum_j w_j x_j = w^T x = x^T w, \tag{2.6}$$

Згідно з правилом Хебба:

$$\Delta w_{ij} = \mu x y, \tag{2.7}$$

Підставимо вираз (2.6) в формулу (2.7) і виберемо для спрощення $\mu = 1$, то отримаємо приріст вектора ваг Δw у вигляді:

$$\Delta w = Cw, \tag{2.8}$$

де $C = xx^T$- це матриця кореляції, яка за визначенням є симетричною і позитивно визначеною і має власні натуральні і невід'ємні значення.

При виконанні операцій, описуваних залежністю (2.8) і повторюваних на позитивно визначеній матриці С, процес стає розбіжним, а значення компонентів вектора прагнуть до нескінченності.

Нестабільність правила Хебба в процесі навчання можна усунути обмеженням вектора ваг за рахунок підбору пропорційного коефіцієнта α на

кожному кроці навчання, щоб $w' = \alpha w$ при $\|w'\| = 1$. Цей метод досить складний і вимагає додаткових трудовитрат на етапі навчання.

Е. Ойя модифікував правило Хебба таким чином, що і без підбору α вектор ваг самостійно прагне $\|w\| = 1$.

Відповідно до правила Ойя корегування ваг проводиться відповідно до виразу:

$$\Delta w = \mu y(x_i - y w_i), \tag{2.9}$$

Це правило нагадує зворотнє поширення, оскільки сигнал x_i модифікується зворотнім сигналом, пов'язаним з вихідним сигналом y нейрона. Для кожного окремо взятого нейрона правило Ойя може вважатися локальним, так як в процесі модифікації x_i приймається до уваги тільки той ваговий коефіцієнт, значення якого підбирається в поточний момент часу.

Доказ обмеженості ваг, що уточнюється за правилом Ойя, можна отримати, замінюючи скалярний вираз (2.9) векторної формою ($\mu = 1$).

$$\Delta w = Cw - (w^T C w)w, \tag{2.10}$$

Стабільність процесу навчання досягається, коли при досить тривалому навчанні забезпечується $\|\Delta w\| = 0$, тобто

$$Cw = (w^T C w)w, \tag{2.11}$$

Якщо власне значення кореляційної матриці C позначити λ, а вектор w підбирати, як пов'язаний з нею власний вектор, то за визначенням власного значення маємо $Cw = \lambda w$.

Підставляючи попередньо згаданий вираз в формулу (2.10), отримуємо:

$$\lambda = w^T Cw = w^T \lambda w = \lambda |w|^2, \tag{2.12}$$

З формули (2.12) слідує, що застосування модифікованого правила Хебба призводить до обмеження модуля вектора w одиницею $|w| = 1$, що забезпечує обмеження значень вагових коефіцієнтів [10, с. 40-44].

2.1.3 Алгоритм зворотного розповсюдження

Алгоритм зворотного поширення помилки визначає стратегію підбору ваг багатошарової мережі із застосуванням градієнтних методів оптимізації. "Винайдений заново" кілька разів, він в даний час вважається одним з найбільш ефективніших алгоритмів навчання багатошарової мережі. Його основу складає цільова функція у вигляді квадратичної суми різниць між фактичними і очікуваними значеннями вихідних сигналів. Нехай, кількість нейронів у вихідному шарі *M*. Тоді, у разі одиничної навчальної вибірки *(x, d)* цільова функція визначається у вигляді:

$$E(w) = \frac{1}{2}\sum_{k=1}^{M}(y_k - d_k)^2, \tag{2.13}$$

При більшій кількості навчальних вибірок j (j =1, 2, …, p) цільова функція перетворюється в суму за всіма вибірками:

$$E(w) = \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{M}\left(y_k^{(j)} - d_k^{(j)}\right)^2, \qquad (2.14)$$

Корегування ваг може проводитися після пред'явлення кожної навчальної вибірки (так званий режим "онлайн") або одноразово після пред'явлення всіх вибірок, що становлять цикл навчання (режим "офлайн"). В подальшому будемо використовувати цільову функцію виду (2.13), яка відповідає актуалізації ваг після пред'явлення кожної вибірки.

Для спрощення можна вважати, що мета навчання полягає в такому визначенні значень ваг нейронів кожного шару мережі, щоб при заданому вхідному векторі отримати на виході значення сигналів y_i, що збігаються з необхідною точністю з очікуваними значеннями d_i, при i = 1, 2, …, M.

Навчання мережі з використанням алгоритму зворотного поширення помилки проводиться в кілька етапів. На першому з них пред'являється навчальна вибірка $\boldsymbol{x}$ і розраховуються значення сигналів відповідних нейронів мережі. При заданому векторі $\boldsymbol{x}$ визначаються спочатку значення вихідних сигналів v_i прихованого шару, а потім значення y_i нейронів вихідного шару. Після отримання значень вихідних сигналів стає можливим розрахувати фактичне значення цільової функції виразом (2.13). На другому етапі мінімізується значення цієї функції.

Якщо прийняти, що цільова функція неперервна, то найбільш ефективними способами навчання виявляються градієнтні методи оптимізації, згідно з якими корегування ваг проводиться за формулою 2.15.

$$w(k+1) = w(k) + \Delta w, \tag{2.15}$$

де $\Delta w = \eta p(w) \eta$ - коефіцієнт навчання, а $p(w)$ - напрямок в багатовимірному просторі $\boldsymbol{w}$.

Навчання багатошарової мережі із застосуванням градієнтних методів вимагає визначення вектора градієнта щодо ваг всіх шарів мережі, що необхідно для правильного вибору напрямку ***p(w).*** Це завдання має очевидне рішення тільки для ваг вихідного шару. Для інших верств створена спеціальна стратегія, яка в теорії штучних нейронних мереж називається алгоритмом зворотного поширення помилки (англ.: error backpropagation), які ототожнюються з процедурою навчання мережі. Відповідно до цього алгоритмом в кожному циклі навчання виділяються наступні етапи:

1. Аналіз нейронної мережі в прямому напрямку передачі інформації при генерації вхідних сигналів, що складають черговий вектор $\boldsymbol{x}$. В результаті такого аналізу розраховуються значення вихідних сигналів нейронів прихованих шарів і вихідного шару, а також відповідні похідні функції активації кожного шару.
2. Створення мережі зворотного поширення помилок шляхом зміни напрямків передачі сигналів, заміна функцій активації їх похідними і подача на колишній вихід (а зараз - вхід) мережі збудження у вигляді різниці між фактичним і очікуваним значенням. Для визначеної таким чином мережі необхідно розрахувати значення необхідних зворотних різниць.
3. Корегування ваг (навчання мережі) проводиться за запропонованими вище формулами на основі результатів, отриманих в пунктах 1 та 2, для оригінальної мережі і для мережі зворотного поширення помилки.
4. Описаний в пункті 2 та 3 процес слід повторити для всіх навчальних вибірок, продовжуючи його аж до виконання умови зупинки алгоритму.

Дія алгоритму завершується в момент, коли норма градієнта впаде нижче апріорі заданого значення ε, що характеризує точність процесу навчання.

Базові формули і їх модифікації для конкретних типів нейронних мереж вважаються класичними для теорії нейронних мереж. З цієї причини ми розглянемо тільки умови, які стосуються мережі з одним прихованим шаром.

Як і раніше, кількість вхідних вузлів позначимо літерою ***N*** кількість нейронів в прихованому шарі ***K***, а кількість нейронів у вихідному шарі ***M***. Будемо використовувати сигмоїдну функцію активації цих нейронів. Основу алгоритму становить розрахунок значення цільової функції як квадратичної суми різниць між фактичними і очікуваними значеннями вихідних сигналів мережі. У разі одиничної навчальної вибірки ***(x, d)*** цільова функція задається формулою (2.13), а для безлічі навчальних вибірок *j (j = 1, 2, ..., p)* - формулою (2.14). Для спрощення викладається будемо використовувати цільову функцію виду (2.13), яка дозволяє уточнювати ваги після пред'явлення кожної навчальної вибірки.

Тепер маємо такий вираз:

$$E = \frac{1}{2}\sum_{k=1}^{M}\left[f\left(\sum_{i=0}^{K} w_{ki}^{(2)} f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_i\right)\right) - d_k\right]^2, \qquad (2.16)$$

Конкретні компоненти градієнта розраховуються диференціюванням залежності (2.16). В першу чергу підбираються ваги нейронів вихідного шару.

Для вихідних ваг отримуємо:

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = (y_i - d_i)\frac{df(u_i^{(2)})}{du_i^{(2)}} v_j, \qquad (2.17)$$

де $u_i^{(2)} = \sum_{j=0}^{K} w_{ij}^{(2)} v_j$.

Якщо ввести позначення $\delta_i^2 = (y_i - d_i)\frac{df(u_i^{(2)})}{du_i^{(2)}}$, то відповідний компонент градієнта ваг нейронів вихідного шару можна представити у вигляді:

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = \delta_i^{(2)} v_j, \tag{2.18}$$

Компоненти градієнта щодо нейронів прихованого шару визначаються за тим же принципом, проте вони описуються іншою, більш складної залежністю, наступною з існування функції:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{M}(y_k - d_k)\frac{dy_k}{dv_i}\frac{dv_i}{dw_{ij}^{(1)}}, \tag{2.19}$$

Після конкретизації окремих складових цього виразу отримуємо:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{M}(y_k - d_k)\frac{df(u_k^{(2)})}{du_k^{(2)}} w_{ki}^{(2)} \frac{df(u_k^{(1)})}{du_i^{(1)}} x_j, \tag{2.20}$$

Введемо позначення

$$\delta_i^{(1)} = \sum_{k=1}^{M}(y_k - d_k)\frac{df(u_k^{(2)})}{du_k^{(2)}} w_{ki}^{(2)} \frac{df(u_k^{(1)})}{du_i^{(1)}}, \tag{2.21}$$

Отримаємо вираз (формула 2.22), що визначає компоненти градієнта щодо ваг нейронів прихованого шарі.

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_i^{(1)} x_j, \tag{2.22}$$

В обох випадках (формули (2.18) і (2.22)) опис градієнта має аналогічну структуру і представляється добутком двох сигналів: перший відповідає початковому вузлу даної зваженого зв'язку, а другий - величиною похибки, перенесеної на вузол, з яким цей зв'язок встановлено. Визначення вектора градієнта дуже важливо для подальшого процесу корегування ваг. [10, с. 51-54]. У класичному алгоритмі зворотного поширення помилки фактор ***p(w)*** враховується в натуральному вираженні (2.15), задає напрям від'ємного градієнта, тому

$$\Delta w = -\eta \nabla E(w), \tag{2.23}$$

2.1.4 Алгоритм найшвидшого спуску

Якщо при розкладанні цільової функції ***E(w)*** в ряд Тейлора обмежитися її лінійним наближенням, то ми отримаємо алгоритм найшвидшого спуску. Для виконання співвідношення $E(w_{k+1}) < E(w_k)$ досить підібрати $g(w_k)^T p < 0$. Умові зменшення значення цільової функції відповідає вибір вектора напрямку:

$$p_k = -g(w_k), \tag{2.24}$$

Саме виразом (2.24) визначається вектор напрямку в методі найшвидшого спуску.

Обмеження доданків першого порядку при розкладанні функції в ряд Тейлора не дозволяє використовувати інформацію про її кривизну. Це обумовлює

повільну збіжність методу (вона залишається лінійної). Зазначений недолік, а також різке уповільнення мінімізації в найближчій околиці точки оптимального рішення, коли градієнт приймає дуже малі значення, роблять алгоритм найшвидшого спуску низько ефективним. Проте з урахуванням його простоти, невисоких вимог до обсягу пам'яті і відносно невеликої обчислювальної складності саме цей метод протягом багатьох років був і залишається в даний час основним способом навчання багатошарових мереж. Підвищити його ефективність вдається шляхом модифікації (як правило, евристичної) виразу, що визначає напрямок. Хороші результати приносить застосування методу навчання з так званим моментом. При цьому підході корегування ваг мережі проводиться з урахуванням модифікованої формули визначення значення:

$$\Delta w_k = \eta_k p_k + \alpha(w_k - w_{k-1}), \tag{2.25}$$

де α - це коефіцієнт моменту, який приймає значення в інтервалі [0, 1].

Перший доданок цього виразу відповідає звичайному навчанню за методом найшвидшого спуску, тоді як друге враховує останню зміну ваг і не залежить від фактичного значення градієнта. Чим більше значення коефіцієнта α, тим більше значення має показник моменту на підбір ваг. Цей вплив істотно зростає на плоских ділянках цільової функції, а також поблизу локального мінімуму, де значення градієнта близько до нуля.

На плоских ділянках цільової функції приросту ваг (при постійному значенні коефіцієнта навчання $\eta_k = \eta$)залишається приблизно одним і тим ж. Це означає, що $\Delta w_k = \eta p_k + \alpha \Delta w_k$, тому ефективне збільшення значень ваг можна описати відношенням

$$\Delta w_k = \frac{\eta}{1-\alpha} p_k, \tag{2.26}$$

При значенні $\alpha = 0{,}9$ це відповідає 10-кратному збільшенню ефективного значення коефіцієнта навчання, а також 10-кратному прискоренню процесу навчання.

Поблизу локального мінімуму показник моменту, не пов'язаний з градієнтом, може викликати дуже велику зміну ваг, що приводить до збільшення значення цільової функції і до виходу з "зони тяжіння" цього мінімуму. При малих значеннях градієнта показник моменту починає домінувати у вираженні (2.25), що призводить до такого приросту ваг Δw_k, яке відповідає збільшенню значення цільової функції, що дозволяє вийти із зони локального мінімуму. Однак показник моменту не повинен повністю домінувати протягом усього процесу навчання, оскільки це призвело б до нестабільності алгоритму. Для запобігання такого надмірного домінування значення цільової функції E контролюється так, щоб допускати його збільшення лише в певних межах, наприклад не більше 4%. При такому підході, якщо на чергових (k-ому та (k+1)-ому) кроках ітерації виконується умова $E(k+1) < 1{,}04\ E(k)$, то зміни ігноруються і вважається, що $(w_k - w_{k-1}) = 0$. При цьому показник градієнта починає домінувати над показником моменту і процес розвивається в напрямку мінімізації, заданому вектором градієнта. Слід підкреслити, що підбір величини коефіцієнта моменту є непростою справою і вимагає проведення великої кількості експериментів, що мають на меті вибрати таке значення, яке найкращим чином відображало б специфіку розв'язуваної проблеми [10, с. 62-63].

2.1.5 Алгоритм спряжених градієнтів

У цьому методі при виборі напрямку мінімізації не використовується інформація про матрицю Гессе. Напрямок пошуку p_k вибирається таким чином, щоб воно було ортогональним і зв'язаних до всіх попередніх напрямків

$p_0, p_1, \dots, p_{k-1}$. Безліч векторів p_i, i=0, 1, …, k буде взаємно зв'язаним відносно матриці **G** (матриця Гессе), якщо

$$p_i^T G p_j = 0, i \neq j, \tag{2.27}$$

Вектор, що задовольняє заданим вище умов, має вигляд:

$$p_k = -g_k + \beta_{k-1} p_{k-1}, \tag{2.28}$$

де $g_k = g(w_k)$ позначає фактичне значення вектора градієнта.

З формули (2.28) випливає, що новий напрямок мінімізації залежить тільки від значення градієнта в точці рішення w_k і від попереднього напрямку пошуку p_{k-1}, помноженого на коефіцієнт спряження β_{k-1}. Цей коефіцієнт відіграє дуже важливу роль, збираючи в собі інформацію про попередні напрямках пошуку. Існують різні правила розрахунку його значення.

Найбільш відомі серед них:

$$\beta_{k-1} = \frac{g_k^T (g_k - g_{k-1})}{g_{k-1}^T g_{k-1}}, \tag{2.29}$$

$$\beta_{k-1} = \frac{g_k^T (g_k - g_{k-1})}{-p_{k-1} g_{k-1}}, \tag{2.30}$$

З огляду на накопичення похибок округлення в послідовних циклах обчислень практичне застосування методу сполучених градієнтів пов'язано з

поступовою втратою властивості ортогональності між векторами напрямків мінімізації. З цієї причини після виконання $\boldsymbol{n}$ ітерацій (значення $\boldsymbol{n}$ розраховується як функція від кількості змінних, що підлягають оптимізації) проводиться рестарт процедури, на першому кроці якої напрямок мінімізації з точки отриманого рішення вибирається за алгоритмом найшвидшого спуску. Метод сполучених градієнтів має збіжність, близьку до лінійної, і він менш ефективний, ніж, наприклад, метод змінної метрики, однак помітно швидше, ніж метод найшвидшого спуску. Він широко застосовується як єдино ефективний алгоритм оптимізації при досить значній кількості змінних, яке може досягати декількох десятків тисяч. Завдяки невисоким вимогам до пам'яті і відносно низькою обчислювальної складності методу сполучених градієнтів дозволяє успішно вирішувати дуже серйозні оптимізаційні задачі [10, с. 67-68].

2.2 Підбір коефіцієнта навчання

Алгоритми, що були розглянуті вище, дозволяють визначити тільки напрямок, в якому зменшується цільова функція, але не говорять нічого про величину кроку, при якому ця функція може отримати мінімальне значення. Після вибору правильного напряму $\boldsymbol{p_k}$ слід визначити на ньому нову точку рішення $\boldsymbol{w_{k+1}}$, в якій буде виконуватися умова $E(w_{k+1}) < E(w_k)$. Необхідно підібрати таке значення η_k, щоб нове рішення $w_{k+1} = w_k + \eta_k p_k$ лежало якомога ближче до мінімуму функції *E(w)* в напрямку p_k. Грамотний підбір коефіцієнта η_k має великий вплив на збіжність алгоритму оптимізації до мінімуму цільової функції. Чим сильніше величина η_k відрізняється від значення, при якому *E(w)* досягає мінімуму в обраному напрямку p_k, тим більша кількість ітерацій потрібно для пошуку оптимального рішення. Занадто мале значення η не дозволяє мінімізувати цільову функцію за один крок і викликає необхідність повторно рухатися в тому ж

напрямку. Занадто великий крок призводить до "перестрибування" через мінімум функції і фактично змушує повертатися до нього.

Існують різні способи підбору значення η, званого в теорії нейронних мереж коефіцієнтом навчання. Найпростіший з них заснований на фіксації постійного значення η на весь період оптимізації. Цей спосіб практично використовується тільки спільно з методом найшвидшого спуску. Він має низьку ефективність, оскільки значення коефіцієнта навчання ніяк не залежить від вектора фактичного градієнта і від напрямку $\boldsymbol{p}$ на даній ітерації. Величина η підбирається, як правило, окремо для кожного шару мережі з використанням різних емпіричних залежностей. Один з підходів полягає у визначенні мінімального значення коефіцієнта η для кожного шару за формулою

$$\eta \leq \min(\frac{1}{n_i}), \tag{2.31}$$

де n_i – кількість входів i-ого нейрона в шарі.

Інший більш ефективний метод заснований на адаптивному підборі коефіцієнта η з урахуванням фактичної динаміки величини цільової функції в результаті навчання. Відповідно до цього методу стратегія зміни значення η визначається шляхом порівняння сумарної похибки ε на i-ій ітерації з її попереднім значенням, причому ε розраховується за формулою

$$\varepsilon = \sqrt{\sum_{j=1}^{M}(y_j - d_j)^2}, \tag{2.32}$$

Для прискорення процесу навчання слід прагнути до безперервного збільшення η при одночасному контролі приросту похибки ε в порівнянні з її значенням на попередньому кроці. Незначне зростання цієї похибки вважається допустимим.

Якщо похибки (i-1) та i-й ітераціях позначити відповідно ε_{i-1} та ε_i, а коефіцієнти навчання на цих же ітераціях - η_{i-1} і η_i, то в разі $\varepsilon_i > k_w\varepsilon_{i-1}$ (k_w-коефіцієнт допустимого приросту похибки) значення η має зменшуватися відповідно до формули

$$\eta_{i+1} = \eta_i p_d, \tag{2.33}$$

де p_d - коефіцієнт зменшення η .

В іншому випадку, коли $\varepsilon_i \leq k_w\varepsilon_{i-1}$, приймається

$$\eta_{i+1} = \eta_i p_i, \tag{2.34}$$

де p_i - коефіцієнт збільшення η.

Незважаючи на деяке зростання обсягу обчислень (необхідних для додаткового розрахунку значень), можливе істотне прискорення процесу навчання.

Цікаво простежити характер зміни коефіцієнта η в процесі навчання. Як правило, на початкових етапах домінує тенденція до його збільшення, однак при досягненні деякого квазістаціонарного стану величина η поступово зменшується, але не монотонно, а циклічно зростаючи і знижуючи в наступних один за одним циклах.

Однак необхідно підкреслити, що адаптивний метод підбору η сильно залежить від виду цільової функції і значень коефіцієнтів k_w, p_d, p_j. Значення, оптимальні для функції одного виду, можуть уповільнювати процес навчання при використанні іншої функції. Тому при практичній реалізації цього методу слід звертати увагу на механізми контролю та управління значеннями коефіцієнтів, підбираючи їх відповідно до специфіки розв'язуваної задачі.

Найбільш ефективний, хоча і найбільш складний, метод підбору коефіцієнта навчання пов'язаний з спрямованої мінімізацією цільової функції в обраному

заздалегідь напрямку p_k. Необхідно так підібрати скалярне значення η_k, щоб нове рішення $w_{k+1} = w_k + \eta_k p_k$ відповідало мінімуму цільової функції в даному напрямку p_k. Насправді отримане рішення w_{k+1} тільки з певним наближенням можна вважати справжнім мінімумом. Це результат компромісу між обсягом обчислень і впливом величини η_k на збіжність алгоритму.

Серед найбільш популярних способів спрямованої мінімізації можна виділити безградієнтні і градієнтні методи. У безградієнтних методах використовується тільки інформація про значеннях цільової функції, а її мінімум досягається в процесі послідовного зменшення діапазону значень вектора *w*. Прикладами можуть служити методи розподілу навпіл, золотого перетину або метод Фібоначчі, що розрізняються способом декомпозиції одержуваних піддіапазонів.

Також заслуговує на увагу метод апроксимації цільової функції *E(w)* в попередньо вибраному напрямку p_k з подальшим розрахунком мінімуму, таким чином одержуємо, функцію однієї змінної η. У цьому випадку, використовується апроксимуючий многочлен другого порядку.

Однак кращим рішенням вважається застосування градієнтних методів, в яких, крім значення функції, враховується також і її похідна вздовж подає вектора p_k. Вони дозволяють значно прискорити досягнення мінімуму, оскільки використовують інформацію про направлення зменшення величини цільової функції. У такій ситуації застосовується, як правило, апроксимуючий многочлен третього порядку [10, с. 68-71].

2.3 Архітектура нейронних мереж

Зазвичай, правильно підібрана архітектура нейронної мережі до конкретної задачі є гарантом гідного результату для вирішення проблеми. У цьому пункті,

розглянемо архітектури, що використовуються частіше інших і є найбільш універсальними у світі нейронних моделей.

2.3.1 Нейронні мережі прямого зв’язку

Нейронні мережі прямого поширення (feed forward neural networks, FF або FFNN) і перцептрони (perceptrons, P) дуже прямолінійні, вони передають інформацію від входу до виходу. Нейронні мережі часто описуються у вигляді листкового торта, де кожен шар складається з вхідних, прихованих або вихідних клітин. Клітини одного шару не пов'язані між собою, а сусідні шари зазвичай повністю пов'язані (рисунок 2.2). Найпростіша нейронна мережа має дві вхідних клітини і одну вихідну, і може використовуватися в якості моделі логічних вентилів. FFNN зазвичай навчаються за методом зворотного поширення помилки, в якому мережа отримує безлічі вхідних і вихідних даних. Якщо у мережі є достатня кількість прихованих нейронів, вона теоретично здатна змоделювати взаємодію між вхідним і вихідними даними. Практично такі мережі використовуються рідко, але їх часто комбінують з іншими типами для отримання нових.

Рисунок 2.2 – Модель перцептрона

2.3.2 Нейронна мережа Хопфілда

Нейронна мережа Хопфілда (Hopfield network, HN) - це повнозв’язна нейронна мережа із симетричною матрицею зв'язків. Під час отримання вхідних

даних кожен вузол є входом, в процесі навчання він стає прихованим, а потім стає виходом (рисунок 2.3). Мережа навчається так: значення нейронів встановлюються відповідно до бажаного шаблону, після чого обчислюються ваги, які в подальшому не змінюються. Після того, як мережа навчилася на одному або декільком шаблонам, вона завжди буде зводитися до одного з них (але не завжди - до бажаного). У кожного нейрона є свій поріг активації, при проходженні якого нейрон приймає одне з двох значень (зазвичай -1 або 1, іноді 0 або 1). Така мережа часто називається мережею з асоціативної пам'яттю; як людина, що бачить половину таблиці, може представити другу половину таблиці, так і ця мережа, отримуючи таблицю, з половиною пошкоджених даних, відновлює її до повної.

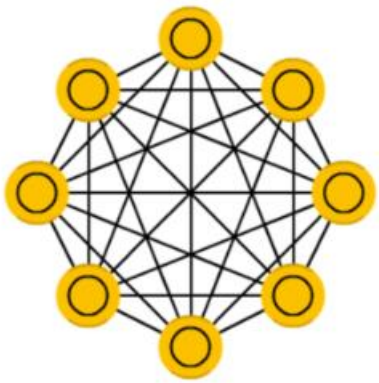


Рисунок 2.3 – Нейронна мережа Хопфілда

2.3.3 Машина Больцмана

Машина Больцмана (Boltzmann machine, BM) дуже схожа на мережу Хопфілда, але в ній деякі нейрони позначені як вхідні, а деякі - як приховані. Вхідні нейрони в подальшому стають вихідними. Машина Больцмана - це стохастична мережа. Навчання проходить за методом зворотного поширення помилки або за

алгоритмом порівняльної розбіжності. В цілому процес навчання дуже схожий на такий, як у мережі Хопфілда [12].

2.3.4 Згорткові мережі

Згорткові нейронні мережі (convolutional neural networks, CNN) і глибинні згорткові нейронні мережі (deep convolutional neural networks, DCNN) сильно відрізняються від інших видів мереж (рисунок 2.4). Зазвичай вони використовуються для обробки зображень, рідше для аудіо. Типовим способом застосування CNN є класифікація зображень: якщо на зображенні є кішка, мережа видає "кішка", якщо є собака - "собака". Такі мережі зазвичай використовують "сканер", що не збирає всі дані за один раз. Наприклад, якщо буде подано зображення 200 × 200, то мережа не буде відразу обробляти всі 40 тисяч пікселів. Замість цього мережа обробляє квадрат розміру 20 x 20 (зазвичай з лівого верхнього кута), потім зрушиться на 1 піксель і обробить новий квадрат, і т.д. Ці вхідні дані потім передаються через згорткові шари, в яких не всі вузли з'єднані між собою. Ці шари мають властивість стискатись з глибиною, причому часто використовуються ступеня двійки: 32, 16, 8, 4, 2, 1. На практиці до кінця CNN прикріплюють FFNN для подальшої обробки даних. Такі мережі називаються глибинними (DCNN).

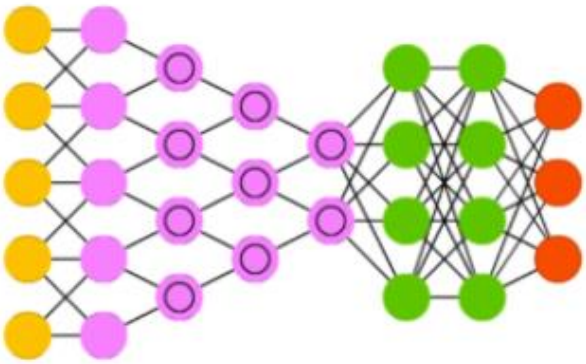


Рисунок 2.4 – Згорткові нейронні мережі

2.3.5 Рекурентні мережі

Рекурентні нейронні мережі (recurrent neural networks, RNN) - це мережі типу FFNN, але з особливістю: нейрони отримують інформацію не тільки від попереднього шару, але і від самих себе попереднього проходу (рисунок 2.5). Це означає, що порядок, в якому подаються дані і навчається мережа, стає важливим. Великою складністю мереж RNN є проблема зникання (або вибухового) градієнта, яка полягає у швидкій втраті інформації з плином часу. Звичайно, це впливає лише на ваги, а не на стан нейронів, але ж саме в них накопичується інформація. Зазвичай мережі такого типу використовуються для автоматичного доповнення інформації [13].

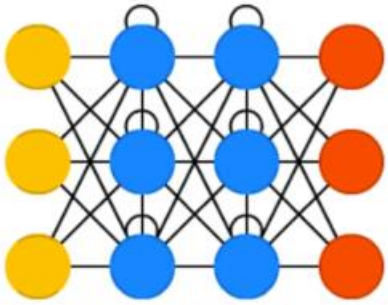


Рисунок 2.5 – Рекурентні нейронні мережі

На рисунку 2.6 розглянемо усі позначення, що використовувались при зображені моделей даних нейронних мереж.

Рисунок 2.6 – Позначення комірок в архітектура мереж

2.4 Висновки до розділу 2

Отже, у даному розділі були розглянуті головні методи, для навчання нейронних мереж, а також їх найпопулярніші підходи до створення архітектури. Завдяки цим методам та моделям нейронні мережи зможуть вирішити конкретну задачу із доволі високою точністю та знайти оптимальний шлях для її вирішення. Також, в залежності від обраного методу, час навчання та точність передбачення можуть змінюватись. У наступному розділі розглянемо архітектури та оберемо відповідну до нашої задачі.

3 ДЕТАЛЬНИЙ ОГЛЯД СУЧАСНИХ МЕРЕЖЕВИХ АРХІТЕКТУР ТА ЇХ ПОРІВНЯННЯ

Глибокі нейронні мережі та глибоке навчання - це потужні та популярні алгоритми. І велика частина їх успіху полягає в ретельному проектуванні архітектури нейронної мережі. Для більш гідного аналізу та порівняння усіх створених мереж, розглянемо графік залежності. На рисунку 3.1 зображено залежність точності результату порівняно з кількістю операцій, необхідних для одного проходу вперед. Розмір краплин пропорційний кількості мережевих параметрів.

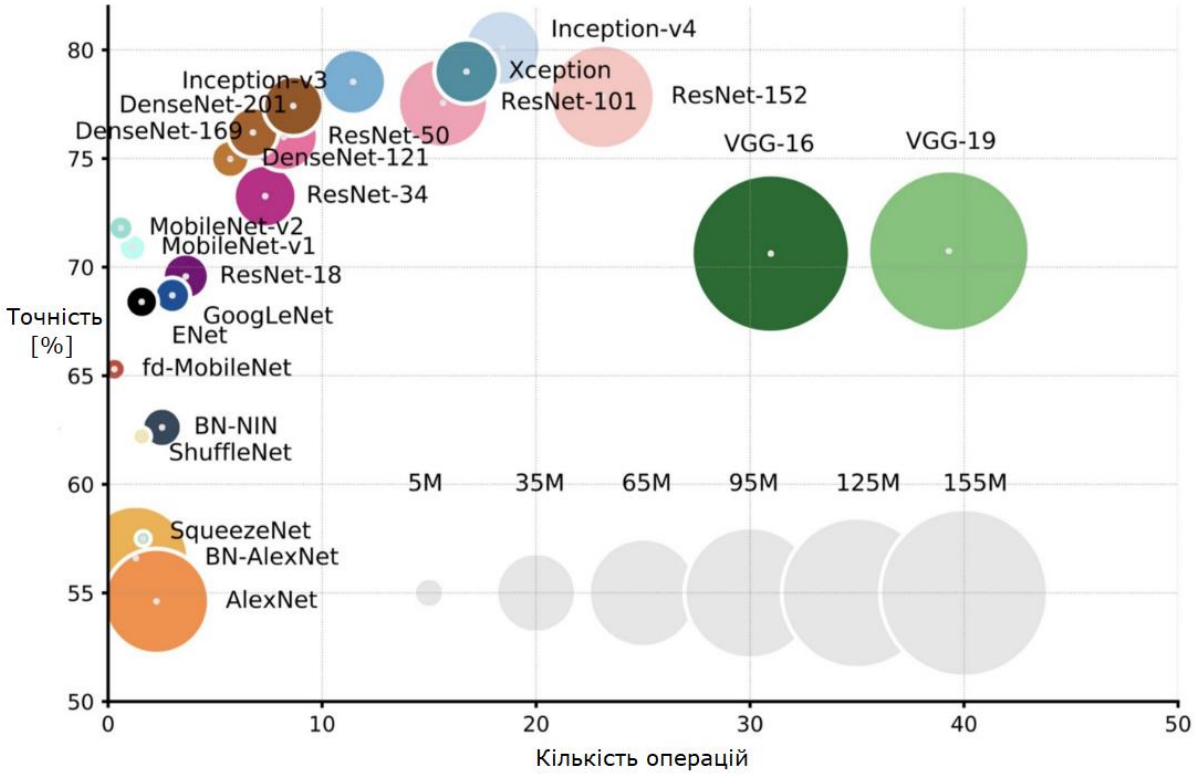


Рисунок 3.1 – Порівняння архітектур нейронних мереж

Щоб обрати гідну стратегію розв’язання, розглянемо детальніше деякі з представлених вище [14].

3.1 Архітектура LeNet-5

1994-й рік і створення однієї з перших згорткових нейронних мереж, що сприяли поглибленому навчанню. Ця новаторська робота Ян Лекуна була названа LeNet5 після багатьох успішних досліджень з 1988 року!

Архітектура LeNet-5 допомогла з’ясувати, що особливості та ознаки зображення розподілені по всьому його периметру. Також, корисною знахідкою був факт того, що згортки зі конкретними параметрами є ефективним способом вилучення подібних ознак у зображені. У той час не було GPU(graphics processing unit), який би допомагав навчанню, і навіть процесори були повільними. Тому можливість збереження параметрів та обчислень була ключовою перевагою. На відміну від використання кожного пікселя як окремого входу великої багатошарової нейронної мережі. LeNet5 показала, що їх не слід використовувати в першому шарі, оскільки зображення є дуже просторово корельованими і це не дасть потрібного результату.

Особливості LeNet-5 можна підсумувати як:

- ця згорткова нейронна мережа використовує послідовність з 3 шарів: згортковий, шар об'єднання, нелінійний;
- використовування згортки для отримання просторових ознак
- підвибірка, що використовує просторове середнє значення карт зображення
- нелінійність у вигляді tanh чи сигмоїда
- багатошарова нейронна мережа (MLP) як кінцевий класифікатор
- рідка матриця з'єднання між шарами, щоб уникнути великих обчислювальних витрат

Загалом ця мережа була джерелом більшості останніх архітектур і була справжнім натхненням для багатьох людей у цій галузі.

Архітектура LeNet-5 складається з двох наборів згорткових і середньо об'єднаних шарів (Average pooling(AP)), потім ще один згортковий шар, потім два повнозв’язні шари (Fully connected(FC)) і класифікатор softmax(3.1). На рисунку 3.2 можна побачити різницю між AP та FC шарами. У таблиці 3.1 розглянемо цілісну архітектуру LeNet -5.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}, \tag{3.1}$$

де i=1,…, K, а $z = (z_1, \ldots, z_k) \in \mathbb{R}^K$

Таблиця 3.1 – Архітектура LaNet-5

Шар		Карта об’єктів	Розмір зображення	Розмір ядра згортки	Крок згортки	Функція активації
Вхідний	Зображення	1	32x32	-	-	-
1	Згортка	6	28x28	5x5	1	tanh
2	AP	6	14x14	2x2	2	tanh
3	Згортка	16	10x10	5x5	1	tanh
4	AP	16	5x5	2x2	2	tanh
5	Згортка	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Вихідний	FC	-	10	-	-	softmax

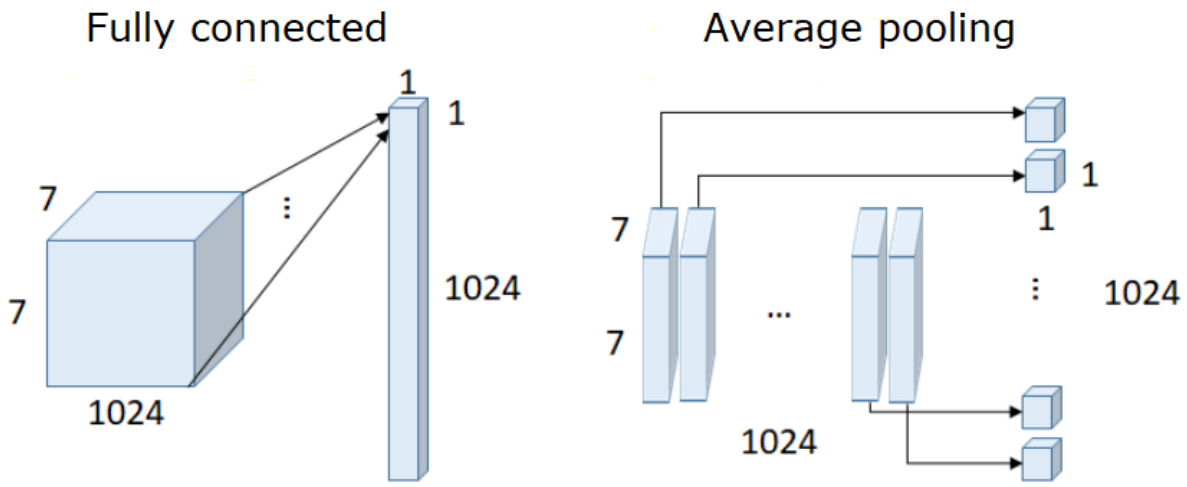


Рисунок 3.2 – Різниця між шаром FC та AP

У роки з 1998 по 2010 рр. дослідження нейронних мереж перебувало в інкубації. Більшість людей не помічали їх зростаючої сили, тоді як багато інших дослідників повільно прогресували. Все більше даних було доступно через зростання кількості камер стільникового телефону та дешевих цифрових камер. Обчислювальна потужність зростала, процесори ставали швидшими, а графічні процесори стали обчислювальним інструментом загального призначення. Обидві ці тенденції додавали прогресу нейронної мережі, хоча і з низькою швидкістю. Але після цього ривку світ знову зацікавився вивченням нейронних мереж [15].

3.2 Архітектура AlexNet

У 2012 році Алекс Крижевський випустив AlexNet та масштабував уявлення LeNet про набагато більшу нейронну мережу, яку можна використовувати для вивчення складних об'єктів та їх ієрархій.

Згорткові нейронні мережі завжди були ідеальною моделлю для розпізнавання об'єктів. Ними легко керувати і навіть простіше їх навчати. Вони не

відчувають надмірного перевантаження в будь-яких масштабах під час використання на мільйонах зображень. Їх продуктивність така ж, як у стандартних нейронних мереж прямого зв’язку однакового розміру. Єдина проблема: їх важко застосувати до зображень високої роздільної здатності. За шкалою ImageNet (набір даних) потрібно запровадити нововведення, яке було б оптимізоване для кількох графічних процесорів та скорочення часу навчання, одночасно покращуючи продуктивність. ImageNet складається з понад 15 мільйонів зображень високої роздільної здатності, позначених 22 тисячами класів [17].

Архітектура AlexNet складається з восьми шарів: п'яти згорткових шарів і трьох повністю з'єднаних шарів (рисунок 3.3), а також функцій max-pooling. Принцип функції max-pooling розглянемо на рисунку 3.4.

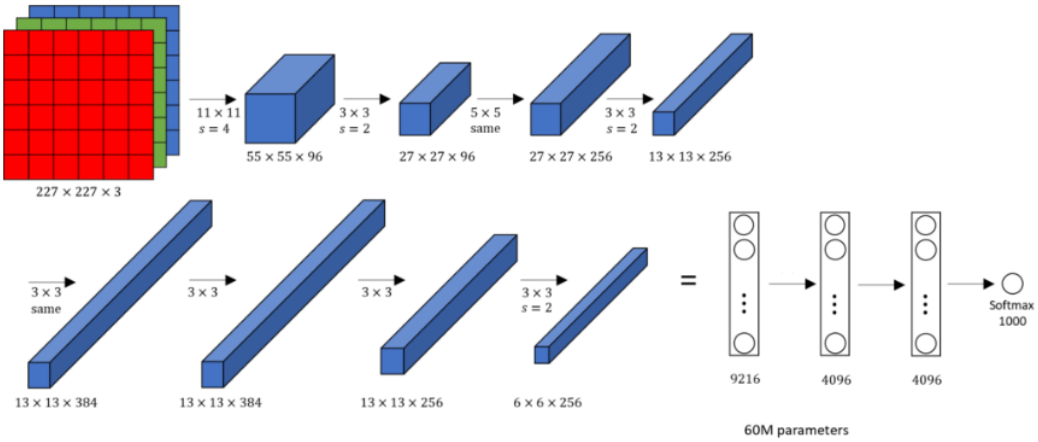


Рисунок 3.3 – Архітектура AlexNet

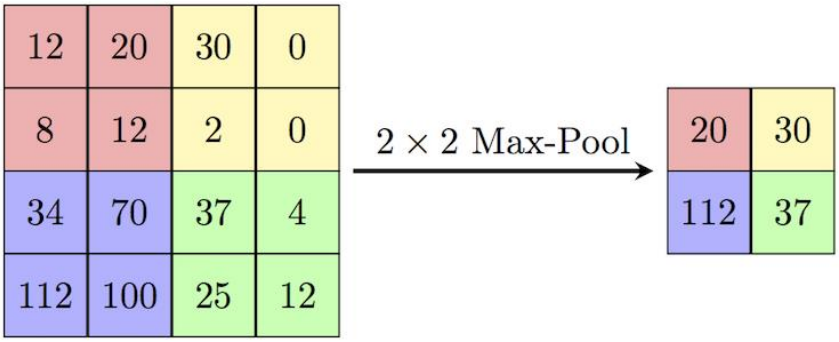


Рисунок 3.4 – Приклад роботи функції max-pooling

Але це не те, що робить AlexNet особливим; ось деякі особливості, які є новими підходами до згорткових нейронних мереж:

– Нелінійність ReLU. AlexNet використовує ReLU (3.2) замість функції tanh, яка була стандартною на той час. Перевага ReLU полягає у навчанні.

$$f(x) = x^+ = \max(0, x), \tag{3.2}$$

– Кілька графічних процесорів. В той час GPU все ще працювали з 3-ома гігабайтами пам’яті. Це було не дуже оптимально, оскільки навчальний набір мав 1.2 мільйона зображень. AlexNet дозволяє проводити навчання з багатьма GPU, тобто ділити кількість нейронів між кількома GPU. Таким чином можна навчити більшу модель, а також скоротити час навчання.
– З’єднання, що перекриваються. Згорткова нейронна мережа традиційно "об'єднує" виходи сусідніх груп нейронів без перекриття. Однак, коли автори ввели перекриття, вони побачили зменшення помилок приблизно на 0,5% і виявили, що моделям із об'єднанням, що перекриваються, взагалі важче переоцінити.
– Збільшення даних. Автори використовували збережені мітки, щоб зробити свої дані більш різноманітними. Зокрема, вони генерували трансляції зображень та горизонтальні відображення, що збільшувало навчальний набір

у 2048 разів. Вони також виконували принциповий аналіз компонентів на значеннях пікселів RGB, для зміни інтенсивності каналів RGB, що зменшувало помилку більше ніж на 1%.

– Dropout. Ця методика складається з "відключення" нейронів із заздалегідь визначеною ймовірністю (наприклад, 50%). Це означає, що кожна ітерація використовує різну вибірку параметрів моделі, що змушує кожен нейрон мати більш надійні функції, які можна використовувати з іншими випадковими нейронами. Однак такий принцип також збільшує час навчання, необхідне для конвергенції моделі [21].

3.3 Архітектура VGG

Мережі VGG першими застосували набагато менші 3 × 3 фільтри в кожному шарі згортки, а також поєднали їх як послідовність (рисунок 3.5).

Це, здається, суперечить принципам LeNet, де великі згортки використовувались для фіксації подібних ознак у зображенні. Замість фільтрів AlexNet 9 × 9 або 11 × 11 фільтри почали ставати меншими, занадто небезпечно близько до ядра згортки 1 × 1, яких LeNet хотів уникнути, принаймні, на перших шарах мережі. Але великою перевагою VGG було розуміння того, що множинне згортання 3 × 3 послідовно може імітувати ефект більших сприятливих полів, наприклад, 5 × 5 та 7 × 7. Ці ідеї будуть також використані в новітніх мережевих архітектурах як Inception і ResNet.

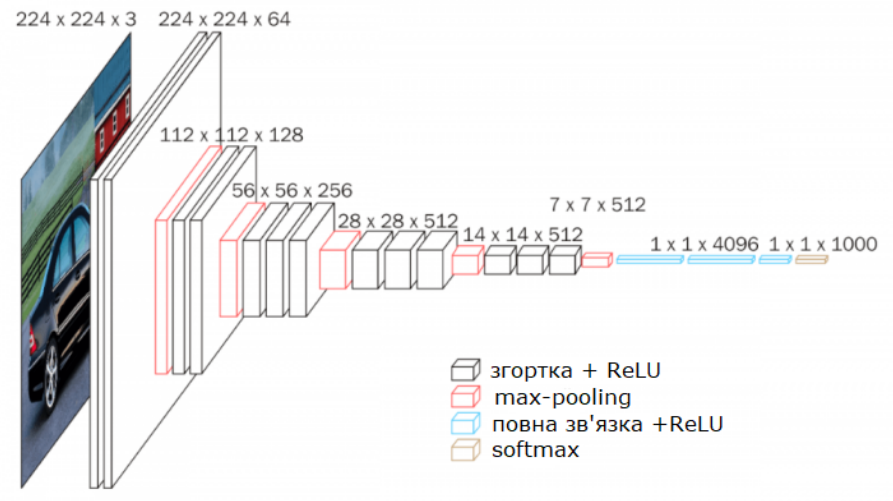


Рисунок 3.5 – Архітектура VGG

Вхідне зображення має фіксований розмір 224 x 224 RGB. Зображення передається через стек згорткових шарів, де використовуються фільтри з дуже невеликим ядром згортки: 3 × 3 (що є найменшим розміром для захоплення поняття лівий / правий, вгору / вниз, центр). Крок згортки фіксується на 1 піксель. Розглянути приклад такої згортки на рисунку 3.6.

Об'єднання мережі здійснюється п'ятьма шарами max-pooling, які слідують за деякими згортковими шарами. Max-pooling виконується над вікном 2 × 2 пікселів, з кроком 2.

Три повнозв'яні шари (FC) йдуть після згорткових (які мають різну глибину в різних архітектурах): перші два мають по 4096 каналів кожен, третій виконує 1000-разову класифікацій і містить 1000 каналів (по одному для кожного класу). Заключний шар - це застосування функції softmax. Конфігурація повнозв’язних шарів однакова у всіх мережах.

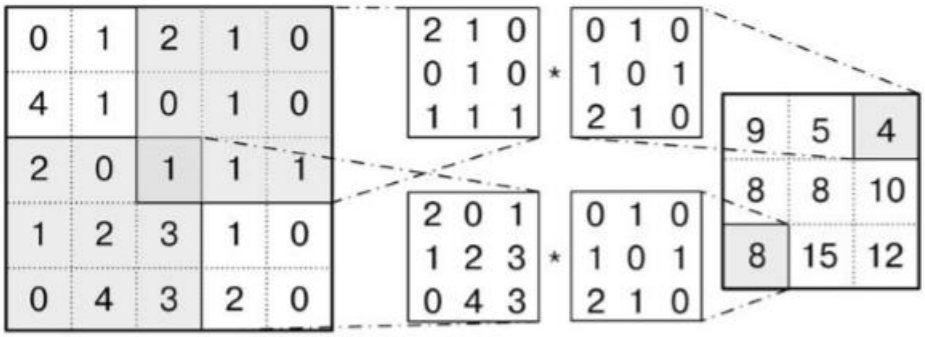


Рисунок 3.6 – Приклад згортки з ядром 3x3 та кроком 1 піксель

На жаль, у VGG є два основних недоліки:

- Дуже повільно тренується
- Ваги мережевої архітектури, зазвичай, досить великі

Завдяки глибині та кількості повністю підключених вузлів VGG перевищує 533 Мб. Це робить розгортання VGG стомлюючим завданням. VGG використовується в багатьох проблемах класифікації зображень; однак, менші мережеві архітектури часто є більш бажаними (наприклад, SqueezeNet, GoogLeNet тощо). Але це чудовий будівельний блок для навчальних цілей, оскільки його легко реалізувати [19].

3.4 Архітектура Network in network

Network in network (NiN) мала чудовий та простий погляд на використання 1x1 згортків для надання більшої комбінаційної потужності особливостям згорткових шарів (рисунок 3.5).

Архітектура NiN використовувала просторові шари прямого зв’язку(ПЗ) після кожного згортання, щоб краще поєднати функції перед іншим шаром. Знову ж можна подумати, що згортки 1x1 суперечать початковим принципам LeNet, але насправді вони допомагають краще поєднувати згорткові особливості, що

неможливо, просто укладаючи більше згорткових шарів. Це допомагає обмежити процес від використання необроблених пікселів та запобігти. Тут згортка 1x1 використовується для поєднання функцій на картах функцій після згортки, тому вони ефективно використовують дуже мало параметрів.

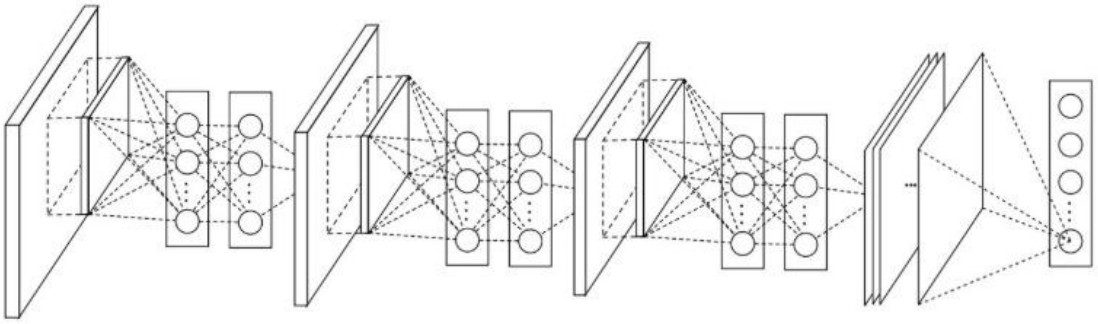


Рисунок 3.5 – Архітектура NiN

Розглянемо детальніше будову шарів з прямим зв’язком (рисунок 3.6)

$$f^1_{i,j,k_1} = \max(0, w^{1^T}_{k_1} x_{i,j} + b_{k_1}),$$

...

$$f^n_{i,j,k_n} = \max(0, w^{n^T}_{k_1} f^{n-1}_{i,j} + b_{k_n}), \quad (3.3)$$

де n - кількість шарів у багатошаровому перцептроні,

(i, j) - індекс пікселів на карті об’єктів,

$x_{i,j}$ означає патч введення, орієнтований у місці розташування (i, j),

k - індексації каналів карти функцій.

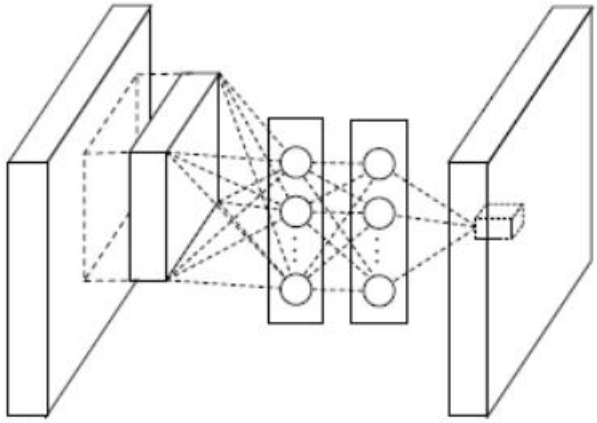


Рисунок 3.6 – Просторовий шар прямого зв’язку

Також потрібно приділити увагу останньому, але не значенням, шару зі звичайною лінійною згорткою (рисунок 3.7).

$$f_{i,j,k}^{1} = \max(0, w_k^T x_{i,j}), \tag{3.4}$$

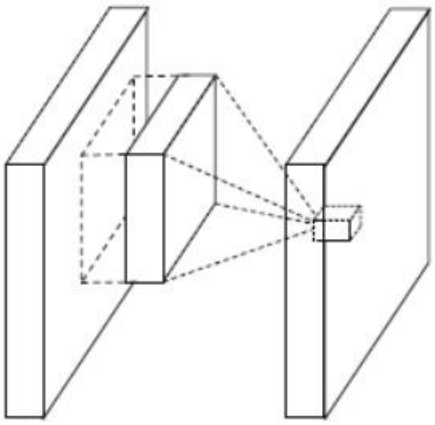


Рисунок 3.7 – Шар лінійної згортки

Потужність ПЗ може значно підвищити ефективність окремих ознак, об'єднавши їх у більш складні групи. Ця ідея буде згодом використана в більшості останніх архітектур як ResNet та Inception та похідних.

NiN також використовував середній шар об'єднання як частину останнього класифікатора - ще одна практика, яка стане загальною. Це було зроблено для усереднення мережі на множину вхідного зображення перед класифікацією [20].

3.5 Архітектури GoogLeNet та Inception

Незабаром Крістіан Сегеді від Google розпочав квест, спрямований на зменшення обчислювальних операцій глибоких нейронних мереж, і створив GoogLeNet першу архітектуру Inception. Він багато думав про способи зменшити навантаження мереж при отриманні найсучаснішого виконання (наприклад, на ImageNet). Або мати можливість зберегти однакову обчислювальну вартість, пропонуючи при цьому покращені показники.

Розглянемо частину GoogLeNet мережі:

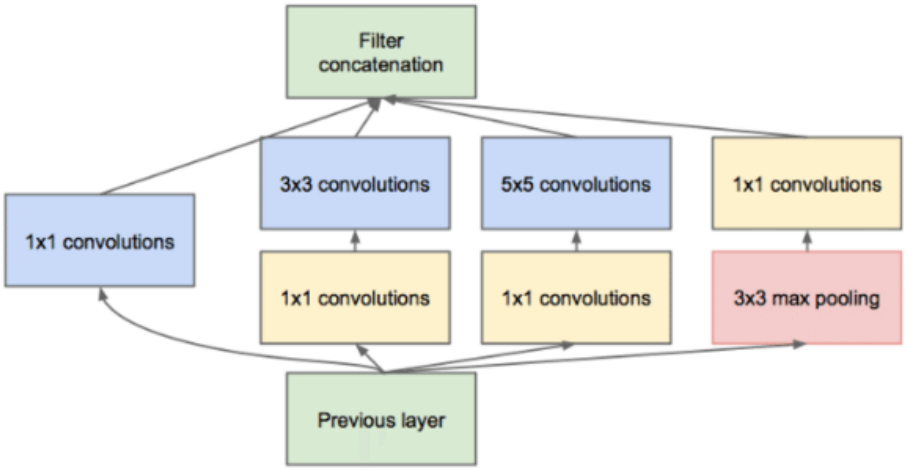
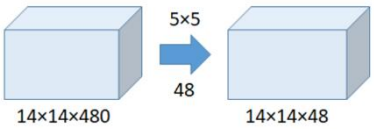
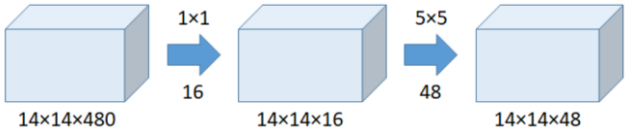


Рисунок 3.8 – Часткова архітектура GoogLeNet, модуль Inception

На перший погляд, це схоже на паралельне поєднання згорткових фільтрів 1x1, 3x3 та 5x5. Але об'єднання робляться загалом для попереднього входу та знову збираються разом на виході. Коли зображення надходить, застосовуються різні розміри згортків, та перевіряється, що є оптимальнішим, після йде шар max-pooling. Після цього всі карти функцій на різних контурах об'єднуються разом як вхід наступного модуля [18].

Але великим кроком стало використання 1x1 згорткових блоків (NiN) для зменшення кількості функцій перед розходженням мережі (паралельними блоками). Це зазвичай називають «вузьким місцем». Прослідкуємо різницю між кількістю операцій із згортковим ядром 1x1 та без у таблиці 3.2.

Таблиця 3.2 – Порівняння мереж зі згортковим ядром 1x1 та без

	Без згорткового ядра 1x1	Із згортковим ядром 1x1
Ілюстрація		
Кількість операцій	(14×14×48)×(5×5×480) = 112.9M	Number of operations for 1×1 = (14×14×16) × (1×1×480) = 1.5M Number of operations for 5×5 = (14×14×48) × (5×5×16) = 3.8M Total number of operations = 1.5M + 3.8M = 5.3M

Розглянемо повну архітектуру GoogLeNet на рисунку 3.9, та детальніше на рисунку 3.10.

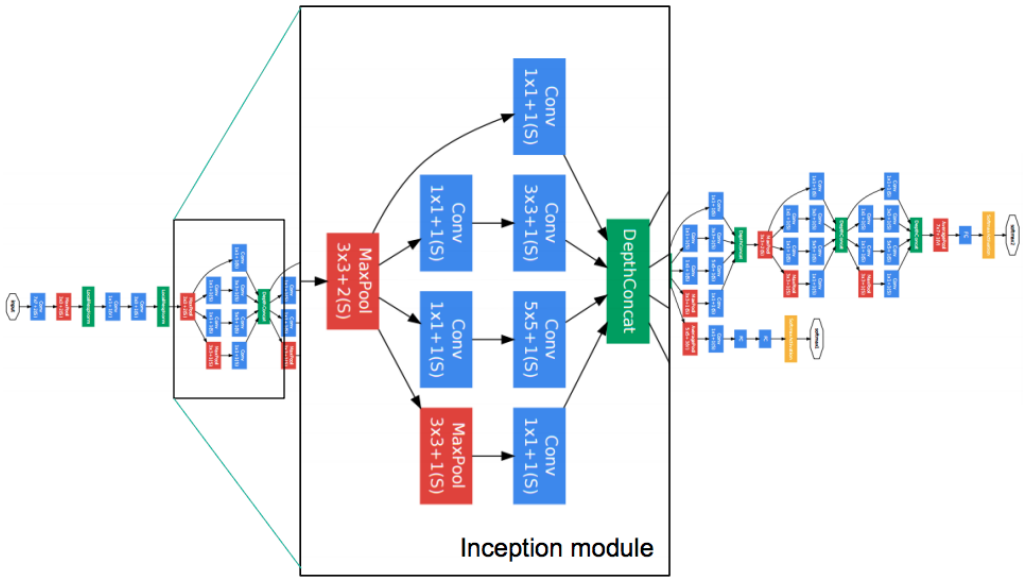


Рисунок 3.9 – Архітектура GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Рисунок 3.10 – Детальний опис кожного шару мережі GoogLeNet

3.6 Архітектура Inception V3(V2)

У лютому 2015 року Batch-normalized Inception було представлено як Inception V2. Пакетна нормалізація (Batch-Normalization) обчислює середнє та стандартне відхилення всіх функціональних карт на виході шару і нормалізує їх відповіді на ці значення. Це відповідає "відбілюванню" даних, і, таким чином, всі нейронні карти мають відповіді в одному діапазоні та з нульовим середнім. Такий підхід допомагає навчанню, оскільки наступному шару не потрібно вчити зміщення вхідних даних, а можна зосередитись на тому, як найкраще поєднувати функції.

Головні принципи мережі:

- максимізувати потік інформації в мережу, ретельно будуючи мережі, які врівноважують глибину та ширину. Перед кожним об'єднанням збільшуйте карти функцій.
- при збільшенні глибини кількість ознак або ширина шару також систематично збільшується
- використовувати збільшення ширини на кожному шарі, щоб збільшити поєднання функцій перед наступним шаром
- використовувати лише згортку 3x3, коли це можливо, враховуючи, що фільтр 5x5 та 7x7 може бути розкладений з декількома 3x3
- модель “Inception” модернізована (рисунок 3.11)
- фільтри також можуть розкладатися за допомогою сплющених згортків на більш складні модулі (рисунок 3.12)

Inception як і раніше використовує об'єднаний шар та softmax як кінцевий класифікатор.

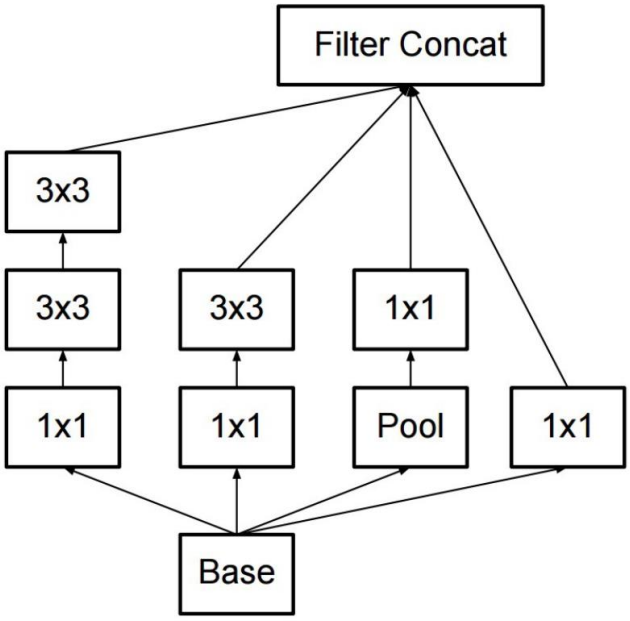


Рисунок 3.11 – Модель Inception для мережі Inception V3

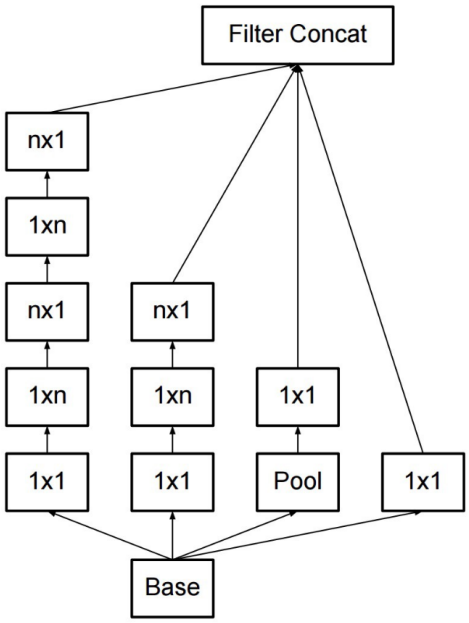


Рисунок 3.12 – Способи розкладу різних фільтрів

3.7 Архітектура ResNet

Ідеї ResNet полягають в наступному:

- подавати на вихід два послідовних згорткових шари
- обхід 2-х згорткових шарів

Для кращого розуміння, розглянемо рисунок 3.13

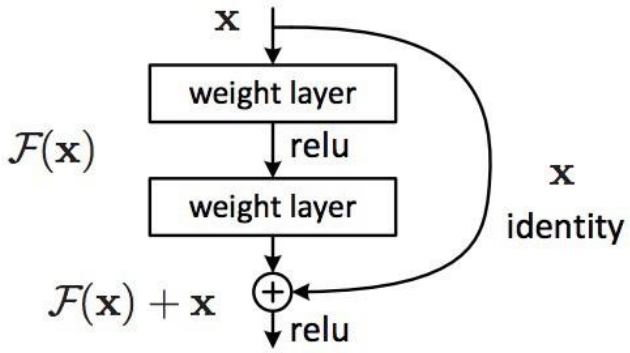


Рисунок 3.13 – Особливість ResNet

Такий підхід здається вже знайомим, але головна особливість у обході двух шарів і використанні при великому об'ємі даних. Обхід двох шарів - ключова інтуїція, оскільки обхід одного не дав особливих поліпшень. На 2-х шарах можна розглянути невеликий класифікатор або Network in network!

ResNet з великою кількістю шарів почав використовувати шар схожий на «вузьке місце», як у модуля Inception (рисунок 3.14).

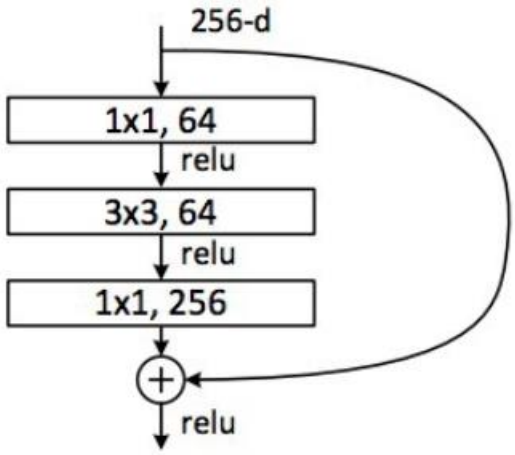


Рисунок 3.14 – Шар «вузького місця» для ResNet

Цей шар зменшує кількість ознак кожного шару, спочатку використовуючи згортку 1x1 з меншим виходом (зазвичай 1/4 вхідного), а потім 3x3 шаром, а потім знову згортання 1x1 на більшу кількість функцій. Як і у випадку з модулями «Inception», це дозволяє тримати обчислення низькими, забезпечуючи при цьому багате поєднання функцій.

Принципи мережі ResNet:

- використовувати досить прості початкові шари на вході
- використовувати шар об'єднання плюс softmax як кінцевий класифікатор
- розглядати як паралельний, так і послідовні модулі. Тобто мережа думає про вхід як про перехід на багато паралельних модулів, тоді як виходи кожного з модулів з'єднані послідовно
- розглядати безліч ансамблів паралельних або послідовних модулів
- працювати на відносно не глибоких блоках ~ 20–30 шарів, які діють паралельно

3.8 Архітектура Xception

Xception вдосконалив модель за допомогою простої та більш елегантної архітектури, настільки ж ефективної, як ResNet та Inception V4.

Розглянемо модель на рисунку:

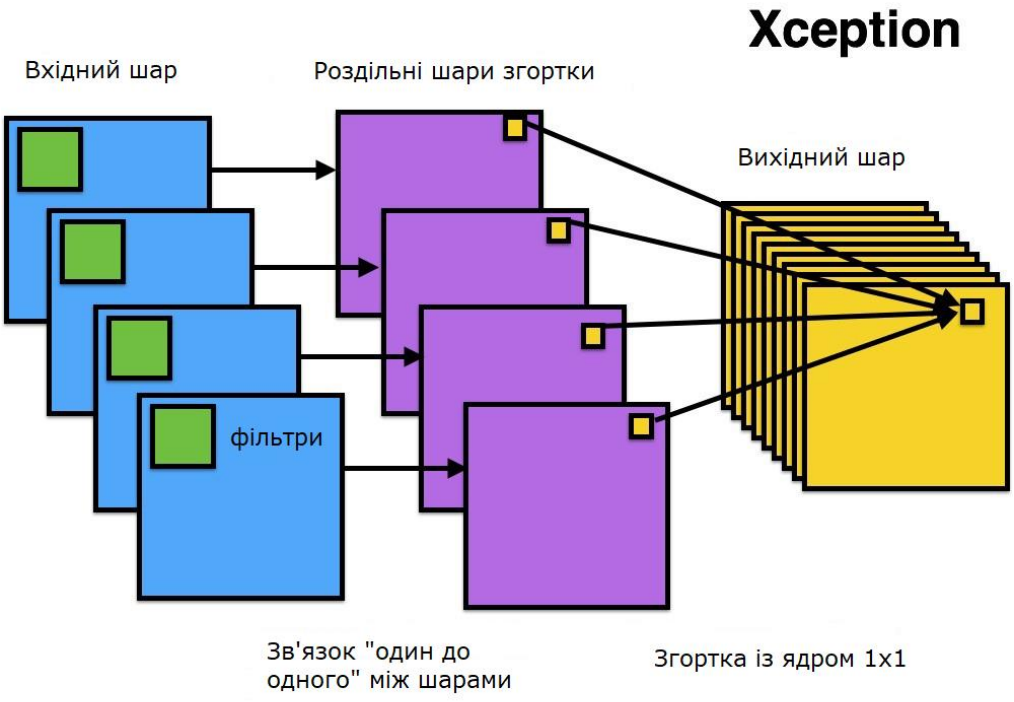


Рисунок 3.15 – Архітектура Xception

Більш детальну архітектуру розглянемо на схемі (рисунок 3.16).

Архітектура має 36 згорткові етапи, що робить її близькою за схожістю з ResNet-34. Але модель і код настільки ж прості, як ResNet і набагато зрозуміліші, ніж будь-яка модифікація Inception.

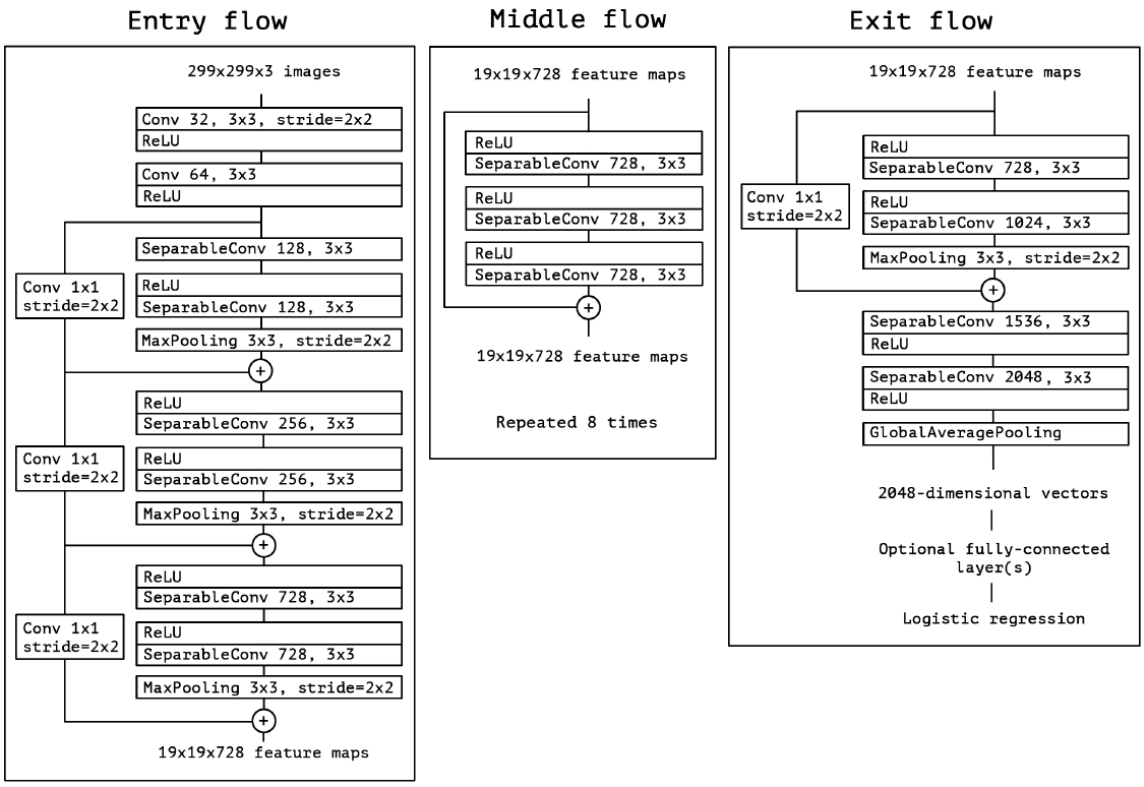


Рисунок 3.16 – Схема моделі Xception

3.9 Архітектура MobileNet

Нова архітектура MobileNet доступна з квітня 2017 року. Ця архітектура використовує окремі згортки для зменшення кількості параметрів. Спостерігається значне зниження параметрів - приблизно 1/2 на FaceNet (набір даних). Повна архітектура моделі зображена на рисунку 3.17.

На жаль, при перевірці цієї мережі в практиці, було помічено, що вона дуже повільна. Порівняння часу роботи розглянемо у наступній таблиці:

Таблиця 3.3 – Порівняння нейронних мереж та часу їх роботи

Мережа	ResNet	AlexNet	VGG	MobileNet
Час (с)	0,002871	0,001003	0,001698	0,033251

Зрозуміло, що MobileNet може зменшити параметри та розмір мережі на диску, але не є швидким. Тепер порівняймо архітектури та точність передбачення, наприклад, перший результат та п’ятий.

Таблиця 3.4 – Порівняння архітектури мережи та точності передбачення

Мережа	Кількість параметрів	Перший результат передбачення (Топ-1)	Топ-5
Xception	22.91M	0.79	0.945
VGG	138.35M	0.715	0.901
MobileNetV1	2.59M	0.672	0.873
MobileNetV2	6.06M	0.750	0.925

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Рисунок 3.17 – Архітектура MobileNet (зліва направо)

3.4 Висновок до розділу 3

Отже, можна помітити, що із плином часу архітектури змінювались та вдосконалювалися, даючи результати кращі, аніж попередники. Маючи представлення про кожну з сучасних архітектур, можна використати найкращі з

них для нашої задачі. Таким чином, ми знайдемо модель, що допоможе найшвидше та найточніше передбачити хворобу.

Для нашої задачі, з конкретними параметрами виберемо одну з найточніших архітектур – MobileNet, а також спробуємо навчати мережу, із структурою як у VGG16. Крім того, спробуємо створити самостійну архітектуру, опираючись на всі попередні данні та дослідження.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ЗА ДОПОМОГОЮ МОДЕЛЕЙ НЕЙРОННИХ МЕРЕЖ

4.1 Загальний огляд програмного продукту

В процесі роботи було розроблено та реалізовано програмний продукт, який допомагає передбачити таку хворобу, як рак шкіри, а також виявити її тип.

Для написання програмного продукту було обрано професійне середовище програмування JupyterLab. JupyterLab - це інтерактивне середовище розробки для роботи з файлами розширення .py та .ipynb (Jupyter-блокноти), кодом і даними. Головні особливості JupyterLab:

- Використовується метод drag-and-drop для того щоб можна було змінювати осередки і копіювати їх між блокнотами
- Високий рівень інтеграції між файлами з різним розширенням
- Блоки коду виконуються інтерактивно прямо з текстових файлів (.py, .R, .md, .tex і т. Д.)
- Консоль коду можна пов'язати з ядром блокнота, щоб вивчати код в інтерактивному режимі, не захаращуючи блокнот тимчасовими правками
- Можливість редагування популярних форматів файлів з попереднім переглядом в реальному часі, такі як Markdown, JSON, CSV, Vega, VegaLite і інші

Програмний продукт розроблявся на мові програмування Python, тому може бути запущений на будь-якій операційній системі, що є дуже великим плюсом для усіх користувачів. Потрібно лише завантажити середовище та мову програмування на свій ПК.

4.2 Інтерфейс програми

Розглянемо основі функції нашої програми. Головне вікно містить кнопку для запису ПП. Користувач має натиснути клавішу «Restart and run all», ви можете побачити її розташування в інтерфейсі на рисунку 4.1.

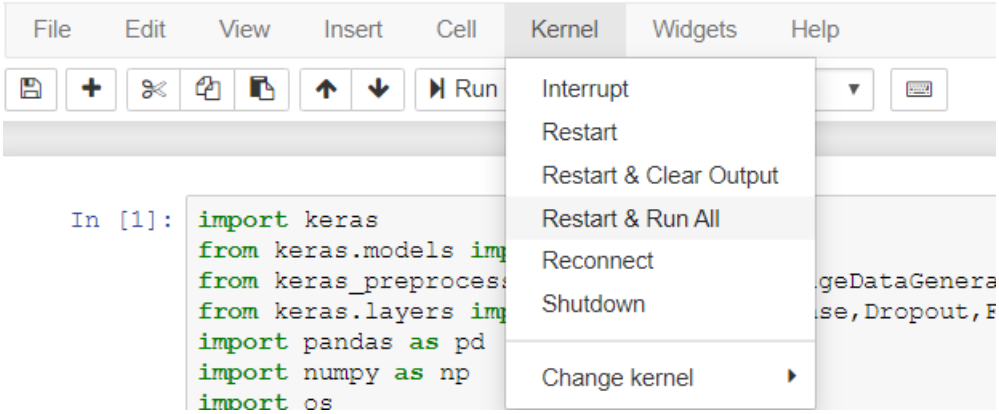


Рисунок 4.1 – Запуск програми

Надалі програма буде виконувати усі дії самостійно, та не буде потребувати додаткового втручання. Розглянемо детальніше, що саме передбачає даний програмний продукт.

Отже, програма завантажує усі потрібні файли та шляхи до набору даних (зображень), для використання їх у подальшому розв'язку задачі. Першим результатом аналізу є діаграма, що показує кількість зображень кожної хвороби у вибірці для навчання нейронної мережі (рисунок 4.2).

Далі програма виконує певні перетворення для роботи із даними та самостійно генерує зображення формату .png, яке показує приклади даних усіх класів, що використовуються для навчання (рисунок 4.3).

Наступний крок – нормалізація даних про кольорові канали кожного зображення. ПП передбачає аналіз зображення таким чином, що можна побачити який кольоровий канал є ведучим у кожному класі (рисунок 4.4).

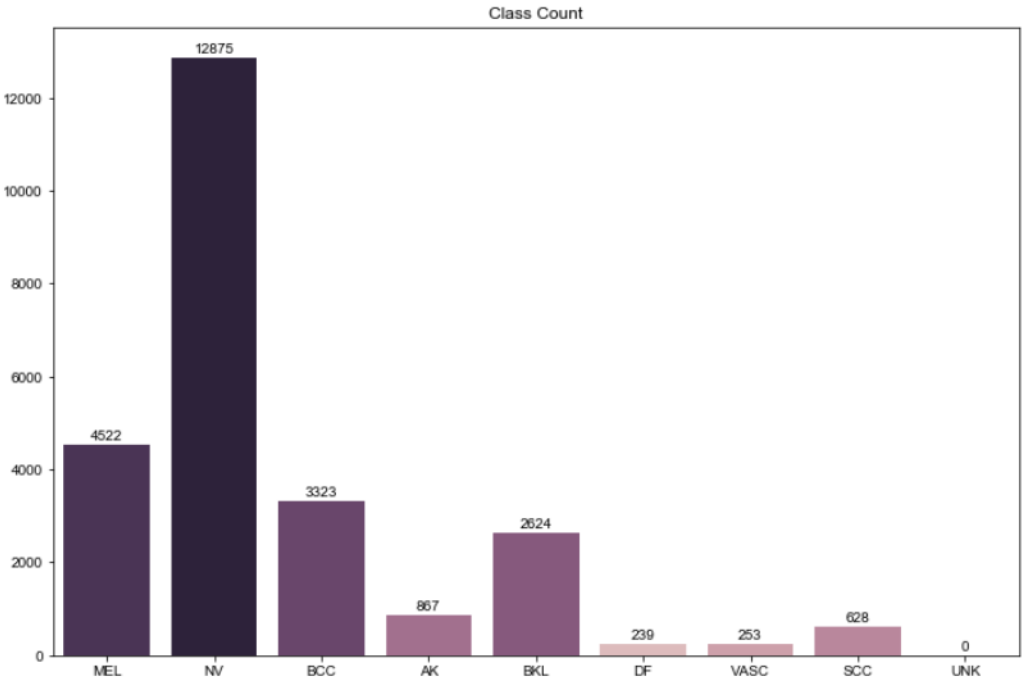


Рисунок 4.2 – Діаграма кількості зображень

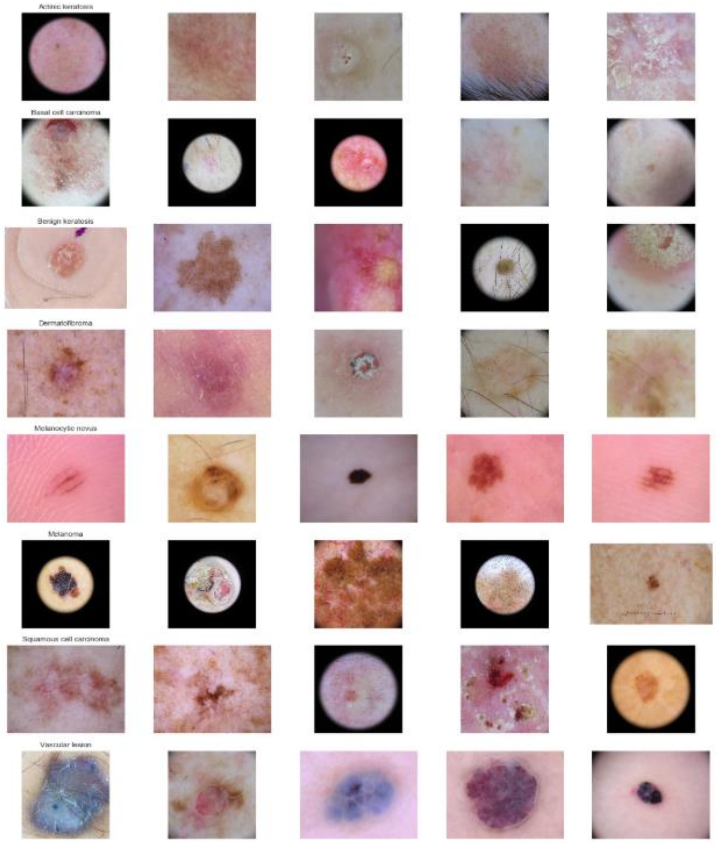


Рисунок 4.3 – Приклади зображень кожного класу

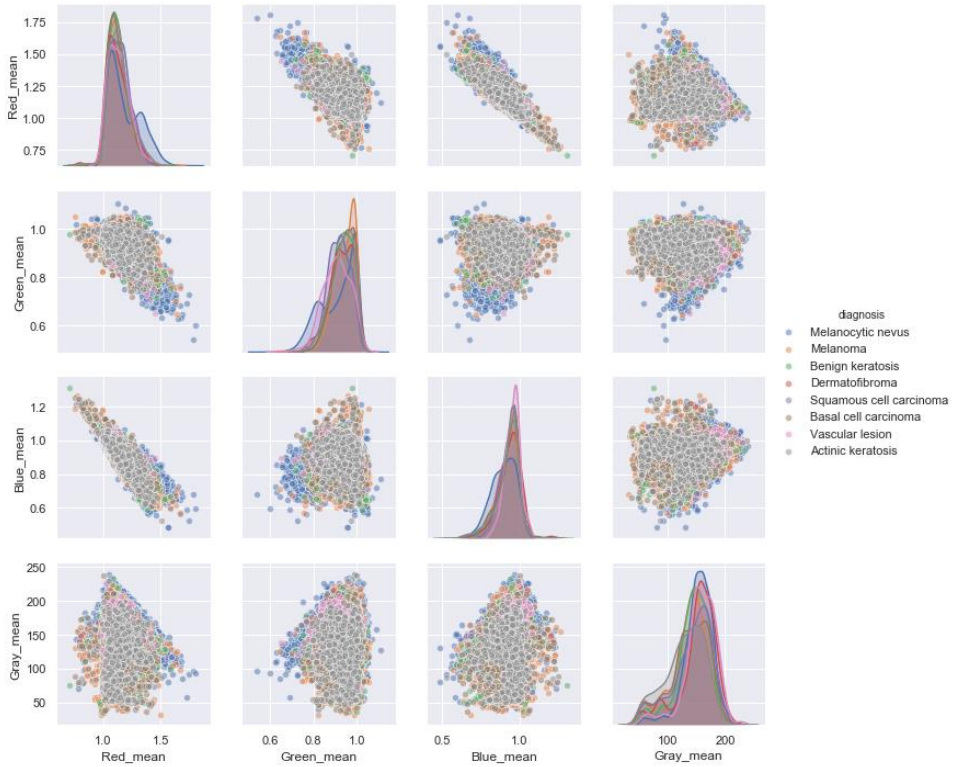


Рисунок 4.4 – Залежність кольорових каналів та класів

Після підготовки усіх даних, програма готує вибірку даних та нейронна мережа починає навчання (рисунок 4.5). При цьому, автоматично завантажує схему нейронної мережі у форматі .png (рисунок 4.6).

```
Epoch 1/10
197/197 [==============================] - 5522s 28s/step - loss: 0.2413 - accuracy: 0.9066 - val_loss: 0.3353 - v
al_accuracy: 0.8601

Epoch 00001: val_accuracy improved from -inf to 0.86010, saving model to Models/multi_output_model_7_weights-01-0.
860.hdf5
Epoch 2/10
197/197 [==============================] - 5063s 26s/step - loss: 0.2212 - accuracy: 0.9137 - val_loss: 0.2884 - v
al_accuracy: 0.8788

Epoch 00002: val_accuracy improved from 0.86010 to 0.87881, saving model to Models/multi_output_model_7_weights-02
-0.879.hdf5
Epoch 3/10
197/197 [==============================] - 5046s 26s/step - loss: 0.2153 - accuracy: 0.9161 - val_loss: 0.2768 - v
al_accuracy: 0.8974

Epoch 00003: val_accuracy improved from 0.87881 to 0.89738, saving model to Models/multi_output_model_7_weights-03
-0.897.hdf5
Epoch 4/10
```

Рисунок 4.5 – Навчання нейронної мережі

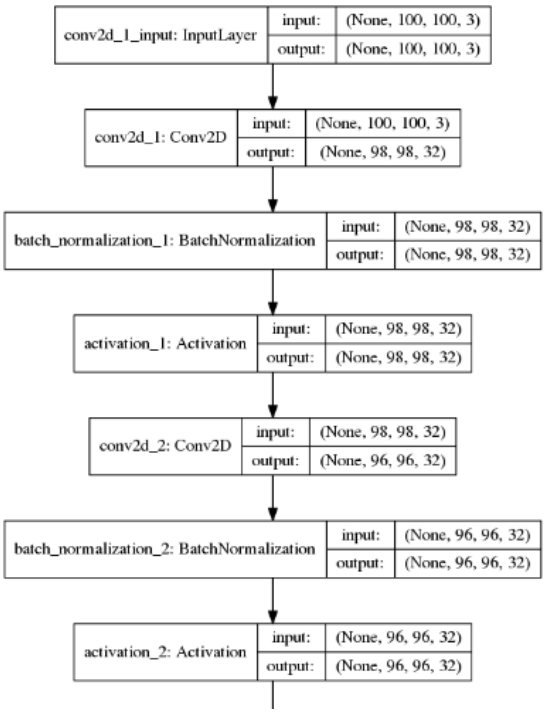


Рисунок 4.6 – Часткова модель нейронної мережі

Наприкінці роботи, програма показує процес усього навчання у вигляді графіку зміни точності передбачення та значення помилки (рисунок 4.7).

Рисунок 4.7 – Графік зміни точності та значення помилки

У кінці програма видає результат у текстовому форматі (рисунок 4.8)

Рисунок 4.8 – Результат роботи програми

4.3 Висновки до розділу 4

В даний час трудомістку роботу, пов’язану с обчисленнями, побудовою графіків, діаграм з легкістю може виконувати комп’ютер. Саме тому зараз вирішити будь-яку задачу набагато легше та займає менше часу, ніж кілька п’ятиліть тому. Але головним завданням, з яким машина досі не може впоратися: постановка задачі, реалізація оптимальних алгоритмів самостійно та правильна інтерпретація результатів.

В процесі роботи для проведення необхідних досліджень було створено власний програмний продукт, який дозволяє передбачити появлення хвороби раку шкіри, а також з’ясувати, до якого підвиду відноситься дане захворювання. Окрім того, програма дає можливість прослідкувати різні залежності між зображеннями та видами діагнозу.

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1 Постановка задачі проектування

Спроектувати програмний продукт для аналізу зображення шкірної ділянки користувача для діагностики хвороби раку шкіри. Він призначений для використання в середовищах що підтримують технології Python або JupyterNotebook в залежності від вибору розробника-реалізатора прототипу.

5.1.1 Обґрунтування функцій та параметрів програмного продукту

Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – завантаження зображення: а) завантаження зображення за допомогою апарату дерматоскопії, б) завантаження зображення, зробленого звичайною камерою.

F_2 – сегментація зображення: а) спектральний аналіз, б) кореляційний аналіз.

F_3 – обробка сегментів: а) нейронна мережа, б) ручна обробка.

F_4 – отримання результатів діагностики за допомогою інтерфейсу: а) визначення наявності хвороби, б) визначення групи хвороби, якщо вона присутня.

5.1.2 Варіанти реалізації основних функцій

Виходячи з представлених варіантів будуємо морфологічну карту (рис.5.1).

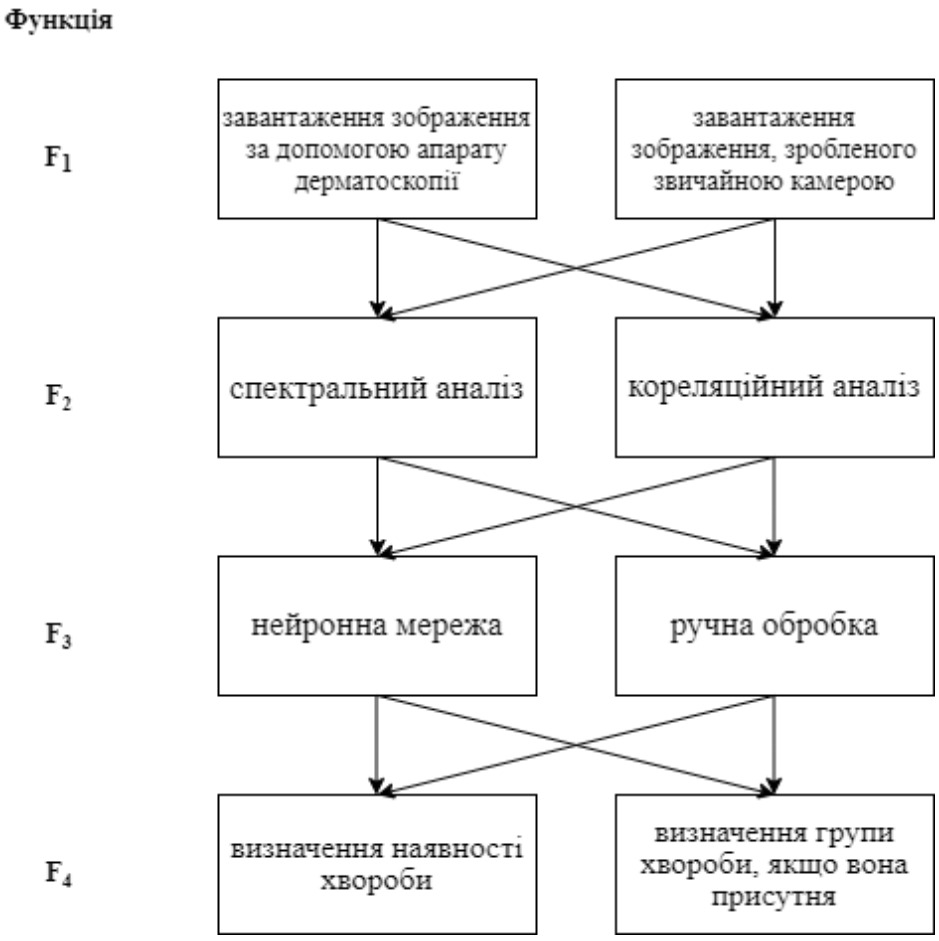


Рисунок 5.1 – Морфологічна карта

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Висока роздільна здатність зображення	Потребує часу та додаткових затрат
	Б	Не потребує окремого апарату	Низька якість зображення
F_2	А	Якісна обробка даних та легкість реалізації	Зображення можуть бути схожими та містити однакові кольорові канали

Кінець таблиці 5.1

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_2	Б	Висока точність аналізу	Велика кількість обчислень та чималі затрати пам’яті
F_3	А	Точність результатів	Потреба у навчанні
	Б	Мала вірогідність помилки	Дуже великі затрати часу
F_4	А	Легкість в реалізації	Мала кількість інформації
	Б	Надання додаткової інформації	Не завжди точний діагноз

5.2 Обґрунтування системи параметрів ПП

5.2.1 Опис параметрів

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X_1 – час написання коду та реалізації усіх потрібних алгоритмів;

X_2 – об’єм пам’яті для збереження та обробки даних;

X_3 – час обробки даних та роботи програмного коду;

X_4 – точність розв’язку.

5.2.2 Кількісна оцінка параметрів

Значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.2. За даними таблиці 5.2 будуються графічні характеристики параметрів – рисунки 5.2 - 5.5.

Таблиця 5.2 – Система параметрів додатку

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметру		
			Гірше	Середнє	Краще
Час написання коду та реалізації усіх потрібних алгоритмів	X_1	Год	12	8	5
Об’єм пам’яті для збереження та обробки даних	X_2	Гб	18	14	10
Час обробки даних та роботи програмного коду	X_3	Год	10	6	2
Точність розв’язку	X_4	%	52	86	92

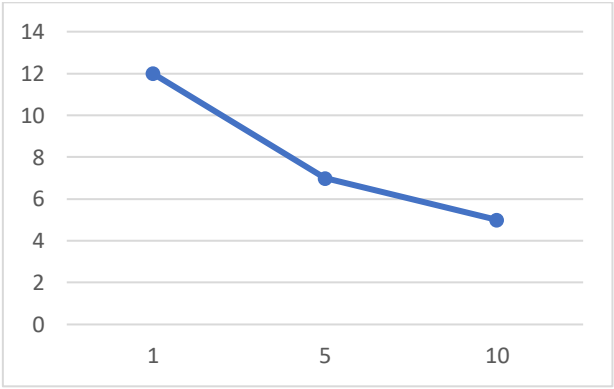


Рисунок 5.2 – X_1, час написання програмного коду

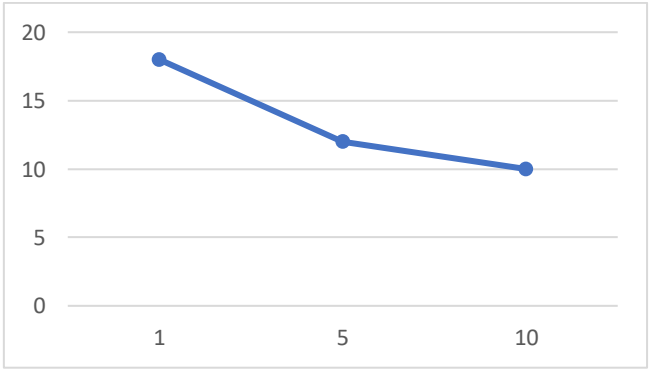


Рисунок 5.3 – X_2, об’єм пам’яті

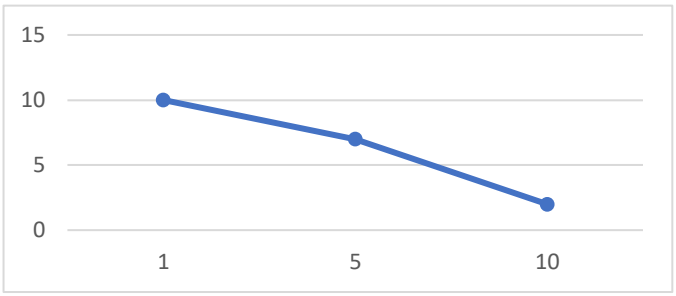


Рисунок 5.4 – X_3, час обробки даних

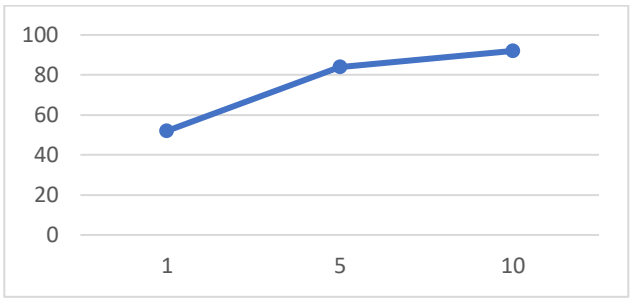


Рисунок 5.5 – X_4, точність розв'язку

5.2.3 Аналіз експертного оцінювання параметрів

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Нехай 1-ий ранг означає те, що параметр не є доволі важливим для вирішення даної задачі, тоді як 4-ий ранг означає, що даний параметр є значущим.

Таблиця 5.3 – Результат оцінки параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	Відхилення, Δ_i	Квадрат відхилення, $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X_1	1	1	2	1	2	1	1	9	-8.5	72.25
X_2	2	2	1	2	1	2	3	13	-4.5	20.25
X_3	3	4	3	3	4	3	2	22	4.5	20.25
X_4	4	3	4	4	3	4	4	26	8.5	72.25
Разом	10	10	10	10	10	10	10	70	0	185

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 185}{7^2(4^3 - 4)} = 0.75 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Проведемо попарне порівняння всіх параметрів.

Таблиця 5.4 – Попарне зрівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 та X_2	<	<	>	<	>	<	<	<	0.5
X_1 та X_3	<	<	<	<	<	<	<	<	0.5
X_1 та X_4	<	<	<	<	<	<	<	<	0.5
X_2 та X_3	<	<	<	<	<	>	>	<	0.5
X_2 та X_4	<	<	<	<	<	<	<	<	0.5
X_3 та X_4	<	>	<	<	>	<	<	<	0.5

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{вi} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

де $b_i = \sum_{i=1}^{N} a_{ij}$.

На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{вi} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри X_i	Параметри X_j				Ітерація №1		Ітерація №2		Ітерація №3	
	X_1	X_2	X_3	X_4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X_1	1.0	0.5	0.5	0.5	2.5	0.16	9.25	0.156	34.125	0.157
X_2	1.5	1.0	0.5	0.5	3.5	0.22	12.25	0.2	44.875	0.2
X_3	1.5	1.5	1.0	0.5	4.5	0.28	16.25	0.275	59.175	0.274
X_4	1.5	1.5	1.5	1.0	5.5	0.34	21.25	0.36	77.875	0.36
Загалом:					16	1	59	1	216	1

5.3 Аналіз рівня якості варіантів реалізації функцій

Враховуючи дані з порівнянь варіантів будемо розглядати такі варіанти використання ПП:

1. $F_1а - F_2а - F_3а - F_4а$
2. $F_1а - F_2а - F_3а - F_4б$

Таблиця 5.6 – Розрахунок показників рівня якості заданих варіантів

Основні функції	Варіант реалізації функції	Параметр	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А	X_1	8	4.2	0.157	0.6594
F_2	А	X_2	14	4.1	0.2	0.82
F_3	А	X_3	6	6.1	0,274	1.6714
F_4	А	X_4	86	7.1	0.36	2.556
	Б	X_4	92	9.4	0.36	3.384

Отже, визначимо рівень якості кожного з варіантів

$К_{К1}$ = 0.6594 + 0.82 + 1.6714 + 2.556 = 5.7014;

$К_{К2}$ = 0.6594 + 0.82 + 1.6714 + 3.384 = 6.5294.

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Реалізація алгоритму створення програмного продукту;
2. Написання програмного коду, з урахуванням затверджених методів та алгоритмів у попередньому пункті;

В свою чергу, варіант II місить завдання:

3. Реалізація алгоритму, що виводить результат з графічною ілюстрацією та розгорнутою інформацією.

Крім того, варіант I містить завдання:

4. Реалізація алгоритму, виводу тексту на екран програмного продукту

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б, завдання 3, 4 – до групи В. За складністю алгоритми, які використовуються в завданні 1, 2 належать до групи 1; завдання 3 – до групи 3, а завдання 4 - до групи 2.

Для першого завдання (степінь новизни А, група складності – 1), тобто $Т_Р$=90 людино-днів, $К_П$ = 1.7, $К_{СК}$ = 1; $К_{СТ}$ = 0.8.

Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90·1.7·1·0.8 = 122.4 людино-днів.

Для другого завдання (степінь новизни Б, група складності – 1), тобто $T_Р$=64 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1; $К_{СТ}$ =0.6:

$T_2 = 64 \cdot 0.9 \cdot 1 \cdot 0.6 = 34.56$ людино-днів.

Для третього завдання (степінь новизни В, група складності – 3), тобто $T_Р$=12 людино-днів, $К_П$ = 0.8; $К_{СК}$ = 1; $К_{СТ}$ = 0.6:

$T_3 = 12 \cdot 0.8 \cdot 1 \cdot 0.6 = 5.76$ людино-днів.

Для четвертого завдання (степінь новизни В, група складності – 2), тобто $T_Р$=19 людино-днів, $К_П$ = 1.2; $К_{СК}$ = 1; $К_{СТ}$ = 0.6:

$T_4 = 19 \cdot 0.72 \cdot 1 \cdot 0.8 = 10.94$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$T_I = (122.4 + 34.56 + 5.76) \cdot 6 = 976{,}32$ людино-годин;

$T_{II} = (122.4 + 34.56 + 10.94) \cdot 6 = 1007{,}4$ людино-годин.

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 14000 грн., один фінансовий аналітик з окладом 12800грн. Визначимо зарплату за годину:

$$С_Ч = \frac{14000 + 14000 + 12800}{3 \cdot 21 \cdot 8} = 80{,}95 \text{ грн.}$$

Тоді, заробітну плата розробників за варіантами становить:

I. $С_{ЗП} = 80{,}95 \cdot 976.32 \cdot 1.2 = 94839.72$ грн.;

II. $С_{ЗП} = 80{,}95 \cdot 1007.4 \cdot 1.2 = 97858.83$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 94839.72 \cdot 0.22 = 20865$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 97858.83 \cdot 0.22 = 21528$ грн.

Тепер, визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 14000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 14000 \cdot 0{,}2 = 33600$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 33600 \cdot (1 + 0.2) = 40320$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 40320 \cdot 0,22 = 8870,4$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 13000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 13000 = 3737,5$ грн.,

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 13000 \cdot 0.05 = 747,5$ грн.,

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 142 – 16) \cdot 8 \cdot 0.9 = 1324.8$ год.,

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1324,8 \cdot 0,156 \cdot 0,61 \cdot 1,53 = 192,88$ грн.,

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 13000 \cdot 0,67 = 8710$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = 40320 + 8870,4 + 3737,5 + 747,5 + 192,88 + 8710 = 62578,3$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 62578,3 / 1324,8 = 47,2$ грн/час.

Отже, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

I. $С_М = 47.2 \cdot 976.32 = 46117.5$ грн.;

II. $С_М = 47.2 \cdot 1007.4 = 47549.3$ грн.

Накладні витрати складають 67% від заробітної плати:

I. $С_Н = 94839.72 \cdot 0,67 = 63542.6$ грн.;

II. $С_Н = 97858.83 \cdot 0,67 = 65565.4$ грн.

Отже, вартість розробки ПП за варіантами становить:

I. $С_{ПП}$ = 94839.72 + 20865 + 46117.5 + 63542.6 = 225364.82 грн.;

II. $С_{ПП}$ = 97858.83 + 21528 + 47549.3 + 65565.4 = 232501.53 грн.

5.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$К_{ТЕРj} = К_{Кj} / С_{Фj}$,

$К_{ТЕР1}$ = 5,6814 / 225364.82 = $2.5 \cdot 10^{-5}$;

$К_{ТЕР2}$ = 6,5094 / 232501.53 = $2,8 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2}$= $2,8 \cdot 10^{-5}$.

5.6 Висновки до розділу 5

Після виконання функціонально-вартісного аналізу програмного комплексу, що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості:

$К_{ТЕР}$ = $2,11 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– завантаження зображення за допомогою апарату дерматоскопії;

– спектральний аналіз для сегментації зображення;

– використання нейронної мережі для обробки даних;

– визначення групи хвороби, якщо вона присутня.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, хороший функціонал і швидкодію.

ВИСНОВКИ ДО РОБОТИ

У ході виконання даної дипломної роботи, були досліджені різноманітні методи визначення раку шкіри. Зупинившись на алгоритмі, що використовує нейронні мережі, було повністю досліджено головні аспекти даного способу та проаналізовані найкращі підходи до навчання. Крім того, була проведена порівняльна характеристика прийомів побудови моделі нейронних мереж.

З даного дослідження, можна зробити висновок, що не завжди мережа, яка показала гідний результат на конкретних навчальних даних, зможе показати такий же на будь-яких. Тобто, мережа, що тренувалася кластеризувати машини, або виявляти громіздкі предмети на зображенні, не дасть потрібного результату, при класифікації хвороби раку шкіри.

Саме тому, інколи, створення власної нейронної мережі є оптимальним варіантом для вирішення конкретної задачі. Такий підхід дозволяє врахувати усі деталі завдання самостійно та корегувати архітектуру, для поліпшення результатів.

У результаті дослідження був розроблений програмний продукт, що розпізнає хворобу раку шкіри та визнає її підвид. При завантаженні зображення, зробленого за допомогою дерматоскопії, програма із точністю 92% ставить узагальнений діагноз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Yuexiang L. Skin Lesion Analysis towards Melanoma Detection Using Deep Learning Network / L. Yuexiang, S. Linlin. // Cornell University. – 2017. – №1. – С. 1–3, 6.

2. Abhishek K. Illumination-based Transformations Improve Skin Lesion Segmentation in Dermoscopic Images / K. Abhishek, G. Hamarneh, M. Drew. // School of Computing Science, Simon Fraser University, Canada. – 2020. – №1. – С. 1–3.

3. Gómez D. Independent Histogram Pursuit for Segmentation of Skin Lesions / D. D.Gómez, C. Butakoff, B. K. Ersbøll, W. Stoecker. // IEEE. – 2011. – №2.

4. Yu L. Automated Melanoma Recognition in Dermoscopy Images via Very Deep Residual Networks /L. Yu, H. Chen, Q. Dou, J. Qin, and P. A. Heng. // IEEE Trans. Med. Imaging. – 2016 – №1.

5. Stanley R. J. A relative color approach to color discrimination for malignant melanoma detection in dermoscopy images/ R. J. Stanley, W. V. Stoecker, R. H. Moss. // Skin Research & Technology. – 2007. – №1. – С. 62-72.

6. Ballerini L. A Color and Texture Based Hierarchical K-NN Approach to the Classification of Non-melanoma Skin Lesions/ L. Ballerini, R. B. Fisher, B. Aldridge, J. Rees. // Springer. – 2013 – №1.

7. Николенко С. Глубокое обучение. Погружение в мир нейронных сетей / С. Николенко, А. Кадурин, Е. Архангельская. – Санкт-Петербург: "Питер", 2018. – 480 с.

8. Bailer-Jones C. An introduction to artificial neural networks / C. Bailer-Jones, R. Gupta, H. Singh. // Cornell University. – 2001 – №2.

9. Созыкин А. В. Обзор методов обучения глубоких нейронных сетей / А. В. Созыкин. // Институт математики и механики им. Н.Н.Красовского. – 2017. – №1. – С. 29–32.

10. Осовский С. Нейронные сети для обработки информации / Станислав Осовский: [пер. с польского И.Д. Рудинский]. – М.: Финансы и статистика, 2002. – 344 с. – (Финансы и статистика).

11. The 8 Neural Network Architectures Machine Learning Researchers Need to Learn [Електронний ресурс] / James Le // KDnuggets – 2018. – Режим доступу до ресурсу: https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html.

12. Разнообразие нейронных сетей. Часть вторая. Продвинутые конфигурации [Електронний ресурс]/ И. Бирюков // Tproger – 2016. – Режим доступу до ресурсу: https://tproger.ru/translations/neural-network-zoo-2/.

13. Шпаргалка по разновидностям нейронных сетей. Часть первая. Элементарные конфигурации [Електронний ресурс] / И. Бирюков // Tproger – 2016. – Режим доступу до ресурсу: https://tproger.ru/translations/neural-network-zoo-1/.

14. Neural Network Architectures [Електронний ресурс] / E. Culurciello // Towards Data Science – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba.

15. LeNet-5 - A Classic CNN Architecture [Електронний ресурс] // Data Science Central – Режим доступу до ресурсу: https://mobilemonitoringsolutions.com/lenet-5%E2%80%8A-%E2%80%8Aa-classic-cnn-architecture/.

16. Simonyan K. Very deep convolutional networks for large-scale image recognition/ K. Simonyan, A. Zisserman. // Visual Geometry Group, Department of Engineering Science, University of Oxford. – 2015. – С. 2,3.

17. #013 В CNN AlexNet [Електронний ресурс] // Data Hacker. – 2018. – Режим доступу до ресурсу: http://datahacker.rs/deep-learning-alexnet-architecture/.

18. Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification) [Електронний ресурс] // Coinmonks – 2018. – Режим доступу до

ресурсу: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7.

19. VGG16 – Convolutional Network for Classification and Detection [Електронний ресурс] // Neurohive – 2018. – Режим доступу до ресурсу: https://neurohive.io/en/popular-networks/vgg16/.

20. Review: NIN — Network In Network (Image Classification) [Електронний ресурс] // Towards Data Science – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/review-nin-network-in-network-image-classification-69e271e499ee.

21. AlexNet: The Architecture that Challenged CNNs [Електронний ресурс] / J. Wei // Towards Data Science – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

22. MobileNet: меньше, быстрее, точнее [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/352804/.

23. Создаем нейронную сеть / Тарик Рашид : [пер. с англ. — СПб.: ООО “Альфа-книга”]. — Санкт-Петербург: “Альфа-книга”, 2017. — 272 с.

24. Суворова Е. Ю. Анализ основных технологий вычислительного интеллекта/ Е. Ю. Суворова. // ЛНУ имени Тараса Шевченка. – №20.

25. Згуровський М. З. Основы вычислительного интеллекта/ М. З. Згуровський, Ю. П. Зайченко. – Киев: Наукова думка, 2013. – 406 с.

26. Segaran T. Programming Collective Intelligence: Building Smart Web 2.0 / Toby Segaran., 2011. – 384 с.

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Дипломна робота
за освітньо-професійною програмою «Інтелектуальний аналіз даних в управлінні проектами»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»
на тему: «Система розпізнавання раку шкіри з використанням нейронних мереж»

Виконала: студентка групи КА-66 Житанська Дар'я Сергіївна
Науковий керівник: доцент, к.ф.-м.н. Шубенкова Ірина Анатоліївна

Мета, об'єкт та предмет дослідження

Мета дослідження: Спростити алгоритм виявлення хвороби раку шкіри. Дослідити сучасні методи та алгоритми встановлення діагнозу та запропонувати власний оптимальний спосіб, який буде доступним для будь-кого та водночас давати результат із високою точністю.

Об'єкт дослідження: Стан окремої медичної галузі, а саме дерматології, на предмет складності виявлення меланомних уражених ділянок шкіри. Сучасні способи вирішення проблеми для швидкого передбачення раку та запобігання летальних випадків.

Предмет дослідження: Інтелектуальні алгоритми та методи виявлення хвороби раку шкіри на основі використання штучних нейронних мереж.

Постановка задачі

- ✓ Провести дослідження останніх оновлень статистики захворювання, наукових зібрань та досліджень на тему меланоми. Крім того, ознайомитися із новітніми стартапами, що перегукуються із метою даної роботи та, хоча б частково, націлені на вирішення задачі даної дипломної роботи.

- ✓ Знайти та проаналізувати вхідні дані, привести їх до конкретного виду, а також виявити способи знаходження кореляції між ними, для подальшого створення алгоритму розв'язку.

- ✓ Виявити оптимальний метод розв'язку задачі, порівнявши їх за найважливішими параметрами, та застосувати його на наших вхідних даних.

- ✓ Створити програмний продукт, що швидко та точно розпізнає хворобу раку шкіри

Актуальність задачі

Рак шкіри є однією з найбільш поширених форм раку у світі. Точне розпізнавання цього захворювання на *ранній стадії* може значно підвищити відсоток виживання пацієнтів. Однак ручне виявлення раку шкіри має чималу потребу у висококваліфікованих фахівцях.

За рахунок великої кількості типів раку шкіри та широкого діапазону його ознак, не завжди поставлений діагноз є правильним. Саме тому варто розробити надійну **автоматичну систему розпізнавання**, яка підвищує точність виявлення та класифікації патологів.

Одним із сучасних та дієвих методів виявлення раку є **дерматоскопія**. Такий метод посилює візуальний ефект ураження шкіри шляхом усунення поверхневого відображення. Проте, автоматичне розпізнавання раку шкіри з дерматоскопії зображень, як і раніше, є складним та затратним завданням.

Саме тому, важливою задачею є модифікація стандартного алгоритму виявлення діагнозу та вживлення сучасних алгоритмів, для покращення результату.

Головні етапи вирішення задачі

При дослідженні останніх робіт, присвячених даній темі, було виявлено, що вирішити задачу такого типу, допоможуть штучні нейронні мережі. Вони самостійно навчаються на основі даних із раніше проведених спостережень та знаходять власне вирішення проблеми. Саме тому, алгоритм розв'язку задачі можна розбити на такі етапи:

1. Знаходження даних (а саме зображень), які вже мають коректний діагноз, для тренування та знаходження кореляції між даними, за допомогою штучних нейронних мереж.

2. Підготовка зображень(даних), які були зроблені за допомогою дерматоскопії.

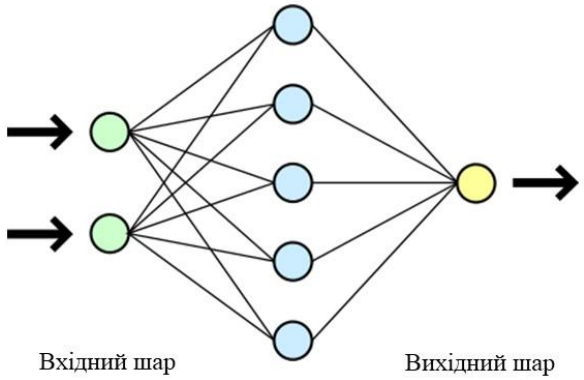
3. Сегментація. На цьому кроці, ми працюємо із ознаками раку шкіри, його властивостями та намагаємось домогтися автоматичного виявлення усіх ознак.

4. Класифікація. На основі дослідження ознак уражених ділянок та діагнозів, ми повинні зробити певні висновки та знайти взаємозв'язок між ними.

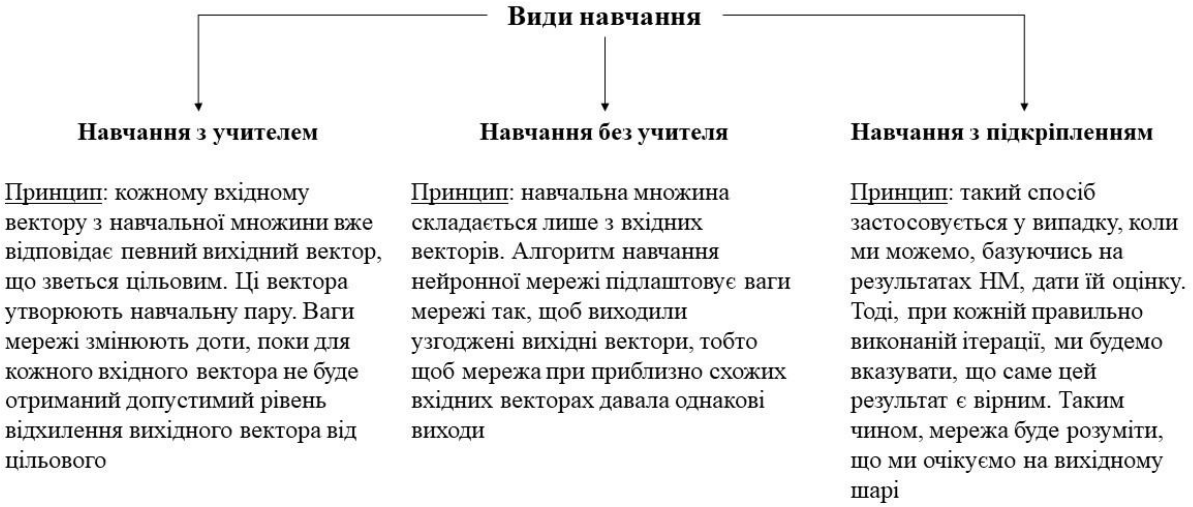
Ключові поняття в теорії нейронних мереж

Штучна нейронна мережа - математична модель, а також її програмне або апаратне втілення, побудована за принципом організації та функціонування біологічних нейронних мереж - мереж нервових клітин живого організму.

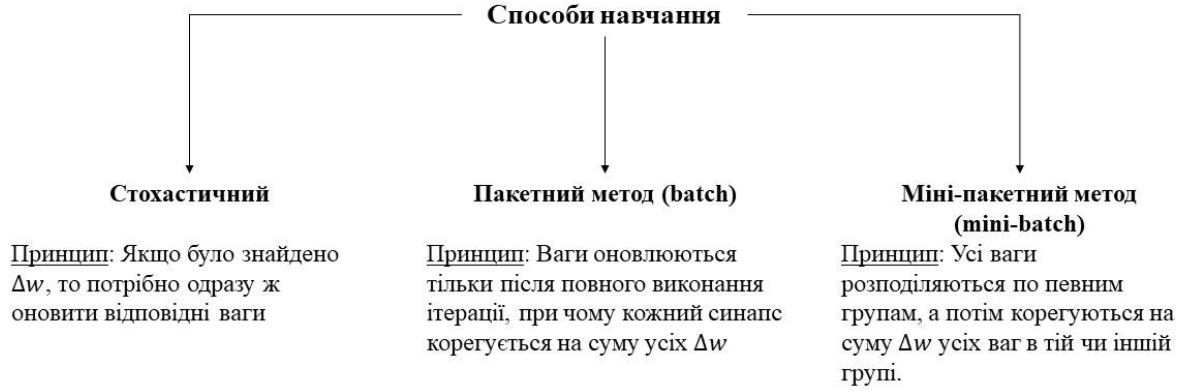
ШНМ складається із **нейронів**, що представляють собою спрощену модель біологічного нейрона. Вони приймають сигнали з багатьох входів, обробляють їх та передають результат на інші. Зв'язки між штучними нейронами називають **синаптичними**, або просто **синапсами**.
У синапсу є один параметр - **ваговий коефіцієнт**, в залежності від його значення і відбувається (або не відбувається) передача інформації. Саме такому підходу вхідна інформація обробляється і перетворюється в результат.
Навчання нейронної мережі засноване на експериментальному підборі такого вагового коефіцієнта для кожного синапсу, який приведе до необхідного результату.



Навчання нейронної мережі



Навчання нейронної мережі

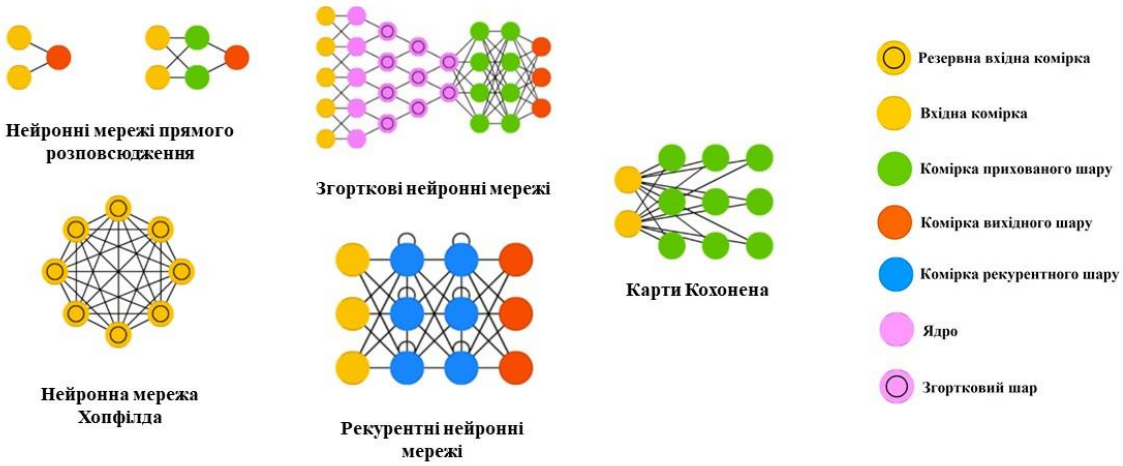


Основні методи навчання нейронних мереж

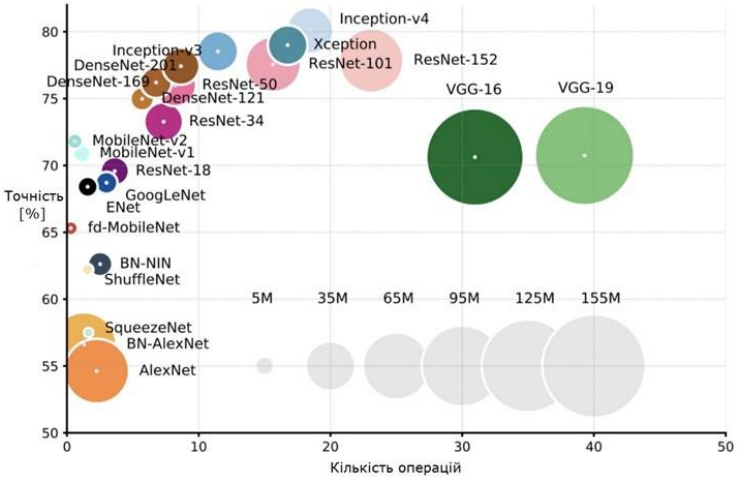
Алгоритм зворотного розповсюдження	Алгоритм найшвидшого спуску	Алгоритм спряжених градієнтів
• Відноситься до методів навчання з учителем • Його основу складає цільова функція у вигляді квадратичної суми різниць між фактичними і очікуваними значеннями вихідних сигналів. • Ваги нейронів мережі коригуються після подачі на її вхід одного навчального прикладу. ЕТАПИ АЛГОРИТМУ: 1. На прямому вхідний вектор поширюється від входів мережі до її виходів і формує певний вихідний вектор, що відповідає поточному (фактичного) стану ваг 2. Потім обчислюється помилка нейронної мережі як різниця між фактичним і цільовим значеннями. 3. На зворотному проході ця помилка поширюється від виходу мережі до її входів, і проводиться корекція ваг нейронів відповідно до конкретного правила	• Модифікація градієнтного методу • Теоретична швидкість збіжності методу не вище швидкості збіжності градієнтного методу з постійним (оптимальним) кроком • Вимагає рішення на кожному кроці завдання одномірної оптимізації. • Вимагає меншого числа операцій, ніж градієнтний метод з постійним кроком. ГОЛОВНІ ІДЕЇ: 1. Рух уздовж антиградієнта відбувається до тих пір, поки значення функції убуває 2. Крок змінюється в залежності від отриманого наближення	ГОЛОВНА ІДЕЯ: На кожному кроці в якості напрямку спуску використовувати не антіградієнта, а його лінійну комбінацію з попереднім напрямом спуску Позначимо через p^k напрямок спуску на k-й ітерації, то послідовність векторів p^k будується наступним чином: *1.* $p^0 = -g^0$, перший крок по антиградієнту *2.* $p^{k+1} = -g^{k+1} + \beta_k p^k$, де $\beta_k = (g^{k+1}, g^{k+1})/(g^k, g^k)$ Тобто зробивши з початкової точки один крок по методу найшвидшого спуску, треба обчислити антиградієнт в новій точці. Потім взяти відношення квадратів довжин нового і старого градієнтів, помножити попереднє значення антиградієнта на це число і результат скласти з новим. Це і буде напрямок спуску.

Схема створення програмного продукту

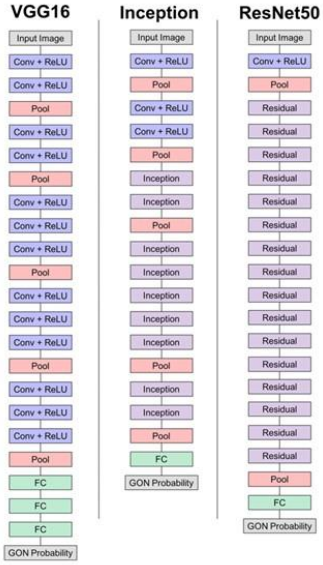
Ключові архітектури нейронних мереж



Сучасні моделі нейронних мереж



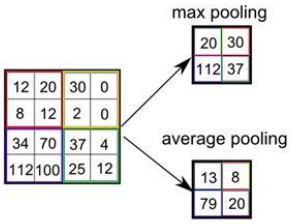
Порівняння архітектур



Conv + ReLU – згортковий шар із функцією активації ReLU

Pool – Max/Average pooling

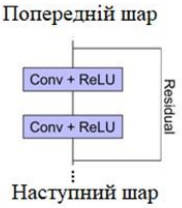
FC – повнозв'язний шар



** з ядром 2x2*

Модуль «Inception»

Модуль «Residual»



Результати навчання

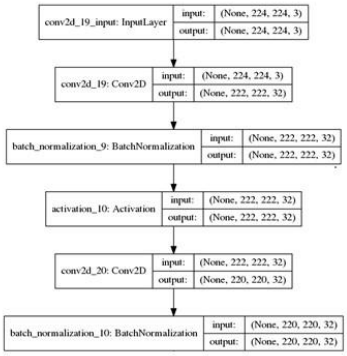
VGG16

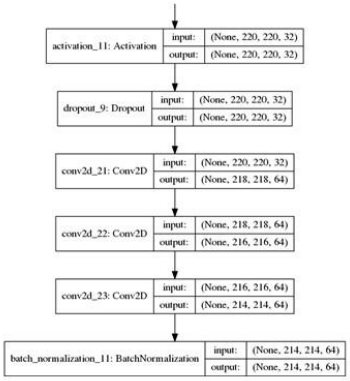
Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.502968	1.484062	0.516901	1.464917
1	0.50567	1.465239	0.517002	1.269898
2	0.504505	1.462749	0.518936	1.355259
3	0.50644	1.461367	0.51394	1.417217
4	0.505161	1.461633	0.517486	1.355599
5	0.506518	1.457538	0.515874	1.442344
6	0.504876	1.462347	0.521354	1.45676
7	0.505405	1.460434	0.512651	1.52712
8	0.505034	1.460706	0.524416	1.373612
9	0.504928	1.45911	0.512651	1.420581

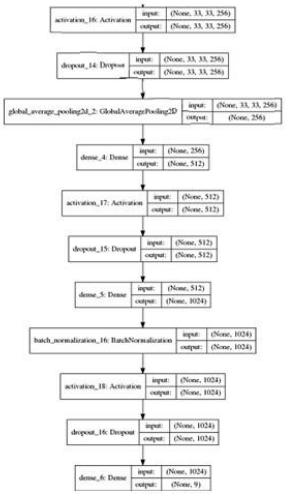
ResNet50

Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.642289	1.026844	0.598055	0.974046
1	0.699947	0.825596	0.442869	1.809946
2	0.725967	0.759391	0.651088	1.032954
3	0.745787	0.704625	0.617405	1.084079
4	0.756333	0.672484	0.702337	0.846883
5	0.768892	0.633461	0.697824	0.95721
6	0.778855	0.609998	0.74585	0.645828
7	0.786634	0.582199	0.729573	0.808383
8	0.803469	0.54106	0.665915	1.214281
9	0.809913	0.519991	0.75552	0.716226

Створення власної архітектури







Примітка: Модуль такого типу повторюється **4 рази**, чергуючи Dropout та Max-Pooling2D

Результати роботи власної мережі

Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.857131	0.341639	0.813561	0.372054
1	0.91145	0.227817	0.900999	0.263456
2	0.914991	0.217868	0.912578	0.206821
3	0.91746	0.212269	0.920924	0.209146
4	0.918864	0.207754	0.920731	0.222788
5	0.919859	0.204358	0.924253	0.188406
6	0.921461	0.201743	0.919188	0.185571

Висновки

Отже, з даного дослідження, можна зробити висновок, що не завжди мережа, яка показала гідний результат на конкретних навчальних даних, зможе показати такий же на будь-яких. Тобто, мережа, що тренувалася кластеризувати машини, або виявляти громіздкі предмети на зображенні, не дасть потрібного результату, при класифікації хвороби раку шкіри.

Саме тому, інколи, створення власної нейронної мережі є оптимальним варіантом для вирішення конкретної задачі. Такий підхід дозволяє врахувати усі деталі завдання самостійно та корегувати архітектуру, для поліпшення результатів.

У результаті дослідження був розроблений програмний продукт, що розпізнає хворобу раку шкіри та визнає її підвид. При завантаженні зображення, зробленого за допомогою дерматоскопії, програма із точністю 92% ставить узагальнений діагноз.

Дякую за увагу!

ДОДАТОК Б ІЛЮСТРАЦІЇ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

Таблиця Б.1 – Результати навчання мережі із архітектурою VGG16

Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.502968	1.484062	0.516901	1.464917
1	0.50567	1.465239	0.517002	1.269898
2	0.504505	1.462749	0.518936	1.355259
3	0.50644	1.461367	0.51394	1.417217
4	0.505161	1.461633	0.517486	1.355599
5	0.506518	1.457538	0.515874	1.442344
6	0.504876	1.462347	0.521354	1.45676
7	0.505405	1.460434	0.512651	1.52712
8	0.505034	1.460706	0.524416	1.373612
9	0.504928	1.45911	0.512651	1.420581

Таблиця Б.2 – Результати навчання мережі із архітектурою VGG16

Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.642289	1.026844	0.598055	0.974046
1	0.699947	0.825596	0.442869	1.809946
2	0.725967	0.759391	0.651088	1.032954
3	0.745787	0.704625	0.617405	1.084079
4	0.756333	0.672484	0.702337	0.846883
5	0.768892	0.633461	0.697824	0.95721
6	0.778855	0.609998	0.74585	0.645828
7	0.786634	0.582199	0.729573	0.808383
8	0.803469	0.54106	0.665915	1.214281
9	0.809913	0.519991	0.75552	0.716226

Таблиця Б.3 – Результати навчання мережі із власною архітектурою

Епоха	Навчальна вибірка		Тренувальна вибірка	
	Точність	Функція помилки	Точність	Функція помилки
0	0.857131	0.341639	0.813561	0.372054
1	0.91145	0.227817	0.900999	0.263456
2	0.914991	0.217868	0.912578	0.206821
3	0.91746	0.212269	0.920924	0.209146
4	0.918864	0.207754	0.920731	0.222788
5	0.919859	0.204358	0.924253	0.188406
6	0.921461	0.201743	0.919188	0.185571

ДОДАТОК В ЛІСТИНГ ПРОГРАМИ

```
#!/usr/bin/env python
# coding: utf-8

import keras
from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import
Input,Conv2D,Dense,Dropout,Flatten,MaxPooling2D,AveragePooling2D,Activation,Ba
tchNormalization,GlobalAveragePooling2D
import pandas as pd
import numpy as np
import os
from sklearn.utils import shuffle
import seaborn as sns
import matplotlib.pyplot as plt
from glob import glob
from skimage.io import imread


pip install nbconvert


# ### ***Загрузка и работа с тренировочными данными. Отображение
результатов предсказания и класификации кожных заболеваний***


train_path = "ISIC_2019_Training_Input/"
test_path = "ISIC_2019_Test_Input/"
ground_truth_path = "Training_GroundTruth.csv"



df = pd.read_csv(ground_truth_path) #считывание файла
df.head()


columns = list(df.columns)[1:] #вывод всех названий болезней
print(columns)
```

```
df_shuffle = shuffle(df,random_state = 7) #вывод случайных строчек
df_shuffle.head()


def show_values_on_bars(axs, h_v, space): #надпись над каждым столбцом
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
            _y = p.get_y() + p.get_height()+ 100
            value = int(p.get_height())
            ax.text(_x, _y, value, ha="center")
    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)


fig = plt.gcf() #обозначене фигуры
fig.set_size_inches(12, 8)#размеры фигуры
plt.title('Class Count')#название диаграммы
data = df[columns].sum() #сумма каждого столбца
pal = sns.cubehelix_palette(9,reverse = True)#вид цветовой карты
rank = data.values.argsort().argsort()
br = sns.barplot(data.index,data.values,palette=np.array(pal[::-1])[rank]) #построение
графика
show_values_on_bars(br, "v", 0.4)
sns.set()


df['dx'] = df[columns].idxmax(axis=1) #запись диагноза
df.head()


base_skin_dir = os.path.join('..', 'Diploma')

image_id_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                      for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))} #путь к
соответствующей фотографии
```

```
lesion_type_dict = {
    'NV': 'Melanocytic nevus',
    'MEL': 'Melanoma',
    'BKL': 'Benign keratosis',
    'BCC': 'Basal cell carcinoma',
    'AK': 'Actinic keratosis',
    'VASC': 'Vascular lesion',
    'DF': 'Dermatofibroma',
    'SCC':'Squamous cell carcinoma'
} #список всех диагнозов

df['path'] = df['image'].map(image_id_path_dict.get) #добавление нового столбца с путем
df['diagnosis'] = df['dx'].map(lesion_type_dict.get) #добавление нового столбца с полным диагнозом
df.head()


# ### ***Отображение примеров каждого заболевания***


df['image_numb'] = df['path'].map(imread) #загрузка фотографий в файл



df['image_numb'].map(lambda x: x.shape).value_counts()#преобразование изображения в массив чисел
df.head()



n_samples = 5 #вывод случайных картинок
fig, m_axs = plt.subplots(8, n_samples, figsize = (4*n_samples, 3*8))
for n_axs, (type_name, type_rows) in zip(m_axs,
                                         df.sort_values(['diagnosis']).groupby('diagnosis')):
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples,
random_state=20).iterrows()):
        c_ax.imshow(c_row['image_numb'])
        c_ax.axis('off')
fig.savefig('category_samples.png', dpi=300)
```

```
# ###  ***Достаем и нормализируем информацию о цветовых каналах. Предобладание каждого цвета в каждом изображении***

rgb_info_df = df.apply(lambda x: pd.Series({'{}_mean'.format(k): v for k, v in
                              zip(['Red', 'Green', 'Blue'],
                                  np.mean(x['image_numb'], (0, 1)))}),1) #нормализация данных
gray_col_vec = rgb_info_df.apply(lambda x: np.mean(x), 1)
for c_col in rgb_info_df.columns:
    rgb_info_df[c_col] = rgb_info_df[c_col]/gray_col_vec
rgb_info_df['Gray_mean'] = gray_col_vec
rgb_info_df.sample(3)  #среднее значение цветовых каналов


for c_col in rgb_info_df.columns:
    df[c_col] = rgb_info_df[c_col].values #копирование значений в таблицу df


sns.pairplot(df[['Red_mean', 'Green_mean', 'Blue_mean', 'Gray_mean', 'diagnosis']],
             hue='diagnosis', plot_kws = {'alpha': 0.5}) #график преобладания цвета в конкретном изображениях диагноза


# ### ***Влияние среднего значения цветового канала(red, green, blue, gray) на изображение***

n_samples = 5
for sample_col in ['Red_mean', 'Green_mean', 'Blue_mean', 'Gray_mean']:
    fig, m_axs = plt.subplots(7, n_samples, figsize = (4*n_samples, 3*7))
    def take_n_space(in_rows, val_col, n):
        s_rows = in_rows.sort_values([val_col])
        s_idx = np.linspace(0, s_rows.shape[0]-1, n, dtype=int)
        return s_rows.iloc[s_idx]
    for n_axs, (type_name, type_rows) in zip(m_axs,
                                             df.sort_values(['diagnosis']).groupby('diagnosis')):

        for c_ax, (_, c_row) in zip(n_axs,
                                    take_n_space(type_rows,
                                                 sample_col,
                                                 n_samples).iterrows()):
```

```
            c_ax.imshow(c_row['image_numb'])
            c_ax.axis('off')
            c_ax.set_title('{:2.2f}'.format(c_row[sample_col]))
        n_axs[0].set_title(type_name)
    fig.savefig('{}_samples.png'.format(sample_col), dpi=350)


# ### ***Тренировка нейронной сети***


from keras.utils import plot_model
from IPython.display import Image
plot_model(model,to_file = 'multi_output_model_2.png',show_shapes=True)
pil_img = Image('multi_output_model_2.png')
display(pil_img) #разобраться!


def name(row): #добавление .jpg к каждому названию в колонке image
    row.image = row.image + '.jpg'
    return row
#подумать про if-else чтобы не добавлялись лишние .jpg

df = df.apply(name,axis = 1)  #применение функции name к каждой строчке файла
df.head()


#создание пакетов данных (шаблонов) тензорного изображения для данных
train_datagen = ImageDataGenerator(rescale=1./255., #масштабирование
                                   horizontal_flip = True, #переворот по горизонтали
                                   width_shift_range = 0.2, #доля от изображения
                                   height_shift_range = 0.2,
                                   fill_mode = 'nearest', #режим заполнения aaaaaaaa | abcd |
dddddddd
                                   zoom_range = 0.3, #диапазон для случайного увелечения
                                   rotation_range = 30 #диапазон поворота изображения
                                    )
valid_datagen = ImageDataGenerator(rescale=1./255.)

#заполнение тензоров данными
train_generator = train_datagen.flow_from_dataframe(dataframe = df[:25231],
                                        directory = train_path,
```

```
                                    x_col = "image",
                                    y_col = columns, #целевые данные
                                    batch_size = 128, #размер пакета данных
                                    seed = 42,
                                    shuffle = True, #перетасовка
                                    class_mode = "raw",
                                    target_size = (100,100) #размер всех изображений после
преобразования
                                    #validate_filenames = False
                                    )

#If class_mode="multi_output", y_col must be a list. Received Index.
valid_generator = valid_datagen.flow_from_dataframe(dataframe=df[22800:25231],
                                        directory = train_path,
                                        x_col= "image",
                                        y_col=columns,
                                        batch_size=128,
                                        seed=42,
                                        shuffle=True,
                                        class_mode="raw",
                                        target_size=(100,100),
                                        validate_filenames = True
                                        )

test_generator = valid_datagen.flow_from_dataframe(dataframe=df[25231:],
                                        directory = train_path,
                                        x_col="image",
                                        #y_col=columns,
                                        batch_size=1,
                                        seed=42,
                                        shuffle=True,
                                        class_mode=None,
                                        target_size=(100,100),
                                        validate_filenames = True
                                        )

#My own model
model = Sequential() #нейронная сеть
model.add(Conv2D(32, (3, 3),input_shape=(100,100,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3)))
model.add(Conv2D(64, (3, 3)))
model.add(Conv2D(64,(3,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3)))
model.add(Conv2D(64,(3,3)))
model.add(Conv2D(64,(3,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3)))
model.add(Conv2D(128,(3,3)))
model.add(Conv2D(128,(3,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(128,(3,3)))
model.add(Conv2D(128,(3,3)))
model.add(Conv2D(128,(3,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Conv2D(256,(3,3)))
model.add(Conv2D(256,(3,3)))
model.add(Conv2D(256,(3,3)))
model.add(Conv2D(256,(3,3)))
model.add(BatchNormalization())
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(GlobalAveragePooling2D())
#model.add(Flatten())

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(9, activation='sigmoid'))
model.compile(keras.optimizers.rmsprop(lr=0.0001, decay=1e-
6),loss="binary_crossentropy",metrics=["accuracy"])

model.summary() #итоговая модель


os.mkdir('History') #создание новой папки
os.mkdir('Models')

from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.callbacks import *

filepath = 'Models/multi_output_model_7_weights-{epoch:02d}-
{val_accuracy:.3f}.hdf5' #запись потерь и точности в файлик
checkpoint1 = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
save_best_only=True, mode='max')
checkpoint2 = CSVLogger('History/Log_model_7.csv')
callbacks_list = [checkpoint1 , checkpoint2]



STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size
```

```
history = model.fit_generator(generator=train_generator,
                              steps_per_epoch=STEP_SIZE_TRAIN,
                              validation_data=valid_generator,
                              validation_steps=STEP_SIZE_VALID,
                              epochs=10,
                              callbacks = callbacks_list
                              )



#VGG16

from keras.optimizers import SGD


model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(100,100,3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
```

```
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(9, activation='softmax'))

from keras.utils import plot_model #схема нейронной сети
from IPython.display import Image
plot_model(model,to_file = 'model_VGG16.png',show_shapes=True)
pil_img = Image('model_VGG16.png')
display(pil_img)

from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.callbacks import *
filepath = 'Models/model_VGG16_weights-{epoch:02d}-{val_accuracy:.3f}.hdf5'
checkpoint1_VGG16 = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
save_best_only=True, mode='max')
checkpoint2_VGG16 = CSVLogger('History/Log_model_VGG16.csv')
callbacks_list = [checkpoint1_VGG16 , checkpoint2_VGG16]

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=["accuracy"])
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size #обучение
нейронной сети
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
history_VGG16 = model.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
```

```
                    epochs=10,
                    callbacks = callbacks_list
                    )

#ResNet50

import tensorflow as tf
from IPython.display import Image
from keras.preprocessing import image
from keras.applications import imagenet_utils
from keras.applications import ResNet50
from keras.models import Model

base_model= ResNet(weights='imagenet',include_top=False) #imports the resnet model and discards the last 1000 neuron layer.
x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
preds=Dense(9,activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=base_model.input,outputs=preds)

# for i,layer in enumerate(model.layers):
#   print(i,layer.name)

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size #обучение нейронной сети
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size
model.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=10,
                    callbacks = callbacks_list)
model.predict(test_generator(1201))
```