Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра прикладної математики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Класифікація фрактальних реалізацій методами машинного навчання на основі побудови графів видимості
(тема)х

Виконав:
студент _2_ курсу, групи _ПМм-21-1
Рижанов В.С.
(прізвище, ініціали)
Спеціальність
113 Прикладна математика
(код і повна назва спеціальності)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Прикладна математика
(повна назва освітньої програми)
Керівник проф. Кіріченко Л.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ПМ _____ Сидоров М.В.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет _інформаційно-аналітичних технологій та менеджменту
Кафедра _прикладної математики
Рівень вищої освіти _другий (магістерський)
Спеціальність _113 Прикладна математика
(код і повна назва)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма _Прикладна математика
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри ПМ _____
(підпис)
“ 07 ” листопада 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _Рижанову Віталію Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи _Класифікація фрактальних реалізацій методами машинного навчання на основі побудови графів видимості
затверджена наказом по університету від 25 жовтня 2022 р. № 1412 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 7 грудня 2022 р.

3. Вихідні дані до роботи _фрактальні реалізації

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області
2. Вибір і обґрунтування методу розв’язання
3. Програмна реалізація
4. Результати обчислювального експерименту
5. Аналіз можливих застосувань

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

1. Актуальність теми роботи
2. Постановка задачі
3. Аналіз предметної області
4. Метод чисельного аналізу
5. Результати обчислювального експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Підбір та вивчення технічної літератури за темою роботи	7 – 13 листопада 2022 р.	виконано
2	Вибір та обґрунтування методу	14 – 20 листопада 2022 р.	виконано
3	Розробка алгоритму і програми	21 – 27 листопада 2022 р.	виконано
4	Проведення аналітичних досліджень та розрахунків	28 листопада – 4 грудня 2022 р.	виконано
5	Робота над текстом пояснювальної записки	28 листопада – 6 грудня 2022 р.	виконано
6	Представлення роботи на рецензію в ЕК	7 грудня 2022 р.	виконано

Дата видачі завдання _7_ листопада 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кіріченко Л.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 68 с., 34 рис., 2 дод., 29 джерел.

БРОУНІВСЬКИЙ РУХ, ГРАФ НАТУРАЛЬНОЇ ВИДИМОСТІ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, КЛАСИФІКАЦІЯ, МАТРИЦЯ СУМІЖНОСТІ, ФРАКТАЛ, ЧАСОВИЙ РЯД, PYTHON, HSV.

Об’єкт дослідження – фрактальні реалізації.

Мета роботи – дослідити метод візуалізації фрактальних реалізацій за допомогою якого буде здійснена класифікація різнокольорових зображень фрактальних реалізацій, використовуючи Deep Machine Learning.

Методи дослідження – класифікація фрактальних реалізацій.

Кваліфікаційна робота присвячена дослідженню застосування візуалізації фрактальних реалізацій за допомогою графів видимості для класифікації, використовуючи Deep Machine Learning. Для обчислюваного експерименту було згенеровано фрактальні реалізації, побудовано графи видимості та репрезентовано їх у матриці суміжності. Машинне навчання, а саме нейронні мережі – це потужний інструмент класифікації та кластеризації даних, тому, для класифікації матриць суміжності представлених у вигляді різнокольорових зображень, використовувалась згорткова нейронна мережа.

ABSTRACT

Introductory note: 68 pages, 34 figures, 2 appendixes, 29 sources.

BROWNIAN MOTION, GRAPH OF NATURAL VISIBILITY, CONVOLVED NEURAL NETWORK, CLASSIFICATION, ADJACENT MATRIX, FRACTAL, TIME SERIES, PYTHON, HSV.

Object of research – fractal realizations.

Purpose of work – to investigate the method of visualization of fractal realizations, which will be used to classify colorful images of fractal realizations, using Deep Machine Learning.

Methods of research – classification of fractal realizations.

The qualification work is devoted to the study of the application of visualization of fractal realizations with the help of visibility graphs for classification, using Deep Machine Learning. For the calculated experiment, fractal realizations were generated, visibility graphs were constructed and represented in the adjacency matrix. Machine learning, namely neural networks is a powerful tool for data classification and clustering, therefore, a convolutional neural network was used to classify adjacency matrices presented in the form of multi-colored images.

ЗМІСТ

С.

ВСТУП

Актуальність теми. Досить новий підхід для аналізу та класифікації часових рядів полягає у перетворенні часових рядів в іншу структуру та вилученні ознак із цієї нової структури. Наприклад, класифікацію можна здійснити за допомогою вейвлет-спектру, рекурентної діаграми або зображення. Нещодавно з'явився новий метод дослідження часових рядів. Він використовує добре розроблені методи комплексного мережевого аналізу, такі як графи видимості, і перетворює інформацію, подану у формі часових рядів, на потужний інструмент теорії графів. Цей метод намагається перетворити інформацію, представлену як часовий ряд, у більш потужну математичну структуру, ніж просто часовий ряд. Введення певних функцій графічного введення в систему класифікації часових рядів може допомогти проаналізувати конкретні характеристики графіка, що класифікується та зовсім по-іншому подивитися на проблему класифікації часових рядів.

Багато складних процесів мають фрактальну структуру і їх динаміка представлена часовими рядами, що володіють фрактальними властивостями. До таких процесів належать різні інформаційні процеси в комунікаційних мережах, у тому числі хакерські атаки.

Важливим етапом у методах виявлення відхилень – є класифікація часового ряду за фрактальними властивостями, зокрема, діапазоном значень параметра Херста. У багатьох випадках мають місце проблеми розпізнавання та класифікації фрактальних рядів. Найчастіше такі задачі вирішуються шляхом оцінювання та аналізу фрактальних характеристик. Проте в останні роки зростає інтерес до методів машинного навчання для аналізу та класифікації фрактальних рядів.

Проблемами класифікації в машинному навчанні є те, що дані впорядковуються за часовим виміром, і тому результативний алгоритм повинен би використовувати цю властивість даних. Тому, тема даної кваліфікаційної роботи є актуальною.

Мета і завдання кваліфікаційної роботи. Метою кваліфікаційної роботи є дослідження візуалізації фрактальних часових реалізацій за допомогою графів видимості з подальшою їх класифікацією методами Deep Machine Learning та розробка програмного забезпечення для практичного підтвердження та демонстрації результатів класифікації.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести огляд і аналіз сучасного стану задачі «класифікація часових рядів»;

– огляд та генерація фрактального броунівського руху;

– огляд алгоритму та побудова натурального графа видимості;

– аналіз та застосування методів подання графів видимості у вигляді, зручному для зорового спостереження та аналізу;

– огляд та використання згорткової нейронної мережі для класифікації фрактальних реалізацій.

Об'єктом дослідження є фрактальні реалізації.

Предметом дослідження є застосування графів видимості для класифікації фрактальних реалізацій методами машинного навчання.

Методи дослідження. У кваліфікаційній роботі використовуються технологія перетворення графів видимості у зображення та методи глибокого машинного навчання.

Публікації. Результати, отримані у кваліфікаційній роботі, було представлено на XVI Міжнародній науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті» (м. Дніпро, 14-15 грудня 2022 р.) [29].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Короткий огляд алгоритмів класифікації часових рядів

Класифікація часових рядів – типове завдання машинного навчання.

Класифікація – це завдання поділу набору об’єктів або спостережень на групи (так звані класи), у кожній групі вони вважаються схожими один на одного, з приблизно однаковими властивостями та характеристиками (рис. 1.1).

Існує багато алгоритмів, призначених для класифікації часових рядів. Залежно від даних один тип може забезпечити вищу точність класифікації, ніж інші типи. Ось чому важливо розглянути низку алгоритмів, занурюючись у проблему класифікації часових рядів. Використання автоматизованої платформи, яка б ретельно досліджувала простір доступних алгоритмів і гіперпараметрів, може заощадити значний час принаймні на початкових етапах дослідження, вказуючи алгоритмічний конвеєр, який забезпечує оптимізовану точність для вхідного набору даних. Такі платформи з можливістю класифікації часових рядів очікуються в найближчому майбутньому.

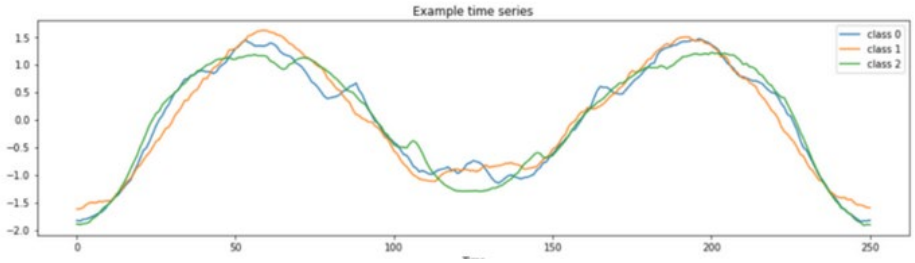


Рисунок 1.1 – Часові ряди, які відносяться до різних классів

1.1.1 Дистанційні підходи класифікації

Міра відстані – це об’єктивна оцінка, яка підсумовує відносну різницю між двома об’єктами в проблемній області. Що менша відстань між двома

об’єктами (зазвичай це дані, що описують щось), то більш схожими є елементи. Деякі типи вимірювань відстані, які зазвичай використовуються в машинному навчанні:

а) евклідова відстань;

б) відстань Хеммінга;

в) манхеттенська відстань;

г) відстань Мінковського.

Ці вимірювання відстані використовуються разом із деякими добре відомими алгоритмами на основі відстані, такими як k-найближчі сусіди (KNN). Він вимірює відстань між тестовим об’єктом і всіма об’єктами в наборі навчальних даних. Потім вибираються k найкоротших відстаней, і новому об’єкту призначається клас, який найбільше представлений у k об’єктах із навчального набору. Коли k встановлено в одиницю, алгоритм зводиться до найближчого сусіда, а тестовому об’єкту призначається клас вибірки навчального набору з найкоротшою відстанню [2, 4].

Існують інші алгоритми на основі ядра, в основі яких використовуються вимірювання відстані. Мабуть, найвідомішим із цих алгоритмів є опорна векторна машина (SVM). Цей алгоритм створює гіперплощину (або лінію у 2-вимірах) для розділення об’єктів на класи. Потім розраховується положення тестового об’єкта відносно гіперплощини та відповідно призначається клас.

Динамічне викривлення часу (DTW) – це алгоритм на основі відстані, який використовується для вимірювання відстані між двома часовими рядами. DTW робить це, обчислюючи відстані між кожною точкою в часовому ряді та підсумовуючи їх для загальної відстані. Алгоритм створено для роботи з незначними зрушеннями між дуже подібними часовими рядами. DTW у поєднанні з 1-NN був золотим стандартом для класифікації часових рядів протягом останнього десятиліття і майже завжди використовується як порівняльний алгоритм у порівняльних дослідженнях.

1.1.2 Shapelet-класифікація

Shapelet – це підпослідовності або невеликі підфігури форми часового ряду, які представляють клас (рис. 1.2). Дані часових рядів часто демонструють характерні форми даних, які вказують на клас часових рядів [1, 2]. Алгоритм shapelet-перетворення може аналізувати підпослідовності часових рядів і генерувати результати, корисні для класифікатора для розрізнення класів. Характерні форми ЕКГ, присутні в підпослідовностях серцевих скорочень і які вказують на захворювання серця, були б ідеальною проблемою для цього типу алгоритму класифікації.

Дані часових рядів, які проходять через алгоритм shapelet, дають вихід, що показує мінімальну відстань між shapelet і всіма підпослідовностями в наборі даних (рис. 1.3). Як такий, це тип алгоритму на основі відстані, який подібний до DTW, за винятком того, що shapelet-перетворення вимірює відстань лише для підпослідовностей даних, а не для всього часового ряду. Деякі з найефективніших класифікаторів часових рядів складаються з кількох класифікаторів (ансамблів), які використовують дані, перетворені за допомогою shapelet-перетворення.

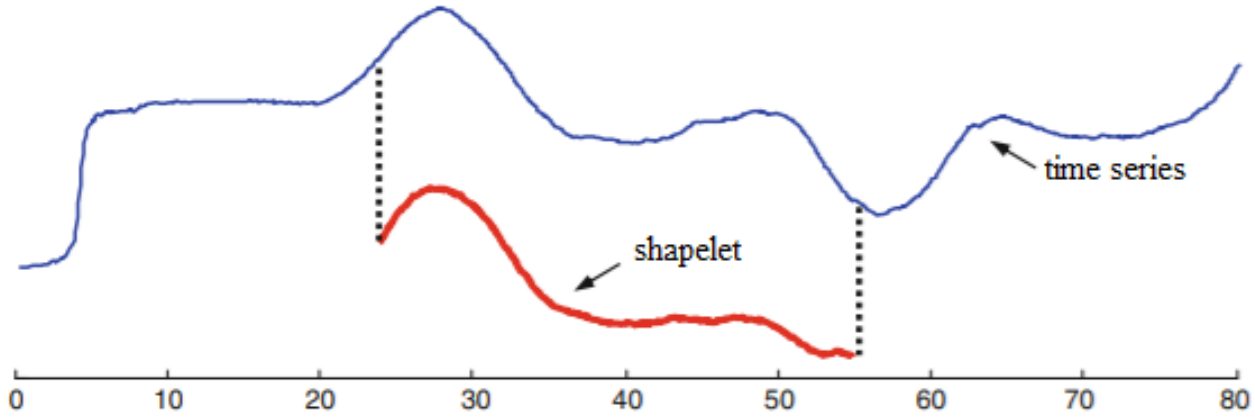


Рисунок 1.2 – Підпослідовності у вигляді Shapelet

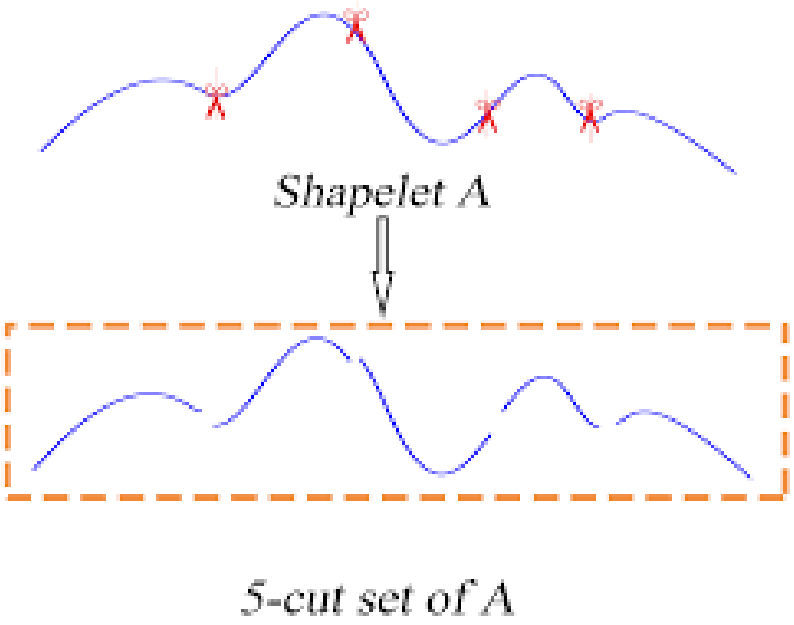


Рисунок 1.3 – Приклад набору Shapelet

1.1.3 Класифікатори на основі модельних ансамблів

Модель ансамблю для класифікації часових рядів – це набір моделей класифікації, кожна з яких виконує власну класифікацію набору даних. Клас, який найчастіше виходить із зібраних класифікаторів, стає класом, застосованим до набору даних. Щоб цей підхід працював, будь-які помилки, створені класифікаторами, не повинні корелюватися. Якщо помилки корельовані, сила підходу втрачається, і ансамбль наближається до точності одного класифікатора.

Якщо кілька типів алгоритмічних класифікаторів об’єднано в ансамбль, можна зменшити вплив корельованої помилки, згрупувавши класифікатори за типом і взявши одну класифікацію з групи. Потім це використовується з класифікацією з інших груп для класифікації ансамблю. Ансамблі такого типу є основою для алгоритму HIVE-COTE, який дуже добре працює з проблемами класифікації відносно невеликих часових рядів, але погано масштабується для більших наборів даних.

1.1.4 Словникові підходи класифікації

Інший популярний тип алгоритмів класифікатора заснований на структурі словника, який використовується для опису значення твору в найбільш поширеному вживанні терміну. Але словниковий підхід може описувати й інші об’єкти, крім слів. Словник може бути використаний для опису кількості входжень певного shapelet в часовий ряд (рис. 1.4).

Структура, дуже схожа на shapelet, називається ядром, використовується в аналізі шаблонів і може бути джерелом словника для подальшої класифікації. Алгоритм, який використовує цей підхід, зараз досить популярний і доступний через sktime ROCKET. ROCKET використовує випадкові згорткові ядра для створення словника, який згодом використовується для навчання класифікатора Ridge (також доступного як частина scikit-learn) для невеликих наборів даних або класифікатора лінійної регресії для великих наборів даних.

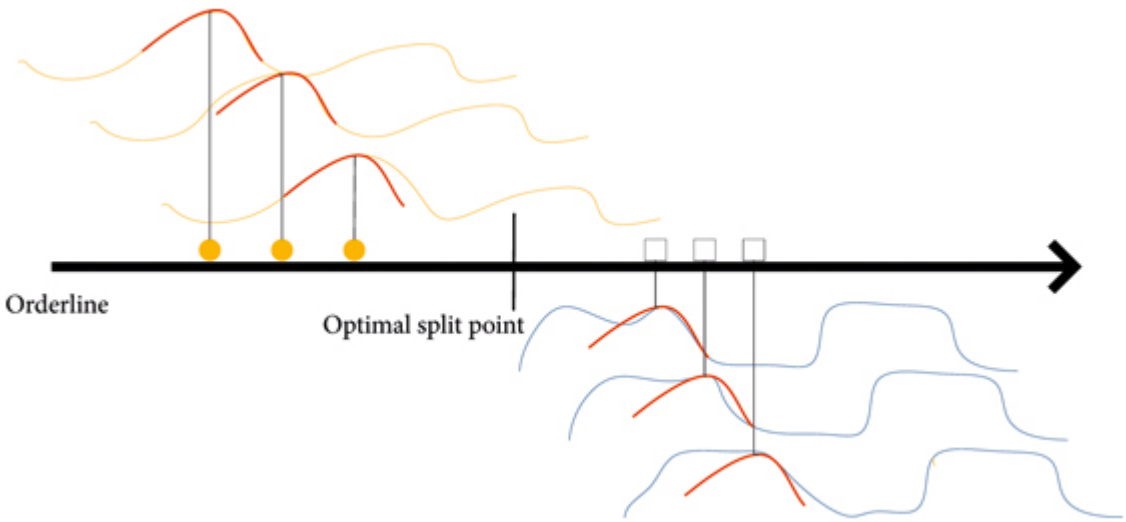


Рисунок 1.4 – Shapelet в часовому ряду

Для класифікації тексту дуже корисним словниковим підходом є алгоритм Bag-of-Words (BoW), який підраховує кількість слів у документі, а потім ця інформація використовується для навчання моделі BoW. Інтуїція тут полягає

в тому, що документи схожі, якщо їхні характеристики (кількість слів) подібні. Подібний підхід, який можна використовувати для даних часових рядів, використовує алгоритм Bag-of-Patterns (BoP), який замість підрахунку слів розглядає амплітуду сигналу часового ряду в певному заданому вікні даних і застосовує просте перетворення до сигналу, щоб представити середнє значення сигналу у вікні. Потім ця функція стає основою для словника, який використовується для навчання класифікатора.

1.1.5 Інтервальні підходи класифікації

Методи на основі інтервалів базуються на поділі часових рядів на окремі інтервали, подібно до методу Bag-of-Patterns, який обговорювався раніше. Потім кожен інтервал використовується для навчання окремої моделі машинного навчання (класифікатора). Цей підхід створює ансамбль класифікаторів, кожен з яких діє на власну підпослідовність/інтервал. Остаточна класифікація призначається на основі найпоширенішого класу, який генерується окремими класифікаторами.

Найпоширенішим алгоритмом на основі інтервалів є ліс часових рядів (TSF). Цей метод використовує дерево рішень для кожного інтервалу, при цьому агреговані дерева рішень є лісом. Кожне дерево рішень – це модель машинного навчання, яка потім призначає клас своєму інтервалу даних. Оскільки дерева рішень навчаються на різних інтервалах загального часового ряду, вони можуть не давати однакову класифікацію, тому необхідний процес голосування за ансамблем.

При цьому підході слід пам’ятати про те, що час виконання лінійно зростає з довжиною вибірки часового ряду, кількістю зразків часового ряду та кількістю функцій на інтервал. Тому, це може призвести до досить довгих обчислень для великих наборів даних.

1.2 Броунівський рух

Окрім вивчення реальних часових рядів, також дуже корисно вивчати деякі відомі математичні ряди, які можуть допомогти зрозуміти іншу більш складну поведінку в реальному світі. Одним із таких математичних рядів є той, що є результатом широко вивченого процесу Вінера, також відомого як броунівський рух у фізиці та інших галузях науки.

Процес Вінера має велике практичне і теоретичне значення, його можна використовувати як модель для багатьох різних явищ реального світу. Наприклад, його можна використовувати для моделювання руху фізичних частинок або для моделювання значень фондового ринку [10].

Вінерівський процес $W(t)$, $t \in \mathbb{R}, t \geq 0$, є математичним стохастичним процесом безперервного часу з такими властивостями [11]:

– $W(0) = 0$;

– $W(t)$ є неперервною функцією в t;

– W має незалежні прирости, якщо при $0 \leq s < t < u < \upsilon$ прирости $W(t) - W(s)$ та $W(\upsilon) - W(u)$ незалежні;

– W має гаусівський приріст, якщо при $0 \leq s < t$ приріст $W(t) - W(s)$ має нормальний розподіл із середнім значенням 0 і дисперсією $t - s : W(t) - W(s) \sim \mathrm{N}(0, t - s)$.

Процес Вінера, як визначено, є сигналом безперервного часу. Є можливість легко побудувати версію $W_t, t \in \mathbb{N}$ у дискретному часі, використовуючи підхід ітераційного випадкового блукання, визначений як:

$$\begin{cases} W_0 = 0, \\ W_i = W_{i-1} + \xi_i, \end{cases} \tag{1.1}$$

де $\xi_1, \xi_2, \ldots$ є незалежними та однаково розподіленими випадковими змінними із середнім 0 і дисперсією 1. Отже, для будь-якого $i : \xi_i \sim N(0,1)$.

Коли $n \to \infty$, W_n поводиться подібно до безперервного вінерівського процесу. Відтепер ми використовуватимемо назву вінерівський процес для позначення цієї дискретної версії. За допомогою цього методу ми можемо побудувати часові ряди (1.1):

$$T_W = \{(1, W_1), (2, W_2), ..., (n, W_n)\}.$$

Цей ітераційний підхід можна легко реалізувати програмно, таким чином дозволяючи нам генерувати часові ряди процесів Вінера будь-якої довжини n, яку ми хочемо (обмежується лише практичними обмеженнями комп'ютера).

Оскільки W_t є стохастичним процесом (стохастична природа виникає завдяки використанню рандомізованих приростів), ми можемо генерувати нескінченну кількість різних часових рядів T_W.

Інший спосіб поглянути на ряди процесів Вінера полягає в тому, що вони побудовані шляхом інтегрування ряду Гаусса з розподілом $N(0,1)$, інтегрування того, що відомо як білий шум Гаусса (рис. 1.5).

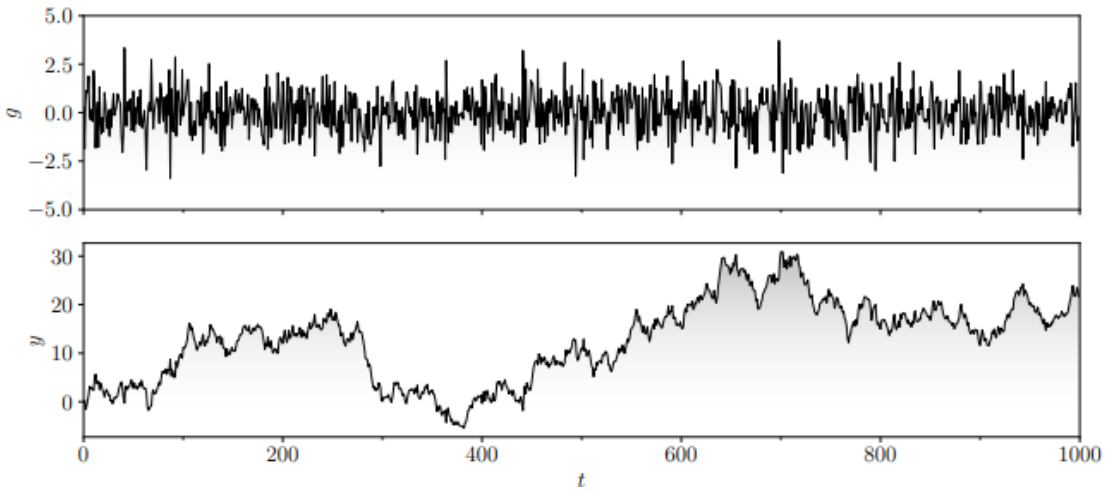


Рисунок 1.5 – Приклад часового ряду гаусового білого шуму з 1000 кроками (вгорі) та його інтегралом (внизу), що призводить до так званого вінерівського процесу або броунівського руху

1.2.1 Фрактальний броунівський рух

Фрактальний броунівський рух (ФБР) є узагальненням раніше описаного броунівського руху. На відміну від класичного броунівського руху, у фрактальному броунівському русі прирости не повинні бути незалежними. Величина залежності (або коваріації) приростів є функцією параметра H в інтервалі $(0,1)$, що називається індексом Херста або параметром Херста [12-14]. Залежно від значення H поведінку процесу ФБР можна розділити на три класи:

– якщо $H < \frac{1}{2}$, то прирости процесу негативно корелюють;

– якщо $H = \frac{1}{2}$, то процес є регулярним процесом Вінера з некорельованими приростами;

– якщо $H > \frac{1}{2}$, то прирости процесу позитивно корелюють.

Чим ближче H до 0 і чим ближче H до 1, тим більшою буде негативна та позитивна кореляція приростів відповідно. Тому, наприклад, якщо процес ФБР зі значенням $H > \frac{1}{2}$ зростає за значенням у певному інтервалі, то, ймовірно, він продовжуватиме зростати подібним чином у наступному інтервалі. Крім того, крайній випадок $H = 1$ призведе до цілком передбачуваної лінійної функції постійного тренду. Приклади процесів ФБР з різними значеннями H показані на рисунку 1.6.

Математично один із способів визначення ФБР-процесів – це використання фрактального інтеграла Рімана-Ліувіля, що призводить до наступного стохастичного представлення [12, 14]:

$$B_H(t) = \frac{1}{\Gamma\left(H + \frac{1}{2}\right)} \int_0^t (t-s)^{H-\frac{1}{2}} dW(s), \qquad (1.2)$$

де $W(s)$ – регулярний вінерівський процес (або броунівський рух), а Γ – гамма-функція, яку можна визначити як $\Gamma(n)=(n-1)!$ для додатних цілих значень. Зверніть увагу, як коли $H=\frac{1}{2}$, тоді $\Gamma\left(H+\frac{1}{2}\right)=1$ і $B_H(t)$ стає регулярним вінерівським процесом $W(t)$.

Можна сказати, що індекс Херста описує нерівність, мінливість або довгострокову пам'ять серії ФБР [14], і ту саму ідею можна екстраполювати для вимірювання цих властивостей для будь-якого типу часових рядів (не лише тих, що є результатом процессів ФБР) через індикатор, відомий як експонента Херста.

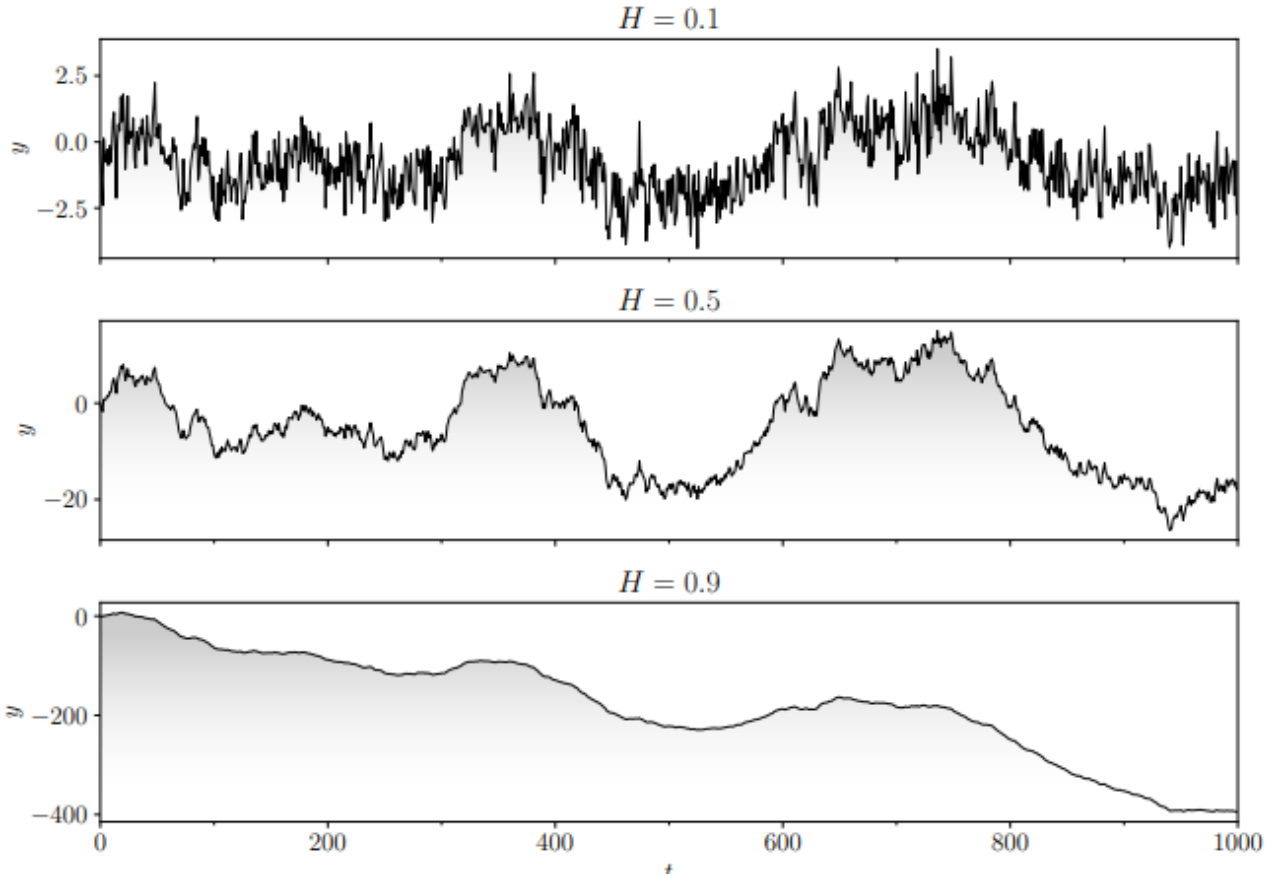


Рисунок 1.6 – Зразки фрактального броунівського руху для різних значень H (зауважте, що вісь y має різний масштаб)

1.3 Колірна модель HSV в комп'ютерній графіці

Колірна модель – це багатовимірне представлення колірного спектру. Найбільш релевантними колірними спектрами є RGB, HSV, HSL і CMYK. Колірна модель може бути представлена як 3D-поверхня (наприклад, для RGB) або перейти до набагато вищих розмірів (наприклад, CMYK). Регулюючи параметри цих поверхонь, ми можемо отримати різні кольори, які ми бачимо в колірному спектрі навколо нас.

Колірна модель HSV є найточнішою кольоровою моделлю за умови, що люди сприймають кольори. Те, як люди сприймають кольори, не схоже на RGB або CMYK. Це просто основні кольори, злиті для створення спектру. H означає відтінок, S означає насиченість, а V означає значення. Уявіть собі конус зі спектром від червоного до синього зліва направо, а від центру до краю зростає інтенсивність кольору. Знизу вгору збільшується яскравість. Таким чином, у центрі верхнього шару виходить білий колір. Піктографічне зображення подано на рисунку 1.7.

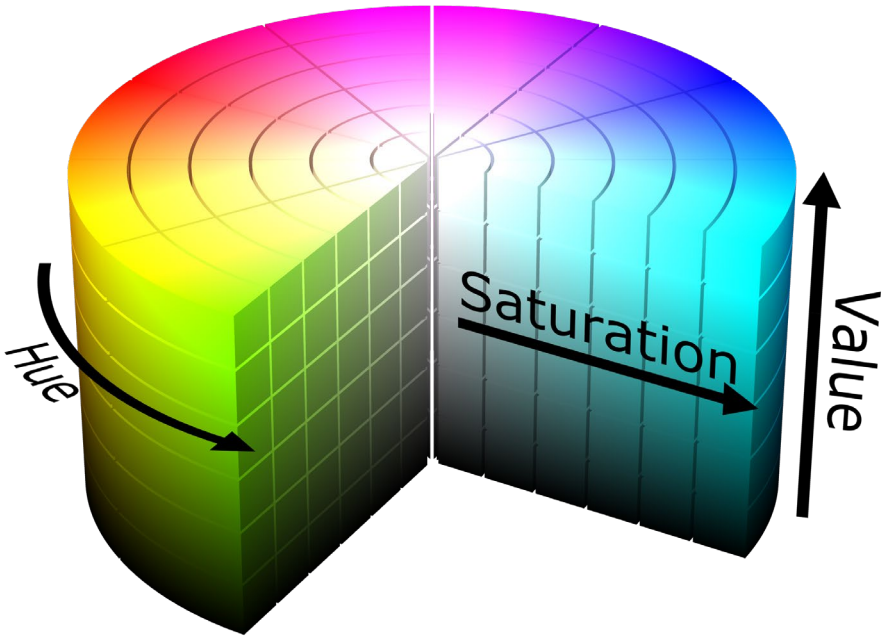


Рисунок 1.7 – Колірна модель HSV

Hue: відтінок вказує, під яким кутом дивитися на циліндричний диск. Відтінок представляє колір. Значення відтінку коливається від 0 до 360 градусів.

Saturation: значення насиченості вказує нам, скільки потрібно додати відповідного кольору. Насиченість 100% означає, що додається повністю чистий колір, тоді як насиченість 0% означає, що колір не додається, що призводить до відтінків сірого.

Value: значення представляє яскравість щодо насиченості кольору. значення 0 означає повну темряву, а значення 100 означає повну яскравість і залежить від насиченості.

1.4 Графи

У галузі теорії графів, граф (також відомий як мережа) – це математична структура, що складається з набору об’єктів, у якому деякі пари цих об’єктів з’єднані або пов’язані певним чином. Для заданого графа G це зазвичай формалізується як $G=(V,E)$, де V – набір вершин (або вузлів), які представляють об’єкти, а E – набір ребер (або зв’язків), які представляють зв’язки серед пар об'єктів. Два вузли, з’єднані ребром, також називаються сусідніми вузлами.

1.4.1 Різновид графів

Орієнтований граф – це граф, де кожне ребро має поняття напрямку, тобто ребра мають початок і призначення, тому для даної пари вузлів a, b є два можливих ребра (a,b) і (b,a) які можуть бути частиною графа. З іншого боку, у неорієнтованому графі немає поняття напрямку, а зв’язок між вузлами визначено як симетричне, тому ребро (a,b) є таким самим, як ребро (b,a).

Зважений граф – це граф, кожне з його ребер має відповідну числову вагу. Ця вага відповідає будь-якій відповідній інформації про з'єднання, такій як відстань, вартість або пропускна здатність. З іншого боку, незважений граф – це граф без пов'язаної інформації про вагу, тому всі з'єднання вузлів вважаються рівними.

Граф може бути будь-якою комбінацією орієнтованого/неорієнтованого та зваженого/незваженого, оскільки ці властивості незалежні.

1.5 Графи видимості

Граф видимості (натуральний) часового ряду T – це граф із n вузлами (що відповідають n спостереженням у T), де межа між двома вузлами існує тоді і тільки тоді, коли немає перешкод, при відстеженні лінії між верхньою частиною відповідних смужок на гістограмі T [8]. Інтуїтивно зрозуміло, що якщо ми вважаємо гістографічну діаграму пейзажом, для кожного вузла ми зв'яжемо його з усіма іншими вузлами (рис. 1.8).

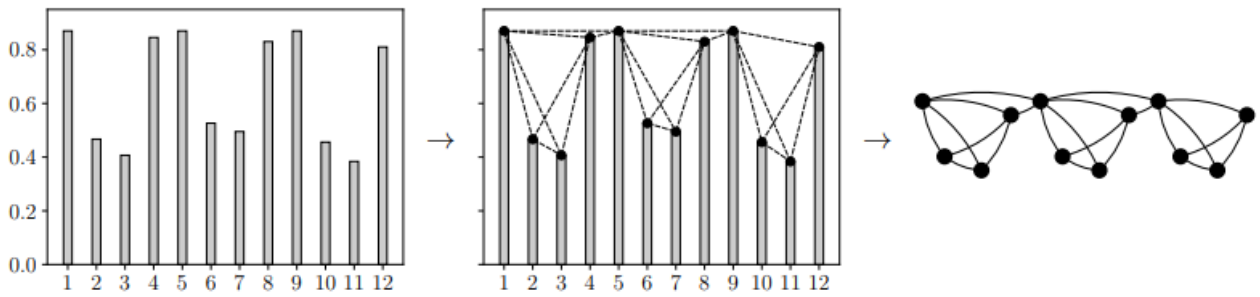


Рисунок 1.8 – Стовпчаста діаграма для прикладу часового ряду (ліворуч), лінії без перешкод (у центрі) та отриманий графік видимості (праворуч)

Іншими словами, у графі видимості $G_T = (V, E)$ з T для кожної пари вузлів $a, b \in V$ із пов'язаними точками даних (t_a, y_a) і (t_b, y_b), з $t_a < t_b$, є ребром

$a, b \in E$ тоді і тільки тоді, коли немає проміжної точки даних, яка перекриває лінію прямої видимості від (t_a, y_a) до (t_b, y_b). Математично, ребро (a,b) є E тоді і тільки тоді, коли всі точки $(t_c, y_c) \in T$ з $t_a < t_c < t_b$ знаходяться нижче двовимірної лінії, яка проходить через точки (t_a, y_a) і (t_b, y_b). Тобто всі проміжні точки (t_c, y_c) повинні виконувати наступну нерівність:

$$y_c < \frac{y_b - y_a}{t_b - t_a}(t_c - t_a) + y_a.$$

1.6 Класифікація графів

Завдання класифікації графів передбачити/задати атрибут кожному графу в колекції графів (рис. 1.9). Наприклад, позначення кожного графіка категоріальним класом (бінарна класифікація або багатокласова класифікація) або прогнозування безперервного числа (регресія). Це можливо зробити при умові, коли модель навчається за допомогою підмножини графів, які мають мітки базової істинності.

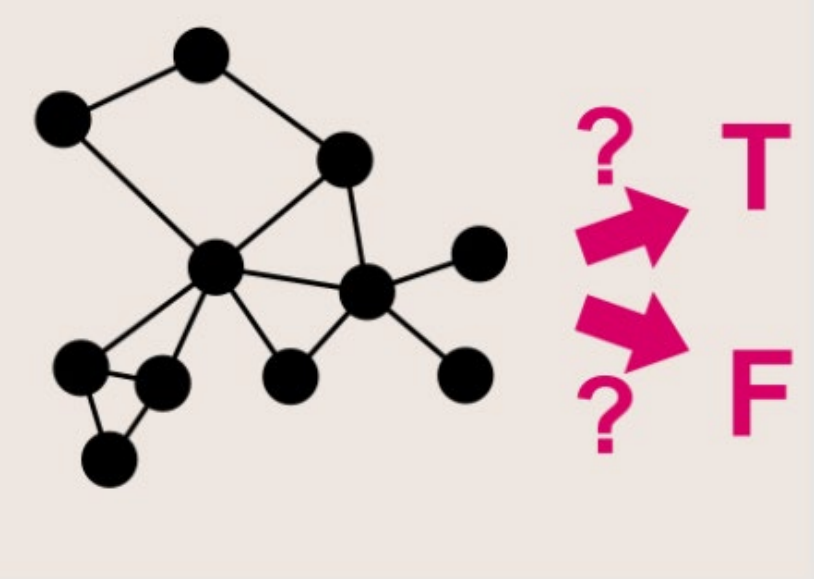


Рисунок 1.9 – Класифікація графів

Знаходження таких спільнот часто є обчислювально складним завданням, і вивчається та використовується багато різних ідей і методів, таких як алгоритми, засновані на мінімальних скороченнях, оптимізація модульності, моделювання відпалу, розкладання власних векторів (спектральна теорія графів) [16].

1.7 Формальна та змістовна постановка задачі

1.7.1 Змістовна постановка задачі

Розробити алгоритм, який класифікує різнокольорові зображенні отримані на основі графів видимості. Найбільшою складністю в машинному навчанні є класифікація часових рядів через упорядкований характер часових даних. Отже, результативні алгоритми повинні скористатися цим фактом, щоб досягти успіху. Відмінності між кількома категоріями даних вимагають пошуку ознак для визначення категорії фрактальної реалізації. Згорткові нейронні мережі відповідають за пошук таких ознак. Фрактальні реалізації використовуються для створення модельних даних, які потім використовуються в дослідженнях, зосереджених на графах природної видимості. Це включає граф натуральної видимості броунівського руху, який представлено графом видимості у виді зображень. Класифікація виконано за допомогою згорткових нейронних мереж, створеної за допомогою TensorFlow і Keras.

1.7.2 Формальна постановка задачі

Нехай X – множина описань об’єктів, де об’єкт – це графічний вигляд часового ряду, котрий отриманий як зображення з симетричної матриці суміжності.

Часовий ряд представляється як послідовний набір часових моментів $\{t_i, i = 1,..., N\}$. Цей набір відповідає часовому ряду, який виглядає так $\{x(t_i) = t_{i+1} - t_i, i = 1,..., N-1\}$. Точки t_i відзначаються на площині на осі X, від яких у перпендикулярному напрямі будуються відрізки довжиною $x(t_i)$. Вузлами графа будь-якого графа видимості являються вершини побудованих вертикальних відрізків. Для графа з n вершинами та m ребрами матрицею суміжності буде матриця $B = [b_{ij}]$ розміру $n \cdot n$, де $b_{ij} = 1$, якщо існує ребро, що йде від вершини i до вершини j, та $b_{ij} = 0$ у іншому випадку.

Нехай Y – множина номерів (чи найменувань) класів. Існує невідома цільова залежність – відображення $y^*: X \to Y$, значення якої відомі тільки на об'єктах кінцевої навчальної вибірки:

$$X^m = \{(x_1, y_1), ..., (x_m, y_m)\},$$

де множина елементів навчальної вибірки розмірністю m.

Потрібно побудувати алгоритм здатний визначити належність довільного об'єкта $x \in X$ до класу $y \in Y$.

1.8 Постановка задач дослідження

Метою цієї кваліфікаційної роботи є виявлення способу візуалізації даних на основі часових рядів, який здатний класифікувати зображення фрактальних реалізацій за допомогою нейронних мереж.

Для досягнення поставлених цілей необхідно виконати наступні кроки:

– дослідити та вивчити основні поняття та характеристики графів;

– дослідити та вивчити методи побудови графів видимості на основі часових рядів;

– дослідити та вивчити методи представлення комп’ютерної графіки;

– розробити програмну реалізацію для генерації зразків фрактального броунівського руху у середовищі Python;

– розробити програмну реалізація для побудови графів видимості у середовищі Python;

– розробити програмну реалізація для побудови зображення з отриманих графів видимості у середовищі Python;

– побудувати нейронну мережу на основі бібліотеки нейронної мережі Keras для класифікації зображень;

– виконати обчислювальні досліди;

– зробити висновки.

2 АЛГОРИТМ ГРАФІЧНОГО ПОДАННЯ ФРАКТАЛЬНИХ РЕАЛІЗАЦІЙ

2.1 Алгоритми побудови графів видимості

При побудові графа видимості, часовий ряд представляється як послідовний набір моментів часу $\{t_i, i = 1,..., N\}$. Наступним кроком зіставляється цьому набору часовий ряд $\{x(t_i) = t_{i+1} - t_i, i = 1,..., N-1\}$. Точки t_i відзначаються на площині на осі X, від яких у перпендикулярному напрямі будуються відрізки довжиною $x(t_i)$. Вузлом вважається найвища точка побудованого перпендикуляра в точці t_i, тобто кількість вузлів дорівнює кількості точок.

2.1.1 Алгоритм побудови графа натуральної видимості

Зв’язок між вершинами NVG-графа вважається існуючим, якщо пряма, що з’єднує вершини відрізків не пересікає жодного з побудованих вертикальних відрізків, що знаходяться між ними (рис. 2.1) [8].

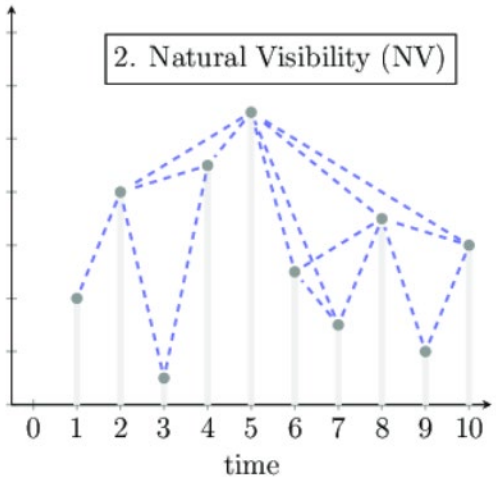


Рисунок 2.1 – Алгоритм натуральної видимості, застосований до часового ряду з 10 точок періодичного часового ряду

На рисунку 2.2 схематично зображений критерій видимості для кута зору $\alpha = \frac{\pi}{2}$. Зв’язки, що відповідають кутам, меншим за кут зору, наприклад α_1, позначені товстими лініями, а ті, що відповідають більшим кутам зору, наприклад α_2, тонкими лініями. Разом обидва ці типи ліній утворюють натуральний граф видимості (NVG).

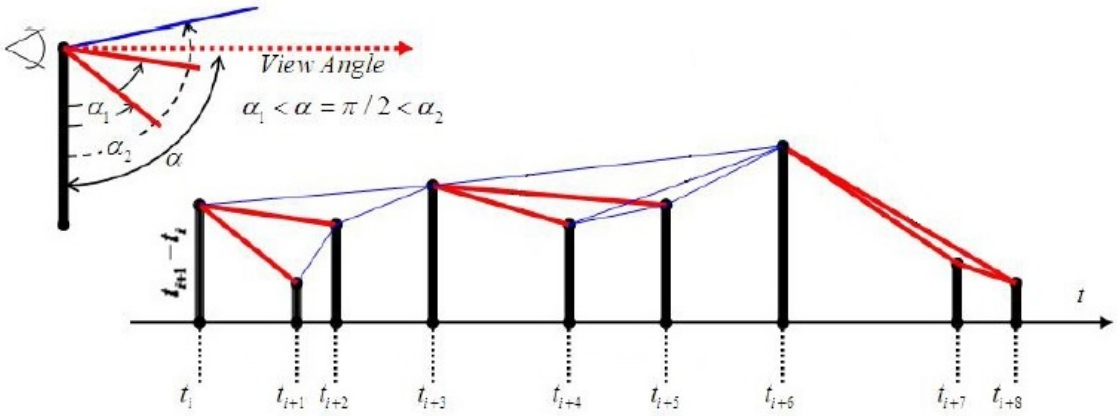


Рисунок 2.2 – Критерій видимості для кута зору α

2.2 Візуалізація графів видимості

Найефективніший спосіб візуалізації графіків – за допомогою моделі вузол-зв’язок, де вузли представляють сутності, а зв’язки – з’єднання. Ці вузли та зв’язки можуть бути чим завгодно – транзакціями між обліковими записами, пристроями в мережі або телефонними дзвінками між друзями.

Неважливо, наскільки великі чи малі дані, що вони містять і де зберігаються. Поки існують зв’язки, які потрібно зрозуміти, візуалізація графіків буде корисною.

Для комп'ютерної обробки та аналізу граф у вигляді вузлів та ребер не підходить. Але комп'ютерна обробка добре справляється з картинками та пікселями в них. Найбільш поширеним способом візуального представлення графі-

ків є їх малювання. Малюнок графа відображає кожну вершину на точку на площині (зазвичай малюється у вигляді маленького кола або іншої форми), а кожне ребро – на криву або відрізок прямої лінії між двома вершинами.

В інформатиці, графи зазвичай представлені однією з двох стандартних структур даних: матрицями суміжності та списками суміжності. На високому рівні обидві структури даних є масивами, індексованими вершинами; це вимагає, щоб кожна вершина мала унікальний цілий ідентифікатор від 1 до V. Формально ці цілі числа є вершинами.

2.2.1 Матриця суміжності

Кращим способом представлення графа є матриця суміжності. Говорять, що вершини i та j у графі суміжні, якщо існує ребро, що їх з'єднує. Говорять, що два ребра суміжні, якщо вони мають спільну вершину. Тоді для графа з n вершинами та m ребрами матрицею суміжності буде матриця $B=[b_{ij}]$ розміру $n \cdot n$, де $b_{ij}=1$, якщо існує ребро, що йде від вершини i до вершини j, та $b_{ij}=0$ у іншому випадку [17]. Оскільки у неорієнтованому графі ребро $\{i, j\}$ веде як від i до j, так і від j до i, тому матриця суміжності такого графа завжди є симетричною. Це проілюстровано на рисунку 2.3, де для графів побудовано матриці суміжності.

(а)

.	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

(б)

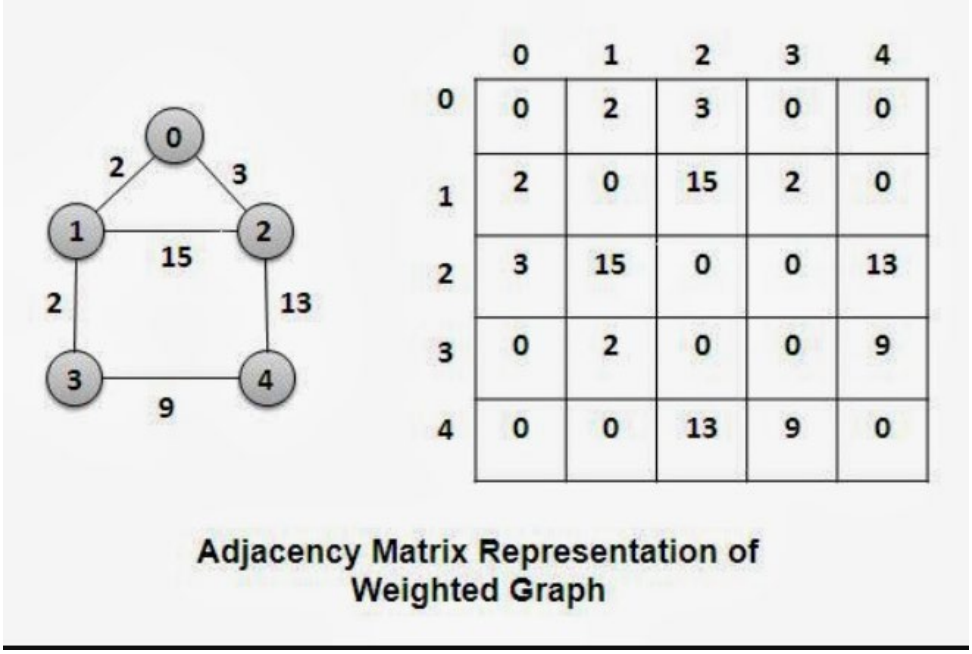
.	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	1
6	0	0	0	1	1	0

Рисунок 2.3 – Матриця суміжності:
а) орієнтованого графа; б) неорієнтованого графа

Основною перевагою матриці суміжності є той факт, що за один крок можна отримати ребро з x у y.

2.2.2 Матриця суміжності для зваженого графа

У разі зваженого графа елемент матриці суміжності s_{ij} дорівнює числу w, якщо існує ребро між вершинами v_i та v_j з вагою w. Елемент s_{ij} дорівнює нулю, якщо ребер між вершинами v_i та v_j не існує.



	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Рисунок 2.4 – Представлення матриці суміжності зваженого графа

2.2.3 Алгоритм побудови зображення на основі матриці суміжності

При побудові зображення (рис. 2.5), отриманого з часового ряду, граф представляється у вигляді матриці суміжності.

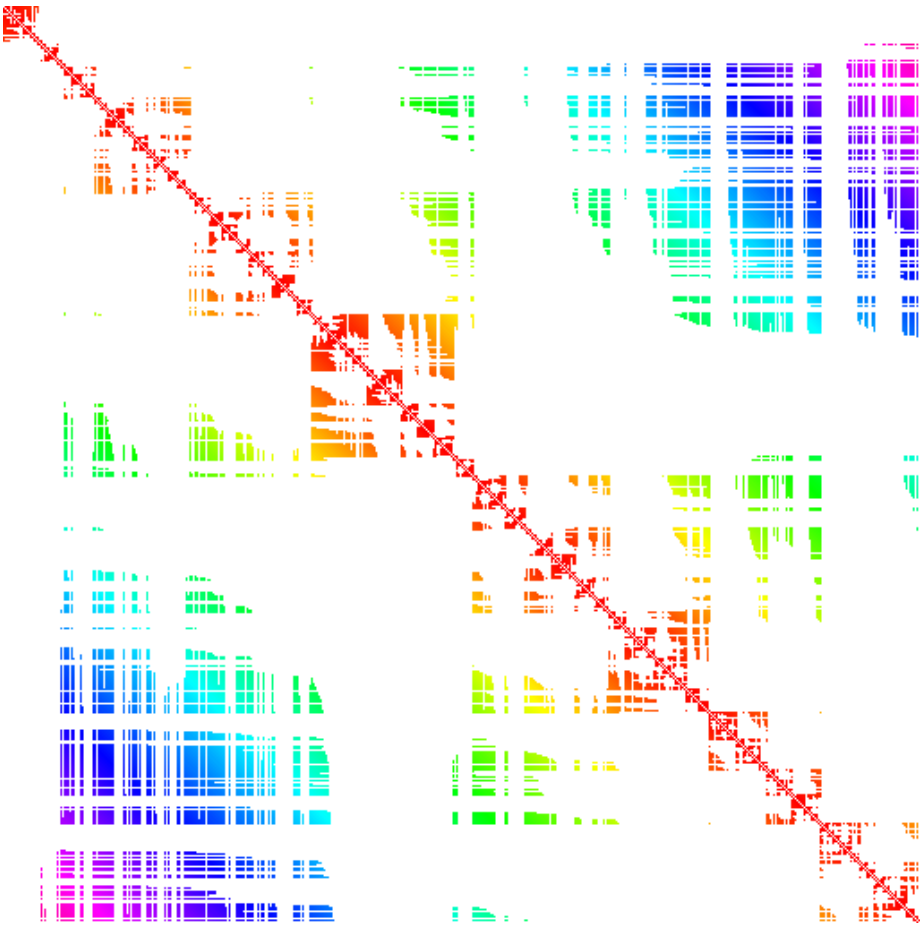


Рисунок 2.5 – Зображення отримане з матриці суміжності

Кожний піксель зображення – це елемент матриці суміжності. Значення елементу зазвичай є 0 або 1, тобто зображення буде отримане чорно-біле. Але для зваженого графа кожному ребру відповідає деяке число (наприклад, довжина дороги або вартість проїзду) – вага ребра. В цьому випадку замість одиниць в матриці суміжності зберігають ваги ребер, а при відсутності ребра зберігають спеціальне значення, наприклад, в багатьох задачах зручно при відсутності ребра зберігати дуже велике число – «нескінченість». Для графа з вагами на дугах, замість 1 заноситься вага ребра. Таким чином можна отримати не тільки чорно-біле, а і різнокольорове зображення (рис. 2.8).

Для того щоб перефарбувати піксель, використовується колірна модель HSV, де H означає відтінок. І цьому відтінку прирівнюється значення ребра, нормалізоване до інтервалу $[0,360]$, де 0 – це мінімальне значення, яке приймає відтінок, а 360 – максимальне.

2.3 Згорткова нейронна мережа

Згорткові нейронні мережі (CNN) – це тип штучних нейронних мереж (ANNs), які показали хорошу ефективність у різноманітних візуальних завданнях, таких як класифікація зображень, сегментація зображень, пошук зображень, виявлення об’єктів, створення підписів до зображень, розпізнавання обличчя, оцінка пози, розпізнавання дорожніх знаків, обробка мовлення, передача нейронних стилів і так далі». Згорткова нейронна мережа (CNN) – це глибока нейронна мережа (DNN), яка використовується для аналізу візуальних зображень у DL і може бути описана наступною формулою

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

де f – вихідна матриця зображення;

g – ядро (матриця) згортки.

Недоліком використання ANN для класифікації зображень є велика кількість обчислень, розгляд локальних пікселів так само, як пікселів, розташованих далеко один від одного, і чутливість до розташування об’єкта на зображенні. Архітектура CNN складається з серії дискретних рівнів, які використовують диференційовану функцію для перетворення вхідного обсягу на вихідний. Шари бувають різних форм і розмірів.

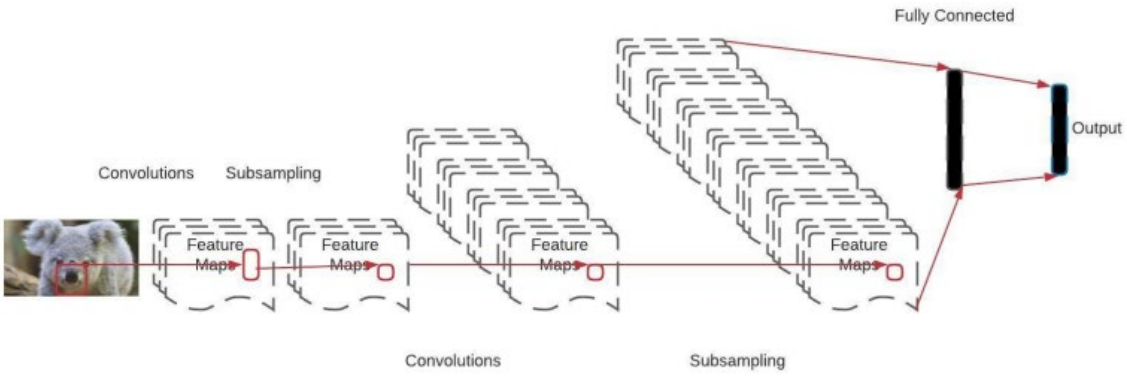


Рисунок 2.6 – Структура згорткової нейронної мережі

CNN – це мережа, яка складається з вхідного рівня, прихованих рівнів і вихідного рівня. Функція активації та кінцева згортка прямої нейронної мережі приховують входи та виходи будь-яких середніх рівнів. Згорткові шари включені в приховані шари згорткової нейронної мережі. Типовим є використання шару, який виконує скалярний добуток ядра згортки та вхідної матриці шару. Вхідними даними для CNN є тензор із формою.

Структура зорової кори головного мозку тварин відображена в сполучній структурі між нейронами. Біологічна діяльність впливала на згорткові мережі так само, як це робить зв'язковий патерн між нейронами. Окремі нейрони кори реагують виключно на стимули, які потрапляють у рецептивне поле, обмежену ділянку поля зору. Рецептивні поля різних нейронів частково перекриваються, що дозволяє їм охоплювати все поле зору. У порівнянні з іншими алгоритмами класифікації зображень, CNN потребують надзвичайно мало попередньої обробки (рис. 2.7).

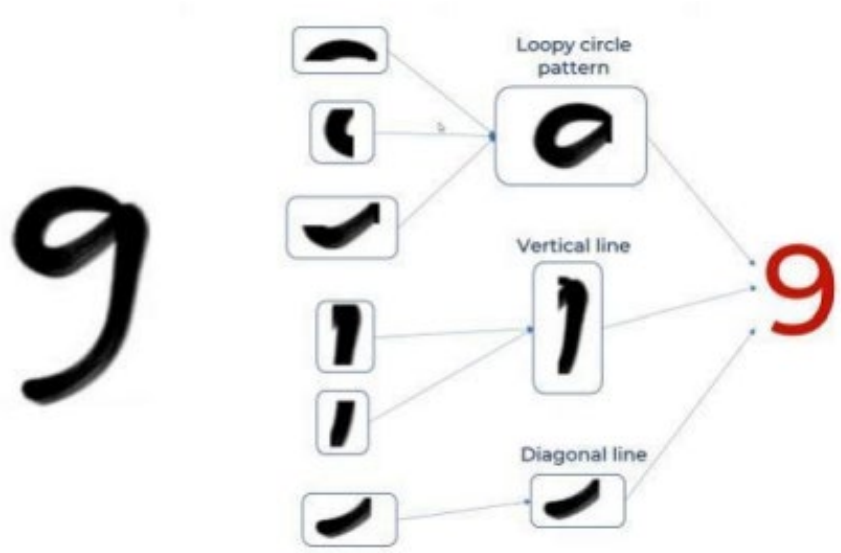


Рисунок 2.7 – CNN на рукописних цифрах

Карти ознак CNN фіксують ефект застосування фільтрів до вхідного зображення. Іншими словами, результатом кожного шару є карта ознак. Мета перевірки карти ознак для конкретного вхідного зображення полягає в тому, щоб краще зрозуміти, як наш CNN знаходить ознаки.

Це означає, що, на відміну від інших методів, мережа використовує автоматичне навчання для вдосконалення фільтрів (або ядер) (рис. 2.8). Ключовою перевагою є той факт, що виділення ознак не покладається на минулі знання чи взаємодію з людьми.

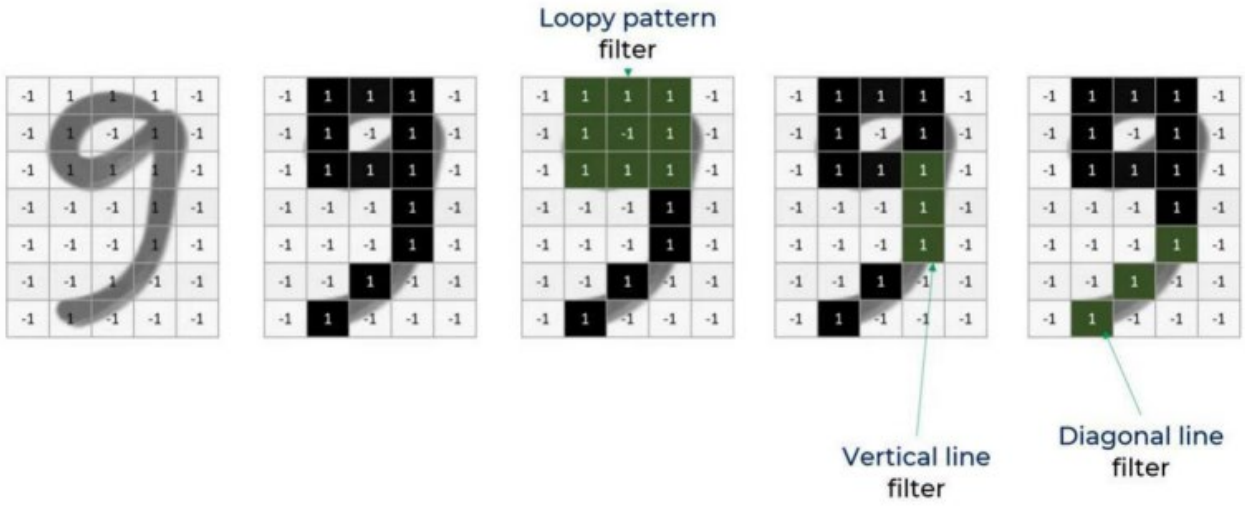


Рисунок 2.8 – Концепція фільтрів

Модель згорткової мережі складається з трьох типів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling, підвибірка) верстви і прошарки «звичайної» нейронної мережі – персептрона [18].

Архітектура згорткових нейронних мереж реалізує три ідеї, які забезпечують інваріантність мережі до невеликих зрушень, змін масштабу і спотворень:

– кожен нейрон отримує вхідний сигнал від локального рецептивного поля (local receptive fields) у попередньому шарі, що забезпечує локальну двовимірну зв’язність нейронів;

– кожен прихований шар мережі складається з множини карт ознак, на яких всі нейрони мають загальні ваги (shared weights), що забезпечує інваріантність до зміщення і скорочення загальної кількості вагових коефіцієнтів мережі;

– за кожним шаром згортки слідує обчислювальний шар, який здійснює локальне усереднення та підвибірку, що забезпечує зменшення розширення для карт ознак.

Робота згорткової нейронної мережі забезпечується двома основними елементами.

– фільтри (filters) (визначники ознак);

– карти ознак (feature maps).

Фільтр – це невелика матриця, що представляє ознаку, яку необхідно знайти на вихідному зображенні. За допомогою верхнього фільтра визначаються частини вихідного зображення з вертикальними лініями, нижній фільтр служить для визначення частин зображення з горизонтальними лініями.

Безпосередньо процес визначення заснований на операції згортки фільтром оригінального зображення. Результати згортки, які визначають місце розташування ознак вихідного зображення, називаються картами ознак.

Мета процесу згортки – зменшити розмірність карти ознак до такої міри, щоб з повним набором ознак могла працювати мережа прямого поширення (в більшості випадків багатошаровий персептрон).

Згортковий шар реалізує ідею локальних рецептивних полів, тобто кожен вихідний нейрон з’єднаний тільки з певною (невеликою) областю вхідної матриці і таким чином моделює деякі особливості людського зору.

2.3.1 Згортковий шар

Центральним для згорткової нейронної мережі є згортковий рівень, який дає назву мережі. Цей шар виконує операцію під назвою «згортка».

У контексті згорткової нейронної мережі згортка – це лінійна операція, яка передбачає множення набору вагових коефіцієнтів на вхідні дані, подібно до традиційної нейронної мережі. Враховуючи те, що методика була розроблена для двовимірного введення, множення виконується між масивом вхідних даних і двовимірним масивом ваг, що називається фільтром або ядром.

Фільтр менший за вхідні дані, а тип множення, застосований між фрагментом розміру фільтра вхідних даних і фільтром, є скалярним добутком. Скаляр-

ний добуток – це поелементне множення між фрагментом розміру фільтра введення та фільтра, який потім підсумовується, що завжди призводить до єдиного значення.

З математичної точки зору згортка – це операція над парою матриць $A_{n_a \times m_a}$ та $B_{n_b \times m_b}$ результатом якої є матриця $C = A * B$:

$$C_{i,j} = \sum_{u=0}^{n_b-1} \sum_{v=0}^{m_b-1} A_{i+u,j+v} B_{u,v},$$

$$i = 1,...,(n_a - n_b + 1),\ j = 1,...,(m_a - m_b + 1),$$

де C – матриця розміром $(n_a - n_b + 1) \times (m_a - m_b + 1)$;

B – фільтр або ядро згортки.

Використання фільтра, меншого за вхідний, є навмисним, оскільки це дозволяє множити той самий фільтр (набір ваг) на вхідний масив кілька разів у різних точках вхідних даних. Зокрема, фільтр систематично застосовується до кожної частини вхідних даних, що перекривається, або ділянки розміру фільтра, зліва направо, зверху вниз.

Це систематичне застосування одного і того ж фільтра на зображенні є потужною ідеєю. Якщо фільтр призначений для виявлення певного типу ознак у вхідних даних, то систематичне застосування цього фільтра до всього вхідного зображення дає фільтру можливість виявити цю функцію будь-де на зображенні. Цю здатність зазвичай називають інваріантністю перекладу, наприклад загальний інтерес до того, чи присутня ознака, а не до того, де вона була присутня.

Результатом одноразового множення фільтра на вхідний масив є одне значення (рис. 2.9–2.10). Оскільки фільтр застосовується кілька разів до вхідного масиву, результатом є двовимірний масив вихідних значень, які представляють фільтрацію вхідних даних. Таким чином, двовимірний вихідний масив від цієї операції називається «картою ознак».

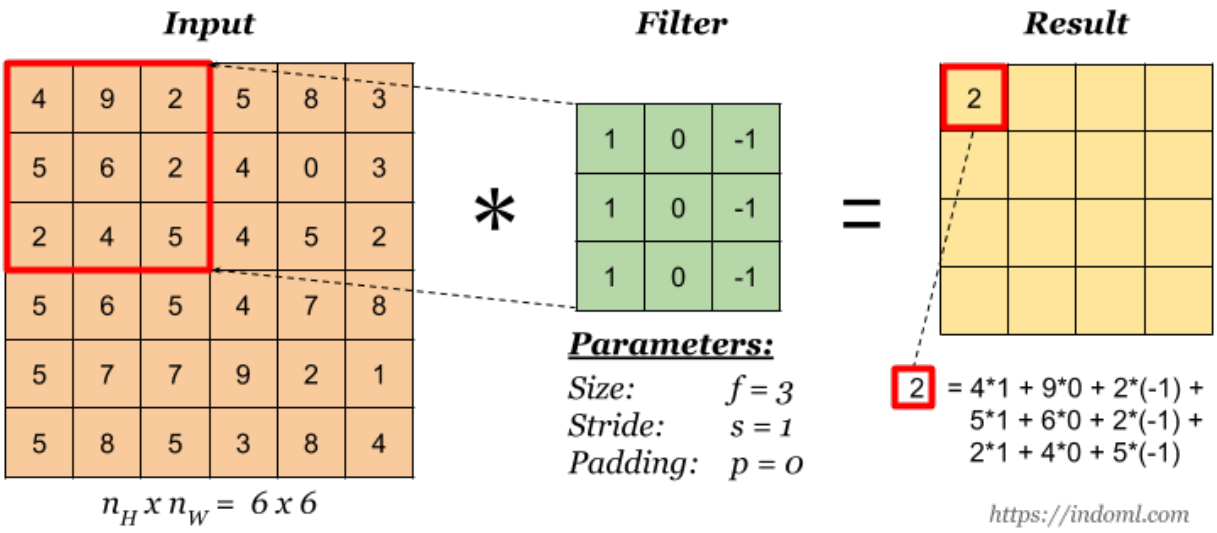


Рисунок 2.9 – Приклад обчислення згортки

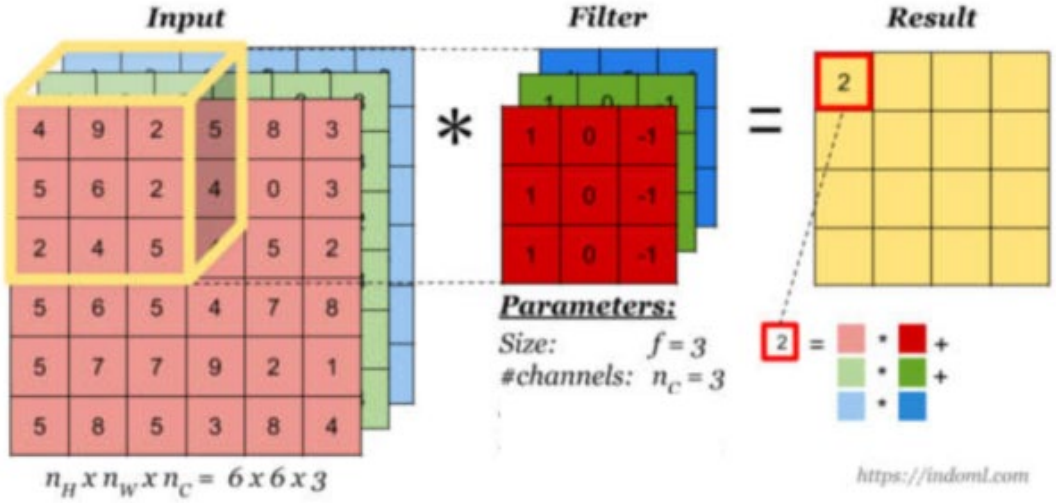


Рисунок 2.10 – Обчислення згортки по багатоканальним даними

Шар згортки включає для кожного каналу свій фільтр, ядро згортки якого обробляє попередній шар за фрагментами. Вагові коефіцієнти ядра згортки невідомі і встановлюються в процесі навчання. Якщо на вхід згорткового шару подаються дані розміром $w_1 \times h_1 \times d_1$ ($w_1 \times h_1$ – розмірність матриці даних, d_1 – кількість каналів), то на виході отримаємо дані розміром $w_2 \times h_2 \times d_2$, де

$$w_2 = \left[\frac{(w_1 + 2p - f)}{s}\right] + 1,\ h_2 = \left[\frac{(h_1 + 2p - f)}{s}\right] + 1,$$

де $f \times f$ – розмірність ядра згортки;

s – страйд;

p – ширина паддінга;

$d_2 = k$ – кількість фільтрів.

Згорткові шари корисні для виділення функцій із зображень, оскільки вони мають справу з просторовою надлишковістю через розподіл ваги. У міру проникнення в мережу функції стають більш ексклюзивними та інформативними, а надмірність зменшується. Насамперед це пов'язано з повторюваними каскадними згортками та стисненням інформації шарами підвибірки. Оскільки надлишковість зменшується, ми отримуємо стиснуте представлення ознак щодо вмісту зображення. Тепер вихідні шари займаються зіставленням цих ознак з необхідними категоріями виводу. Ця функція відображення більше не потребує розподілу ваги, оскільки для прийняття інформативного рішення потрібен увесь вектор ознак. Стандартною практикою є перетворення вивчених функцій із згорткових екстракторів у вектор, який працює як дескриптор зображення. Один із способів полягає в тому, щоб просто переформувати всі активації останнього шару екстрактора ознак в одновимірний тензор [19, 22]. Другий метод полягає у використанні повномасштабного середнього об'єднання. Для карти активації роздільної здатності $h \times w$ середній пул із ядром такої самої роздільної здатності $h \times w$ зменшить карту до скалярного значення, що означає загальну активацію [20, 21]. Таким чином останній шар можна відобразити на векторі ознак. Потім цей вектор з'єднується з вихідним класифікатором. Класифікатор – це стандартний багатошаровий тип, що складається з додаткових прихованих шарів і вихідного шару з необхідною кількістю вихідних нейронів, які відображаються з дескриптора ознак у вихідний простір.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Python як потужний інструмент для машинного навчання

Python – це найпопулярніша високорівнева мова програмування з динамічною семантикою. Він досить простий для роботи: його використання знижує вартість розробки та обслуговування програм. Python вважається найпростішим мовою програмування – саме тому він найпоширеніший.

Машинне навчання – це технологія, яка допомагає додаткам на основі штучного інтелекту навчатися і видавати результати автоматично, без людського втручання. Робота фахівця по машинному навчання полягає в тому, що він повинен збирати, систематизувати і аналізувати дані, а потім на основі отриманої інформації створювати алгоритми для штучного інтелекту.

Python найкраще підходить для виконання таких завдань, тому що він досить зрозумілий в порівнянні з іншими мовами. Більш того, у нього відмінна продуктивність при обробці даних. Одна з основних причин, чому Python використовується для машинного навчання полягає в тому, що у нього є безліч фреймворків, які спрощують процес написання коду і скорочують час на розробку. У наукових розрахунках використовується Numpy, в просунутих обчисленнях – SciPy, в добуванні і аналізі даних – SciKit-Learn. Ці бібліотеки працюють в таких фреймворків, як TensorFlow, CNTK і Apache Spark. Існує фреймворк для Python, розроблений спеціально для машинного навчання – це PyTorch.

Завдяки лаконічності Python і зручності читання він добре підходить для навчання штучного інтелекту при розробці ПЗ. Крім того, Python добре підходить для машинного навчання, тому що самі алгоритми машинного навчання складні для розуміння. При роботі з Python розробнику не потрібно приділяти багато уваги безпосередньо написанню коду: всю увагу він може зосередити на вирішенні більш складних завдань, пов'язаних з машинним навчанням. Простий синтаксис мови Python допомагає розробнику тестувати складні алгоритми з мінімальною витратою часу на їх реалізацію.

Ще одна перевага Python – це велика підтримка та якісна документація. Існує безліч корисних ресурсів про Python, на яких програміст може отримати допомогу і консультацію, перебуваючи на будь-якому етапі розробки.

Наступна перевага Python в машинному навчанні полягає в його гнучкості: наприклад, у розробника є вибір між об'єктно-орієнтованим підходом і скриптами. Python допомагає об'єднувати різні типи даних. Більш того, Python особливо зручний для тих розробників, які більшу частину коду пишуть за допомогою IDE.

3.2 Опис використаних бібліотек для написання програми

3.2.1 Бібліотека Matplotlib

Matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується). Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy і IPython надає можливості, подібні до MATLAB. В даний час пакет працює з декількома графічними бібліотеками, включаючи wxWindows і PyGTK.

Пакет підтримує багато видів графіків і діаграм:

– графіки (line plot);

– діаграми розсіювання (scatter plot);

– стовпчасті діаграми (bar chart) і гістограми (histogram);

– секторні діаграми (pie chart);

– діаграми «Стовбур-листя» (stem plot);

– контурні графіки (contour plot);

– поля градієнтів (quiver);

– спектральні діаграми (spectrogram).

3.2.2 Бібліотека Numpy

NumPy – це основний пакет наукових обчислень на Python. Це бібліотека Python, яка забезпечує багатовимірний об'єкт масиву, різні похідні об'єкти (наприклад, масковані масиви та матриці) та асортимент підпрограм для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з фігурами, сортування, вибір, введення / виведення, дискретні перетворення Фур'є, основна лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого [23].

NumPy повністю підтримує об’єктно-орієнтований підхід, починаючи, знову ж таки, з ndarray. Наприклад, ndarray – це клас, що має безліч методів та атрибутів [23]. Багато його методів відображаються функціями в самому зовнішньому просторі імен NumPy, що дозволяє програмісту кодувати будь-яку парадигму, яка їм більше подобається. Ця гнучкість дозволила діалекту масиву NumPy та класу NumPy ndarray стати фактичною мовою багатовимірного обміну даними, що використовується в Python.

3.2.3 Бібліотека Scikit-learn

Бібліотека Scikit-learn – найпоширеніший вибір для вирішення завдань класичного машинного навчання [24]. Вона надає широкий вибір алгоритмів навчання з учителем і без вчителя. Одне з основних переваг бібліотеки полягає в тому, що вона працює на основі декількох поширених математичних бібліотек, і легко інтегрує їх один з одним. Ще однією перевагою є широка спільнота і докладна документація. Scikit-learn широко використовується для промислових систем, в яких застосовуються алгоритми класичного машинного навчання, для досліджень, а так само для новачків, які тільки робить перші кроки в області машинного навчання.

Scikit-learn спеціалізується на алгоритмах машинного навчання для вирішення завдань навчання з учителем: класифікації (прогноз ознаки, множина допустимих значень якого обмежена) і регресії (прогноз ознаки з речовими значеннями), а також для задач навчання без учителя: кластеризації (розбиття даних по класах , які модель визначить сама), зниження розмірності (подання даних в просторі меншої розмірності з мінімальними втратами корисної інформації) і детектування аномалій [24]. Це – лише базовий список. Крім цього, Scikit-learn містить функції для розрахунку значень метрик, вибору моделей, попередньої обробки даних та інші.

3.2.4 Бібліотека TensorFlow / Keras

Одним з найбільш поширених застосувань TensorFlow і Keras є розпізнавання і класифікація зображень.

TensorFlow – це бібліотека з відкритим вихідним кодом, створена для Python командою Google Brain. TensorFlow компілює безліч різних алгоритмів і моделей, дозволяючи користувачеві реалізувати глибокі нейронні мережі для використання в таких завданнях, як розпізнавання і класифікація зображень, а також обробка природної мови.

TensorFlow – це потужний фреймворк, який функціонує шляхом реалізації ряду вузлів обробки, кожен з яких представляє математичну операцію, а весь ряд вузлів називається «графом».

Говорячи про Keras, це високорівнева API (інтерфейс прикладного програмування), який може використовувати функції TensorFlow (а також інші бібліотеки ML, такі, як Theano) [25]. Keras був розроблений з зручністю і модульність в якості керівних принципів. З практичної точки зору Keras дозволяє реалізувати безліч потужних, але часто складних функцій TensorFlow максимально просто, до того ж він налаштований для роботи з Python без будь-яких серйозних змін або налаштувань.

3.2.5 Бібліотека OpenCV

OpenCV – це величезна бібліотека з відкритим кодом для комп'ютерного зору, машинного навчання та обробки зображень. OpenCV підтримує широкий спектр мов програмування, таких як Python, C++, Java тощо. Він може обробляти зображення та відео, щоб ідентифікувати об'єкти, обличчя чи навіть почерк людини. Коли його інтегровано з різними бібліотеками, такими як Numpy, яка є високооптимізованою бібліотекою для числових операцій, тоді кількість зброї збільшується у вашому арсеналі, тобто будь-які операції, які можна виконувати в Numpy, можна поєднувати з OpenCV.

imread() є одним із найбільш корисних і часто використовуваних методів бібліотеки OpenCV-Python. Використовується для завантаження зображення в програму Python із зазначеного файлу. Він повертає numpy.ndarray (N-вимірний масив NumPy) після успішного завантаження зображення. Цей numpy.ndarray є 3-вимірним масивом, коли завантажене зображення є кольоровим зображенням, і 2-вимірним масивом, коли завантажене зображення є зображенням у градаціях сірого.

Читання зображень у форматі .JPEG залежить від версії бібліотеки OpenCV, встановленої в системі, платформі чи середовищі (наприклад, x86/ARM) тощо. І найголовніше те, що тип зображення визначається не розширенням файлу зображення, а за вмістом numpy.ndarray, який повертає метод cv2.imread().

3.2.6 Бібліотека Collections (Counter)

Counter – це підклас словника для підрахунку хешованих об'єктів. Це колекція, де елементи зберігаються як ключі словника, а їх кількість зберігається як значення словника. Підрахунки можуть бути будь-якими цілими значеннями, включаючи нульові чи від'ємні значення.

3.2.7 Модуль OS

Обробка файлів у Python за допомогою модуля OS включає створення, перейменування, переміщення, видалення файлів та папок, а також отримання списку всіх файлів та каталогів та багато іншого.

В індустрії програмного забезпечення більшість програм обробляють файли: створюють їх, перейменовують, переміщують і так далі. Модуль OS використовується як для роботи з файлами. Він включає безліч методів та інструментів для інших операцій: обробки змінних середовища, управління системними процесами, а також аргументи командного рядка і навіть розширені атрибути файлів, які є тільки в Linux.

3.3 Опис програми

Програмна реалізація класифікації візуалізованих графів видимості побудованих на основі фрактальних реалізацій за допомогою Deep Machine Learning складається з двох програм:

– генерація та візуалізація фрактальних реалізацій (Додаток А);

– класифікація різнокольорових зображень на основі Deep Machine Learning (Додаток Б).

Під час написання програм були використані деякі сторонні бібліотеки для розширення стандартного функціоналу Python: numpy, tensorflow, networkx, scikit-learn та matplotlib.

Перша програма «генерація та візуалізація фрактальних реалізацій» складається с трьох модулів: VisibilityGraphService, BrownianMotionSeriesService, main.

Модуль main відповідає за основну структуру побудови зображень на основі фрактальних реалізацій. Цей модуль є початковою точко. для початку роботи програми. А саме він складається з послідовних викликів методів для отримання бажаного результату.

VisibilityGraphService – модуль, котрий відповідає за роботу з графами. Методи цього модулю дають змогу побудувати графи горизонтальної, динамічної та натуральної видимості, а також надається змога відобразити отриманий граф та зберегти його у файловій системі за вказаною адресою. Додатково передбачені методи для аналізу графа та збереження у CSV-файл.

BrownianMotionSeriesService – модуль, котрий містить деякий функціонал для роботи безпосередньо з часовими рядами та фрактальнимим реалізаціями. За допомогою методів цього модулю можна згенерувати зразки траєкторії фрактального броунівського руху за допомогою методу Девіса Харта, вказавши необов’язкові параметри: тривалість часу (у роках), кількість часових кроків у часовому проміжку та параметр Херста. Будь-який часовий ряд може бути збережений та зчитаний з файлу за вказаною адресою, та у свою чергу відображений як граф чи збережений у файловій системі для подальшого використання чи аналізу.

3.4 Опис роботи додатку та результати обчислювального експерименту

3.4.1 Генерація зразків фрактального броунівського руху

Перший етап даної роботи представляє собою важливу технологію Information graphics, яка стосується аналізу, обробки та перетворення даних у графічне візуальне подання.

Для реалізації першого етапу, спочатку було сформовано файли з даними фрактальних реалізацій та збережені у файловій системі. Кожний часовий ряд має довжину 512 значень. Результати зображено на рисунках 3.1 – 3.3.

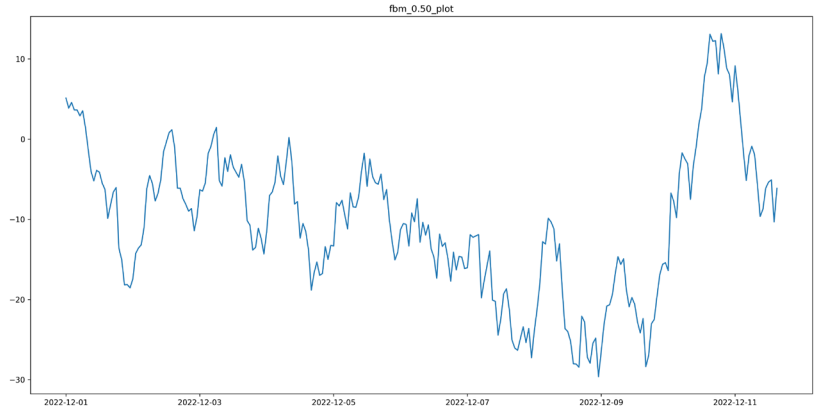


Рисунок 3.1 – Модель фрактального броунівського руху з параметром Херста (0,5)

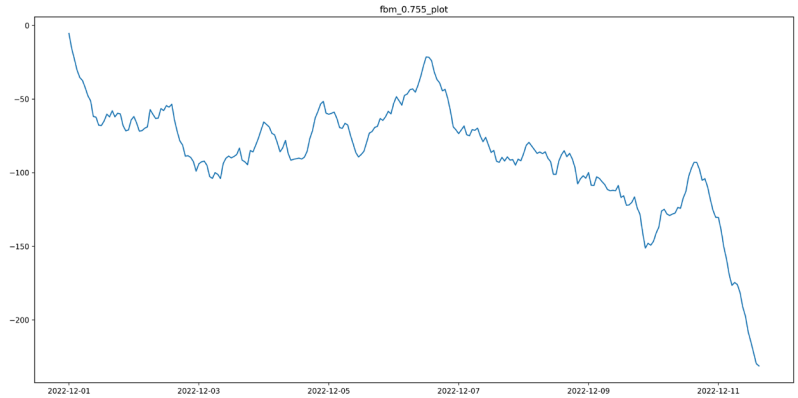


Рисунок 3.2 – Модель фрактального броунівського руху з параметром Херста (0,75)

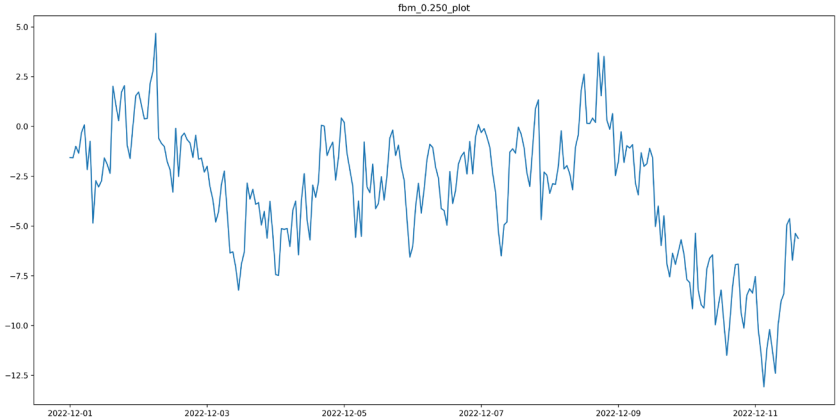


Рисунок 3.3 – Модель фрактального броунівського руху з параметром Херста (0,2)

Реалізація Python симуляції дробового броунівського руху (FBM) з використанням методу Девіса-Харта для генерації зразків фрактального шуму Гауса. FBM виходить шляхом взяття кумулятивних сум відібраного фрактального гаусового шуму.

Цей метод отримує вибірки фрактального гаусового шуму, шляхом вбудовування коваріаційної матриці в «циркулянтну коваріаційну матрицю» розміром $2N$, що дозволяє ефективно обчислювати «квадратний корінь» з коваріаційної матриці. Власні значення цієї матриці використовуються для створення вибірок фрактального гаусового шуму. Цей алгоритм використовує швидке перетворення Фур'є для підвищення ефективності. Складність цього алгоритму має порядок $\mathrm{O}\big(N\log\big(N\big)\big)$.

3.4.2 Візуалізація фрактальних реалізацій

Наступним кроком даної роботи є побудова графа видимості натуральної видимості (NVG) для усіх наведених вище фрактальних реалізацій. Зв'язок між вершинами NVG-графа вважається існуючим, якщо пряма, що з'єднує вершини відрізків, не перетинає жодного з побудованих вертикальних відрізків, що знаходяться між ними. Отримані результати NVG-графів зображено на рисунках 3.4 – 3.6.

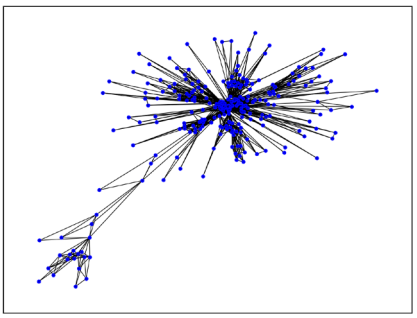


Рисунок 3.4 – NVG-граф для фрактального броунівського руху з параметром Херста (0,5)

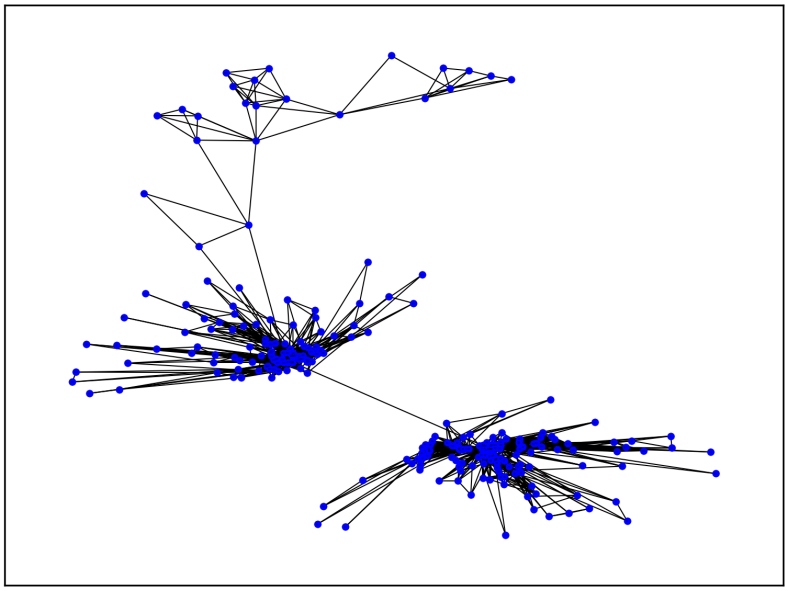


Рисунок 3.5 – NVG-граф для фрактального броунівського руху з параметром Херста (0,75)

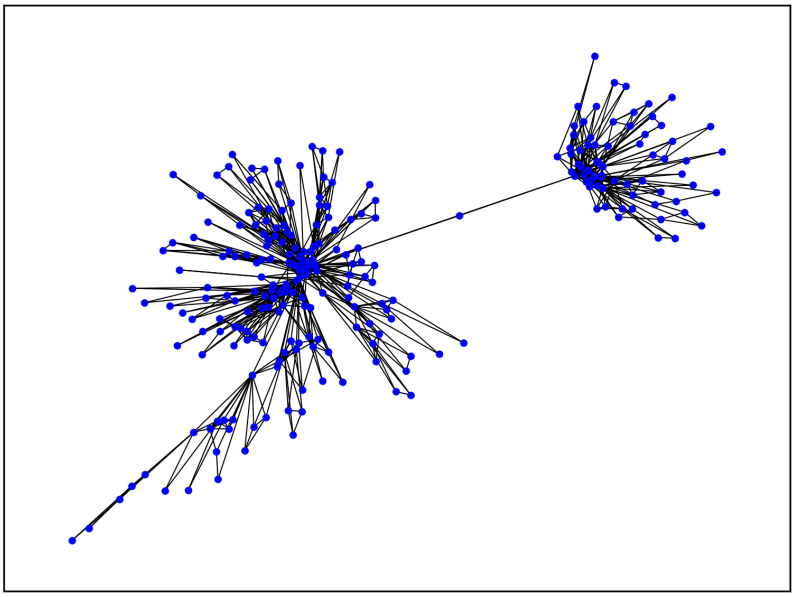


Рисунок 3.6 – NVG-граф для фрактального броунівського руху з параметром Херста (0,2)

Отже, результати побудови NVG-графів показують, що використання графів видимості ефективні для наступного аналізу, оскільки відмінність між графами є помітними та є вірогідність, що згорткова нейронна мережа зможе розрізняти зображення фрактальних реалізацій.

Наступний етап – візуалізація даних (рис. 3.7 – 3.9). Завдяки візуалізації вихідних даних, користувач може переглядати часові ряди, щоб отримати “відчуття” властивостей цих даних.

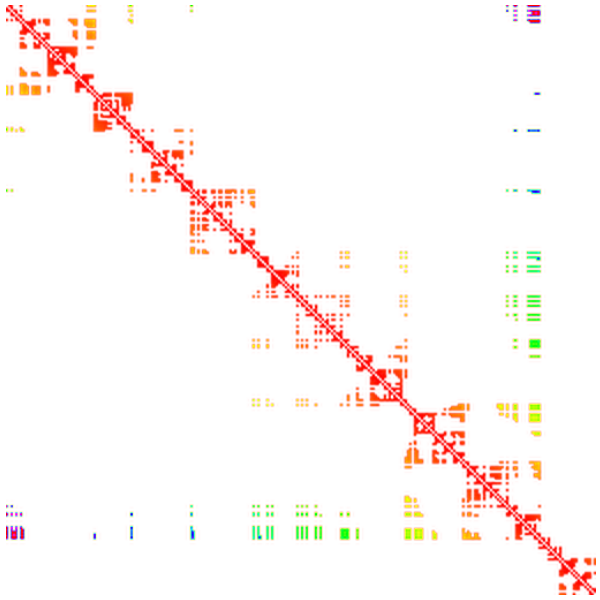


Рисунок 3.7 – Зображення отримане з матриці суміжності
для фрактального броунівського руху
з параметром Херста (0,5)

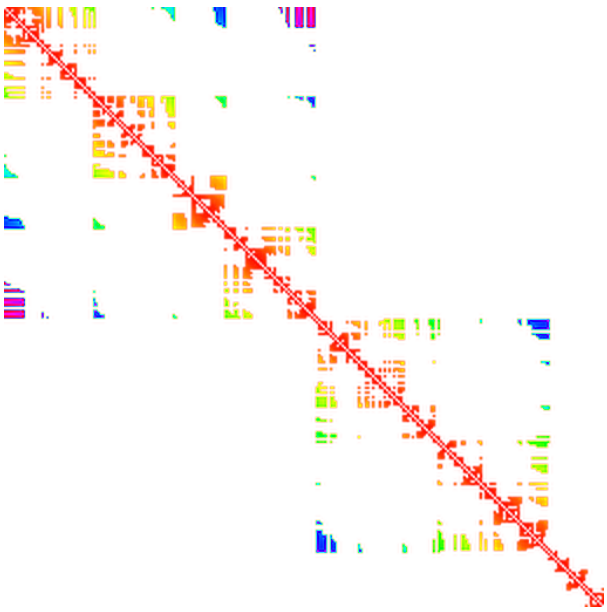


Рисунок 3.8 – Зображення отримане з матриці суміжності
для фрактального броунівського руху
з параметром Херста (0,75)

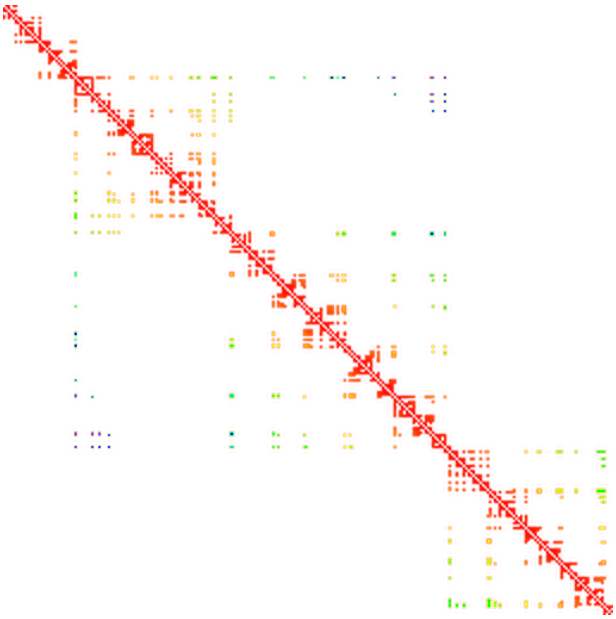


Рисунок 3.9 – Зображення отримане з матриці суміжності
для фрактального броунівського руху
з параметром Херста (0,2)

3.4.3 Класифікатор зображень на основі Deep Machine Learning

Наступний та найголовніший крок даної кваліфікаційної роботи – це класифікація отриманих зображень фрактальних реалізацій. З цим завданням впорається згорткова нейронна мережа. Першим етапом є побудова моделі згорткової нейронної мережі з використанням бібліотеки Keras (рис. 3.10). Представлення структури побудованої моделі представлено на рисунку 3.11.

Після завантаження зображень, представлення зображень у вигляді матриці та розбиття данних на класи, розіб’ємо дані на тестові та тренувальні. Тренувальний набір даних: набір даних призначений для навчання нейронної мережі; тестовий набір даних: набір даних призначений для перевірки ефективності роботи нейронної мережі. В даній задачі було розбито дані на 50% тренувальних та 50% тестових зображень, 300 та 300 зображень відповідно.

```
model = Sequential()
model.add(Conv2D(256, kernel_size=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(128, (2, 2), activation='relu'))
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Dropout(0.4))
model.add(MaxPooling2D(pool_size=(5, 5)))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
```

Рисунок 3.10 – Модель згорткової нейронної мережі

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 512, 512, 256)     1024

 conv2d_1 (Conv2D)           (None, 511, 511, 128)     131200

 dropout (Dropout)           (None, 511, 511, 128)     0

 conv2d_2 (Conv2D)           (None, 509, 509, 128)     147584

 dropout_1 (Dropout)         (None, 509, 509, 128)     0

 max_pooling2d (MaxPooling2D  (None, 101, 101, 128)    0
 )

 dropout_2 (Dropout)         (None, 101, 101, 128)     0

 flatten (Flatten)           (None, 1305728)           0

 dense (Dense)               (None, 128)               167133312

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 167,413,507
Trainable params: 167,413,507
```

Рисунок 3.11 – Структура згорткової нейронної мережі

На рисунку 3.12 бачимо результати навчання на кожній епосі, де loss – функція втрат, accuracy – точність на тренувальній вибірці, val_loss і val_accuracy – функція втрат і точність на тестовій вибірці відповідно.

```
start fit
Epoch 1/13
38/38 [==============================] - 198s 5s/step - loss: 858.8102 - accuracy: 0.2633 - val_loss: 1.0988 - val_accuracy: 0.233
Epoch 2/13
38/38 [==============================] - 197s 5s/step - loss: 1.0835 - accuracy: 0.3933 - val_loss: 1.0476 - val_accuracy: 0.3167
Epoch 3/13
38/38 [==============================] - 197s 5s/step - loss: 0.9374 - accuracy: 0.5467 - val_loss: 0.9576 - val_accuracy: 0.6467
Epoch 4/13
38/38 [==============================] - 197s 5s/step - loss: 0.7461 - accuracy: 0.7033 - val_loss: 0.7764 - val_accuracy: 0.7567
Epoch 5/13
38/38 [==============================] - 197s 5s/step - loss: 0.5364 - accuracy: 0.7833 - val_loss: 0.6885 - val_accuracy: 0.7167
Epoch 6/13
38/38 [==============================] - 196s 5s/step - loss: 0.2003 - accuracy: 0.9400 - val_loss: 0.4373 - val_accuracy: 0.9133
Epoch 7/13
38/38 [==============================] - 196s 5s/step - loss: 0.1193 - accuracy: 0.9567 - val_loss: 0.1793 - val_accuracy: 0.9900
Epoch 8/13
38/38 [==============================] - 197s 5s/step - loss: 0.1228 - accuracy: 0.9533 - val_loss: 0.4342 - val_accuracy: 0.8767
Epoch 9/13
38/38 [==============================] - 197s 5s/step - loss: 0.0696 - accuracy: 0.9800 - val_loss: 0.1324 - val_accuracy: 0.9933
Epoch 10/13
38/38 [==============================] - 197s 5s/step - loss: 0.0961 - accuracy: 0.9700 - val_loss: 0.1898 - val_accuracy: 0.9767
Epoch 11/13
38/38 [==============================] - 196s 5s/step - loss: 0.1166 - accuracy: 0.9667 - val_loss: 0.1146 - val_accuracy: 0.9733
Epoch 12/13
38/38 [==============================] - 197s 5s/step - loss: 0.0482 - accuracy: 0.9833 - val_loss: 0.0923 - val_accuracy: 0.9833
Epoch 13/13
38/38 [==============================] - 196s 5s/step - loss: 0.0092 - accuracy: 1.0000 - val_loss: 0.0229 - val_accuracy: 0.9933
```

Рисунок 3.12 – Процес навчання нейронної мережі

Модель закінчила навчання нейронної мережі. Бачимо, що точність на тестовій вибірці вийшла val_accuracy: 0.99, тобто 99.33% точність класифікації зображень фрактальних реалізацій (рис. 3.13 – 3.15). Можна зробити висновок, що нейронна мережа працює правильно. Щоб поліпшити нейронну мережу – краще розпізнавати зображення, необхідно збільшити кількість даних та їх розмір та повторити попередні кроки.

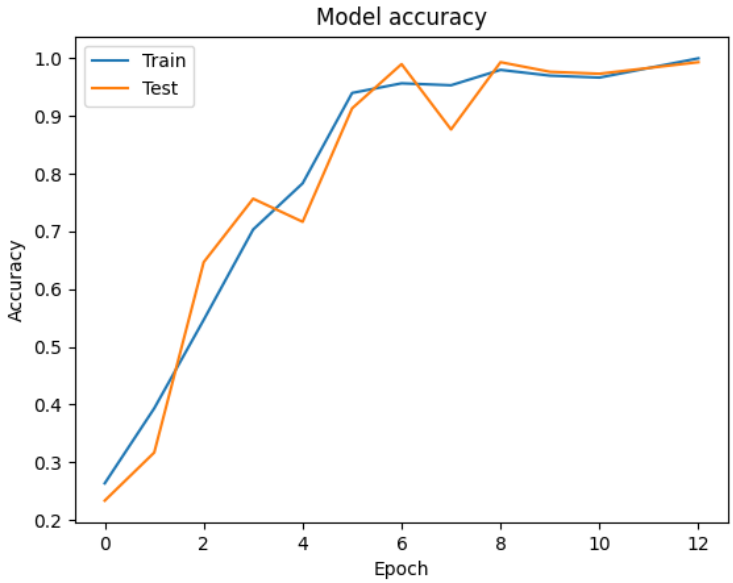


Рисунок 3.13 – Графік точності

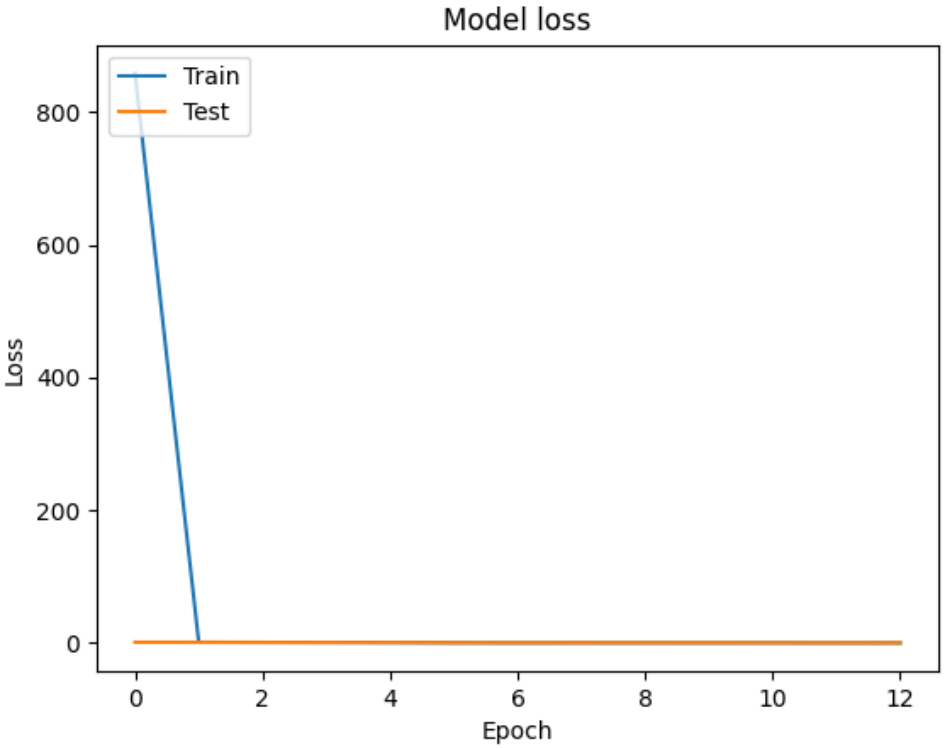


Рисунок 3.14 – Графік функції втрат

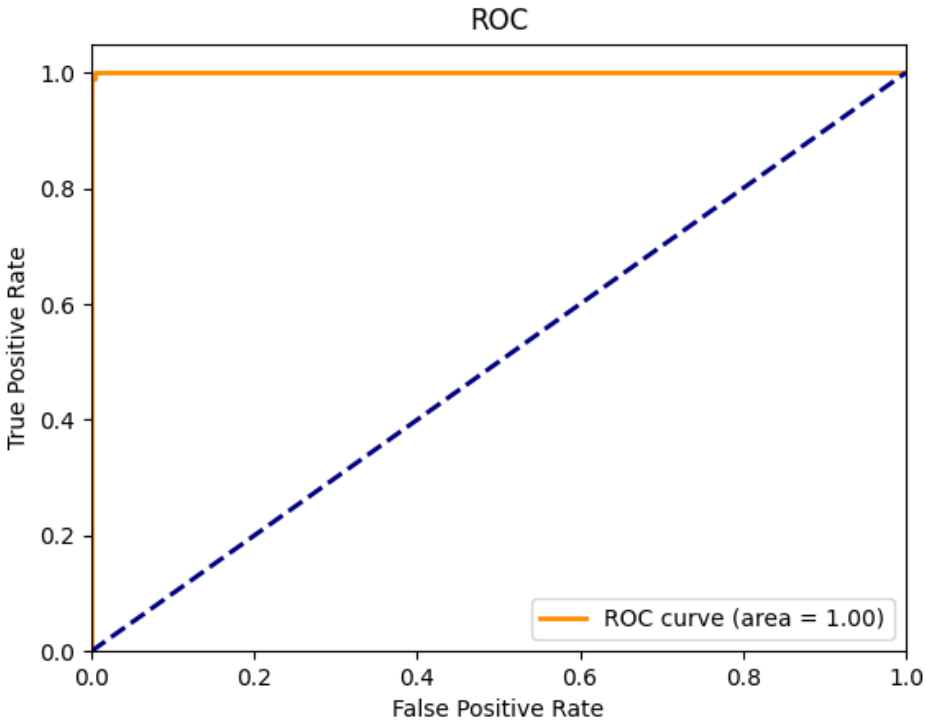


Рисунок 3.15 – ROC-крива

ВИСНОВКИ

У даній кваліфікаційній роботі був проведений аналіз існуючих алгоритмів аналізу фрактальних реалізацій, що розширило ряд можливостей та знань в проектуванні та розробці в подібних сферах застосування. Зразки симуляції фрактального броунівського руху були використані як вхідні часові ряди. Часові ряди були поділені на кілька класів залежно від показника Херста. Згорткова нейрона мережа використовувалась для класифікації зразків. Результати класифікації показали перевагу методу машинного навчання над традиційним методом оцінка показника Херста, особливо з короткою довжиною часовий ряд.

Для розроблення відповідного програмного забезпечення були розв’язані наступні задачі:

– генерація зразків симуляції фрактального броунівського руху;

– побудова натурального графа видимості;

– побудова матриці суміжності;

– перетворення матриці суміжності в кольорове зображення;

– створення нейронної мережі для класифікації часових рядів;

– аналіз результатів.

Отже, як результат, було розроблено програмне забезпечення, яке дозволяє класифікувати фрактальні реалізації у графічному вигляді.

У майбутніх дослідженнях є намір зосередитися на збільшенні розмірів зображення, оптимізації алгоритмів побудови графів видимості та оптимізації нейронної мережі.

Аналіз часових рядів можна покращити за допомогою візуальних представлень. Звичайні графіки важко інтерпретувати, але часові ряди легко зрозуміти за допомогою графічних зображень. Цей метод дозволяє полегшити класифікацію, прогнозування та оцінювання числових даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bagnall A., Bostrom A. The great time series classification bakes off: a review and experimental evaluation of recent algorithmic advances // Data Mining and Knowledge Discovery. 2017. Vol. 31, No. 3. P. 606–660.

2. Bagnall A., Flynn M. On the Usage and Performance of the Hierarchical Vote Collective of Transformation-based Ensembles. 2019. P. 105–107.

3. Fawaz H. Deep learning for time series classification: a review // Data Mining and Knowledge Discovery. 2019. Vol. 33, No. 4. P. 917–963.

4. Ye L., Keogh E. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification // Data Mining and Knowledge Discovery. 2011. Vol. 22. P. 149–182.

5. Rodriguez J., Kuncheva L., Alonso J. Rotation Forest. A New Classifier Ensemble Method // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2006. Vol. 28. P. 1619–1630.

6. Bagnall A., Flynn M. Is rotation forest the best classifier for problems with continuous features. 2018. P. 35–38.

7. Deng H. A Time Series Forest for Classification and Feature Extraction // Information Sciences 239. 2013. 153 p.

8. Lacasa L., Luque B., Ballesteros F. Time Series to Complex Networks: The Visibility Graph // Proceedings of the National Academy of Sciences of U.S.A. 2008. 4972 p.

9. Luque B., Lacasa L., Ballesteros F. Horizontal Visibility Graphs: Exact Results for Random Time Series. Physical Review. 2009. 70 p.

10. Durrett R. Probability: Theory and Examples, 5th ed. Cambridge University Press. 2019. 490 p.

11. Lalley S. Mathematical finance 345 – lecture 5: Brownian motion. 2019. 12 p.

12. Lim S., Eab H. Some fractional and multifractional gaussian processes: A brief introduction // International Journal of Modern Physics: Conference Series. 2015. Vol. 36. 14 p.

13. Mandelbrot B., Ness J. Fractional Brownian motions, fractional noises and applications // SIAM Review. 1968. Vol. 10, No. 4. P. 422–437.

14. McGonegal B. Fractional Brownian motion and the fractional stochastic calculus. 2014. 44 p.

15. Способи представлення графів. URL : https://disted.edu.vn.ua/courses/learn/250 (дата звернення: 27.10.2022).

16. Fortunato S. Community detection in graphs // Physics Reports. 2010. Vol. 486, No. 3-5. P. 75–174.

17. Матриця суміжності. URL : https://disted.edu.vn.ua/courses/learn/7508 (дата звернення: 27.10.2022).

18. Krizhevsky A., Sutskever I., Hinton G. Imagenet classification with deep convolutional neural networks. NIPS. 2012. 1114 p.

19. Gradient-based learning applied to document recognition / Y. Lecun, L. Bottou, Y. Bengio, P. Haffner // in Proceedings of the IEEE. 1998. Vol. 86, No.11, P. 2278-2324.

20. Deep residual learning for image recognition / K. He, X. Zhang, S. Ren, J. Sun // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. P. 770–778.

21. Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition / C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. 2015. P. 1–9.

22. Krizhevsky A., Sutskever I., Hinton E. ImageNet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems. 2012. P. 1097–1105.

23. What is NumPy? – NumPy v1.20 Manual. URL : https://numpy.org/doc/stable/user/whatisnumpy.html (дата звернення: 27.10.2022).

24. Введение в Scikit-learn. URL : https://neurohive.io/ru/osnovy-data-science /vvedenie-v-scikit-learn/ (дата звернення: 27.10.2022).

25. Распознавание изображений на Python с помощью TensorFlow и Keras. URL : https://evileg.com/ru/post/619/ (дата звернення: 27.10.2022).

26. Снарский А. А., Ландэ Д. В. Графы видимости – инструмент сетевого анализа рядов измерений // Реєстрація, зберігання і обробка даних. 2013. Т. 15, № 2. С. 29–34.

27. Kirichenko L., Radivilova T., Bulakh V. Classification of fractal time series using recurrence plots. In 2018 International Scientific-Practical Conference Problems of Infocommunications // Science and Technology (PIC S&T). 2018, October. P. 719-724.

28. Kirichenko L., Abed Saif Ahmed A., Radivilova T. Generalized Approach to Analysis of Multifractal Properties from Short Time Series // International Journal of Advanced Computer Science and Applications. 2020. Vol. 11(5). P. 183–198.

29. Рижанов В.С. Класифікація фрактальних реалізацій методами машинного навчання на основі побудови графів видимості // Тези XVI Міжнародної науково-практичної конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті» (м. Дніпро, 14-15 грудня 2022 р.). Дніпро : ДІІТ, 2022. С. 103.