Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: **«ВИКОРИСТАННЯ ПРИХОВАНИХ МАРКОВСЬКИХ МОДЕЛЕЙ У ЗАДАЧАХ ПОКРАЩЕННЯ ЯКОСТІ ДАНИХ»**

Виконав: студент 2-го року навчання
освітньо-наукової програми
«Системний аналіз»,
спеціальності 124 Системний аналіз

Картавий Микола Олексійович

Керівник: Крюкова Г. В.,
кандидат фіз.-мат. наук, доцент

Рецензент_____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2022

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на магістерську роботу

студенту Картавому М. О. факультету інформатики 2 курсу МП
ТЕМА: Використання прихованих марківських моделей у задачах покращення якості даних
Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Календарний план
Анотація
Вступ
РОЗДІЛ 1: Загальна інформація про ПММ
РОЗДІЛ 2: Постановка задачі
РОЗДІЛ 3: Використані підходи
РОЗДІЛ 4: Реалізація алгоритму
РОЗДІЛ 5: Отримані результати
Висновки
Список використаної літератури

Дата видачі „___” _____ 2022р. Керівник _____
(підпис)

Тема: Використання прихованих марківських моделей у задачах покращення якості даних

Календарний план виконання роботи:

№ п/п	Назва етапу дипломної роботи	Термін виконання етапу
1.	Отримання завдання на дипломну роботу.	16.10.2021
2.	Огляд технічної літератури за темою роботи.	16.11.2021
3.	Аналіз існуючих досліджень за темою роботи.	14.12.2021
3.	Розробка програмного продукту.	10.01.2022
4.	Написання вступу та плану роботи.	05.04.2022
6.	Написання основних розділів роботи.	09.04.2022
7.	Створення слайдів для доповіді та написання доповіді.	11.05.2022
8.	Коригування роботи відповідно до вимог щодо оформлення робіт.	13.05.2022
9.	Остаточне оформлення пояснювальної роботи та слайдів.	16.05.2022
10.	Коригування роботи згідно із зауваженнями керівника.	26.05.2022
11.	Передхахист курсової роботи	17.06.2022

Студент Картавий М. О.
Керівник Крюкова Г. В.

Оглавление

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ **2**
АНОТАЦІЯ ДО КУРСОВОЇ РОБОТИ **5**
ВСТУП **6**
РОЗДІЛ 1 ..... **7**
1.1 ЗАГАЛЬНА ІНФОРМАЦІЯ ПРО ПММ **7**
1.2 МАРКОВСЬКІ ЛАНЦЮГИ **8**
1.3 ПРИХОВАНІ МАРКІВСЬКІ МОДЕЛІ **9**
1.4 ДЕКОДУВАННЯ ПРИХОВАНИХ СТАНІВ АЛГОРИТМОМ ВІТЕРБІ **10**
РОЗДІЛ 2 ..... **12**
2.1 ПОСТАНОВКА ЗАДАЧІ **12**
РОЗДІЛ 3 ..... **13**
3.1 ПРЕДСТАВЛЕННЯ СТАНУ ЯК ЗНАЧЕННЯ КОНКРЕТНОГО ПІКСЕЛЮ **13**
3.2 ПРЕДСТАВЛЕННЯ СТАНУ ЯК ЗНАЧЕННЯ НАБОРУ ПІКСЕЛІВ **14**
3.3 РОЗБИТТЯ НА ЗОНИ ДЛЯ ПЕРЕДБАЧЕННЯ **15**
РОЗДІЛ 4 ..... **16**
4.1 ОПИС ПОБУДОВИ ПММ **16**
4.2 АЛГОРИТМ ВІДНОВЛЕННЯ ЗОБРАЖЕННЯ **17**
4.3 ПСЕВДОКОД АЛГОРИТМУ **19**
РОЗДІЛ 5 ..... **26**
5.1 ОТРИМАНІ РЕЗУЛЬТАТИ **26**
5.2 ПОРІВНЯННЯ З ІСНУЮЧИМИ РЕЗУЛЬТАТАМИ **30**
5.3 СПОСОБИ ПОКРАЩЕННЯ РЕЗУЛЬТАТІВ **30**
ВИСНОВКИ **31**
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **32**

Анотація до курсової роботи

В дипломній роботі я розглянув приховані марковські моделі та спробував застосувати їх до покращення якості даних, а саме покращення якості зображень. Оскільки використання прихованих марківських моделей для покращення якості зображень вже описано в багатьох наукових статтях, тому задача полягала в тому, щоб визначити недоліки даних алгоритмів, їх обмеження і запропонувати та реалізувати з урахуванням цих недоліків нову версію використання прихованих марківських моделей для покращення якості картинок. Головна умова для цього - було створення алгоритму, що буде працювати швидше, економніше по пам’яті та показувати кращий результат за деякі існуючі реалізації використання прихованих марківських моделей в задачах покращення якості зображень.

Мета дослідження: створення алгоритму з використанням прихованих марківських моделей для покращення якості зображень з мінімальними необхідними ресурсами для цього та обгрунтування, чому саме такий підхід є кращим для покращення якості даних.

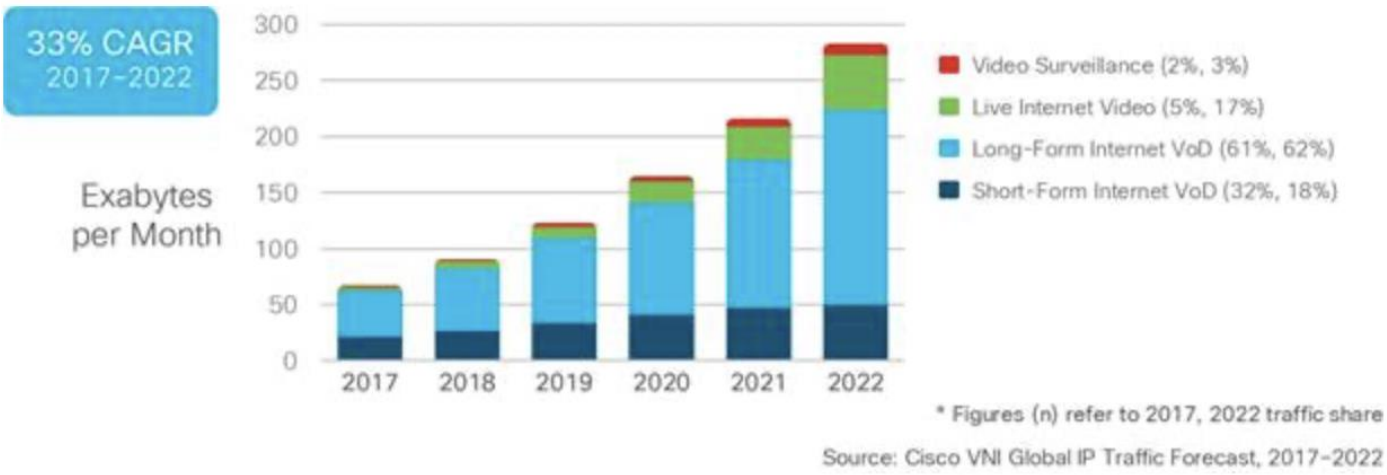
Джерела для дослідження: використані іноземні джерела – наукова література та статті. Більш детально можна ознайомитися в розділі «Список використаних джерел».

Що було зроблено: я опрацював наукові статті з використанням прихованих марківських моделей для покращення якості зображень, дослідив їх проблеми та розробив підхід для використання прихованих марківських моделей з використанням меншої кількості ресурсів та меншої кількості часу з отриманням допустимого результату.

Вступ

Вже впродовж довгого часу задачі покращення якості даних залишаються актуальними й не мають оптимального вирішення. Для цього сучасні методи вимагають використання складних нейронних мереж або складних алгоритмів з використанням великої кількості даних, пам’яті, ресурсів загалом, тощо. Виникає питання: чи можливо розробити щось, що буде давати допустимі результати з використанням менших ресурсів з допустимим часом обробки. Ми дослідимо існуючі методи покращення якості зображень з використанням прихованих марківських моделей, проаналізуємо та спробуємо створити свій алгоритм для відновлення якості даних, розглянемо можливі варіанти підходу, проблеми кожного з підходів та їх вирішення.

Нижче можна побачити як протягом останніх 5-ти років кожного року виростало по 33% обсяг відео даних, це зумовлено появою нових додатків та збільшення вироблення кількості відеоматеріалів, що використовуються в бізнесі, рекламі, освіті, тощо.



Малюнок 1 Графік зросту кількості відеоданих протягом 2017 - 2022

Розділ 1

1.1 Загальна інформація про ПММ

ПММ — це статистичні моделі для захоплення прихованої інформації з спостережуваних послідовних символів (наприклад, нуклеотидної послідовності). Вони мають багато застосувань в аналізі послідовностей, зокрема для прогнозування екзонів та інтронів у геномній ДНК, ідентифікації функціональних мотивів (доменів) у білках (профіль ПММ), вирівнювання двох послідовностей (пара ПММ). У ПММ система, що моделюється, вважається марковським процесом з невідомими параметрами, і завдання полягає у визначенні прихованих параметрів із спостережуваних параметрів. Хороша ПММ точно моделює реальне джерело спостережуваних реальних даних і має можливість моделювати джерело. При його застосуванні послідовність моделюється як результат дискретного стохастичного процесу, який проходить через серію станів, які «приховані» від спостерігача.

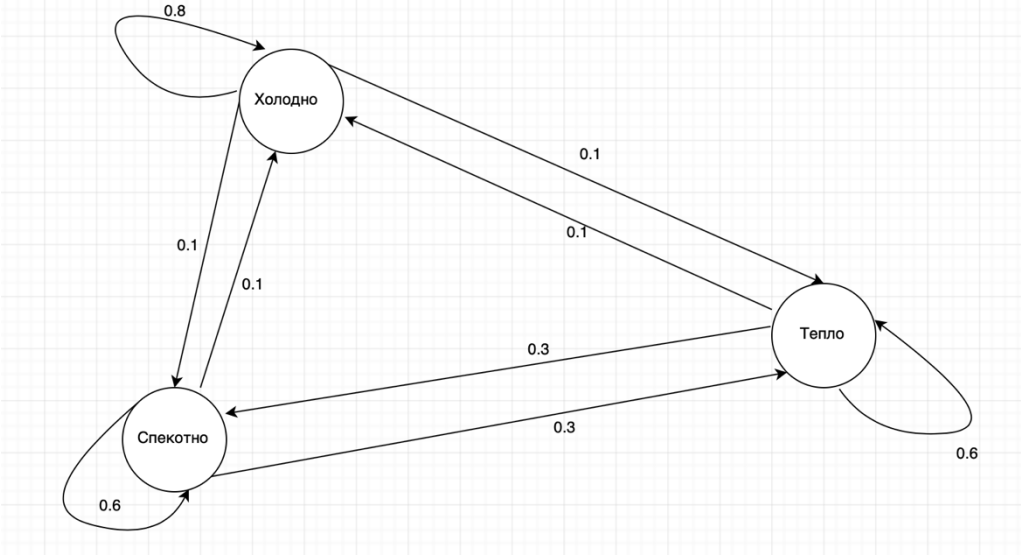
ПММ заснований на збільшенні ланцюга Маркова. Ланцюг Маркова — це модель, яка розповідає нам щось про ймовірності послідовностей випадкових величин, станів, кожен з яких може приймати значення з деякого набору. Ці набори можуть бути словами, тегами або символами, що представляють будь-що, наприклад погоду. Ланцюг Маркова робить дуже сильне припущення, що якщо ми хочемо передбачити майбутнє в послідовності, важливо лише поточний стан. Стани до поточного стану не впливають на майбутнє, крім поточного стану. Це ніби передбачити завтрашню погоду, ви могли б подивитися на сьогоднішню погоду, але вам не дозволили дивитися на вчорашню.

1.2 Марковські ланцюги

Більш формально розглянемо послідовність змінних стану q1,q2,...,qi. Модель Маркова втілює марковське припущення щодо ймовірностей цієї послідовності: при передбаченні майбутнього не має значення минуле, лише сьогодення.

Припущення Маркова: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$ (A.1)

На малюнку 1 показано ланцюг Маркова для присвоєння ймовірності послідовності погодних подій, словник якої складається зі «Спекотно», «Холодно» і «Тепло»



Малюнок 2

Стани представлені у вигляді вузлів у графі, а переходи з їхніми ймовірностями — у вигляді ребер. Переходи є ймовірностями: значення дуг, що виходять із заданого стану, мають бути рівними одиниці.

Формально ланцюг Маркова задається такими компонентами:

$Q = q_1q_2 ...q_N$ - набір з N станів

$A=a_{11}a_{12}...a_{n1}...a_{nn}$ - матриця ймовірності переходу A, кожна a_{ij} представляє ймовірність переходу зі стану i в стан j, так що $\sum_{j=1}^{n} aij = 1 \ \forall$ i

$\pi = \pi_1,\pi_2,...,\pi_N$ - початковий розподіл ймовірності за станами. πi – ймовірність того, що ланцюг Маркова почнеться в стані i. Деякі стани j можуть мати π j = 0, що означає, що вони не можуть

бути початковими станами. Також $\sum_{i=1}^{N} \pi i = 1$

1.3 Приховані марківські моделі

Ланцюг Маркова корисний, коли нам потрібно обчислити ймовірність для послідовності спостережуваних подій. Однак у багатьох випадках події, які нас цікавлять, приховані: ми не спостерігаємо їх безпосередньо. Наприклад, ми зазвичай не спостерігаємо члени частини мови в тексті. Швидше, ми бачимо слова і повинні вивести члени з послідовності слів. Ми називаємо члени прихованими, тому що вони не спостерігаються.

Прихована модель Маркова (HMM) дозволяє нам говорити як про спостережувані події (наприклад, слова, які ми бачимо у вхідних даних), так і про приховані події (наприклад, члени частини мови), які ми вважаємо причинними факторами в нашій імовірнісній моделі. ПММ визначається такими компонентами:

$Q = q_1 q_2 \ldots q_N$ - набір з N станів

$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$ - матриця ймовірності переходу A, кожна a_{ij} представляє ймовірність переходу зі стану i в стан j, так що $\sum_{j=1}^{n} aij = 1 \; \forall$ i

$O = o_1 o_2 \ldots o_T$ - послідовність T спостережень, кожне з яких отримано зі словника $V = v_1, v_2, \ldots, v_V$

B = b_i (o_t) - послідовність ймовірностей спостережень, також званих імовірностями викидів, кожна з яких виражає ймовірність того, що спостереження генерується із стану *i*

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ - початковий розподіл ймовірності за станами. π_i – ймовірність того, що ланцюг Маркова почнеться в стані *i*. Деякі стани *j* можуть мати

$\pi_j = 0$, що означає, що вони не можуть бути початковими станами.

Також $\sum_{i=1}^{N} \pi i = 1$.

Прихована марківська модель першого порядку створює два спрощувальні припущення.

По-перше, як і у випадку з ланцюгом Маркова першого порядку, ймовірність певного стану залежить тільки від попереднього стану:

$P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1})$

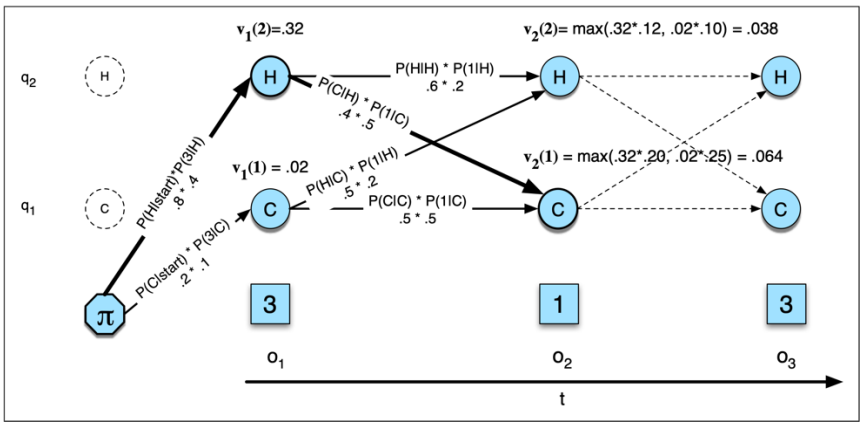
По-друге, ймовірність вихідного спостереження o_i залежить лише від стану, який викликав спостереження q_i, а не від будь-яких інших станів чи будь-яких інших спостережень:

$P(o_i|q_1 \ldots q_i,\ldots,q_T, o_1,\ldots,o_i,\ldots,o_T) = P(o_i|q_i)$

1.4 Декодування прихованих станів алгоритмом Вітербі

Для будь-якої моделі, наприклад ПММ, яка містить приховані змінні, завдання визначення того, яка послідовність змінних є основним джерелом деякої послідовності спостережень, називається завданням декодування. У задачах для покращення якості зображень – це знаходження станів (множини пікселів), що передували зображенню низької якості. Враховуючи вхідні дані ПММ $\lambda = (A, B)$ і послідовність спостережень $O = o_1, o_2, \ldots, o_T$, необхідно знайти найбільш вірогідну послідовність станів $Q = q_1q_2q_3 \ldots q_T$.

Ми можемо запропонувати знайти найкращу послідовність наступним чином: для кожної можливої послідовності прихованих станів ми можемо запустити прямий алгоритм і обчислити ймовірність послідовності спостереження з урахуванням цієї послідовності прихованих станів. Тоді ми могли б вибрати послідовність прихованого стану з максимальною ймовірністю спостереження. Натомість найпоширенішим алгоритмом декодування для ПММ є алгоритм Вітербі. Як і прямий алгоритм, Вітербі є різновидом динамічного програмування, який використовує решітку динамічного програмування. Вітербі також сильно нагадує інші варіанти динамічного програмування.



Малюнок 3

Опис Малюнку 2:

Решітка Вітербі для обчислення найкращого шляху через прихований простір станів для подій-пікселів 3->1->3. Приховані стани – у колах, спостереження – у квадратах. Білі

(незаповнені) кола вказують на неможливі переходи. На малюнку показано обчислення v_t (j) для двох станів за два кроки часу. Обчислення в кожній комірці слідує рівнянню $v_t(j) = max1 \leq i \leq N-1\ v_t-1(i)\ a_{ij}\ b_j(ot)$. Отримана ймовірність, виражена в кожній комірці, є рівнянням $v_t(j) = P(q_0,q_1,...,q_{t-1},o_1,o_2,...,o_t,q_t = j|\lambda)$.

На малюнку 2 показано приклад решітки Вітербі для обчислення найкращої послідовності прихованого стану для послідовності спостереження 3 1 3. Ідея полягає в тому, щоб обробити послідовність спостережень зліва направо, заповнюючи решітку. Кожна клітинка решітки, vt (j), представляє ймовірність того, що ПММ перебуває в стані j після того, як побачив перші t спостережень і пройшов через найбільш імовірну послідовність станів q1,...,qt−1, враховуючи автомат λ. Значення кожної комірки vt(j) обчислюється шляхом рекурсивного вибору найбільш ймовірного шляху, який може привести нас до цієї комірки. Формально кожна клітинка виражає ймовірність $v_t(j) = \max P(q_1...q_{t-1},o_1,o_2 ...o_t,q_t = j|\lambda)$ для $q_1 \ldots q_{t-1}$

Зверніть увагу, що ми представляємо найбільш імовірний шлях, беручи максимум для всіх можливих попередніх послідовностей стану

$\max(q_1 \ldots q_{t-1})$. Як і інші алгоритми динамічного програмування, вітербі заповнює кожну клітинку рекурсивно. Враховуючи, що ми вже обчислили

ймовірність перебувати в кожному стані в момент t − 1, ми обчислюємо ймовірність Вітербі, вибираючи найбільш ймовірне з продовжень шляхів, які ведуть до поточної комірки. Для даного стану q_j у момент t значення $v_t(j)$ обчислюється як $v_t(j) = \max v_{t-1}(i)\ a_{ij}\ b_j(o_t)$, *i* = 1-N (3)

Три множники, помножені в рівнянні (3) для розширення попередніх шляхів для обчислення ймовірності Вітербі в момент часу t є:

- $v_{t-1}(i)$ ймовірність попереднього шляху Вітербі з попереднього кроку часу
- a_{ij} ймовірність переходу з попереднього стану q_i в поточний стан q_j
- $b_j(o_t)$ імовірність спостереження стану символу спостереження o_t з урахуванням поточного стану *j*

Отже, ми можемо дати формальне визначення рекурсії Вітербі наступним чином:

1) Ініціалізація
    a. $v_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$
    b. $bt_1(j) = 0 \quad 1 \leq j \leq N$
2) Рекурсія
    a. $v_t(j) = \max_{vt-1}(i)\ a_{ij} b_j(o_t); \quad 1 \leq j \leq N,\ 1 < t \leq T$
    b. $bt_t(j) = \text{argmax}\ v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$
3) Зупинка
    a. Найкращий скор: $P^* = \max v_T(i)$, i = 1-N
    b. Початок зворотного трасування: $q_T^* = \text{argmax}\ v_T(i)$, i = 1-N

РОЗДІЛ 2

2.1 Постановка задачі

Необхідно використати ПММ для покращення якості зображення. На вхід подається зображення низької якості, попередньо розтягнуте до роздільної здатності кінцевого зображення, на виході маємо отримати зображення з більшою роздільною здатністю. Необхідно побудувати матрицю переходів, матрицю емісій, початкову ймовірність та використати алгоритм Вітербі для визначення прихованих станів, котрі виступають як пікселі оригінального зображення. Для цього необхідно визначити множину станів, множину прихованих станів. Оскільки зображення – це множина така що,$A = \{A_1, A_2, A_3\} \mid A_i = \{M \times N\}$, $x_{mn} \in [0,255]$, $m \in M$, $n \in N$,
то способи задавання стану та прихованого стану можуть бути:

1) Конкретний піксель, що набуває значення від 0 до 255
2) Набір пікселів, що набувають значення від 0 до 255

Необхідно проаналізувати ефективність таких підходів, а саме:

1) Дослідити ймовірносний розподіл для кожного з підходів
2) Складність та час виконання кожного

Ймовірносний розподіл дає оцінку для кожного з підходів, а саме імовірність вибору наступного стану повинна задовільняти більшість зображень, у випадку, коли станів 256 – це може не дати задовільного результату, оскільки ми можемо отримати неоднозначний розподіл. Складність та час виконання залежить від розмірностей матриці переходу, матриці емісій. У випадку з першим підходом, це матриці розміру 256*256, що є не складні для обрахунку, проте чи буде давати такий підхід задовільний результат. Для другого підходу, якщо стан складається з двох пікселів, то ми маємо матриці (256*256)*(256*256), матриці таких розмірностей важко обраховувати, проте вони можуть дати кращий результат.

Складність та час виконання напряму залежать від розмірів множини станів та множини спостережень. Алгоритм Вітербі має складність $O(K^2T)$, де K – множина станів, T – множина спостережень. У випадку з представленням стану як значення конкретного пікселю ми маємо K = 256, тоді як у випадку з представленням стану як набору пікселів – K = 256*256.

РОЗДІЛ 3

3.1 Представлення стану як значення конкретного пікселю

Формалізуємо задачу для конкретного підходу

Q = 0,1,2,3 ...255 - набір з 256 станів

$A=a_{01}a_{02}...a_{2551}...a_{255255}$ - матриця ймовірності переходу A, кожна a_{ij} представляє ймовірність переходу зі стану i в стан j, так що $\sum_{j=1}^{n} aij = 1 \; \forall$ i

O = 0,1,2,3 ...255 - послідовність T=256 спостережень, кожне з яких отримано зі словника $V = v_1,v_2,...,v_V$

B = b_i (o_t) - послідовність ймовірностей спостережень, також званих імовірностями викидів, кожна з яких виражає ймовірність того, що спостереження генерується із стану *i*

$\pi = \pi_0,\pi_1,...,\pi_{255}$ - початковий розподіл ймовірності за станами. π_i – ймовірність того, що ланцюг Маркова почнеться в стані *i*. Деякі стани *j* можуть мати

$\pi_j = 0$, що означає, що вони не можуть бути початковими станами.

Також $\sum_{i=0}^{255} \pi i = 1$.

Підготовка матриць ймовірностей відбувалася на датасеті з 5600 зображень з роздільною здатністю 300*300, як оригінал зображення та 5600 зображень з початковою роздільною здатністю 100*100, 150*150, ефектом розмиття.

При дослідженні ймовірносних розподілів переходу зі стану *i* в стан *j* розподіл має характер нормального, тому ймовірність вибору стану *i* зі стану *j* найвища у випадку *i=j*. Такий розподіл дозволяє передбачати дані, що мають дуже плавні зміни, що не є підходящим для нашої задачі. Головна проблема в зображеннях низької якості – це «розмитість», що утворюється за допомогою вибору наступного пікселю максимально схожого до попереднього, тому чіткі границі, наприклад, чорний текст на білому фоні вже не має різкого переходу, що дає нечіткість зображення. Тому використовування такого підходу не принесло хоч якихось результатів, оскільки всі спроби визначення

послідовності станів, що передували певним спостереженням призводили лише до погіршення результатів.

3.2 Представлення стану як значення набору пікселів

Для визначення стану візьмемо набір з двох пікселів по вісі X.

Формалізуємо задачу для конкретного підходу

Q = 0,1,2,3 ...255….65535- набір з 65536 станів

A=$a_{01}a_{02}...a_{655351}….a_{6553565535}$ - матриця ймовірності переходу A, кожна a_{ij} представляє ймовірність переходу зі стану i в стан j, так що $\sum_{j=1}^{n} aij = 1 \; \forall$ i

O = 0,1,2,3…255…65535 - послідовність T=65536 спостережень, кожне з яких отримано зі словника V = $v_1,v_2,...,v_V$

B = b_i (o_t) - послідовність ймовірностей спостережень, також званих імовірностями викидів, кожна з яких виражає ймовірність того, що спостереження генерується із стану *i*

π = $\pi_0,\pi_1,...,\pi_{65535}$ - початковий розподіл ймовірності за станами. π_i – ймовірність того, що ланцюг Маркова почнеться в стані *i*. Деякі стани *j* можуть мати

$\pi_j = 0$, що означає, що вони не можуть бути початковими станами.

Також $\sum_{i=0}^{65535} \pi i = 1$.

Підготовка матриць ймовірностей відбувалася на датасеті з 5600 зображень з роздільною здатністю 300*300, як оригінал зображення та 5600 зображень з початковою роздільною здатністю 100*100, 150*150 та ефектом розмиття.

При дослідженні ймовірносних розподілів переходу зі стану *i* в стан *j* розподіл має характер альфа, тому ймовірність вибору стану *i* зі стану *j* найвища у конкретному стані, що дозволяє робити чіткіші передбачення.

Основною проблемою при такому підході стає складність підрахунків, а саме велика кількість станів. Тому необхідно зробити оптимізацію, котра дозволила б зменшити час виконання алгоритму та це б не впливало на результати.

3.3 Розбиття на зони для передбачення

Для того, щоб зменшити кількість прихованих станів, необхідно розбити картинку на підзони по проміжках значення пікселів та передбачати послідовність прихованих станів у кожній підзоні окремо. Таким чином ми можемо зменшити кількість станів та збільшити кількість проходів, що дозволить зменшити час виконання, оскільки складність алгоритму Вітербі степенева і кількість станів є більш критичним показником аніж кількість проходів. Оптимальним вибором по кількості моделей та проміжках є співвідношення 4-х моделей ПММ та проміжок з 64 елементів, це будуть кодовані проміжки: 0-63, 64–127, 128 – 191, 192 – 255

Таким чином, ми маємо побудувати ПММ для трьох каналів: red, green, blue та по 4-и проміжки відповідно, на виході маємо 12 моделей. Оскільки дані процеси є незалежними, тому їх можна запустити паралельно, що підвищить швидкість ще в рази. Таким чином, ми маємо 4-и основні зони на зображенні певного каналу. На малюнку 3 нижче зображено поділ зон (зліва) для передбачення, таким чином, маємо:

- значення пікселів 0-64, червоний колір
- значення пікселів 64-128, зелений колір
- значення пікселів 128-192, синій колір
- значення пікселів 192-256, чорний колір

Також є білі зони – це зони, що не підпадають під жодну з зон, наприклад, множина пікселів {0, 255,4,192} – така множина пікселів вже має різкий перехід, тому об’єкти на зображенні будуть мати чіткі границі, наша задача полягає в тому, щоб уникнути дублювання пікселів в зображеннях - додати різкості для покращення якості зображення.

Малюнок 4: Зліва - зони для передбачення, зправа - оригінальне зображеня

РОЗДІЛ 4

4.1 Опис побудови ПММ

Позначимо зображення високої роздільної здатності як HR – high rate зображення, та зображення низької роздільної здатності LR – low rate зображення.
В якості HR датасету використовувалися 4600 зображень з роздільною здатністю 300*300, з різними типами об’єктів та різними показниками різкості, контрастності, яркості, тощо, в якості LR зображень використовувалися ці ж самі 4600 зображень зменшених до роздільної здатності 100*100 та за допомогою бікубічної інтерполяції збільшено роздільну здатність до 300*300. В якості валідаційного датасету HR використовувалися 1000 зображень з роздільною здатність 300*300, з різними типами об’єктів та різними показниками різкості, контрастності, яркості, тощо, в якості валідаційного датасету LR зображень використовувалися ці ж самі 1000 зображень зменшених до роздільної здатності 100*100 та за допомогою бікубічної інтерполяції збільшено роздільну здатність до 300*300.
Для побудови матриці переходів HR зображення для кожної горизонтальної стрічки набору пікселів в циклі з кроком = 4, обиралося по 4-и сусідні пікселі для побудови двох станів. Тобто матриця переходів A = {M x N}, x_{mn} є 0, 1, 2,….,4095 | x_{mn} – стан, що кодується двома сусідніми пікселями по вісі X. Для пікселів, значення котрих завжди в межах від K до K+64, де K – це стартова точка для рахунку для певної зони передбачення, кількість станів як прихованих так і спостережуваних буде завжди

дорівнювати 4096. В залежності від зони стани будуть кодуватися різними проміжками: 0-64, 64-128, 128-192, 192-255 відповідно.
Початковий розподіл ймовірностей – це відношення частоти певного стану до загальної кількості станів загалом. Матриця емісій – це умовна ймовірність:

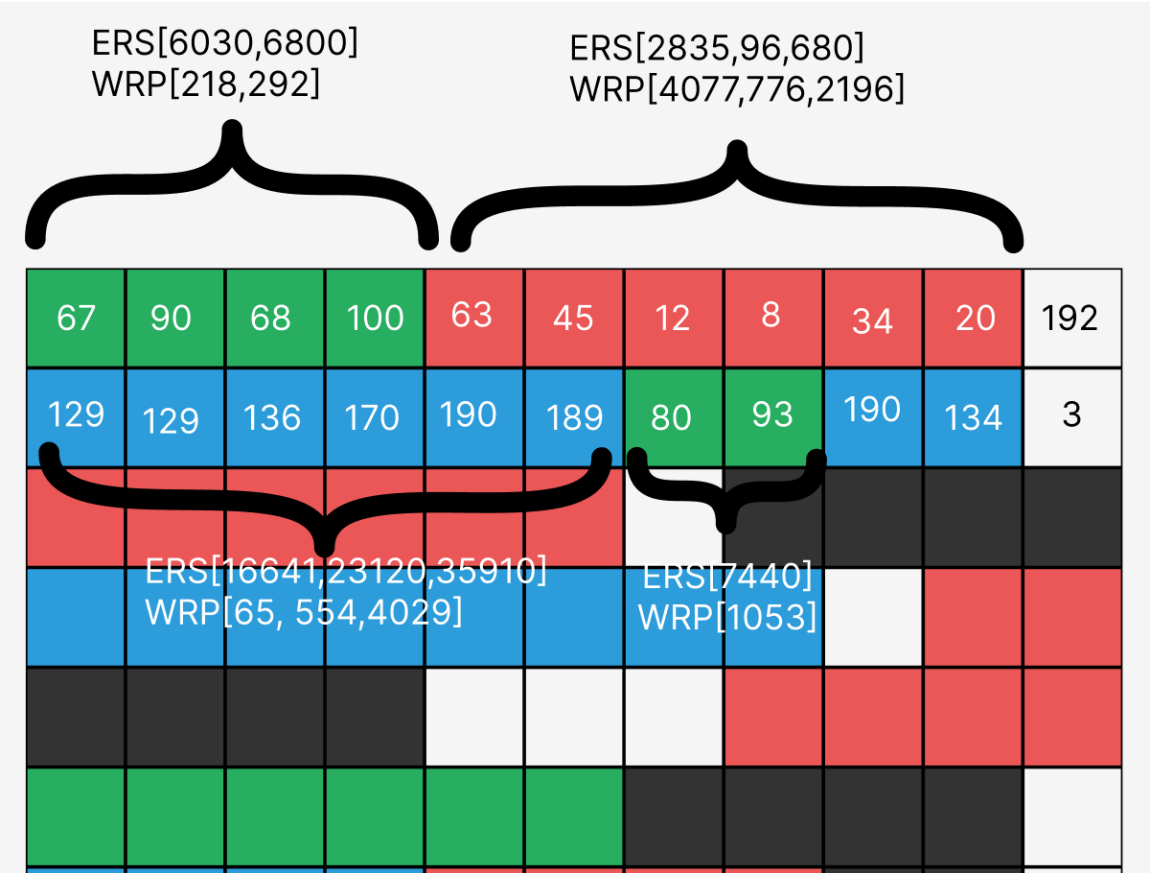
$P(B|A) = \frac{P(AB)}{P(A)}$ - ймовірність події A при умові, що подія B відбулася.

В контексті нашої задачі, B – спостережуваний стан, тобто закодоване значення двох пікселів зображення LR, A – це прихований стан, тобто закодоване значення двох пікселів зображення HR.

4.2 Алгоритм відновлення зображення

Тепер необхідно порахувати матриці емісій, переходів, розподіли початкової ймовірності для кожного з 4-х проміжків для кожного з 3-х каналів кольору і побудувати моделі, що загалом дає 12 ПММ.
Для відновлення зображення необхідно розділити зображення LR на три канали: червоний, зелений, синій і по кожному пройтися окремо.
Фіксуємо матрицю кожного каналу та кодуємо пікселі по вісі X по два сусідні пікселі в набір кодованих станів. Кожен кодований стан буде на проміжку від 0 до 65536, позначимо цей проміжок як ERS – encoded range state, за кожним кодованим станом закріплено його робочий проміжок для передбачення, позначимо як WRP – working range for prediction, на котрому працює модель, це проміжок від 0 до 4096. Таким чином, ERS стани від 0 до 4096 будуть відноситися до першої моделі, що працює на проміжку значень пікселів 0-64, кодовані стани від 4096 до 16384 будуть давати набір пікселів 64-128, від 16384 до 36864 – набір пікселів від 128-192, від 36864 до 65536 –

набір пікселів від 192 до 256.



Малюнок 5 - приклад того, як кодується значення ERS та WRP, кольорами виділено ділянки пікселів для конкретної моделі для передбачення. Текстове значення в кожному з квадратів – значення пікселів від 0 до 255, білим позначено пікселі, що не відносяться до жодної з зон. Фігурними дужками виділено початок та кінець зони для передбачення.

На малюнку вище можна побачити як працює розбиття матриці певного каналу на зони для передбачення та кодування як стану для певної моделі.
Отже, тепер при проходженні в циклі по 2 пікселі, ми кодуємо їх на проміжку від 0 до 65536, і зберігаємо в матрицю, така матриця для зображення розміром M x N, де M – ширина зображення, N -висота, буде мати розмір M/2 x N. Після цього, при проходженні в циклі цією матрицею кожного разу ми шукаємо найбільшу послідовність одного з проміжків ERS, після цього ми її (послідовність) переводимо в WRP і передбачаємо певною моделлю, що закріплена за кожним з проміжків, передбачення з моделі складаємо переводимо в набір пікселів від 0 до 255 та складаємо в матрицю для відновлення. Таким чином, виконуємо передбачення для кожного з каналів: червоний, зелений, синій та будуємо загальне зображення роздільної здатності, що і зображення високої якості. Оскільки за результатами дослідження було помічено, що в деяких випадках алгоритм передбачав послідовність, котра мала меншу ймовірність ніж послідовність, котру подавали як вхідний параметр, то для відбудови кінцевого зображення не можна використовувати усі результати алгоритму. Для вирішення цієї проблеми після передбачення алгоритмом Вітербі ми оцінюємо сумарну ймовірність послідовності отриманої та сумарну ймовірність послідовності, котру передавали на вхід і обираємо серед них ту, чия сумарна ймовірність більша. Оскільки це ще підвищує складність алгоритму, а саме

Після цього порівнюємо MSE для зображення високої якості та зображення, що було розтягнене за допомогою бікубічної інтерполяції та MSE для зображення високої якості та зображення, що було відновлено нашим алгоритмом та порівнюємо результати. В окремих випадках алгоритм може показати гірший результат, це пов’язано з великою кількістю однотипних зон на зображенні, тобто, наприклад, 80% зображення має набір значень пікселів від 128 до 192, в такому випадку алгоритм не показує задовільні результати в цілому. Оскільки складність алгоритму Вітербі $O(K^2T)$, де K – множина станів, T – множина спостережень, то час виконання лінійно залежить від довжини послідовності для передбачення, таким чином, алгоритм буде витрачати більше часу на відновлення зображення для вищої роздільної здатності.

4.3 Псевдокод алгоритму

Розглянемо псевдокод алгоритму:

```
SET restored_img_r TO [] #Створюємо порожні масиви для зберігання червоного каналу
SET restored_img_g TO [] #Створюємо порожні масиви для зберігання зеленого каналу
SET restored_img_b TO [] #Створюємо порожні масиви для зберігання синього каналу
SET picture_index TO 0
В циклі по картинкам валідайціного датасету
FOR img IN images_valid:
    picture_index+=1
    SET y_pxls1 TO array(img)
    SET x_pxls TO array(img.filter(ImageFilter.GaussianBlur(1)))#знижуємо якість
    #RED
    SET pxls TO x_pxls[:,:,0] #обираємо червоний канал зображення
    SET y_pxls TO y_pxls1[:,:,0] #обираємо червоний канал зображення
    SET new_m TO [] #створюємо матрицю для заповнення
    FOR y IN range(300):
        SET arr_list TO [] #кодуємо пікселі в ERS
        FOR k IN range(0,300,2):
            SET item TO pxls[y,k:k+2]
            arr_list.append(ers_map[(item[0],item[1])])
        new_m.append(arr_list)
    SET rec_m TO [] #створюємо матрицю для реконструкції зображення
    FOR y IN range(300):
        SET recovered TO []
        x=0
        while(x<150):
            #Заповнюємо масив станів WRP для декодування певною моделлю
```

```
#Як тільки робоча модель змінюється, тобто ERS стан має значення на певному
#з проміжків, то виконується декодування моделі і складання результату в масив для
#відновлення
        SET decoded_state TO ers_map_reverse[new_m[y][x]]
        SET last_model TO state_encoder_model[decoded_state]
        SET row_states TO []
        if(last_model==0):
           recovered.append(decoded_state[0])
           recovered.append(decoded_state[1])
           x+=1
        ELSE:
           FOR i IN range(x,150,1):
              SET decoded_state TO ers_map_reverse[new_m[y][i]]
              if(state_encoder_model[decoded_state]==last_model):
                 x+=1
                 if(last_model==1):
                    row_states.append(wrp_map1[(decoded_state)])
                 elif(last_model==2):
                    row_states.append(wrp_map2[(decoded_state)])
                 elif(last_model==3):
                    row_states.append(wrp_map3[(decoded_state)])
                 elif(last_model==4):
                    row_states.append(wrp_map4[(decoded_state)])

              ELSE:
                 break
           if(last_model==1):
              # декодування алгоритмом Вітербі
              SET res TO r1.decode([row_states],algorithm='viterbi')[1]
              FOR item IN res:
                 recovered.append(wrp_map_reverse1[item][0])
                 recovered.append(wrp_map_reverse1[item][1])
           elif(last_model==2):
      # декодування алгоритмом Вітербі
              SET res TO r2.decode([row_states],algorithm='viterbi')[1]
              FOR item IN res:
                 recovered.append(wrp_map_reverse2[item][0])
                 recovered.append(wrp_map_reverse2[item][1])
           elif(last_model==3):
      # декодування алгоритмом Вітербі
              SET res TO r3.decode([row_states],algorithm='viterbi')[1]
```

```
            FOR item IN res:
               recovered.append(wrp_map_reverse3[item][0])
               recovered.append(wrp_map_reverse3[item][1])
         elif(last_model==4):
     # декодування алгоритмом Вітербі
            SET res TO r4.decode([row_states],algorithm='viterbi')[1]
            FOR item IN res:
               recovered.append(wrp_map_reverse4[item][0])
               recovered.append(wrp_map_reverse4[item][1])
   rec_m.append(recovered)
restored_img_r.append(rec_m)
#GREEN аналогічним методом виконується реконструкція для зеленого каналу
SET pxls TO x_pxls[:,:,1]
SET y_pxls TO y_pxls1[:,:,1]
SET new_m TO []
FOR y IN range(300):
   SET arr_list TO []
   FOR k IN range(0,300,2):
      SET item TO pxls[y,k:k+2]
      arr_list.append(ers_map[(item[0],item[1])])
   new_m.append(arr_list)
SET rec_m TO []
FOR y IN range(300):
   SET recovered TO []
   x=0
   while(x<150):
      SET decoded_state TO ers_map_reverse[new_m[y][x]]
      SET last_model TO state_encoder_model[decoded_state]
      SET row_states TO []
      if(last_model==0):
         recovered.append(decoded_state[0])
         recovered.append(decoded_state[1])
         x+=1
      ELSE:
         FOR i IN range(x,150,1):
            SET decoded_state TO ers_map_reverse[new_m[y][i]]
            if(state_encoder_model[decoded_state]==last_model):
               x+=1
               if(last_model==1):
                  row_states.append(wrp_map1[(decoded_state)])
               elif(last_model==2):
```

```
                row_states.append(wrp_map2[(decoded_state)])
            elif(last_model==3):
                row_states.append(wrp_map3[(decoded_state)])
            elif(last_model==4):
                row_states.append(wrp_map4[(decoded_state)])
          ELSE:
            break

        if(last_model==1):
          SET res TO g1.decode([row_states],algorithm='viterbi')[1]
          FOR item IN res:
            recovered.append(wrp_map_reverse1[item][0])
            recovered.append(wrp_map_reverse1[item][1])
        elif(last_model==2):
          SET res TO g2.decode([row_states],algorithm='viterbi')[1]
          FOR item IN res:
            recovered.append(wrp_map_reverse2[item][0])
            recovered.append(wrp_map_reverse2[item][1])
        elif(last_model==3):
          SET res TO g3.decode([row_states],algorithm='viterbi')[1]
          FOR item IN res:
            recovered.append(wrp_map_reverse3[item][0])
            recovered.append(wrp_map_reverse3[item][1])
        elif(last_model==4):
          SET res TO g4.decode([row_states],algorithm='viterbi')[1]
          FOR item IN res:
            recovered.append(wrp_map_reverse4[item][0])
            recovered.append(wrp_map_reverse4[item][1])
    rec_m.append(recovered)
    progress(y,300,'GREEN: {}'.format(picture_index))
restored_img_g.append(rec_m)
#BLUE аналогічним методом виконується реконструкція для синього каналу
SET pxls TO x_pxls[:,:,2]
SET y_pxls TO y_pxls1[:,:,2]
SET new_m TO []

FOR y IN range(300):
    SET arr_list TO []
    FOR k IN range(0,300,2):
        SET item TO pxls[y,k:k+2]
        arr_list.append(ers_map[(item[0],item[1])])
```

```
    new_m.append(arr_list)
SET rec_m TO []
FOR y IN range(300):
    SET recovered TO []
    x=0
    while(x<150):
        SET decoded_state TO ers_map_reverse[new_m[y][x]]
        SET last_model TO state_encoder_model[decoded_state]
        SET row_states TO []
        if(last_model==0):
            recovered.append(decoded_state[0])
            recovered.append(decoded_state[1])
            x+=1
        ELSE:
            FOR i IN range(x,150,1):
                SET decoded_state TO ers_map_reverse[new_m[y][i]]
                if(state_encoder_model[decoded_state]==last_model):
                    x+=1
                    if(last_model==1):
                        row_states.append(wrp_map1[(decoded_state)])
                    elif(last_model==2):
                        row_states.append(wrp_map2[(decoded_state)])
                    elif(last_model==3):
                        row_states.append(wrp_map3[(decoded_state)])
                    elif(last_model==4):
                        row_states.append(wrp_map4[(decoded_state)])
                ELSE:
                    break
            if(last_model==1):
                SET res TO b1.decode([row_states],algorithm='viterbi')[1]
                FOR item IN res:
                    recovered.append(wrp_map_reverse1[item][0])
                    recovered.append(wrp_map_reverse1[item][1])
            elif(last_model==2):
                SET res TO b2.decode([row_states],algorithm='viterbi')[1]

                FOR item IN res:
                    recovered.append(wrp_map_reverse2[item][0])
                    recovered.append(wrp_map_reverse2[item][1])
            elif(last_model==3):
                SET res TO b3.decode([row_states],algorithm='viterbi')[1]
```

```
            FOR item IN res:
                recovered.append(wrp_map_reverse3[item][0])
                recovered.append(wrp_map_reverse3[item][1])
        elif(last_model==4):
            SET res TO b4.decode([row_states],algorithm='viterbi')[1]
            FOR item IN res:
                recovered.append(wrp_map_reverse4[item][0])
                recovered.append(wrp_map_reverse4[item][1])
    rec_m.append(recovered)
  restored_img_b.append(rec_m)
```

Оскільки моделі намагаються передбачити стани, що передували нинішнім, то інколи відбувається так, що модель передбачує послідовність станів, котра за алгоритмом є кращою, але насправді не є такою. Тоді виходить так, що необхідно замінити передбачення тим, що ми маємо в бікубічній інтерполяції. В такому випадку необхідно порахувати ймовірність кожної з послідовностей і обрати ту, що буде мати більшу сумарну ймовірність послідовності. Для цього необхідно оцінити ймовірність отриманої послідовності та вхідної і скласти у нове відновлене зображення. Нижче представлено псевдокод для реконструкції зображення з урахуванням вищесказанної логіки.

```
FOR img_i IN range(len(restored_img_r)):
  SET img TO images_high[5161+img_i]
  SET y_pxls1 TO np.array(np.array(img),int)
  SET x_pxls TO np.array(np.array(img.filter(ImageFilter.GaussianBlur(1)))[:,:],int)
  SET pxls TO x_pxls[:,:,0]
  SET y_pxls TO y_pxls1[:,:,0]
  SET rec_m TO restored_img_r[img_i]
  SET sp_r TO []
  FOR y IN range(300):
    SET row TO []
    SET row1_hmm TO []
    SET row2_hmm TO []
    SET row3_hmm TO []
    SET row4_hmm TO []
    SET row1_def TO []
    SET row2_def TO []
    SET row3_def TO []
    SET row4_def TO []
    SET last_worked TO 0
    FOR x IN range(0,300,2):
```

```
        SET x1_px TO rec_m[y,x]
        SET x2_px TO rec_m[y,x+1]
        SET y1_px TO pxls[y,x]
        SET y2_px TO pxls[y,x+1]
      #В циклі заповнюємо максимально довгу послідовність станів відновлених через
#алгоритм та вхідних станів
#Після чого порівнюємо сумарну ймовірність та обираємо кращу послідовність
       if(x1_px<px_range1 and x2_px<px_range1 and y1_px<px_range1 and
y2_px<px_range1):
          if(last_worked!=1):
      row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_h
      mm,row3_def,row4_hmm,row4_def))
            SET row1_def TO []
            SET row2_def TO []
            SET row3_def TO []
            SET row4_def TO []
          row1_hmm.append(wrp_map1[(x1_px,x2_px)])
          row1_def.append(wrp_map1[(y1_px,y2_px)])
          last_worked=1
       elif(x1_px<px_range2 and x2_px<px_range2 and y1_px<px_range2 and
y2_px<px_range2
          and x1_px>=px_range1 and x2_px>=px_range1 and y1_px>=px_range1 and
y2_px>=px_range1):
          if(last_worked!=2):
row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_hmm,ro
w3_def,row4_hmm,row4_def))
            SET row1_def TO []
            SET row2_def TO []
            SET row3_def TO []
            SET row4_def TO []
          row2_hmm.append(wrp_map2[(x1_px,x2_px)])
          row2_def.append(wrp_map2[(y1_px,y2_px)])
          last_worked=2
       elif(x1_px<px_range3 and x2_px<px_range3 and y1_px<px_range3 and
y2_px<px_range3
          and x1_px>=px_range2 and x2_px>=px_range2 and y1_px>=px_range2 and
y2_px>=px_range2):
          if(last_worked!=3):
row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_hmm,ro
w3_def,row4_hmm,row4_def))
            SET row1_def TO []
```

```
            SET row2_def TO []
            SET row3_def TO []
            SET row4_def TO []
          row3_hmm.append(wrp_map3[(x1_px,x2_px)])
          row3_def.append(wrp_map3[(y1_px,y2_px)])
          last_worked=3
        elif(x1_px<px_range4 and x2_px<px_range4 and y1_px<px_range4 and
y2_px<px_range4
          and x1_px>=px_range3 and x2_px>=px_range3 and y1_px>=px_range3 and
y2_px>=px_range3):
          if(last_worked!=4):
row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_hmm,ro
w3_def,row4_hmm,row4_def))
            SET row1_def TO []
            SET row2_def TO []
            SET row3_def TO []
            SET row4_def TO []
          row4_hmm.append(wrp_map4[(x1_px,x2_px)])
          row4_def.append(wrp_map4[(y1_px,y2_px)])
          last_worked=4
        ELSE:
row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_hmm,ro
w3_def,row4_hmm,row4_def))
          row.append(x1_px)
          row.append(x2_px)
          last_worked=0
row.extend(add_px(last_worked,row1_hmm,row1_def,row2_hmm,row2_def,row3_hmm,ro
w3_def,row4_hmm,row4_def))
      sp_r.append(row)
```

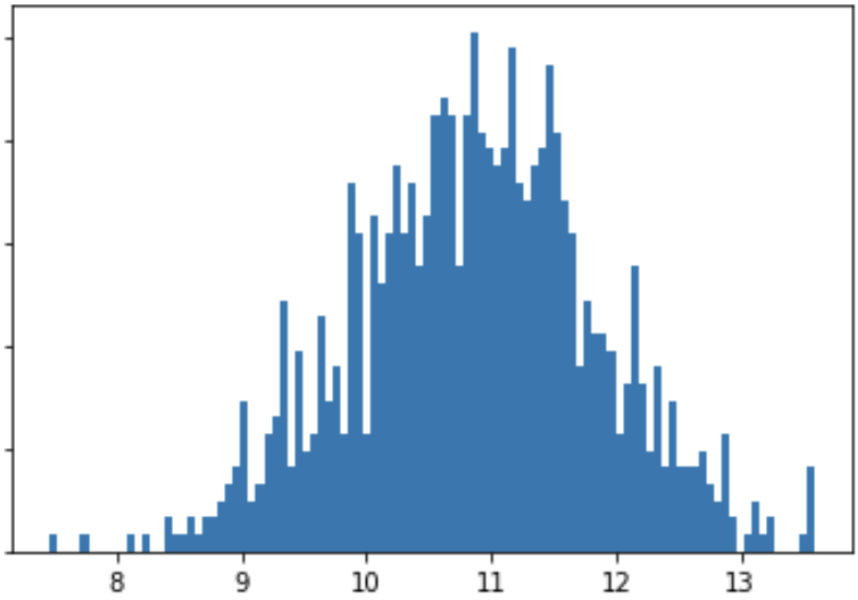
РОЗДІЛ 5

5.1 Отримані результати

В якості валідаційного датасету було обрано 1000 зображень, котрі не використовувалися для побудови моделей. Зображення мають роздільну здатність 300*300 та різні показники яскравості, контрастності, тощо. Нижче представлено таблицю порівнянь MSE для обраних випадковим чином трьох зображень відтворених за допомогою бікубічної інтерполяції та за допомогою нашого алгоритму. Усереднені результати враховуються по всій вибірці

Номер зображення	MSE для бікубічної інтерполяції	MSE для нашого алгоритму	Відсоток покращення
1	166.9	149.2	11.4
2	129.77	116.3	11.2
3	205.68	184.4	11.3
Усереднення	171.5	152.63	11.003

Так, наприклад, для узагальнення результатів можна побудувати графік з процентним покращенням зображення. Як можна бачити він має схожий на нормальний розподіл з центром в 11%.

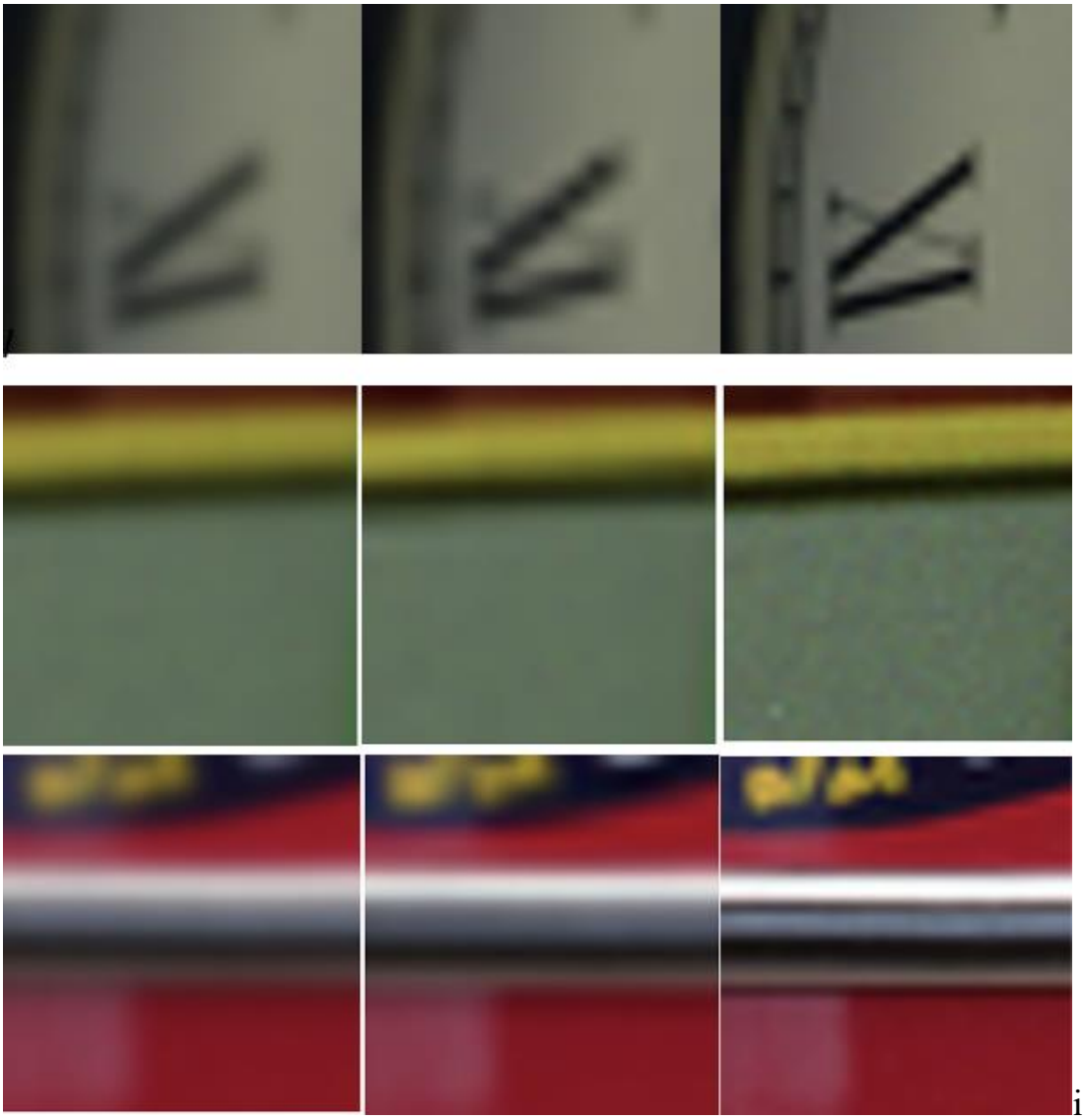


Малюнок 6 Графік відношення кількості до відсоткового покращення якості зображення

Для демонстрації було відібрано декілька найкращих результатів, нижче представлено приклади зображень до і після алгоритму а також оригінал зображення: зліва – зображення LR, посередині – зображення відтворене через наш алгоритм, справа – оригінал зображення HR

Малюнок 7 Зображення: зліва – зображення LR, посередині – зображення відтворене через наш алгоритм, справа – оригінал зображення HR.

На перший погляд може здатися, що зображення посередині має чіткіший результат, але якщо ми виберемо окремі ділянки зображення, то хоч зображення і стало чіткіше – деякі моменти, наприклад, текст не стає більш читабельним. Тобто для зображень, що мають якісь дрібні елементи, даний алгоритм не дасть задовільного результату, в таких випадках краще використати нейронні мережі.



Малюнок 8 Ділянки зображення для детальнішого порівняння: зліва - LR зображення, посередині - зображення відтворене нашим алгоритмом, справа - HR зображення

Вище зображено приклад конкретної ділянки зображення для більш детального порівняння, як можна бачити, що переходи з одного кольору в інший відбуваються більш точно і тоді зображення стає чіткіше, натомість у випадку з однотонним переходом – майже змін не помітно. Хоча зображення зліва має і більш пологий характер, оком воно більш приємно сприймається, це дає в загальному зображенні розмиті контури, тоді як посередині зображення має більш різкі границі переходів кольору в загальному зображенні це дає більш чіткі контури, а відповідно і чіткість. Деякі зображення, що мають однотонний характер, наприклад, небо, вода, трава тощо – такі зображення мають гірший показник MSE ніж середній.

5.2 Порівняння з існуючими результатами

Деякі вивчені мною роботи, мали простий алгоритм кодування станів з набором по 4-и пікселі, що призводило до великих витрат по пам’яті та часу виконання. Описані в них результати є кращими за мої, проте практичного використання або прикладів коду я не знайшов, тому порівнювати з ними не вважаю коректним. Робота, з використанням ПММ та байєсівської системи оцінки виявлення, мала також кращі результати, але практичне використання цієї роботи підходить виключно для даних, котрі мають декілька репрезентацій одного об’єкту, наприклад, відео, таким чином з серії зображень реконструювалося одне з високою роздільною здатністю.

5.3 Способи покращення результатів

Під час дослідження оптимального кодування станів виявилося, що кодування стану двома пікселями для мене дало найкращі результати. Так, наприклад, кодування стану квадратом з 4-х пікселів в проміжку значень пікселя по 8, не давало кращих результатів, це пов’язано з тим, що для побудови матриць ймовірності необхідно було мати більшу кількість різних зображень, котрі б дали змогу побудувати більш точну матрицю ймовірностей. Результати можна покращити, якщо визначити семантику зображення: чи є зображення однотонним, чи багато об’єктів на зображенні, тощо і тренувати модель для певного типу зображення. Також якщо зробити гібрид-модель з нейронної мережі та нашого алгоритму, в такому випадку це може дати досить непогані результати.

Висновки

В даній роботі ми дослідили використання ПММ в задачах покращення якості даних, а саме зображень, визначили актуальність проблеми використання алгоритмів для покращення якості даних, визначили проблеми існуючих алгоритмів для покращення якості зображень та побудували власний алгоритм для покращення якості зображень. Отримали результати, згідно з якими ми маємо в середньому 11% покращення, проте це не є задовільним результатом, котрий можна було б використовувати в додатках. Також співвідношення продуктивності цього методу до його результатів не є високим, а отже, алгоритм не є ефективним. Вважаю, що дослідження використання прихованих марківських моделей у сфері зображень, а саме покращення якості зображень не є виправданим. В першу чергу не через результат, а саме через продуктивність цього методу, велика кількість ресурсів та часу потребує даний алгоритм, але враховуючи, що це дало результати, то ці дослідження можна буде продовжити у випадку, якщо алгоритм Вітербі покращать і пришвидшать, в такому випадку вважаю, що можна буде продовжити дослідження в даному напрямку.

Список використаних джерел

1) https://www.researchgate.net/publication/27355897_Super-Resolution_Using_Hidden_Markov_Model_and_Bayesian_Detection_Estimation_Framework
2) https://web.stanford.edu/~jurafsky/slp3/A.pdf
3) https://wiki.math.uwaterloo.ca/statwiki/index.php?title=markov_Random_Fields_for_Super-Resolution
4) https://www.researchgate.net/publication/301679307_A_2D_hidden_Markov_model_for_patch-based_super_resolution