КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

Випускна кваліфікаційна робота магістра

за спеціальністю 113 Прикладна математика

на тему:

Реставрація розфокусованого зображення

Виконала студентка 2-го курсу
Ніколаєва Олена Володимирівна

Науковий керівник:
доцент, кандидат фізико-математичних наук
Матвієнко Володимир Тихонович

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол №__8__ від "__03__" __травня__ 2023 р.

Завідувач кафедри ДО проф. Іксанов О.М.

Київ-2023

Реферат

Обсяг роботи 80 сторінок, 14 ілюстрацій, 2 таблиці, 10 джерел посилань, 6 додатків.

ЗГОРТКА, ДЕКОНВОЛЮЦІЯ, РЕСТАВРАЦІЯ ЗОБРАЖЕННЯ, АРТЕФАКТИ, РОЗМИТТЯ, ЯДРО РОЗМИТТЯ, ДЕКОНВОЛЮЦІЯ ВІНЕРА ТА РІЧАРДСОНА-ЛЮСІ.

Метою роботи є реалізація алгоритму для одновимірної задачі оптимізація реставрації розфокусованого зображення, розгляд ітераційних алгоритмів та реалізація додаткових методів для покращення зображення. Також метою є проведення експериментів для майбутньої реалізації навчання нейронних мереж для реставрації розфокусованих зображень та створити користувацький інтерфейс для можливості удосконалення реальних зображень, що вимагають більш детального вивчення та глибшого відновлення.

Зміст

Вступ **5**

1 Одновимірна задача оптимізації **6**

1.1 Розфокусування та розмиття зображення за Гаусом 7

1.2 Згортка і деконволюція 8

1.2.1 Згортка 8

1.2.2 Деконволюція 9

1.3 Ітераційні методи відновлення зображення 10

1.4 Ітераційний метод Річардсона-Люсі 15

1.5 Деконволюція Вінера 16

1.6 Індекс структурної подібності (SSIM) 17

1.7 Метод пошуку по сітці 18

2 Проблеми, що виникають у процесі реставрації розфокусованого зображення **21**

3 Багатопараметрична задача відновлення розфокусованого зображення **23**

3.1 Попередня обробка 23

3.2 Обчислення рівня шуму 24

3.3 Моделі розмиття 24

3.4 Генерація ядра 25

3.5 Обробка в частотній області 27

3.6 Деконволюція 27

3.7 Зворотне перетворення 27

3.8 Збільшення різкості 28

3.9 Післяобробка 29

4 Розробка користувацького інтерфейсу 30

5 Експеременти 33

5.1 Результати для штучно розфокусованого зображення 33

5.2 Результати для штучно розфокусованого зображення у русі . . 37

5.3 Результати багатопараметричної задачі 40

Висновки 45

Джерела 47

Додатки 48

Вступ

У сучасну цифрову епоху зображення відіграють фундаментальну роль у різних сферах, включаючи журналістику, криміналістику, стеження та наукові дослідження. Однак у певних ситуаціях зняти чіткі та різкі зображення може бути складно, особливо коли зображення розфокусовані. Розфокусовані зображення характеризуються недостатньою різкістю та чіткістю, що часто є наслідком таких факторів, як тремтіння камери, обмежена глибина різкості або навмисне розмиття.

Відновлення розфокусованих зображень є життєво важливим завданням у сфері обробки зображень, спрямованим на відновлення втрачених деталей і покращення візуальної якості таких зображень. Цей процес передбачає використання вдосконалених алгоритмів і математичних методів для усунення ефектів розмиття та відновлення початкової різкості, тим самим покращуючи загальне візуальне сприйняття та забезпечуючи глибший аналіз вмісту.

Відновлення розфокусованих зображень має значні наслідки у різних областях, особливо у сценаріях, коли якість зображення безпосередньо впливає на важливі процеси прийняття рішень. В умовах війни в Україні реставрація розфокусованих зображень набуває особливої актуальності, враховуючи проблеми, пов'язані зі зніманням чітких і точних зображень у зонах бойових дій.

Метою кваліфікаційної роботи є дослідження та внесок у сферу реставрації розфокусованих зображень, зосередившись на її актуальності та застосуванні у контексті війни в Україні. Метою є розробка та впровадження ефективних методів реставрації, які можуть підвищити якість і чіткість розфокусованих зображень. Таким чином, це дослідження сприятиме наближенню до точного документуванню, криміналістичного аналізу та історичному збереженню візуальних записів, пов'язаних з війною.

1 Одновимірна задача оптимізації

У контексті відновлення розфокусованого зображення задачу одновимірної оптимізації можна сформулювати як пошук оптимального параметра розмиття (наприклад, стандартне відхилення розмиття за Гауссом, σ), який призводить до найкращого відновлення розфокусованого зображення.

Щоб вирішити цю проблему, потрібно визначити цільову функцію, яка вимірює якість відновленого зображення, таку як середня квадратична помилка (MSE) або показник індексу структурної подібності (SSIM). Мета полягає в тому, щоб мінімізувати MSE або максимізувати SSIM між відновленим зображенням і оригінальним зображенням (або еталонним зображенням, якщо вихідне зображення недоступне).

Враховуючи розфокусоване зображення $I_{defocused}$ і функцію відновлення $R(I_{defocused}, \sigma)$, яка приймає розфокусоване зображення та параметр розмиття σ як вхідні дані, одновимірна задача оптимізації може бути визначається як:

$$\sigma^* = \arg\min_{\sigma} \text{MSE}(R(I_{defocused}, \sigma), I_{original})$$

або

$$\sigma^* = \arg\max_{\sigma} \text{SSIM}(R(I_{defocused}, \sigma), I_{original})$$

Тут σ^* — це оптимальне значення параметра розмиття, яке забезпечує найкращу якість відновлення.

Щоб вирішити задачу оптимізації, можна виконати пошук у сітці, оцінивши цільову функцію для діапазону значень σ і вибравши значення, яке призводить до мінімального MSE або максимального SSIM.

1.1 Розфокусування та розмиття зображення за Гаусом

Розфокусування зображення — це процес розмивання або зменшення різкості зображення, часто внаслідок розфокусування об'єктива або тремтіння камери. Розмиття за Гаусом — це широко використовувана техніка для імітації розфокусування цифрових зображень. Він працює шляхом згортання зображення за допомогою функції Гауса, яка є неперервною функцією, яка наближає форму дзвоноподібної кривої.

Функція Гауса визначається як:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

де:

- x і y - горизонтальна і вертикальна координати,
- σ — стандартне відхилення розподілу Гауса, яке контролює ступінь розмиття,
- $G(x, y)$ — значення функції Гауса в точці (x, y).

У контексті розфокусування зображення розмиття за Гаусом застосовується шляхом виконання згортки між зображенням і ядром Гауса.

Ядро Гауса — це дискретна апроксимація функції Гауса, яка зазвичай представлена у вигляді квадратної матриці непарного розміру (наприклад, 3x3, 5x5, 7x7, тощо). Розмір і стандартне відхилення ядра визначають ступінь розмиття, застосованого до зображення.

Операцію згортки можна визначити як:

$$I_{blurred}(x, y) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(x - m, y - n) G(m, n)$$

де:

- $I(x, y)$ - інтенсивність вихідного зображення в точці (x, y),

- $I_{blurred}(x, y)$ - інтенсивність розмитого зображення в точці (x, y),
- $G(m, n)$ - значення ядра Гауса в точці (m, n),
- k — це напівширина ядра (наприклад, для ядра 3x3 $k = 1$).

Застосування розмиття за Гаусом до зображення призводить до розфокусованої версії вихідного зображення, причому ступінь розмиття контролюється розміром і стандартним відхиленням ядра Гауса. Ця техніка зазвичай використовується в програмах обробки зображень і комп'ютерного зору для імітації розфокусування, зменшення шуму або створення візуальних ефектів.

1.2 Згортка і деконволюція

Згортка та деконволюція є фундаментальними математичними операціями, які широко використовуються в обробці зображень і сигналів. Вони відіграють важливу роль у фільтрації, згладжуванні та підвищенні різкості зображень або сигналів, а також у відновленні оригінальної інформації з пошкоджених або змінених даних.

1.2.1 Згортка

Згортка — це математична операція, яка поєднує дві функції (в контексті обробки зображень, зображення та ядро) для отримання третьої функції, що представляє інтеграл від поточкового множення двох вхідних функцій. У обробці зображень згортка використовується для застосування фільтра (ядра) до зображення для отримання трансформованого зображення.

Для двох дискретних функцій $f(x, y)$ і $g(x, y)$ іхня згортка визначається як:

$$(h * g)(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n)g(x - m, y - n)$$

Згортка — це лінійна та інваріантна операція зсуву, що означає, що вихідні дані прямо пропорційні вхідним даним і на них не впливає положення вхідних функцій.

1.2.2 Деконволюція

Деконволюція - це операція, обернена до згортки. Вона спрямований на відновлення вихідного сигналу або зображення зі згорнутої (трансформованої) версії. Під час обробки зображень деконволюція зазвичай використовується для відновлення зображень, які були погіршені такими факторами, як розфокусування, розмиття руху або шум.

Математично проблему деконволюції можна сформулювати як знаходження вихідного зображення $f(x, y)$ за деградованим зображенням $h(x, y)$ і відомим ядром $g(x, y)$ таким чином, що:

$$h(x, y) = (f * g)(x, y)$$

Деконволюція є неправильно поставленою проблемою, оскільки процес деградації часто включає втрату інформації. Тому для отримання стабільного та значущого рішення потрібна регулярізація або додаткові обмеження.

Було розроблено різні алгоритми деконволюції для вирішення різних типів деградації зображення, включаючи фільтр Вінера, алгоритм Річардсона-Люсі та сліпу деконволюцію. Ці методи використовують різні припущення, моделі та стратегії оптимізації для оцінки оригінального зображення від погіршеного.

У контексті відновлення розфокусованого зображення розуміння принципів згортання та деконволюції має вирішальне значення для розробки ефективних алгоритмів для відновлення оригінального чіткого зображення з розмитої версії. Застосовуючи відповідні методи деконволюції, можна протидіяти ефекту розфокусування та відновити візуальну інформацію зображення.

1.3 Ітераційні методи відновлення зображення

Ітераційними методами (методами послідовних наближень) рішення задач називають такі методи, у яких по відомому наближенню шукається наступне, більш точне наближення. Найпростішим прикладом ітераційного алгоритму може бути перетворення рішення f^k в рішення f^{k+1} за допомогою поправок, обрахованих розкладом $f(x)$ у деякий ряд.

Ітераційні методи останнім часом знаходять все більш широке застосування у практиці відновлення зображення, так як у деяких випадках вони допускають просте урахування важливих для задач відновлення обмежень безпосередньо в схемі ітераційного алгоритму і тим самим представляють собою альтернативу методам нелінійного програмування. Основним запитанням застосування ітераційних методів являється збіжність ітерації. Для деяких типів ядер основного інтегрального рівняння доведені загальні положення про збіжність послідовних наближень.

Наприклад, нехай $h(x, \xi)$ - симетричне, достатньо визначене ядро, що належить L_2 , при чому $a \leq x, \xi \leq b$ і нехай рівняння

$$\int_a^b h(x, \xi) f(\xi)\, d\xi = g(x); g(x) \in L_2[a, b] \tag{1.1}$$

однозначно розв'язується. Тоді послідовність $\{f_k(x)\}$, визначається співвід-

ношенням

$$f^k(x) = f^{k-1}(x) + \lambda^{-1}[g(x) - g^{k-1}(x)]; k = 1, 2, ...$$

де

$$g^{k-1}(x) = \int_a^b h(x, \xi) f^{k-1}(\xi)\, d\xi$$
$$f^0 \in L_2[a, b]; 0 < 1/\lambda < 2/\lambda_{\max}$$

і $\lambda_{\max}$ - найбільше власне число ядра, збігається у середньому до розв'язку рівняння (1.1).

Розглянемо (1.1) у певному вигляді $Af = g$ і припустимо, що існує обернений оператор

$$f = A^{-1}g. \tag{1.2}$$

Обернений оператор можна розкласти у ряд Неймана:

$$A^{-1} = I + \sum_{n=1}^{\infty} (I - A)^n$$

де вираз $(I - A)^n$ відповідає цілій степені оператора, I - одиничний оператор, тобто $If = f$. Тоді вираз $f = A^{-1}g$ може бути представлений у вигляді:

$$f = g + \sum_{n=1}^{\infty} (I - A)^n g$$

Таким чином, процес знаходження розв'язка можна інтерпретувати, як знаходження поправок до спотвореного зображення:

$$f^0 = g;$$

$$f^1 = g + (I - A)g;$$

$$f^2 = g + (I - A)g + (I - A)^2 g.$$

На жаль, збіжність ряда Неймана залежить від вагової функції системи формування. Якщо не цікавитbсь питанням збіжності наближень, то розклад у ряді Неймана можна застосувати до розв'язку любого інтегрального рівняння, виділивши ту частину ядра рівняння, яка відрізняється від δ - функції. Це легко зробити для рівняння типу згортки, врахувавши, що перетворення Фур'є від δ - функції рівне одиниці. Беручи до уваги те, що

$$f(x) = F^(-1)\{(G(w))/(H(w))\},$$

$$f(x) = F^(-1)\{G(w)[1/(H(w)) - 1]\} + F^(-1)\{G(w)\},$$

$$f(x) = g(x) + 1/2\pi * \int_{-\infty}^{\infty} [1/(H(w)) - 1]G(w)\exp\{iwx\}\,dw \qquad (1.3)$$

Якщо система формування слабо спотворює зображення, то другий член являється лише невеликим доповненням до першого і, таким чином, можна застосувати метод розкладу у ряд Неймана.

Неважко побачити, що розклад оператора у ряд Неймана відповідає представленню члена $1/(H(w)) - 1$ у вигляді геометричної прогресії:

$$\frac{1}{H(w)} - 1 = \frac{1 - H(w)}{1 - [1 - H(w)]} = \sum_{n=1}^{\infty}[1 - H(w)]^n \qquad (1.4)$$

Якщо врахувати тільки перший член ряда (1.4), то розв'язок буде виглядіти так:

$$f(x) = g(x) + g(x) - h(x) * g(x).$$

Використовуючи біноміальну формулу, отримаємо

$$[1 - H(w)]^n = 1 - nH(w) + \frac{n(n-1)}{2!}H^2(w) - \frac{n(n-1)(n-2)}{3!}H^3(w) + ...$$

Розв'язок, що дозволяє врахувати n членів ряда, можна представити у вигляді $f(x) = g(x) + [ng(x) - h_n(x) * g(x)]$, де функція $h_n(x)$ виражається

наступним чином:

$$h_n(x) = \frac{(n+1)n}{2!}h(x) - \frac{(n+1)n(n-1)}{3!}h_1(x) + \frac{(n+1)n(n-1)(n-2)}{4!}h_2(x) + ...$$

Легко впевнитись, що за допомогою наведених виразів можна отримати послідовність розв'язків $\{f^k(x)\}$, що задовольняють рекурентним співвідношенням вигляду:

$$f^{k+1}(x) = g(x) + f^k(x) - h(x) * f^k(x)$$

$$f^0(x) = 0; f^1(x) = g(x); k = 0, 1, ...$$

або

$$f^{k+1)}(x) - f^k(x) = [f^k(x) - f^{k-1}(x)] - h(x) * [f^k(x) - f^{k-1}(x)]; k = 1, 2, ... \quad (1.5)$$

Можна визначити поправки до початкового розв'язку, розкладаючи $1/H(w) - 1$ у випадковий функціональний ряд:

$$\frac{1}{H(w)} - 1 = \sum_n a_n U_n \quad (1.6)$$

Тут важливо, щоб такий розклад збігався і дозволило при обчисленні поправок замінити операції над $f(x)$, визначені $1/H(w) - 1$, сукупність інших, визначених послідовністю $U_n(w)$, які більш зручніші для обчислення. Підставляючи (1.6) у (1.3) і змінюючи порядок сумування та інтегрування, отримаємо

$$f(x) = g(x) + \sum_n a_n F^{-1}\{U_n(w)G_n(w)\}. \quad (1.7)$$

Розглянемо розклад у степеневий ряд

$$1/H(w) - 1 = \sum_n a_n w^n$$

. По аналогії зі (1.7) знаходимо

$$f(x) = g(x) + \sum_n a_n F^{-1}\{w^n G(w)\}$$

Враховуючи властивість перетворення Фур'є

$$F^{-1}\{(iw)^n F(w)\} = \frac{d^n f(x)}{dx^n}$$

отримаємо

$$f(x) = g(x) + \sum_n \frac{a_n}{i^n} \frac{d^n g(x)}{dx^n}$$

Таким чином, розклад $1/H(w) - 1$ у степеневий ряд відповідає представленню $f(x)$ через $g(x)$ і його похідні. Отже, використання такого розкладу можна розглядати як прийом, що дозволяє перейти від інтеграла згортки до породжуючого його диференціального рівняння.

Зміст ітераційних алгоритмів у їх простому вигляді, що розглядався вище, зводиться до того, що пряме застосування оберненого оператора (у випадку рівняння згортки – інверсного фільтра) заміняється на послідовність кроків, при чому на кожному наступному кроці розв'язок все більше наближається до вихідного. Математично цей прийом означає відкидання у ряді Неймана членів, що перевищують деякий номер N, що дає обмеження смуги частот інверсного фільтра. Таким чином у викладеному варіанті ітераційні алгоритми відновлення – у тому числі у порівнянні із фільтрацією із обмеженням смуги частот.

Ітераційні алгоритми, набувають ряд значних переваг, якщо їх структура ускладнюється. Виявляється, що у загальну структуру ітераційного алгоритму можна ввести нелінійні обмеження, зробивши тим самим алгоритм нелінійним.

1.4 Ітераційний метод Річардсона-Люсі

Метод Річардсона-Люсі — це ітераційний алгоритм відновлення зображення, який використовується для відновлення чіткого зображення з погіршеного або розмитого зображення. Він припускає лінійну, змінно-інваріантну модель деградації з додатковим шумом. Алгоритм має на меті знайти оцінку максимальної правдоподібності справжнього зображення з урахуванням спостережуваного погіршеного зображення та моделі погіршення.

Модель деградації:

$$f = H * g + n$$

де f — спостережене погіршене зображення, g — справжнє базове зображення, H — оператор деградації (наприклад, функція розкиду точок), а n — додатковий шум.

Зворотна модель деградації:

$$g = H^{-1} * f$$

де $H^{-}1$ — псевдообернений або обернений оператор H.

Тоді рівняння Річардсона-Люсі:

$$g_{k+1} = g_k * (H^T * (f/(H * g_k)))/(H^T * (H * g_k))$$

де g_k — оцінка справжнього зображення на ітерації k, H^T — транспонування H, * позначає згортку, а / позначає поелементний поділ.

Метод ітераційно оновлює оцінку справжнього зображення за допомогою рівняння оновлення Річардсона-Люсі. Він уточнює оцінку, враховуючи модель деградації та спостережуване деградоване зображення. Ітерації тривають до тих пір, поки не буде виконано критерій зупинки, такий як максимальна кількість ітерацій або критерій конвергенції.

Алгоритм Річардсона-Люсі поступово покращує оцінку справжнього зображення, використовуючи знання про модель деградації та враховуючи спостережене деградоване зображення. Він зближується до відновленого зображення, яке мінімізує різницю між оціненим погіршеним зображенням і спостережуваним погіршеним зображенням, враховуючи характеристики шуму. Алгоритм прагне відновити чіткі деталі вихідного зображення з його розмитої версії.

1.5 Деконволюція Вінера

Деконволюція Вінера — це техніка лінійної фільтрації, яка використовується для відновлення зображень, погіршених такими факторами, як розмиття та шум. Він заснований на принципах оцінки найменших середніх квадратів і мінімальної середньоквадратичної помилки (MMSE). Метод деконволюції Вінера припускає, що вихідне зображення, ядро розмиття та шум є стаціонарними стохастичними процесами, і він прагне мінімізувати середню квадратичну помилку між оціненим зображенням і вихідним зображенням.

Дано деградоване зображення $h(x, y)$, яке є згорткою вихідного зображення $f(x, y)$ і ядра розмиття $g(x, y)$ разом із додатковим шумом $n(x, y)$, співвідношення можна виразити так:

$$h(x, y) = (f * g)(x, y) + n(x, y)$$

У частотній області це співвідношення стає таким:

$$H(u, v) = F(u, v)G(u, v) + N(u, v)$$

де $H(u, v)$, $F(u, v)$, $G(u, v)$ і $N(u, v)$ є перетвореннями Фур'є $h(x, y)$, $f(x, y)$, $g(x, y)$ і $n(x, y)$ відповідно.

Фільтр деконволюції Вінера призначений для мінімізації середньоквадратичної помилки між оціненим зображенням $\hat{F}(u,v)$ і вихідним зображенням $F(u,v)$. Фільтр можна виразити так:

$$W(u,v) = \frac{G^*(u,v)S_{ff}(u,v)}{|G(u,v)|^2 S_{ff}(u,v) + S_{nn}(u,v)}$$

де $G^*(u,v)$ — комплексно спряжене $G(u,v)$, $S_{ff}(u,v)$ — спектральна щільність потужності (PSD) вихідного зображення, і $S_{nn}(u,v)$ — PSD шуму. Фільтр Вінера враховує наявний у зображенні шум під час оцінки вихідного зображення, що забезпечує кращі результати відновлення порівняно з простою зворотною фільтрацією.

Щоб отримати відновлене зображення, застосовується фільтр Вінера до деградованого зображення у частотній області:

$$\hat{F}(u,v) = W(u,v)H(u,v)$$

Далі обчислюється зворотне перетворення Фур'є $\hat{F}(u,v)$, щоб отримати відновлене зображення у просторовій області.

1.6 Індекс структурної подібності (SSIM)

Індекс структурної подібності (SSIM) — це широко використовуваний показник для оцінки якості зображення шляхом його порівняння з еталонним зображенням. Він призначений для оцінки сприйманої якості зображень шляхом врахування змін у структурній інформації, яскравості та контрастності, які є важливими аспектами зорової системи людини. SSIM забезпечує більш точне відображення сприйнятої якості зображень порівняно з традиційними показниками, такими як середня квадратична помилка (MSE) або пікове спів-

відношення сигнал/шум (PSNR), які базуються на піксельних відмінностях і не завжди узгоджуються з людським сприйняттям.

Для двох зображень x і y, SSIM обчислюється за такою формулою:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

де:

- μ_x і μ_y локальні середні значення зображень x і y відповідно,
- σ_x^2 і σ_y^2 локальні дисперсії зображення x і y відповідно,
- σ_{xy} — локальна коваріація між зображенням x і y,
- C_1 і C_2 є малими константами для стабілізації ділення, коли знаменник близький до нуля. Зазвичай вони визначаються як $C_1 = (K_1L)^2$ і $C_2 = (K_2L)^2$, де L — динамічний діапазон значень пікселів (наприклад, 255 для 8-бітних зображень), а K_1 і K_2 — малі константи (наприклад, $K_1 = 0,01$ і $K_2 = 0,03$).

Значення SSIM коливається від -1 до 1, де 1 означає ідеальну відповідність між зображеннями, а нижчі значення вказують на більшу відмінність.

SSIM використовується для оцінки якості відновленого зображення, порівнюючи його з вихідним зображенням. Оптимізуючи значення SSIM під час процесу відновлення, можна знайти найкращі параметри відновлення, щоб отримати візуально подібний результат до вихідного зображення, враховуючи інформацію про структуру, яскравість і контраст, важливу для людського сприйняття.

1.7 Метод пошуку по сітці

Метод пошуку по сітці — це простий, але потужний метод оптимізації для пошуку найкращих параметрів для заданої проблеми. Він включає у

себе вичерпний пошук у попередньо визначеному наборі значень параметрів, щоб визначити комбінацію, яка дає найвищу продуктивність відповідно до визначеної метрики оцінки.

У контексті задачі одновимірної оптимізації для відновлення розфокусованого зображення мета полягає у тому, щоб знайти найкращий радіус розмиття (або параметр розфокусування), який забезпечує найвищу якість зображення після відновлення. Метод пошуку сітки відбувається наступним чином:

1. Визначити простір пошуку для радіуса розмиття, який є набором значень, наприклад, $R = \{r_1, r_2, ..., r_n\}$.

2. Для кожного потенційного радіуса розмиття $r_i \in R$ виконуються наступні дії:

   - Відновлюється розфокусоване зображення, використовуючи поточний радіус розмиття r_i.

   - Обчислюється оціночну метрику (SSIM) між відновленим і вихідним зображеннями.

3. Обирається радіус розмиття, який максимізує показник оцінки.

Математично метод пошуку сітки формулюється так:

$$r^* = \arg\max_{r_i \in R} f(r_i)$$

де:

- r^* - оптимальний радіус розмиття,

- R - набір значень радіуса розмиття,

- $f(r_i)$ — метрика оцінки (SSIM), обчислена для відновленого зображення з радіусом розмиття r_i.

Метод пошуку по сітці є дорогим з точки зору обчислень, оскільки вимагає оцінки продуктивності для кожного значення у просторі пошуку. Однак це гарантує, що буде знайдено найкраще рішення у попередньо визначеному просторі пошуку. Цей метод може бути особливо корисним у випадках, коли проблема оптимізації не має відомого аналітичного рішення або простір пошуку нелегко піддається методам оптимізації на основі градієнта.

2 Проблеми, що виникають у процесі реставрації розфокусованого зображення

Процес відновлення розфокусованих зображень є складним завданням, яке передбачає розв'язання кількох проблем і складнощів. Деякі з ключових проблем, які виникають у процесі відновлення розфокусованого зображення:

1. **Посилення шуму:** Під час процесу відновлення шум, присутній у спостережуваному розфокусованому зображенні, може посилюватися. Це відбувається тому, що алгоритми відновлення спрямовані на відновлення високочастотних деталей і країв, які також чутливі до шуму. Таким чином, у процесі відновлення необхідно знайти баланс між зменшенням розмиття та збереженням деталей зображення при мінімізації посилення шуму.

2. **Обчислювальна складність:** алгоритми відновлення зображень можуть потребувати інтенсивних обчислень, особливо при роботі з великими зображеннями або складними моделями деградації. Цей процес часто включає розв'язання проблем оптимізації, виконання згорток і застосування складних математичних операцій. Ефективні алгоритми та обчислювальні методи необхідні для забезпечення розумного часу обробки для реального часу або великомасштабних програм.

3. **Невідповідність моделі:** точне відновлення вимагає хорошого розуміння процесу деградації та відповідної математичної моделі. Однак на практиці часто важко мати точні знання про точне ядро розмиття розфокусування або характеристики шуму. Невідповідність моделі може призвести до неоптимальних результатів відновлення, оскільки припущення, зроблені в алгоритмі відновлення, можуть не повністю відповідати реаль-

ній деградації.

4. **Варіації глибини:** розмиття розфокусування змінюється залежно від глибини сцени. Об'єкти на різних відстанях від камери демонструють різний ступінь розмиття. Робота з варіаціями глибини вносить додаткові складності в процес відновлення, оскільки ядро розмиття може знадобитися оцінити або адаптувати для різних глибин зображення.

5. **Компроміс між різкістю та артефактами:** Відновлення розфокусованого зображення передбачає компроміс між збільшенням різкості та введенням артефактів. Агресивна деконволюція або підвищення різкості може покращити високочастотні деталі, але також може викликати артефакти дзвінка або посилення шуму. Збалансування між різкістю та артефактами має вирішальне значення для досягнення візуально приємних результатів відновлення без артефактів.

Розв'язання цих проблем вимагає розробки розширених алгоритмів відновлення, що включають методи регуляризації, методи обробки шуму, надійні підходи до оптимізації та адаптивні моделі.

3 Багатопараметрична задача відновлення розфокусованого зображення

Через велику кількість проблем, що виникають під час реставрації зображень було розглянуто більш детальну багатопараметричну задачу. Алгоритм, який було реалізовано для відновлення розфокусованого зображення, складається з наступних кроків:

3.1 Попередня обробка

Спочатку вхідне зображення попередньо обробляється для зменшення артефактів на краях, спричинених розмиттям. Це робиться шляхом застосування розмиття за Гаусом до зображення, а потім поєднання його з вихідним зображенням за допомогою середньозваженого.

Формула розмиття за Гаусом:
Враховуючи вхідне зображення I і ядро розмиття K, операцію розмиття за Гаусом можна визначити як:

$$I_{\text{blur}} = I * K,$$

де I_{blur} представляє розмите зображення, а $*$ позначає операцію згортання.

У разі двовимірного розмиття за Гаусом ядро K генерується на основі розподілу за Гаусом. Формула для двовимірної функції Гауса:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

де x і y представляють координати пікселів, а σ — стандартне відхилення, яке контролює ступінь розмиття.

Щоб застосувати розмиття за Гаусом, ядро K центрується в місці кожного пікселя вхідного зображення, а операція згортки виконується для обчислення середньозваженого сусідніх пікселів. Отримане значення представляє інтенсивність розмитих пікселів у вихідному зображенні I_{blur}.

3.2 Обчислення рівня шуму

Рівень шуму в розмитому зображенні оцінюється на основі даного відношення сигнал/шум (SNR) у децибелах. SNR є мірою відношення потужності сигналу до потужності шуму в зображенні. Вищий SNR вказує на вищу якість зображення з меншим шумом, тоді як нижчий SNR вказує на більше шуму в зображенні.

Тоді рівень шуму обчислюється за такою формулою:

```
noise = 10 ** (-0.1 * SNR_db)
```

Тут 'SNR_db' - це співвідношення сигнал/шум у децибелах. Формула перетворює SNR з децибел на лінійну шкалу, а потім обчислює відповідний рівень шуму на основі цього лінійного значення. Рівень шуму використовується на етапі регулярізації, щоб запобігти діленню на малі чи нульові значення, що може спричинити чисельну нестабільність. Оцінюючи рівень шуму на основі вхідного SNR, алгоритм може ефективно зменшити шум у відновленому зображенні, зберігаючи деталі зображення.

3.3 Моделі розмиття

Алгоритм підтримує два типи моделей розмиття - розмиття у русі та розмиття з розфокусуванням. Розмиття у русі імітує ефект руху камери або

об'єкта під час експозиції, тоді як розмиття у вигляді розфокусування імітує ефект розфокусування.

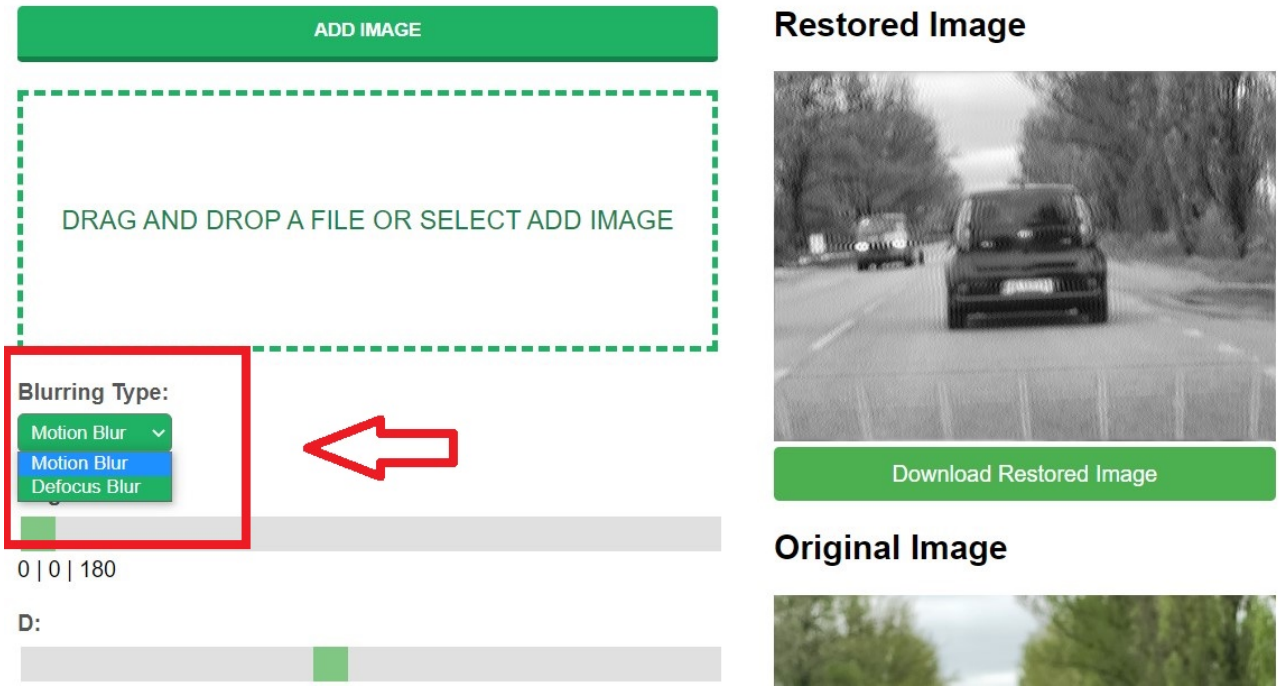

Рис. 1: Вибір типу моделі розмиття

3.4 Генерація ядра

Залежно від обраної моделі розмиття, алгоритм генерує відповідне ядро розмиття. Для розмиття руху створюється лінійне ядро на основі вказаного кута та довжини. Для розмиття дефокусування створюється кругле ядро на основі вказаного діаметра.

1. **Розмиття за Гаусом:**

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

де $G(x, y)$ — функція Гауса, x і y — відстань від центру, а σ — стандартне відхилення функції Гауса.

2. **Ваги відстані:**

$$w(x,y) = \begin{cases} \frac{d(x,y)}{d}, & \text{якщо } d(x,y) \leq d \\ 1, & \text{інакше} \end{cases}$$

де $w(x,y)$ — вага у позиції (x,y), $d(x,y)$ — відстань від межі зображення у позиції (x,y), а d — радіус розмиття.

3. **Матриця обертання:**

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

де θ - кут повороту.

4. **Ядро розмиття у русі:**

$$K(x,y) = \begin{cases} 1. & \text{якщо } x = \frac{s-1}{2} \\ 0, & \text{інакше} \end{cases}$$

де $K(x,y)$ — це ядро в позиції (x,y), s — довжина ядра, а ядро — це горизонтальна лінія одиниць з центром у ядрі. Потім це ядро повертається на кут θ для імітації розмиття руху в певному напрямку.

5. **Ядро розмиття розфокусування:**

$$K(x,y) = \begin{cases} 1, & \text{якщо } (x-c_x)^2 + (y-c_y)^2 \leq r^2 \\ 0, & \text{інакше} \end{cases}$$

де $K(x, y)$ — ядро у позиції (x, y), (c_x, c_y) — центр ядра, r — радіус кола, а ядро — круговий диск з одиницями всередині та нулями зовні. Потім це ядро нормалізується, щоб мати значення від 0 до 1.

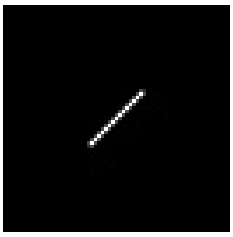

Рис. 2: Ядро руху

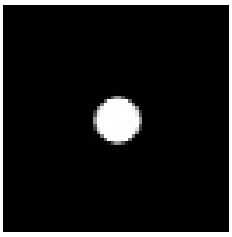

Рис. 3: Ядро розфокусування

3.5 Обробка в частотній області

Розмите зображення перетворюється в частотну область за допомогою швидкого перетворення Фур'є (ШПФ). Це дозволяє ефективно обробляти та аналізувати зображення у частотній області.

3.6 Деконволюція

Щоб відновити чітке зображення з розмитого зображення, алгоритм застосовує метод деконволюції Вінера.

3.7 Зворотне перетворення

Після процесу деконволюції відновлене зображення отримується шляхом перетворення модифікованого представлення частотної області назад у просторову область за допомогою зворотного перетворення Фур'є.

3.8 Збільшення різкості

```
def edge_enhancement(image):
    sharpen_filter = np.array([[-1, -1, -1],
                               [-1, 9, -1],
                               [-1, -1, -1]])

    sharp_image = cv2.filter2D(image, -1, sharpen_filter)

    return sharp_image
```

Функція 'edge_enhancement' у наданому фрагменті коду призначена для застосування фільтра збільшення різкості до вхідного зображення для покращення його країв і видалення шуму. Фільтр, який використовується у цьому випадку, є простим фільтром високих частот, представленим ядром 3x3.

Ядро, позначене як H, визначається як:

$$H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Метою цього ядра є посилення високочастотних компонентів у зображенні, одночасно зменшуючи низькочастотні компоненти (наприклад, гладкі області).

Щоб застосувати фільтр збільшення різкості до зображення, використовується операцію згортки між вхідним зображенням, позначеним як I, і ядром H. Отримане чітке зображення, позначене як I_{sharp}, можна обчислити як:

$$I_{sharp}(x, y) = (I * H)(x, y) = \sum_{i=-a}^{a} \sum_{j=-a}^{a} I(x - i, y - j)H(i, j)$$

Тут $*$ позначає операцію згортання, (x, y) — координати точки на зображенні, а a — половина ширини ядра, яка в даному випадку дорівнює 1.

Фільтр підвищення різкості працює шляхом множення значення центрального пікселя на позитивну вагу (у цьому випадку 9) і віднімання від нього значень сусідніх пікселів (помножених на -1). Це підкреслює відмінності між центральним пікселем і його сусідами, таким чином підкреслюючи краї та роблячи їх більш помітними.

3.9 Післяобробка

Отримане відновлене зображення коригується та нормалізується до потрібного діапазону (від 0 до 255), щоб забезпечити належне візуальне представлення.

Алгоритм поєднує методи обробки зображень, обробки сигналів і математичної оптимізації для відновлення розфокусованих зображень. Вибір методу деконволюції Вінера мотивований його ефективністю у розв'язання проблеми зменшення розмитості, враховуючи присутність шуму на зображенні. Включаючи ядро розмиття та оцінку шуму, деконволюція Вінера забезпечує надійний підхід до відновлення чіткого зображення з розмитої версії.

4 Розробка користувацького інтерфейсу

Метою створення інтерфейсу користувача для відновлення розфокусованих зображень у воєнний час може бути підвищення ефективності та результативності військових дій. Під час війни військовослужбовцям може знадобитися робити знімки у складних умовах, наприклад у поганому освітленні або під час руху. Ці умови можуть призвести до розмитих або розфокусованих зображень, що може ускладнити ідентифікацію цілей або точну оцінку ситуації.

Забезпечивши інтерфейс користувача для відновлення розфокусованих зображень, військовослужбовці можуть швидко та легко відновити розмиті або розфокусовані зображення до більш зручного стану. Це може допомогти їм точніше визначати цілі, ефективніше оцінювати ситуацію та приймати більш обґрунтовані рішення.

Крім того, наявність інтерфейсу користувача для відновлення розфокусованих зображень може зменшити потребу у спеціалізованому обладнанні або персоналі, який має спеціальну підготовку з відновлення зображень. Це може зробити відновлення зображень більш доступним для ширшого кола військовослужбовців, потенційно збільшуючи кількість осіб, які можуть використовувати цей алгоритм.

Інтерфейс користувача виглядає як веб-сторінка для видалення розмитості зображень. Далі опис інтерфейсу:

1. Основний розділ інтерфейсу містить форму, яка дозволяє користувачам завантажувати файл зображення зі свого локального комп'ютера. Форма містить кнопку «Додати зображення», область перетягування для завантажених зображень.

2. Існує випадаюче меню «Blurring Type», яке дозволяє користувачеві вибрати «Розмиття у русі» або «Розмиття з розфокусуванням».

3. Випадаюче меню «Noice», яке дозволяє користувачеві чи застосовувати додатково метод видалення шуму.

4. Інтерфейс містить три повзунки для налаштування параметрів розмиття: «Кут», «Діаметр/Довжина» і «SNR (дБ)». Кожен повзунок має мітку, діапазон значень і числове зчитування поточного значення.

5. Кнопка «Deblur», яка ініціює процес відновлення зображення.

6. Під формою є два контейнери зображень: один для відновленого зображення та один для оригінального зображення. Контейнер відновленого зображення містить кнопку завантаження для збереження файлу відновленого зображення.

7. Відображається ядро

8. Розмір контейнерів зображень динамічно змінюється відповідно до розміру завантажених зображень.

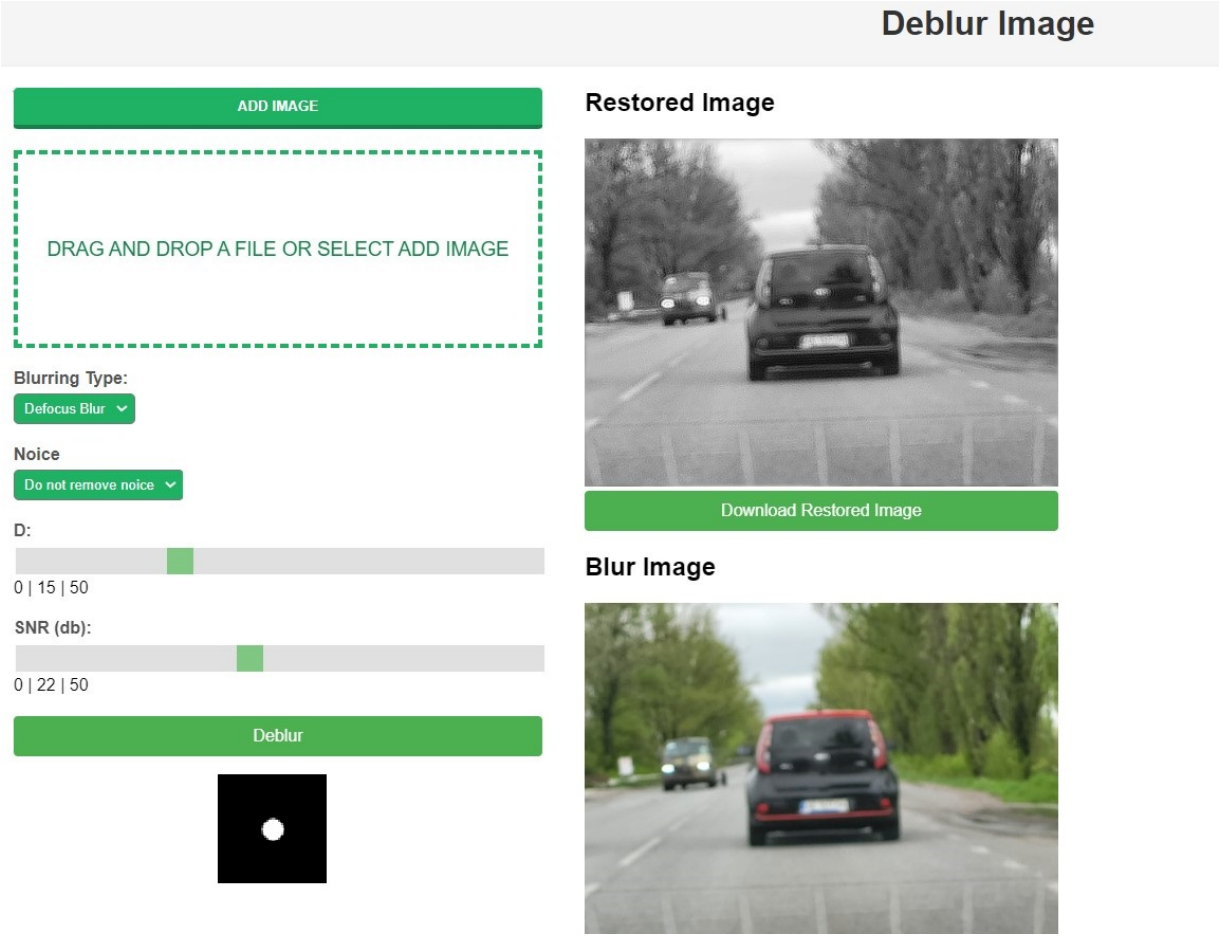

Рис. 4: Користувацький інтерфейс

Користувальницький інтерфейс є простим і легким у використанні, з чіткими мітками та інтуітивно зрозумілими елементами керування для налаштування параметрів розмиття.

5 Експеременти

Далі наведені результати експеременту реалізованого алгоритму для одновимірної задачі оптимізації. Наступні експеременту будуть проведені для даного зображення:

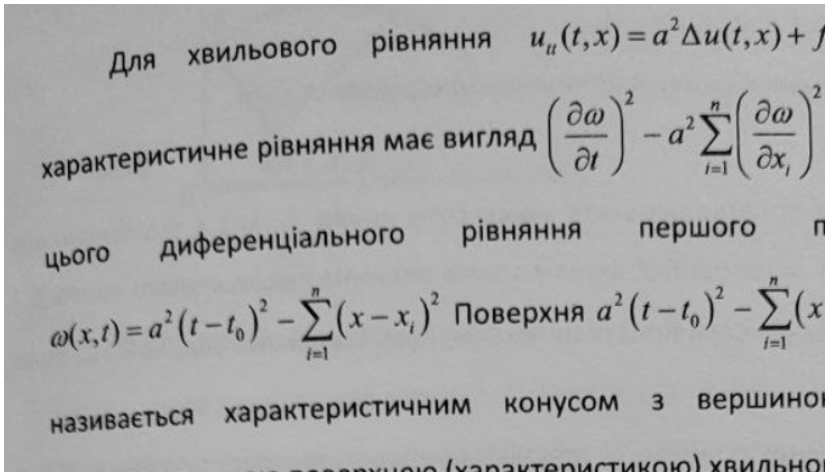

Рис. 5: Оригінальне зображення 1

5.1 Результати для штучно розфокусованого зображення

Алгоритм використовує деконволюцію Вінера для відновлення розфокусованих зображень. Функція приймає розфокусоване зображення та параметр розмиття (σ) як вхідні дані та повертає відновлене зображення. Ядро розмиття за Гауссом генерується за допомогою заданої σ, а потім для відновлення зображення застосовується деконволюція Вінера. Розфокусоване зображення відновлюється за допомогою оптимального значення σ, знайденого під час пошуку по сітці. Алгоритм виконує задачу одновимірної оптимізації, щоб знайти найкраще значення σ, яке максимізує SSIM між вхідним зображенням і відновленим зображенням, надаючи оцінку оптимальних параметрів для від-

новлення розфокусованого зображення.

Результат для значення $\sigma = 2$ та 200 ітерацій:

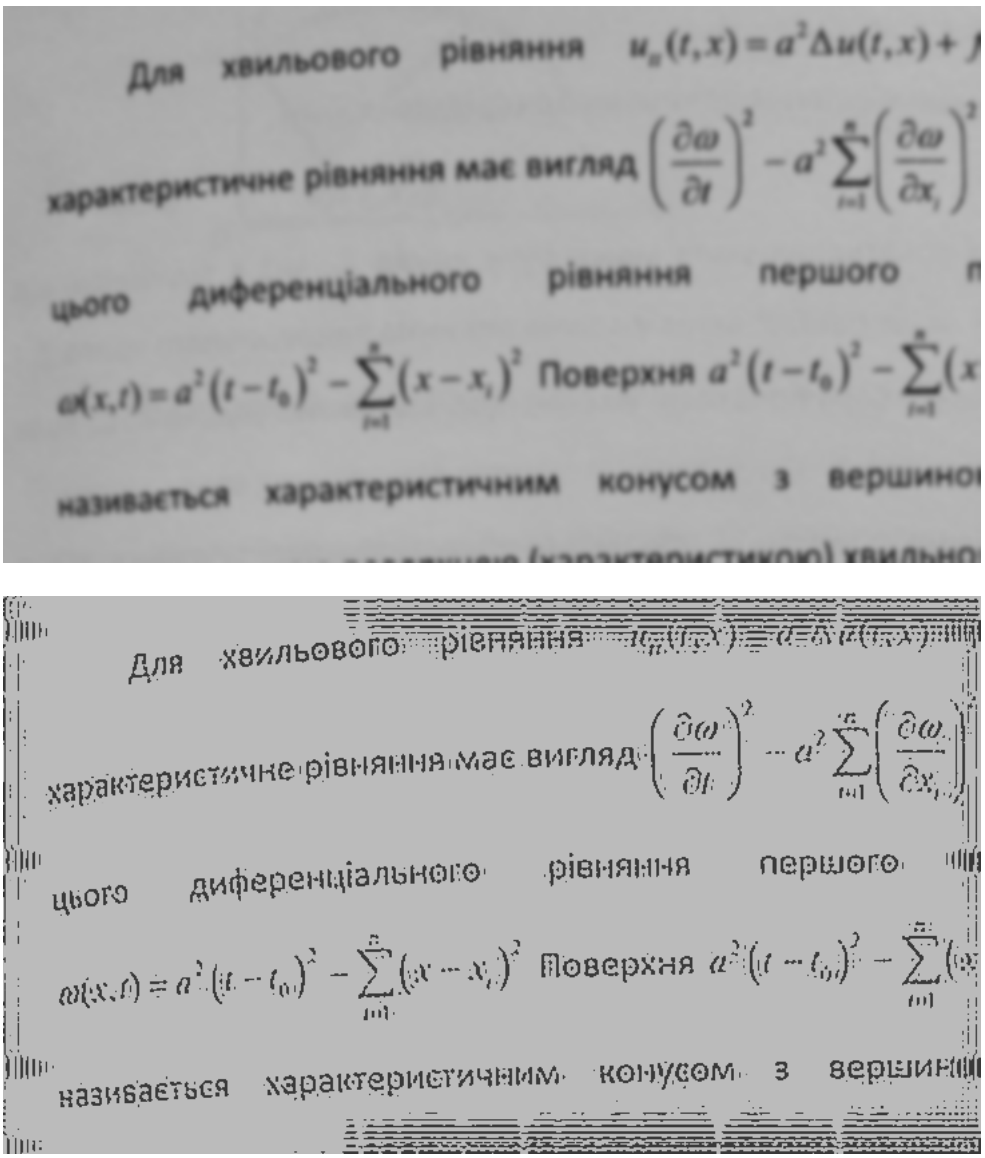

Рис. 6: Optimal sigma: 2.6733668341708543, Best score: 0.7072650422110272

Результат для значення $\sigma = 2.5$ та 200 ітерацій:

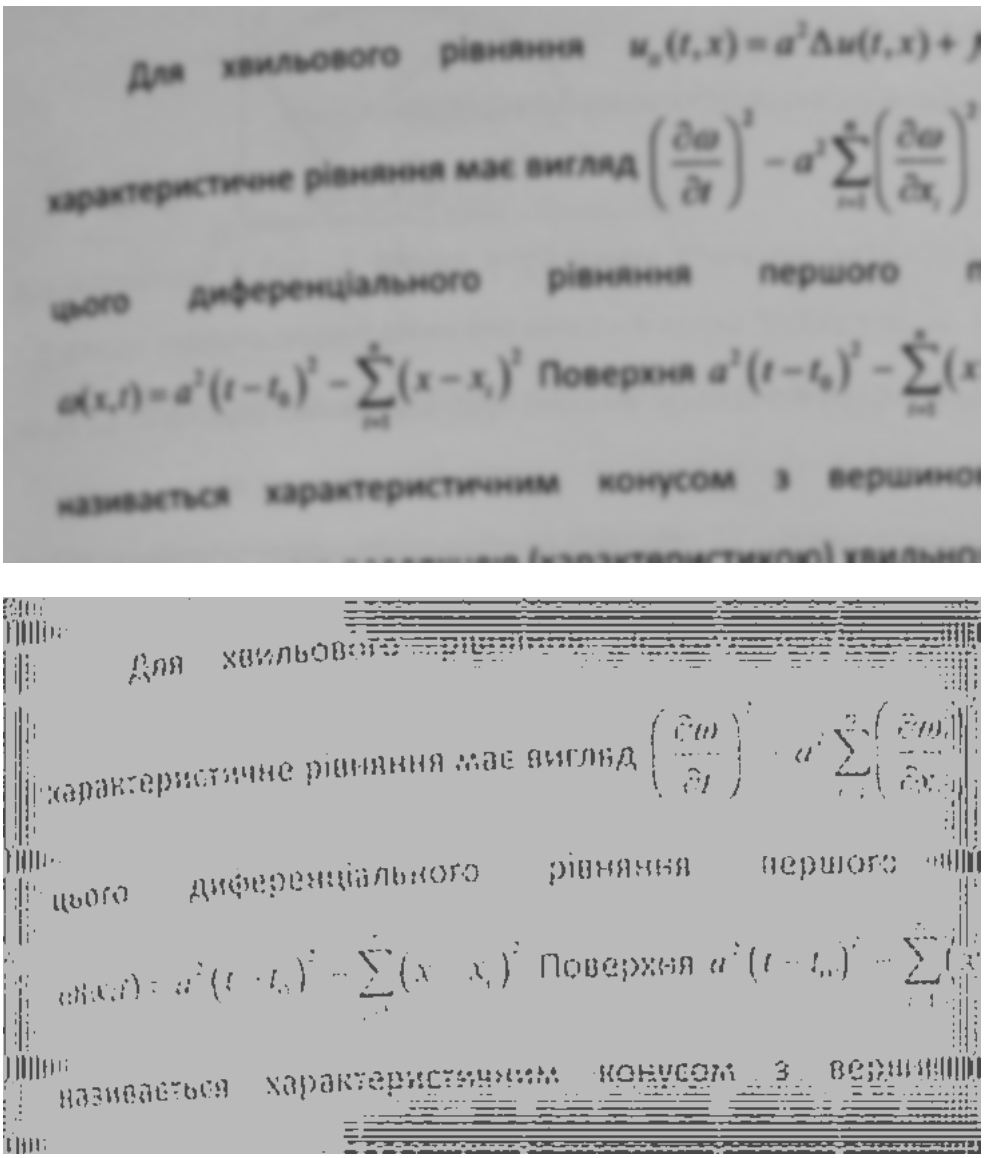

Рис. 7: Optimal sigma: 3.035175879396985 Best score: 0.6189428050595108

Результат для значення $\sigma = 2.5$ та 500 ітерацій: Optimal sigma: 2.677354709418837 Best score: 0.7053659286854167. Зображення майже не відрізняються від результату у 200 ітерацій. З наведеної нижче таблиці 1 результатів можна зробити наступні висновки: чим менша σ тим кращі результати SSIM (Best Score), у жодному випадку оптимальна σ не відповідає заданій σ.

Табл. 1: Результат алгоритму розфокусування

Sigma Blur	Optimal Sigma	Best Score
0.5	1.0	0.782194
0.6	1.045226	0.788058
0.7	1.135678	0.791398
0.8	1.361809	0.800880
0.9	1.407035	0.798622
1.0	1.452261	0.789309
1.1	1.678392	0.796151
1.2	1.678392	0.789319
1.3	1.768844	0.775793
1.4	2.040201	0.770776
1.5	2.040201	0.770575
1.6	2.040201	0.752376
1.7	2.356784	0.740418
1.8	2.356784	0.736387
1.9	2.356784	0.722174
2.0	2.673367	0.707265
2.1	2.673367	0.703018
2.2	2.673367	0.683265
2.3	2.673367	0.660710
2.4	3.035176	0.639625
2.5	3.035176	0.618943

5.2 Результати для штучно розфокусованого зображення у русі

Алгоритм використовує ітераційний метод Річардсона-Люсі для відновлення розфокусованих зображень. Функція приймає розфокусоване зображення, параметр розмиття (σ) та кут розмиття ($angle$)

Результат для значення $\sigma = 15$ та $angle = 45$:

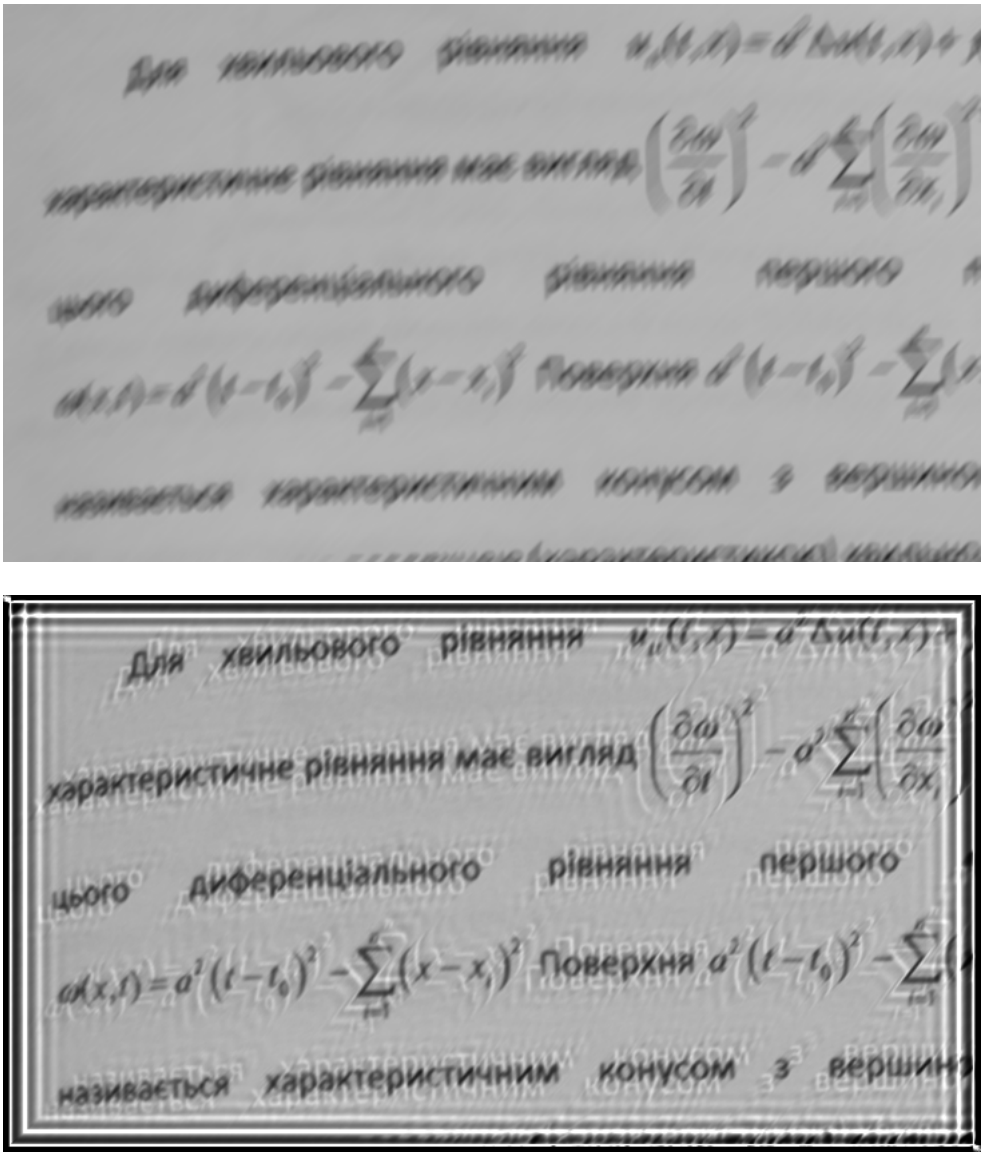

Рис. 8: Optimal kernel size: 15 Optimal angle: 43.0 Best score: 0.5908195857280865

Результат для значення $\sigma = 15$ та $angle = 88$:

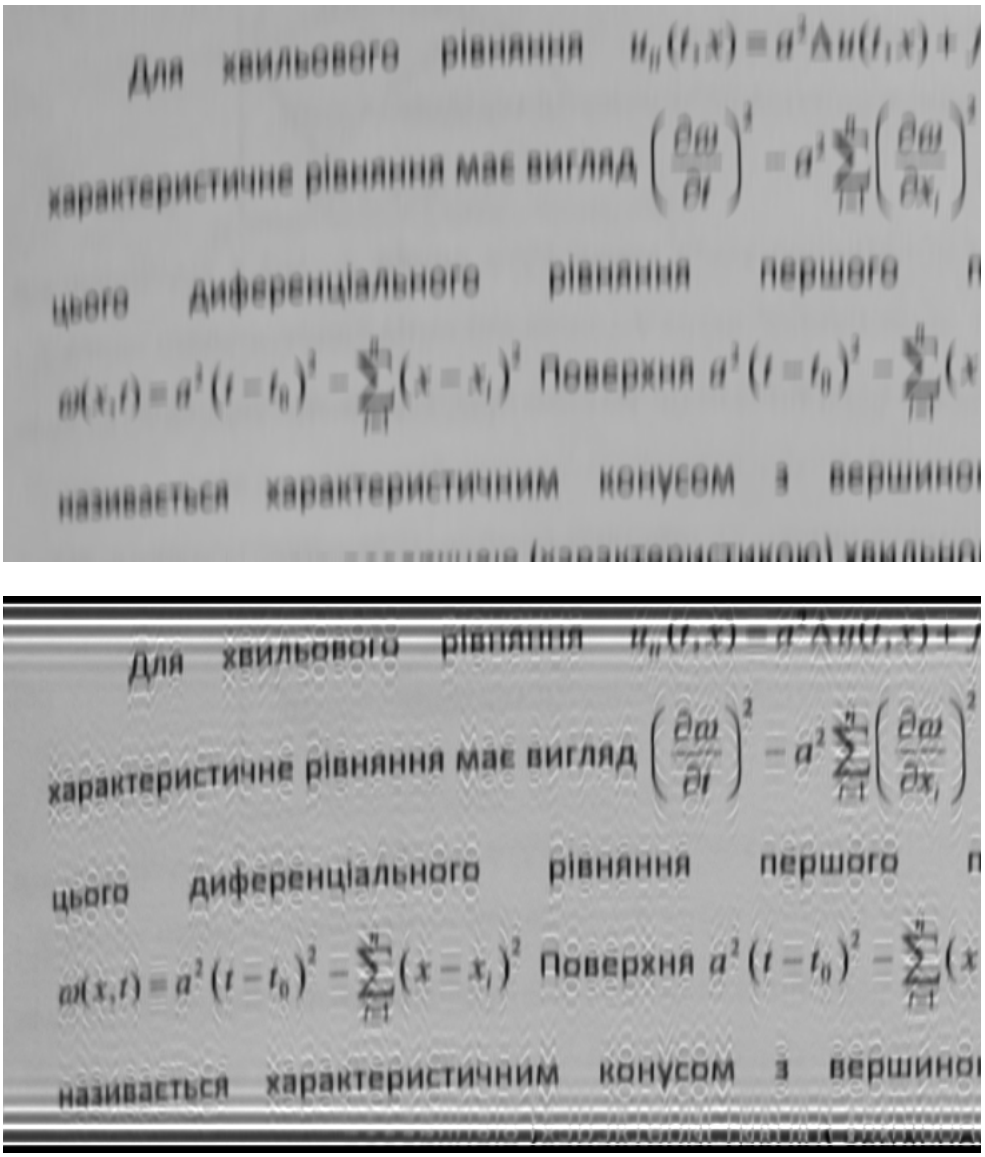

Рис. 9: Optimal kernel size: 13 Optimal angle: 89.0 Best score: 0.6516324132780327

Результат для значення $\sigma = 20$ та $angle = 88$:

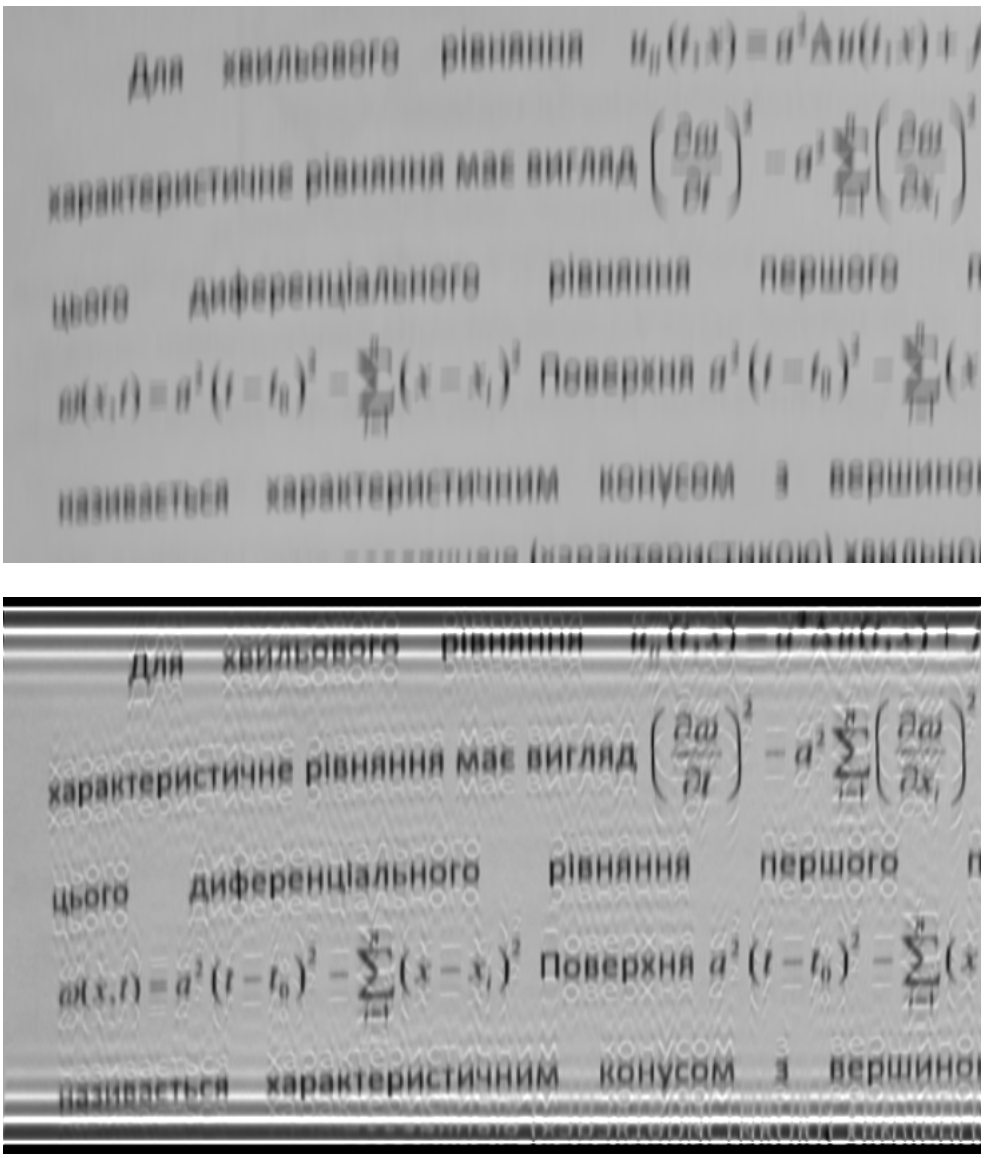

Рис. 10: Optimal kernel size: 17 Optimal angle: 88.0 Best score: 0.5777777571259233

За результатами таблиці 2 можна зробити наступні висновки: оптимальна σ майже у всіх випадках менша за початкову σ та оптимальний кут у більшості випадках збігається з початковим. Також за цією таблицею можна сказати, що SSIM є кращим у більшості випадків за SSIM розфокусування.

Табл. 2: Результат алгоритму розмиття у русі при заданому куті 88

Kernel	Optimal Kernel Size	Optimal Angle	Best Score
5	3	85.0	0.902171
7	7	87.0	0.817276
9	9	87.0	0.756282
11	11	88.0	0.713487
13	13	88.0	0.679236
15	13	89.0	0.651632
17	15	88.0	0.617382
19	17	88.0	0.578134
21	19	88.0	0.548983

5.3 Результати багатопараметричної задачі

Далі приклад реальних розфокусованих зображень та їх реставрацій за допомогою реалізованого алгоритму для багатопараметричної задачі:

Рис. 11: Розфокусоване та реставроване зображення 2

Рис. 12: Розфокусоване зображення 3

Рис. 13: Розфокусоване та реставроване зображення 3

Рис. 14: Номера авто

Висновки

Відновлення розфокусованого зображення є важливою проблемою в обробці зображень. За останні роки досягнуто значного прогресу в розробці алгоритмів і методик відновлення розфокусованих зображень. Кожне розфокусоване реальне зображення вимагає "індивідульного"підходу у реставрації, тому питання реставрації росфокусованих зображень є досить складною і дуже актульною задачею у сучасному світі.

Попри досягнутий прогрес, все ще існує багато проблем у відновленні розфокусованого зображення, включаючи наявність шуму, складність визначення ядра розмиття та обчислювальну складність деяких алгоритмів. Тому були досліджені та проаналізовані різні методи відновлення, такі як деконволюція Вінера та ітераційні методи відновлення зображень. Однак вибір найефективнішої техніки значною мірою залежить від типу та тяжкості розмиття, наявності шуму та наявності попередньої інформації.

Було реалізовано два алгоритми для одновимірної задачі оптимізації (для розфокусованого зображення та для розфокусованого у русі). Проведено багато експерементів та наведені таблиці з результами. Експеременти показали: чим менше розмиття, тим краще відновлення за умови "видалення"лише розмиття. Якщо розмиття є сильним, то зображення вимагають додаткової обробки за допомогою додаткових алгоритмів. Дані алгоритми є корисними для навчання нейроних мереж, що будуть спеціалізуватися на реставрації зображень.

Через індивідальність у питанні реставрації було реалізовано алгоритм для багатопараметричної задачі реставрації зображень з користувацьким інтерфейсом, де також був використаний метод деконволюції Вінера та методи для поглинення шуму та артефактів.

Реалізація зручного інтерфейсу для відновлення розфокусованих зображень може значно підвищити доступність і ефективність процесу відновлення, дозволяючи людям з обмеженими знаннями обробки зображень легко відновлювати розфокусовані зображення.

Джерела

[1] W.Prett. Digital Image Processing Redwood Shores. California. 1991p. - 698c.

[2] Jiaya Jia. Single Image Motion Deblurring Using Transparency. -8.

[3] R.Gonzalez, R.Woods. Digital Image Processing Prentie Hall Upper Saddle River, New Jersey. 2006. -1070c.

[4] Fan Lin, Yingpin Chen. Image Deblurring under Impulse Noise via Total. 2019p. -24c Generalized Variation and Non-Convex Shrinkage

[5] Fasheng Yang, Yongmei Huang. Robust Image Restoration for Motion Blur of Image Sensors.

[6] Abdullah Abuolaim, Mahmoud Afifi, and Michael S Brown. Improving single-image defocus deblurring. 2022p .-1231–1239c.

[7] Sebastian Berisha. Iterative Methods for Image Restoration. 2017p. -60c

[8] [Електронний ресурс] : https://www.mathworks.com/

[9] [Електронний ресурс] : https://en.wikipedia.org/wiki/Deconvolution

[10] [Електронний ресурс] : https://en.wikipedia.org/wiki/Wiener_deconvolution

Додатки

Додаток А

```
import numpy as np
import cv2
from skimage.metrics import structural_similarity as ssim
from skimage.restoration import wiener


def gaussian_blur(image, sigma):
    return cv2.GaussianBlur(image, (0, 0), sigma)


def generate_gaussian_kernel(size, sigma):
    ax = np.linspace(-(size // 2), size // 2, size)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-(xx ** 2 + yy ** 2) / (2 * sigma ** 2))
    return kernel / kernel.sum()


def wiener_deconvolution(image, kernel, K):
    restored_image = wiener(image, kernel, K)
    return restored_image

def restore_defocused_image(image_defocused, sigma):
    kernel_size = int(6 * sigma) | 1
```

```
    gaussian_kernel = generate_gaussian_kernel(kernel_size, sigma)
    K = 10 ** -6
    restored_image = wiener_deconvolution
        (image_defocused, gaussian_kernel, K)
    min_val, max_val = np.min(image_defocused),
        np.max(image_defocused)
    restored_image = np.interp(restored_image,
        (restored_image.min(), restored_image.max()),
            (min_val, max_val))
    restored_image = np.clip(restored_image,
        min_val, max_val).astype(np.uint8)

    return restored_image

def evaluate_mse(image1, image2):
    return np.mean((image1 - image2) ** 2)

def evaluate_ssim(image1, image2):
    return ssim(image1, image2, data_range=image2.max() - image2.min())

def find_optimal_sigma(image_defocused, image_original, sigma_values):
    best_score = None
    best_sigma = None

    for sigma in sigma_values:
        restored_image =
```

```
            restore_defocused_image(image_defocused, sigma)
        score = evaluate_ssim(image_original, restored_image)
        if best_score is None or score > best_score:
            best_score = score
            best_sigma = sigma

    return best_sigma, best_score

if __name__ == '__main__':
    image_original = cv2.imread('12.png', 0)
    sigma_blur = 0.5
    image_defocused = gaussian_blur(image_original, sigma_blur)
    sigma_values = np.linspace(1, 10, 500)
    optimal_sigma, best_score =
        find_optimal_sigma(image_defocused,
            image_original, sigma_values)
    print("Optimal sigma:", optimal_sigma, "Best score:", best_score)
    image_restored =
        restore_defocused_image(image_defocused, optimal_sigma)
    cv2.imshow("Original", image_original)
    cv2.imshow("Defocused", image_defocused)
    cv2.imshow("Restored", image_restored)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Додаток Б

```
import numpy as np
import cv2
from skimage.metrics import structural_similarity as ssim
from skimage.restoration import richardson_lucy

def generate_motion_blur_kernel(size, angle):
    kernel = np.zeros((size, size), dtype=np.float32)
    center = size // 2
    kernel[center, :] = 1.0
    kernel = cv2.warpAffine(kernel,
        cv2.getRotationMatrix2D(
            (center, center), angle, 1.0), (size, size))
    kernel /= kernel.sum()
    return kernel

def restore_motion_blurred_image(image_blurred,
    kernel_size, angle, num_iter=50):
    motion_blur_kernel =
        generate_motion_blur_kernel(kernel_size, angle)
    restored_image =
        richardson_lucy(image_blurred, motion_blur_kernel,
            num_iter=num_iter, clip=False)
    restored_image = np.clip(restored_image, 0, 255).astype(np.uint8)
    return restored_image
```

```
def evaluate_mse(image1, image2):
    return np.mean((image1 - image2) ** 2)

def evaluate_ssim(image1, image2):
    return ssim(image1, image2, data_range=image2.max() - image2.min())

def find_optimal_kernel_size_angle(image_blurred,
    image_original, kernel_sizes, angles):
    best_score = None
    best_kernel_size = None
    best_angle = None

    for kernel_size in kernel_sizes:
        for angle in angles:
            restored_image = restore_motion_blurred_image
                (image_blurred, kernel_size, angle)

            score = evaluate_ssim(image_original, restored_image)
            if best_score is None or score > best_score:
                best_score = score
                best_kernel_size = kernel_size
                best_angle = angle

    return best_kernel_size, best_angle, best_score

if __name__ == '__main__':
```

```
image_original = cv2.imread('12.png', 0)
kernel_size = 20
angle = 88
motion_blur_kernel =
    generate_motion_blur_kernel(kernel_size, angle)
image_blurred = cv2.filter2D(image_original,
    -1, motion_blur_kernel)
kernel_sizes = range(17, 23, 2)
angles = np.linspace(85, 90, 6)
optimal_kernel_size, optimal_angle, best_score =
    find_optimal_kernel_size_angle(image_blurred,
        image_original, kernel_sizes, angles)
print("Optimal kernel size:", optimal_kernel_size,
    "Optimal angle:", optimal_angle, "Best score:", best_score)

image_restored = restore_motion_blurred_image
    (image_blurred, optimal_kernel_size, optimal_angle)
cv2.imshow("Original", image_original)
cv2.imshow("Blurred", image_blurred)
cv2.imshow("Restored", image_restored)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Додаток В

```
from __future__ import print_function
import base64
import cv2
import numpy as np

def apply_blur(image, d=31):
    height, width = image.shape[:2]

    # Pad the image
    padded_image = cv2.copyMakeBorder
        (image, d, d, d, d, cv2.BORDER_WRAP)

    # Apply Gaussian blur
    blurred_image = cv2.GaussianBlur(padded_image,
        (2 * d + 1, 2 * d + 1), -1)[d:-d, d:-d]

    # Compute weights based on distance from image boundaries
    y, x = np.indices((height, width))
    distance = np.dstack([x, width - x - 1, y,
        height - y - 1]).min(-1)
    weights = np.minimum(np.float32(distance) / d, 1.0)

    # Blend original image and blurred image using weights
    blended_image = image * weights + blurred_image * (1 - weights)
```

```
    return blended_image


def generate_motion_kernel(angle, length, size=65):
    # Create a kernel of ones with dimensions (1, length)
    kernel = np.ones((1, length), np.float32)

    # Compute the cosine and sine of the angle
    cos_angle, sin_angle = np.cos(angle), np.sin(angle)

    # Create a transformation matrix for rotation
    transformation_matrix = np.float32([[cos_angle, -sin_angle, 0],
        [sin_angle, cos_angle, 0]])

    # Compute the center position of the kernel
    half_size = size // 2
    transformation_matrix[:, 2] =
        (half_size, half_size) - np.dot(transformation_matrix[:, :2],
            ((length - 1) * 0.5, 0))

    # Apply the rotation transformation to the kernel
    kernel = cv2.warpAffine(kernel,
        transformation_matrix, (size, size), flags=cv2.INTER_CUBIC)

    return kernel
```

```
def generate_defocus_kernel(diameter, size=65):
    # Create an empty kernel with dimensions (size, size)
    kernel = np.zeros((size, size), np.uint8)

    # Draw a circle on the kernel
    cv2.circle(kernel, (size, size), diameter,
        255, -1, cv2.LINE_AA, shift=1)

    # Normalize the kernel to values between 0 and 1
    kernel = np.float32(kernel) / 255.0

    return kernel

def restore_image(filename, motion_angle,
    blur_radius, SNR_db, defocus_blur):
    # Load the image
    image = cv2.imread(filename, 0)
    if image is None:
        print('Failed to load image:', filename)
        return None

    # Convert the image to floating point in the range [0, 1]
    image = np.float32(image) / 255.0

    # Display the input image
    cv2.imshow('Input', image)
```

```
cv2.namedWindow('Input')

# Apply blur to the image
blurred_image = apply_blur(image)

# Compute the Fourier transform of the blurred image
transformed_blurred_image =
    cv2.dft(blurred_image, flags=cv2.DFT_COMPLEX_OUTPUT)

# Compute the noise level
noise = 10 ** (-0.1 * SNR_db)

# Generate the appropriate blur kernel
if defocus_blur:
    blur_kernel = generate_defocus_kernel(blur_radius)
else:
    blur_kernel = generate_motion_kernel(motion_angle, blur_radius)

# Normalize the blur kernel
normalized_blur_kernel = blur_kernel / blur_kernel.sum()

# Pad the blur kernel to match the image size
padded_blur_kernel = np.zeros_like(image)
kernel_height, kernel_width = normalized_blur_kernel.shape
padded_blur_kernel[:kernel_height,
    :kernel_width] = normalized_blur_kernel
```

```
# Compute the Fourier transform of the blur kernel
transformed_blur_kernel =
    cv2.dft(padded_blur_kernel,
        flags=cv2.DFT_COMPLEX_OUTPUT, nonzeroRows=kernel_height)

# Compute the squared magnitude of the Fourier
# transform of the blur kernel
squared_magnitude_blur_kernel =
    (transformed_blur_kernel ** 2).sum(-1)

# Compute the inverse Fourier transform of the blur kernel
# with noise regularization
inverse_transform_blur_kernel =
transformed_blur_kernel /
    (squared_magnitude_blur_kernel + noise)[..., np.newaxis]

# Multiply the Fourier transform of the blurred image
# and the inverse Fourier transform of the blur kernel
multiplied_spectra =
    cv2.mulSpectrums(transformed_blurred_image,
        inverse_transform_blur_kernel, 0)

# Compute the inverse Fourier transform of the multiplied spectra
restored_image =
    cv2.idft(multiplied_spectra,
```

```
            flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)

    # Shift the result to the center of the image
    restored_image = np.roll(restored_image, -kernel_height // 2, 0)
    restored_image = np.roll(restored_image, -kernel_width // 2, 1)

    # Convert the restored image to the range [0, 255]
    restored_image = restored_image * 255

    # Encode the restored image as PNG and convert it to base64
    retval, buffer = cv2.imencode('.png', restored_image)
    encoded_restored_image = base64.b64encode(buffer).decode('utf-8')

    # Encode the blur kernel as PNG and convert it to base64
    retval, encoded_blur_kernel =
        cv2.imencode('.png', normalized_blur_kernel * 255.0)
    kernel_base64 =
        base64.b64encode(encoded_blur_kernel).decode('utf-8')

    # Return the base64 encoded restored image and blur kernel
    # return '' + encoded_restored_image + 'kernel
    return '' + encoded_restored_image

if __name__ == '__main__':
    restore_image()
```

Додаток Г

```
import base64

from flask import Flask, render_template, request,
        redirect, url_for, send_file, jsonify
from werkzeug.utils import secure_filename
import plot_restoration

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'}


# Function to check if the file extension is allowed
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in
                app.config['ALLOWED_EXTENSIONS']


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/upload', methods=['POST'])
```

```
def upload():
    # Check if a file is selected in the upload form
    if 'file' not in request.files:
        return redirect(request.url)

    file = request.files['file']
    angle = int(request.form['angle'])
    diam = int(request.form['diam'])
    SNR_db = int(request.form['SNR_db'])
    blur_type = request.form['blur_type']

    filepath_name = None
    output_files = None
    output_files_split = ''
    # Check if the file is allowed
    if file and allowed_file(file.filename):
        # Save the uploaded file
        filename = secure_filename(file.filename)
        filepath_name = app.config['UPLOAD_FOLDER'] + '/' + filename
        file.save(filepath_name)

        # Perform image processing
        # output_filename = image_processing.process_image(filename)
        defocus = blur_type == 'out-of-focus'
        output_files = plot_restoration.restore_image(
                filepath_name, angle, diam, SNR_db, defocus)
```

```
    return jsonify(restored_image_path=output_files)


def getBase64(image_path):
    with open(image_path, 'rb') as binary_file:
        binary_file_data = binary_file.read()
        base64_encoded_data = base64.b64encode(binary_file_data)
        base64_message = '' +
            base64_encoded_data.decode('utf-8')

    return base64_message


def getStrFromBase64(b64):
    base64_message = '' + b64.decode('utf-8')

    return base64_message


if __name__ == '__main__':
    app.run(debug=True)
```

Додаток Д

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jque
    <script>
        $(document).ready(function() {
            const input = document.getElementById('fileInput');
            const preview = document.getElementById('image-preview');
            //const preview = document.getElementById('image-kernel');

            // Listen for file selection
            input.addEventListener('change', function() {
              // Make sure a file was selected
              if (input.files && input.files[0]) {
                // Create a FileReader object
                const reader = new FileReader();

                // Set up the reader to display the image as a data URL
                reader.readAsDataURL(input.files[0]);

                // When the reader is done, set the preview image source
                reader.onload = function() {
                  preview.src = reader.result;
                }
              }
```

```
});

// Function to update the restored image and kernel image
function updateImages() {
    var kernelType = $("#kernelType").val();
    var kernelSize = $("#kernelSize").val();
    var numIterations = $("#numIterations").val();
    var imagePath = "{{ original_image_path }}";
    var fileInput = document.getElementById('fileInput');
    var snrDbInput = $('#SNR_db');
    var diamInput = $('#diam');
    var angleInput = $('#angle');
    var blur_typeInput = $('#blur_type');

    // Create a new FormData object
    var formData = new FormData($("#my-form")[0]);

    // Add the file to the FormData object
    formData.append('file', fileInput.files[0]);
    formData.append('SNR_db', snrDbInput.val());
    formData.append('diam', diamInput.val());
    formData.append('angle', angleInput.val());
    formData.append('blur_type', blur_typeInput.val());

    // AJAX request to update the images
    $.ajax({
```

```
            type: "POST",
            url: "/upload",
            data: formData,
            processData: false,
            contentType: false,
            success: function(response) {
                // Update the restored image source
                $("#restoredImage").attr("src",
                    response.restored_image_path);
                //$("#kernelImage").attr("src",
                    response.kernel_image_path);
                $("#restoredImageDownload").attr("href",
                    response.restored_image_path);
            },
            error: function(xhr, status, error) {
                console.error("Error updating images:", error);
            }
        });
    }
    // Event listener for parameter change
    $(".parameter").change(function() {
        updateImages();
    });
    $(".btn-submit").click(function() {
        updateImages();
    });
```

```
            // Initial image update
            updateImages();
        function readURL(input) {
  if (input.files && input.files[0]) {

    var reader = new FileReader();

    reader.onload = function(e) {
      $('.image-upload-wrap').hide();

      $('.file-upload-image').attr('src', e.target.result);
      $('.file-upload-content').show();

      $('.image-title').html(input.files[0].name);
    };

    reader.readAsDataURL(input.files[0]);

  } else {
    removeUpload();
  }
}

function removeUpload() {
  $('.file-upload-input').replaceWith($('.file-upload-input').clone());
```

```
  $('.file-upload-content').hide();
  $('.image-upload-wrap').show();
}
$('.image-upload-wrap').bind('dragover', function () {
    $('.image-upload-wrap').addClass('image-dropping');
  });
  $('.image-upload-wrap').bind('dragleave', function () {
    $('.image-upload-wrap').removeClass('image-dropping');
});


            $(".slider").on("input", function() {
    var id = $(this).attr("id");
    var val = $(this).val();
    $("#" + id + "-current").text(val);
});
        });


    </script>
    <title>Deblur Image</title>
    <link rel="stylesheet" type="text/css"
        href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
<h1 class="title">Deblur Image</h1>
<div class="container">
    <form action="http://127.0.0.1:5000/upload" method="post"
```

```
    id="my-form" enctype="multipart/form-data">
<div class="file-upload">
    <button class="file-upload-btn" type="button"
        onclick="$('.file-upload-input').trigger( 'click' )">
        Add Image </button>

    <div class="image-upload-wrap form-group">
        <input type="file" name="file" id="fileInput"
            accept="image/*" class="file-upload-input" required>
        <div class="drag-text">
            <h3>Drag and drop a file or select add Image</h3>
        </div>
    </div>
    <div class="file-upload-content">
        <img class="file-upload-image" src="#"
            alt="your image"/>
        <div class="image-title-wrap">
            <button type="button" onclick="removeUpload()"
                class="remove-image">Remove <span
                class="image-title">
                    Uploaded Image</span></button>
        </div>
    </div>
</div>

<div style="margin-bottom: 20px;">
```

```
    <label for="blur_type">Blurring Type:</label>
    <select id="blur_type" name="blur_type"
        class="parameter my-select-menu">
        <option value="motion">Motion Blur</option>
        <option value="out-of-focus">Defocus Blur</option>
    </select>
</div>

<div class="form-group slidecontainer">
    <label for="angle">Angle:</label>
    <input type="range" min="0" max="180" value="135"
        class="slider parameter" id="angle" name="angle">
    <span class="slider-values"><span
        id="angle-min">0</span> | <span
            id="angle-current">135</span> | <span
            id="angle-max">180</span></span>
</div>
<div class="form-group slidecontainer">
    <label for="diam">D:</label>
    <input type="range" min="0" max="50" value="22"
        class="slider parameter" id="diam" name="diam">
    <span class="slider-values"><span
            id="diam-min">0</span> | <span
                id="diam-current">22</span> | <span
            id="diam-max">50</span></span>
</div>
```

```
    <div class="form-group slidecontainer">
        <label for="SNR_db">SNR (db):</label>
        <input type="range" min="0" max="50" value="25"
                class="slider parameter" id="SNR_db" name="SNR_db">
        <span class="slider-values"><span
                id="SNR_db-min">0</span> | <span
                    id="SNR_db-current">25</span> | <span
                id="SNR_db-max">50</span></span>
    </div>
    <div>
        <button type="button" class="btn-submit">Deblur</button>
    </div>
</form>

<div>
    <div class="image-container">
        <h2 style="margin-top: 0">Restored Image</h2>
        <img src="{{ restored_image_path }}" id="restoredImage"
                alt="Restored Image" style="height: 35vh">
        <a href="{{ restored_image_path }}"
                download="restored_image.jpg"
                style="text-decoration: none"
           id="restoredImageDownload">
            <button type="button" class="btn-submit"
                    >Download Restored Image</button>
        </a>
```

```
        </div>
        <div class="image-container">
            <h2>Original Image</h2>
            <img src="{{ original_image_path }}"
                    alt="Original Image" id="image-preview"
                    style="height: 35vh">
        </div>
    </div>
</div>
</body>
</html>
```

Додаток Ж

```
body {
    background-color: #f5f5f5;
    font-family: Arial, sans-serif;
}

form {
    max-width: 500px;
    margin-right : 40px;
}

.container {
    display: flex;
    margin: 0 auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.title {
    text-align: center;
    color: #333;
}

.form-group {
```

```
    margin-bottom: 20px;
}

label {
    display: block;
    font-weight: bold;
    margin-bottom: 5px;
    color: #555;
}

input[type="file"],
input[type="number"] {
    width: 100%;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

.btn-submit {
    display: block;
    width: 100%;
    padding: 10px;
    background-color: #4CAF50;
    color: #fff;
    border: none;
```

```
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}

.btn-submit:hover {
    background-color: #45a049;
}
.slidecontainer {
    width: 100%; /* Width of the outside container */
}

.slider {
    -webkit-appearance: none;  /* Override default CSS styles */
    appearance: none;
    width: 100%; /* Full-width */
    height: 25px; /* Specified height */
    background: #d3d3d3; /* Grey background */
    outline: none; /* Remove outline */
    opacity: 0.7; /* Set transparency (for mouse-over effects on hover)
    -webkit-transition: .2s; /* 0.2 seconds transition on hover */
    transition: opacity .2s;
}

.slider:hover {
    opacity: 1; /* Fully shown on mouse-over */
```

```
}

.slider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 25px;
    height: 25px;
    background: #4CAF50;
    cursor: pointer;
}

.slider::-moz-range-thumb {
    width: 25px;
    height: 25px;
    background: #4CAF50;
    cursor: pointer;
}


.file-upload {
  background-color: #ffffff;
  width: 500px;
  margin: 0 auto;
}

.file-upload-btn {
```

```
  width: 100%;
  margin: 0;
  color: #fff;
  background: #1FB264;
  border: none;
  padding: 10px;
  border-radius: 4px;
  border-bottom: 4px solid #15824B;
  transition: all .2s ease;
  outline: none;
  text-transform: uppercase;
  font-weight: 700;
}

.file-upload-btn:hover {
  background: #1AA059;
  color: #ffffff;
  transition: all .2s ease;
  cursor: pointer;
}

.file-upload-btn:active {
  border: 0;
  transition: all .2s ease;
}
```

```
.file-upload-content {
  display: none;
  text-align: center;
}

.file-upload-input {
  position: absolute;
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
  outline: none;
  opacity: 0;
  cursor: pointer;
}

.image-upload-wrap {
  margin-top: 20px;
  border: 4px dashed #1FB264;
  position: relative;
}

.image-dropping,
.image-upload-wrap:hover {
  background-color: #1FB264;
  border: 4px dashed #ffffff;
```

```
}

.image-title-wrap {
  padding: 0 15px 15px 15px;
  color: #222;
}

.drag-text {
  text-align: center;
}

.drag-text h3 {
  font-weight: 100;
  text-transform: uppercase;
  color: #15824B;
  padding: 60px 0;
}

.file-upload-image {
  max-height: 200px;
  max-width: 200px;
  margin: auto;
  padding: 20px;
}

.remove-image {
```

```
  width: 200px;
  margin: 0;
  color: #fff;
  background: #cd4535;
  border: none;
  padding: 10px;
  border-radius: 4px;
  border-bottom: 4px solid #b02818;
  transition: all .2s ease;
  outline: none;
  text-transform: uppercase;
  font-weight: 700;
}

.remove-image:hover {
  background: #c13b2a;
  color: #ffffff;
  transition: all .2s ease;
  cursor: pointer;
}

.remove-image:active {
  border: 0;
  transition: all .2s ease;
}
```

```
.my-select-menu {
    color: white;
    background-color: #1FB264;
    padding: 5px;
    border-radius: 5px;
}

option{
    padding: 10px;
    margin-top: 5px;
    border: 1px solid #1FB264;
    border-radius: 5px;
}
```