Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів обробки зображень
Методи реалістичного заміщення об'єктів
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-5

Білий М.Д.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Каук В.І.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Білому Михайлу Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів обробки зображень. Методи реалістичного заміщення об'єктів»
Затверджена наказом по університету від 29.03.2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 21.06.2024
3. Вихідні дані до роботи дослідження алгоритмів заміщення та домальовування зображень такі як PatchMatch, використання планарної структури, генеративні змагальні мережі, нейроні мережі з контекстуальною увагою, Gated згортка та Stable Diffusion у контексті використання реалістичного заміщення об'єктів на зображенні, програмна реалізація, пояснювальна записка
4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі, постановка задачі, дослідження класичних методів та нейронних мереж, постановка задачі, проектування програмної системи, універсальна програмна реалізація

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір методів для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.05 – 01.06.24	*виконано*
10	Підготовка презентації та доповіді	26.04 – 2.05.24	*виконано*
11	Нормоконтроль	01.06 – 11.06.24	*виконано*
12	Рецензування	09.06 – 16.06.24	*виконано*
13	Занесення диплома в електронний архів	15.05.2024	*виконано*
14	Попередній захист	18.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	19.06.2024	*виконано*

Дата видачі завдання 29 березня 2024р.

Студент (ка) _____ Білий М.Д.
(підпис)

Керівник роботи _____ доц. Каук В.І.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 78 с., 26 рис., 24 табл., 7 додатків, 8 джерел.

ГЕНЕРАТИВНІ МЕРЕЖІ, ЗАМІЩЕННЯ ОБ’ЄКТІВ, ЗГОРТКОВІ НЕЙРОНІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ШТУЧНИЙ ІНТЕЛЕКТ, INPAINTING, CONTEXTUAL ATTENTION, GATED CONVOLUTION, DEEPFILL V2, STABLE DIFFUSION, LATENT DIFUSION MODEL.

Об'єкт дослідження є сучасні методи заміщення об'єктів на зображеннях за допомогою алгоритмів штучного інтелекту, зосереджуючись на гейтових згортках та контекстуальній увазі у нейронних мережах.

Мета дослідження – це аналіз методів для ефективного заміщення об'єктів в зображеннях, що вимагає розуміння та адаптації до контексту за допомогою машинного навчання, із застосуванням згорткових нейронних мереж та генеративних моделей.

В результати дослідження, встановленно високої ефективності представлених методів для заміщення об'єктів на зображеннях, здатності до врахування складних контекстуальних відносин та забезпечення високоякісного візуального результату, що відрізняється природністю інтеграції у вихідне зображення.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, OBJECT SUBSTITUTION, NEURAL NETWORKS, CONVOLUTION NEURAL NETWORKS, GENERATIVE NETWORKS, INPAINTING, CONTEXTUAL ATTENTION, GATED CONVOLUTION, DEEPFILL V2, STABLE DIFFUSION, LATENT DIFUSION MODEL

The object of research is modern methods of replacing objects in images using artificial intelligence algorithms, focusing on gate convolutions and contextual attention in neural networks.

The purpose of the research is to analyze methods for effective replacement of objects in images, which requires understanding and adapting to the context with the help of machine learning, using convolutional neural networks and generative models.

In the results of the research, the high efficiency of the presented methods for replacing objects in images, the ability to take into account complex contextual relations and the provision of a high-quality visual result, distinguished by the naturalness of integration into the original image, were established.

Я, Білий Михайло Дмитрович, студент гр.ІПЗм-22-5, здобувач вищої освіти на другому (магістерському) рівні, кафедра програмної інженерії, заявляю: представлена моя робота, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Цифрова ера народжує незліченні можливості для творчості та інновацій, зокрема в галузі обробки зображень. Впровадження штучного інтелекту та машинного навчання відкрило новий вимір у цьому напрямку, піднімаючи можливості традиційної обробки зображень на небачений досі рівень.

Реалістичне заміщення об'єктів у зображеннях є важливим завданням, що знайшло своє застосування у різноманітних сферах - від реставрації зображень до розваг. Такі технології відіграють ключову роль у формуванні майбутнього інформаційних систем та мультимедіа.

Ця робота має на меті детально вивчити і оцінити алгоритми заміщення об'єктів, зокрема, розглянути підходи на основі нейронних мереж, що дозволяють досягати вражаючої реалістичності та точності в обробці зображень.

У подальших розділах буде представлено аналіз існуючих методів, огляд найсучасніших досліджень у цій області, а також оцінка потенціалу запропонованих рішень для практичного застосування.

Також буде реалізована програмна система, яка дає можливість використовувати різні методи для реалістичного заміщення об’єктів.

1 ОГЛЯД НАУКОВОЇ ТА ПАТЕНТНОЇ ЛІТЕРАТУРИ

1.1 Аналіз предметної галузі

Розвиток обробки зображень являє собою вражаючу історію інновацій та вдосконалень. Початкові етапи цієї галузі зосереджувалися на простих методах, таких як фільтрація та контрастне вдосконалення. Згодом, з появою комп'ютерної графіки, з'явилися складніші техніки, такі як морфологічні перетворення та сегментація.

Вступ цифрових технологій ввів концепції згорткових нейронних мереж (CNN), які значно покращили якість обробки зображень. Наступним великим кроком стало використання генеративних змагальних мереж (GANs), що дозволило створювати реалістичні та високоякісні зображення.

Такі інновації, як стробовані згортки та контекстуальна увага, сьогодні є вершиною досягнень у цій галузі, відкриваючи нові можливості для редагування, відновлення та оптимізації зображень.

У наступних розділах розглянемо кожен метод детальніше та їх переваги та недоліки.

1.1.1 PatchMatch

PatchMatch - це передовий алгоритм, що ефективно виявляє відповідності між маленькими квадратними сегментами (патчами) в зображеннях. Широко застосовуваний у різних сферах, таких як видалення об'єктів зі знімків, редагування та переміщення елементів у межах зображень, зміна розмірів та пропорцій, а також у створенні аналізів оптичного потоку та стерео-зображень.

Основна ціль алгоритму полягає у створенні поля найближчих сусідів (NNF) - карти зміщень, де кожна точка одного зображення зіставлена із зсунутою точкою на іншому зображенні. Індивідуальний пошук відповідностей для окремих сегментів є простим, але задача ускладнюється, коли необхідно обробити ціле зображення. Щоб оптимізувати цей процес, PatchMatch використовує рандомізований підхід, значно прискорюючи обчислення.

Алгоритм PatchMatch включає в себе три основні етапи:

– ініціалізація NNF: Поле найближчих сусідів спочатку заповнюється випадковими зміщеннями або за допомогою попередньо зібраної інформації;
– ітеративне оновлення: Відбувається процес ітеративного оновлення NNF, де ефективні зміщення патчів розповсюджуються на сусідні пікселі, покращуючи точність відповідностей;
– випадковий пошук: Проводиться випадковий пошук навколо найкращих зміщень, щоб знайти ще кращі варіанти і доповнити точність відповідностей.

У якості критерію відповідності D двох патчей f та g можна використати різні функції. Найбільш вживаними є:

– сума квадратів різниць (Sum of Squared Differences, SSD) (формула 1.1)

$$D = \sum_{i,j \epsilon R} (f(i,j) - g(i,j))^2 \tag{1.1}$$

– сума модулей різниць (Sum of Absolute Differences, SAD) (формула 1.2)

$$D = \sum_{i,j \epsilon R} |f(i,j) - g(i,j)| \tag{1.2}$$

– сума квадратів різниць або сума модулей різниць із нульовим середнім (ZSSD, ZSAD) – для кожного патча визначається та видаляється середнє значення (формула 1.3)

$$D = \sum_{i,j \epsilon R} ((f(i,j) - \hat{f}) - (g(i,j) - \hat{g}))^2 \tag{1.3}$$

– нормалізована взаємна кореляція (Normalized Cross Correlation, NCC) (формула 1.4)

$$D = \frac{\sum_{i,j \epsilon R} f(i,j) * g(i,j)|}{\hat{f} * \hat{g}} \quad (1.4)$$

Під час початкового етапу алгоритму PatchMatch, ініціалізація здійснюється за допомогою випадкових зсувів, розподілених рівномірно по всій площі зображення B. Цей метод дозволяє уникнути необхідності використання первинних припущень про відповідності між зображеннями та зменшує ризик застрягання в локальних мінімумах в процесі пошуку оптимальних рішень.

На наступному етапі, алгоритм переходить до ітеративного удосконалення поля найближчих сусідів. Кожен цикл ітерацій обробляє пікселі зображення в строго визначеному порядку - від лівого верхнього кута до правого нижнього. Цей процес включає два ключові моменти: фазу поширення, де ефективні зсуви розповсюджуються між сусідніми пікселями, та фазу випадкового пошуку для знаходження ще більш точних відповідностей.

На цьому етапі метою є удосконалення значення функції $f(x,y)$ в конкретній точці (x,y), використовуючи вже відомі значення сусідніх точок $f(x-1,y)$ і $f(x,y-1)$. Припускаємо, що відомі сусідні значення схожі між собою. Таким чином, алгоритм вибирає для $f(x,y)$ нове значення, що мінімізує розбіжності $D(f(x,y))$ порівняно з $D(f(x-1,y))$ та $D(f(x,y-1))$. Якщо дане значення $f(x,y)$ є коректним і знаходиться в координованій області R, то всі пікселі, розташовані нижче та праворуч від $f(x,y)$, будуть також заповнені відповідними правильними значеннями звуку. Додатково, на парних ітераціях процес розширення інвертується, і система шукає нові значення, що мінімізують $D(f(x,y))$ серед $D(f(x+1,y))$, $D(f(x,y+1))$.

У фазі випадкового пошуку, позначимо $u_0 = f(x,y)$ як поточне значення сзуву в даній точці. Наша мета - поліпшити значення $f(x,y)$, для чого ми перевіряємо кілька потенційних кандидатів, зменшуючи відстань до u_0 за допомогою експоненціально зменшуваного радіуса пошуку (формула 1.5):

$$u_i = u_0 + w\alpha^i R_i \quad (1.5)$$

де R_i – це випадкове число, однаково розподілене у діапазоні від -1 до 1 по обом осям,

w – це початковий радіус вікна пошуку, який відповідає розміру зображення (і може бути обмежений, якщо є додаткова інформація, що визначає межі пошуку);

α – це коефіцієнт зменшення радіуса вікна пошуку з кожною ітерацією, який зазвичай встановлюється як ½.

Ця частина алгоритму допомагає забезпечити, що $f(x,y)$ виходить із локального мінімуму в процесі оптимізації.

Зазвичай, критерій зупинки алгоритму визначається лімітом на кількість ітерацій, який часто встановлюють на рівні приблизно чотирьох до п'яти. Навіть з таким обмеженим числом ітерацій, алгоритм продовжує показувати задовільні результати. Можливо також встановити граничне значення для помилки, досягнення якого також стане сигналом до завершення роботи алгоритму. Приклад роботи алгоритму можна побачити на рисунку 1.1.

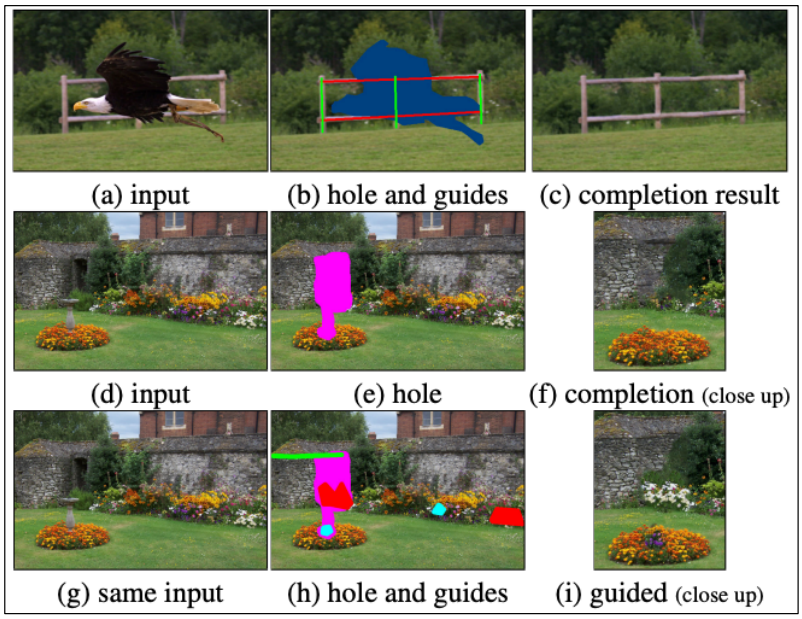

Рисунок 1.1 – Приклад завершення зображення за допомогою PatchMatch[2]

На цьому прикладі птах є видалено з зображення (a). Користувач позначає область завершення і позначає обмеження на пошук у (b), створюючи результат

(c) у кілька секунд. Квіти видаляються з входу (d) за допомогою наданої користувачем маски (e), що призводить до результату (f). Починаючи з того самого введення (g), користувач позначає обмеження на квітах і лінії даху (h), створюючи результат (i) зі зміненим кольором квітки та лінією даху.

1.1.2 Завершення зображення за допомогою навігації планарної структури

Виявлення плоских поверхонь і регулярності у зображеннях важливе для завдань з завершення зображень. Це допомагає зрозуміти геометричні та структурні аспекти сцени, що є критично важливим для створення реалістичних і послідовних завершень зображень. Ідентифікуючи плоскі поверхні, алгоритм може краще розуміти перспективу та вирівнювання різних елементів зображення. Розпізнавання регулярних візерунків допомагає точно відтворювати ці візерунки в завершеному зображенні, підвищуючи загальну якість і реалізм реставрації чи завершення зображення.

Було запропоновано багато методів для ідентифікації та виправлення площин [3], тобто перетворення перспективно викривленої площини у фронтопаралельну версію. У цьому методі використовуємо техніку, яка передбачає виділення відрізка лінії, оцінку точки сходу і групування на основі точок сходу. Спочатку виявляємо краї та підганяємо сегменти ліній у відомій області зображення. Потім виявляємо до трьох точок зникнення (VP – vanishing point) за допомогою підходу голосування на основі RANSAC. Це означає, що ми припускаємо, що в сцені існує лише до трьох різних орієнтацій площини (див. рисунок 1.2).

За наявності трьох VP ми можемо відновити до трьох площинних орієнтацій, по одному з кожної пари виявлених VP. Компактно представимо параметри площини m за допомогою лінії, що зникає, l_∞^m (зображення нескінченної прямої на світовій площині, що з’єднує дві різні VP).

Далі визначаємо опору кожної площини шляхом локалізації позиції, де два набори відрізків, що відповідають дві VP перекривають один одного.

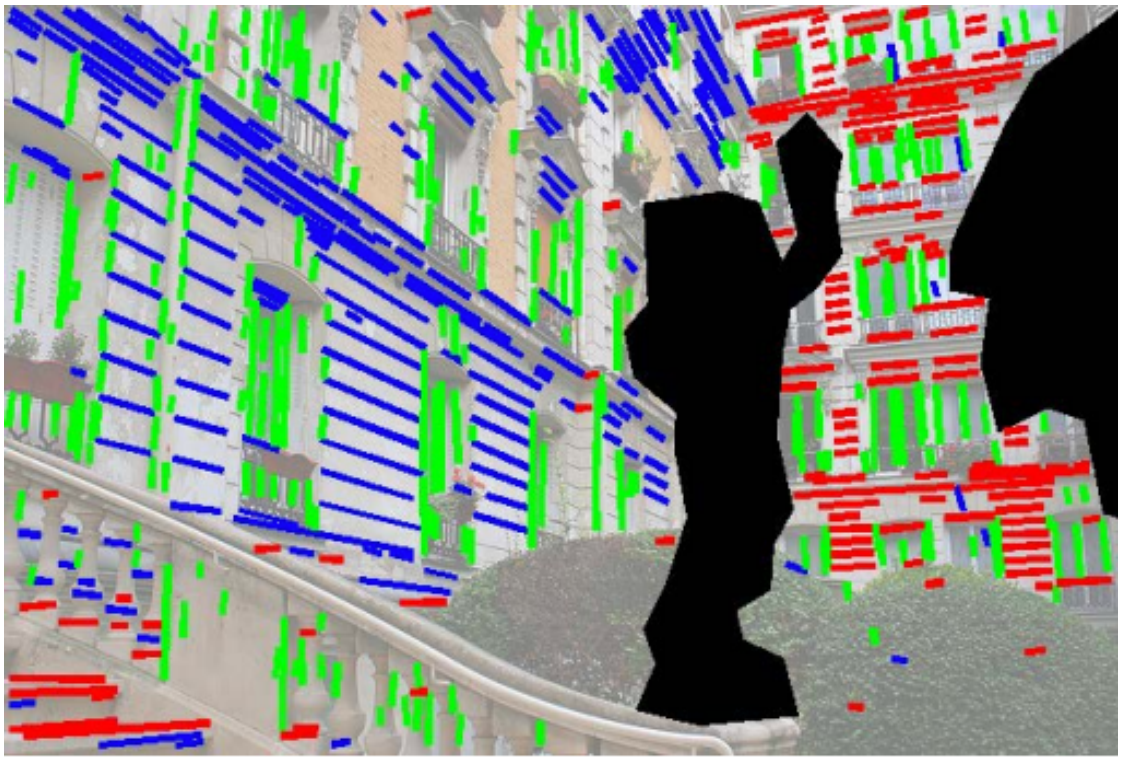

Рисунок 1.2 – Виявлення точки зникнення в техногенному середовищі[3]

Червоний, зелений і синій сегменти лінії відповідають трьом виявленим точкам зникнення відповідно.

Спочатку ми оцінюємо просторову опору кожної VP, розповсюджуючи її відповідні відрізки за допомогою широкого ядра Гауса. Тоді ми оцінити просторову опору для площин, виконавши поелементне множення мап щільності опорних ліній її VP. Ці карти продукту мають високий відгук, де два набори лінії сегменти, накладені один на одного. Зауважте, що ми завжди додаємо фронтально-паралельну площину з параметрами $l_{\infty}^{0} = [0,0,1]^{T}$ і призначити фіксоване значення щільності 10-5 рівномірно по всьому зображенню. Ми тоді виконати попіксельну нормалізацію цієї карти добутку щільності так, щоб сума ймовірності приналежності до площини дорівнює 1; ми називаємо це «апостеріорна ймовірність» $Pr[m|x]$ для призначення площині m у пікселі x. Цей процес проілюстровано на рисунку 1.3. Тут задня частина розподіли ймовірностей показані як кольорові карти щільності правий стовпець.

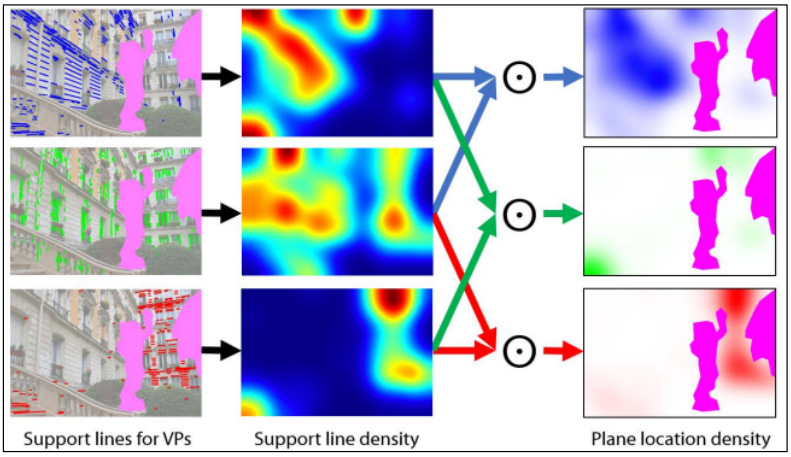

Рисунок 1.3 – Локалізація площини у відомій області[3]

Оскільки лінії можна виявити лише у відомій області зображення, апостеріорні ймовірності в межах невідомої області є високими ненадійний. Щоб вирішити цю проблему, ми призначаємо кожному відсутньому пікселю ймовірності найближчого граничного пікселя. Апостеріорна ймовірнісна карта прикладу зображення показана на рисунку 1.4.

Рисунок 1.4 – Апостеріорна ймовірнісна карта[3]

Повторювані структури в штучних середовищах зазвичай рівновіддалені в 3D. Однак рівний інтервал не зберігається в просторі зображення (і, отже, у наших збігах функцій) через спотворення перспективи. Ми скасовуємо це спотворення, афінно виправляючи позиції узгоджених характерних точок. Зміщення двох випрямлених характерних точок тепер є просторово інваріантним, і, отже, ми можемо виявити трансляційне повторення: якщо існують певні

регулярні структури, ці вектори зміщення утворюють щільний кластер у 2D афінному випрямленому просторі. Використовуємо алгоритм середнього зсуву для виявлення цих режимів (встановлюючи параметр пропускної здатності на 10 пікселів і відхиляючи помилкові режими з менш ніж 10 пікселями). Множину мод позначимо як $D_m = \{d_i\}$, де $d_i \in R^2$ – вектор переміщень у випрямленому просторі.

На рисунку 1.5 показано виявлені в режимі як у випрямленому (ліворуч), так і в простір, вирівняний по осі (праворуч). У (e-f) ми також показуємо їхні позиції відносно цільової ділянки на зображенні (білий квадрат). Ця фігура підкреслює важливість перспективної корекції при обчисленні векторів зміщення. На додаток до пропозиції про точне положення, параметри площини чітко визначають, як вихідні плями повинні просторово деформуватися.

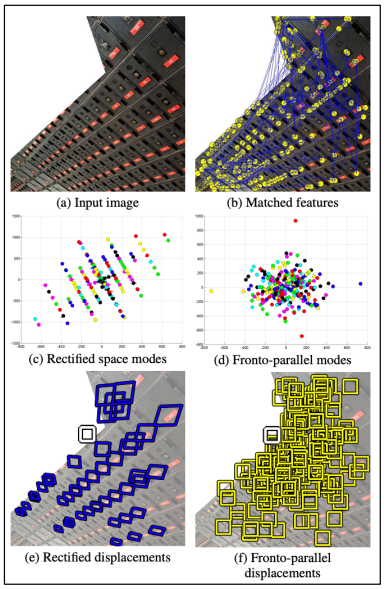

Рисунок 1.5 – Виявлення закономірності за модами векторів зсуву між узгодженими ознаками[3]

Завданням цільової функції є вдосконалення процесу доповнення зображень, що досягається двома шляхами. По-перше, відстань між фрагментами підсилюється за допомогою орієнтовного елемента. По-друге, ми розширюємо простір пошуку завдяки індексу рівнини, який керує перетвореннями фрагментів. Формула цільової функції виглядає так (формула 1.6):

$$\min_{\{t_i,s_i,m_i\}} \sum_{i\in\overline{\Omega}} E_{color}(s_i,t_i,m_i) + E_{guide}(s_i,t_i,m_i) \quad (1.6)$$

де Ω та $\overline{\Omega}$ позначають множини відомих і невідомих індексів пікселів відповідно,

$t_i = (t_i^x, t_i^y)^T$ обозначає центральну позицію цільового фрагменту в $\overline{\Omega}$, $s_i = (s_i^x, s_i^y)^T$ визначає центральну позицію відповідного джерельного фрагменту в Ω,;

m_i – індекс рівнини невідомого цільового фрагменту t_i;

E_{color} та E_{guide} – це терміни, що представляють візуальні та направляючі аспект, які в сукупності формують відстань між фрагментами.

А тепер порівняємо результати роботи (рисунок 1.6) з іншими алгоритмами: Photoshop, Image Melding, He and Sun.

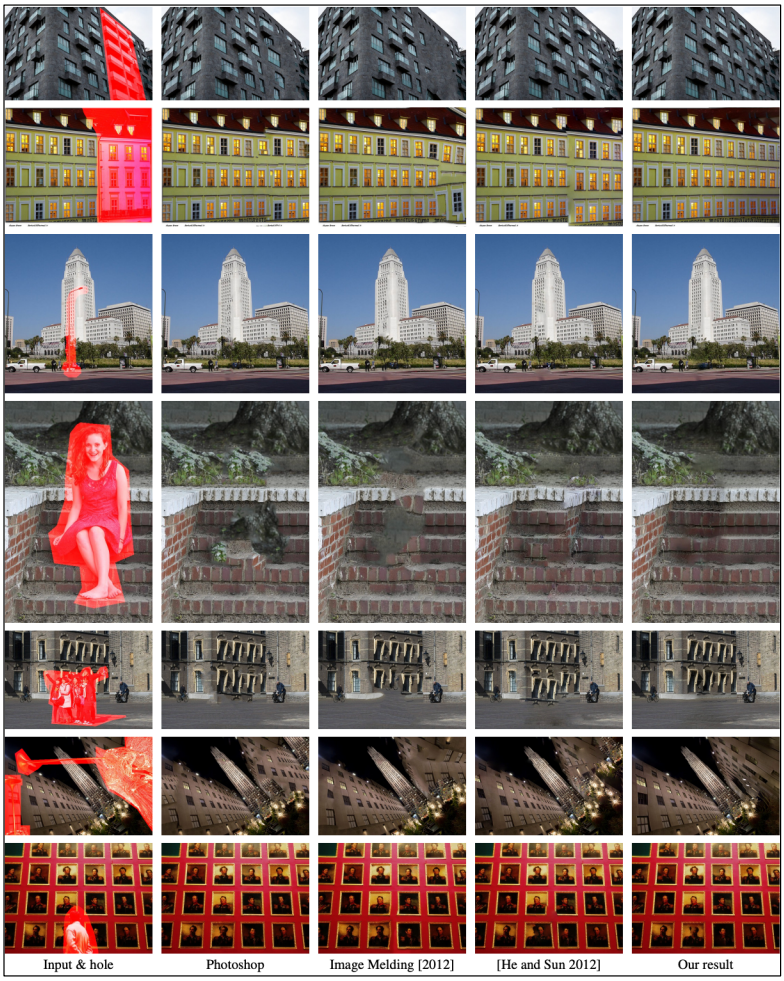

Рисунок 1.6 – Порівняння Photoshop, Image Melding, He and Sun[3]

1.1.3 Генеративні змагальні мережі

Генеративні змагальні мережі, відомі як GANs, представляють собою категорію алгоритмів у галузі AI, розроблену для використання у машинному навчанні без вчителя. Система базується на двох нейронних мережах, що взаємодіють як опоненти в ігровій теоретичній моделі з нульовим виграшем. Запропоновані Яном Ґудфелоу у 2014 році, ці мережі здатні створювати візуально переконливі зображення, які можуть вважатися реалістичними на перший погляд, хоча люди часто можуть розрізнити вироблені зображення від справжніх.

У генеративних змагальних мережах, "генератор" створює пробні дані, а "дискримінатор" оцінює їх, намагаючись відрізнити справжні дані від синтетично створених. Генератор навчається відтворювати дані з латентного простору, в той час як дискримінатор розрізняє від фальшивих. Взаємне навчання мереж веде до покращення їх здатності: генератор намагається обманути дискримінатор, а дискримінатор покращує свою здатність розпізнавати справжність. Цей процес поступово веде до створення все більш точних імітацій даних.

Генеративні змагальні мережі знаходять застосування в генерації фотореалістичних зображень для демонстрації нових концепцій в області інтер'єрного та промислового дизайну, створенні віртуальних аксесуарів та одягу, а також елементів декору в відеоіграх. Вони також застосовуються в аналізі та поліпшенні відео, реконструкції 3D-моделей та в астрономічних дослідженнях для удосконалення якості зображень.

Генеративні змагальні мережі (GAN) ефективно використовуються в техніці "image inpainting", яка полягає в відновленні пошкоджених або відсутніх частин зображень. Вони навчаються відтворювати реалістичні текстури та деталі, заміщаючи пропущені сегменти так, що відновлене зображення здається цілісним і неперервним. Це досягається шляхом "навчання" однієї частини мережі (генератора) виробляти зображення, в той час як інша частина (дискримінатор) оцінює їх якість, покращуючи з часом результат. Архітектуру мережі можна побачити на рисунку 1.7.

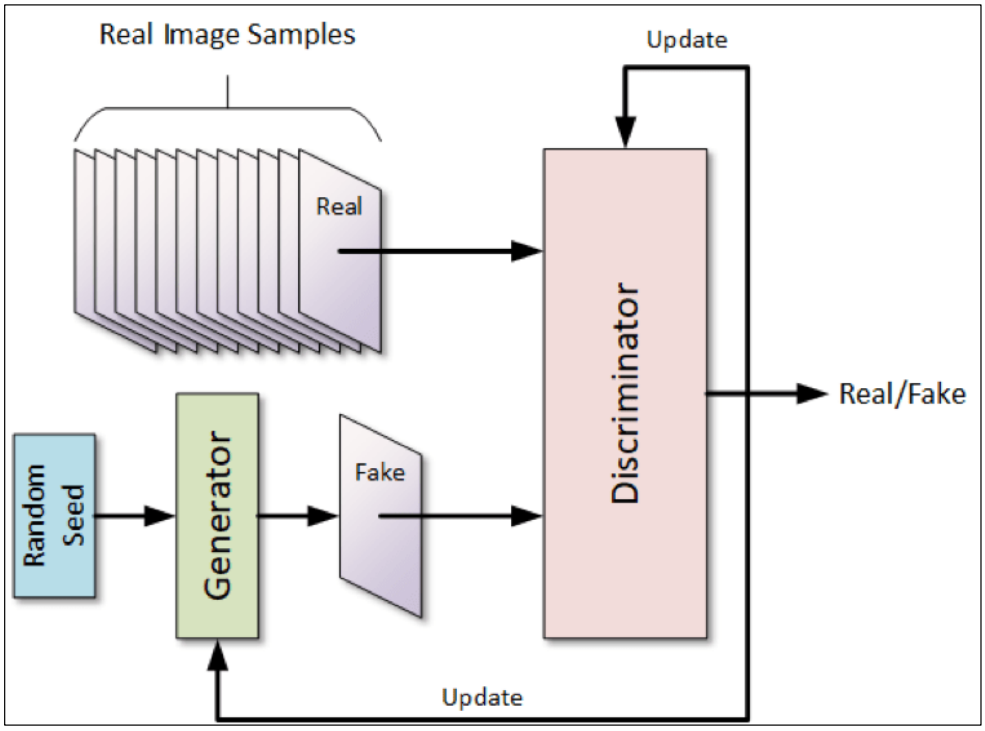

Рисунок 1.7 – Архітектура GAN

1.1.4 Генеративне відновлення зображень із контекстуальною увагою

Згорткові нейронні мережі обробляють особливості зображення з локальним згортковим ядром шар за шаром, отже, не ефективні для запозичення ознак із віддалених просторових місць. Щоб подолати обмеження, в цьому методі ми розглядаємо механізм уваги та вводимо новий контекстний рівень уваги в глибокій генеративній мережі.

Рівень контекстної уваги дізнається, де запозичити або скопіювати інформацію про функції з відомих фонових латок, щоб створити відсутні латки. Він диференційований, тому його можна навчити в глибоких моделях, і повністю згортковий, що дозволяє тестувати на довільних роздільностях.

Розглядаємо проблему, у якій хочемо зіставити характеристики відсутніх пікселів (передній план) з оточенням (фон). Як показано на рисунку 1.8, спочатку витягуємо патчі (3 × 3) у фоновому режимі та змінюємо їх як згорткові фільтри. Щоб зіставити патчі переднього плану $\{f_{x,y}\}$ із фоновими $\{b_{\acute{x},\acute{y}}\}$, ми вимірюємо нормалізований внутрішній добуток (косинус подібності).

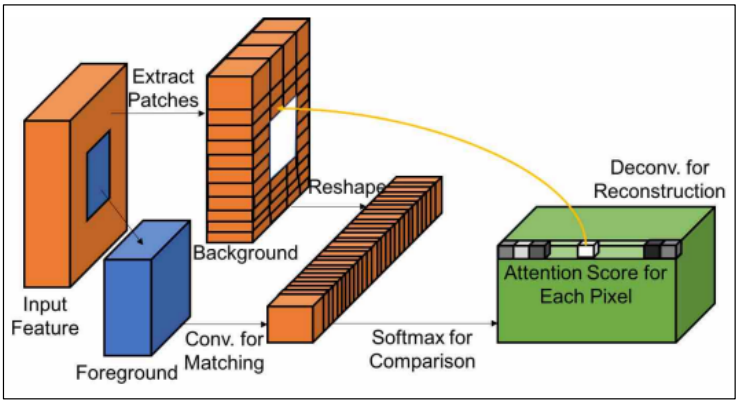

Рисунок 1.8 – Ілюстрація рівня контекстної уваги[4]

По-перше, використовуємо згортку для обчислення оцінки збігу патчів переднього плану з патчами фону (як згорткові фільтри). Потім застосовуємо softmax, щоб порівняти та отримати оцінку уваги для кожного пікселя. Нарешті, реконструюємо патчі переднього плану з патчами фону, виконуючи деконволюцію оцінки уваги. Рівень контекстуальної уваги диференційований і повністю згорнутий (формула 1.7).

$$s_{x,y,\acute{x},\acute{y}} = \langle \frac{f_{x,y}}{\|f_{x,y}\|}, \frac{b_{\acute{x},\acute{y}}}{\|b_{\acute{x},\acute{y}}\|} \rangle, \tag{1.7}$$

де $s_{x,y,\acute{x},\acute{y}}$ представляє подібність фрагмента з центром на фоні $(\acute{x}, \acute{y})$ і передньому плані (x, y).

Потім зважуємо подібність із масштабованим softmax уздовж $\acute{x}, \acute{y}$ - розміру, щоб отримати оцінку уваги для кожного пікселя $s^*_{x,y,\acute{x},\acute{y}} = softmax_{\acute{x},\acute{y}}(\lambda s_{x,y,\acute{x},\acute{y}})$, де λ є постійним значенням. Це ефективно реалізовано як згортка та softmax по каналу. Нарешті, повторно використовуємо витягнуті патчі $\{b_{\acute{x},\acute{y}}\}$ як деконволюційні фільтри для реконструкції переднього плану. Значення пікселів, що перекриваються, усереднюються.

Для поширення уваги заохочуємо когерентність уваги шляхом злиття. Ідея когерентності полягає в тому, що зсув у ділянці переднього плану, швидше за все,

відповідає рівному зсуву у ділянці фону для уваги. Наприклад, $s^*_{x,y,\acute{x},\acute{y}}$ зазвичай мають близьке значення до $s^*_{x+1,y,\acute{x}+1,\acute{y}}$. Щоб змоделювати та заохочувати когерентність карт уваги, робимо поширення ліворуч-праворуч, а потім – зверху вниз із розміром ядра k. Візьмемо як приклад розповсюдження зліва направо, отримаємо новий показник уваги за допомогою (формула 1.8):

$$\hat{s}_{x,y,\acute{x},\acute{y}} = \sum_{i\in\{-k,\dots,k\}} s^*_{x+i,y,\acute{x}+\iota,\acute{y}} \tag{1.8}$$

Розповсюдження ефективно реалізовано як згортка з одиничною матрицею як ядро. Поширення уваги значно покращує результати малювання в тестуванні та збагачує градієнти в навчанні.

Щоб інтегрувати модуль уваги, представляємо два паралельні кодери, як показано на рисунку 1.9.

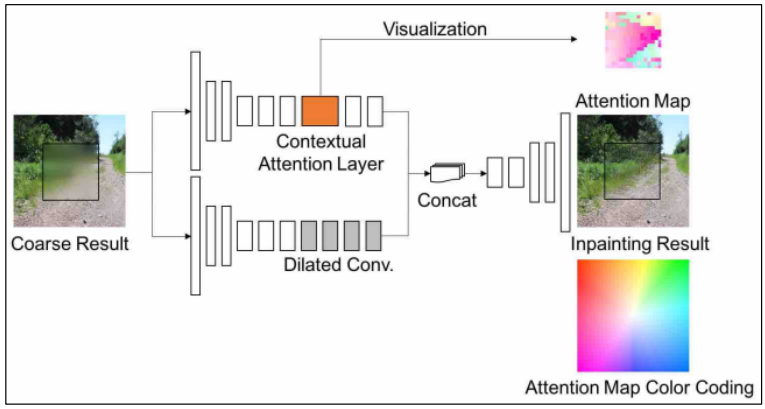

Рисунок 1.9 – Інтегрування модулю уваги[4]

Нижній кодер спеціально фокусується на галюцинаційному вмісті з пошаровим (розширеним) згортанням, тоді як верхній намагається звернути увагу на фонові особливості інтерес. Вихідні характеристики з двох кодерів агрегуються та подаються в один декодер для отримання остаточного результату. Щоб інтерпретувати контекстну увагу, ми візуалізуємо його так, як показано на рисунку 1.9. Використовуємо колір, щоб вказати відносне розташування найбільш

цікавого фонового фрагмента для кожного пікселя переднього плану. Наприклад, білий (центр карти кодування кольорів) означає, що піксель стежить сам за собою, рожевий – унизу ліворуч, зелений – угорі справа. Значення зсуву масштабується по-різному для різних зображень, щоб найкраще візуалізувати найцікавіший діапазон.

Порівняння результатів з іншими методами таблиці 1.1, а також результат роботи метода на рисунку 1.10.

Таблиця 1.1 – Порівняння Contextual Attention з PatchMatch

Метод	l_1 loss	l_2 loss	PSNR	TV Loss
PatchMatch	16.1%	3.9%	16.62	25.0%
Contextual Attention	8.6%	2.1%	18.91	25.3%

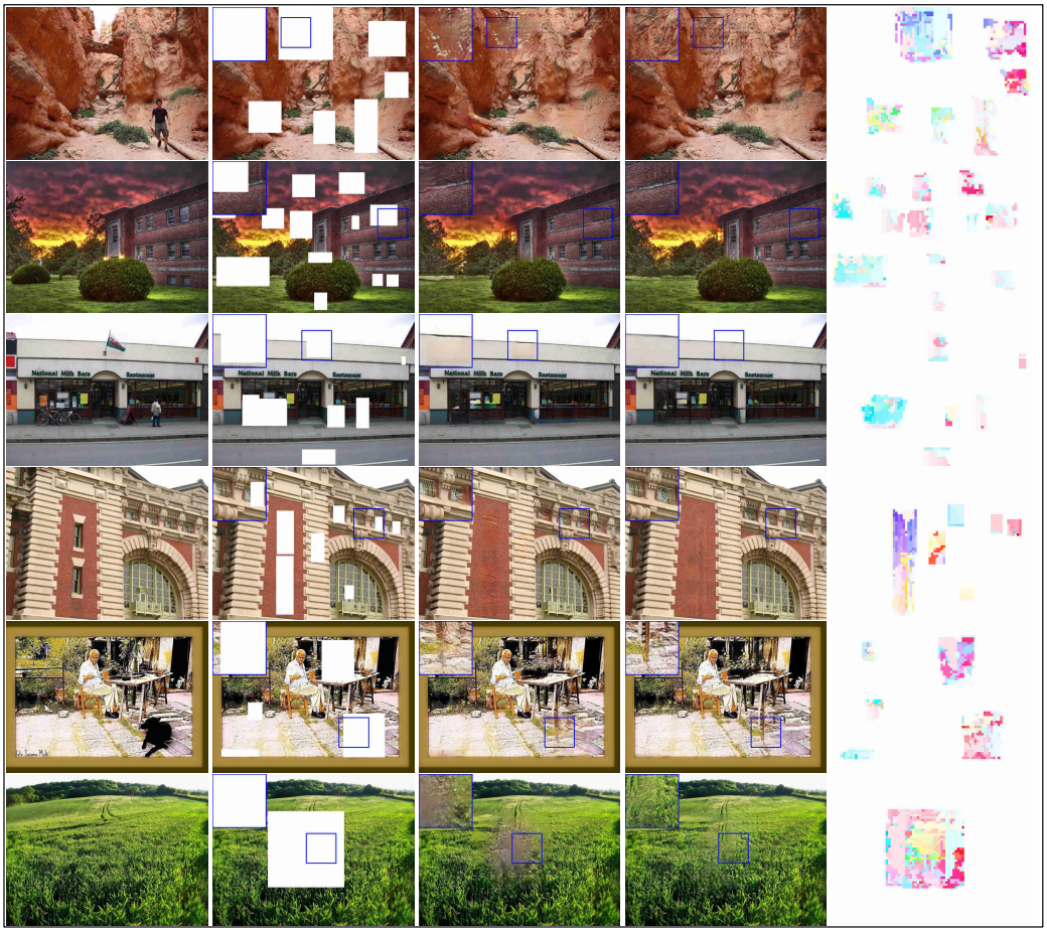

Рисунок 1.10 – Результат роботи Contextual Attention[4]

1.1.5 Довільне малювання зображень зі Gated Convolution

Ванільні згортки погано підходять для завдання малювання зображень вільної форми. Розглянемо згортковий шар, у якому банк фільтрів застосовано до вхідної карти функцій як вихід. Припустимо, що вхід є C-каналом, кожен піксель, розташований (y, x) у вихідній карті $\acute{C}$-каналу, обчислюється як (формула 1.9):

$$O_{y,x} = \sum_{i=-\acute{k}_h}^{\acute{k}_h} \sum_{j=-\acute{k}_w}^{\acute{k}_w} W_{\acute{k}_h+i,\acute{k}_w+j} * I_{y+i,x+j} \tag{1.9}$$

де x, y представляє вісь x, вісь y вихідної карти,

k_h і k_w — розмір ядра (наприклад, 3 × 3),

$\acute{k}_h = \frac{k_h-1}{2}, \acute{k}_w = \frac{k_w-1}{2}$, $\mathrm{W} \in \mathbb{R}^{k_h * k_w * \acute{C} * C}$ представляє згорткові фільтри, $I_{y+i,x+j} \in \mathbb{R}^C$ та $O_{y,x} \in \mathbb{R}^{\acute{C}}$ входами та виходами.

Для простоти зміщення в згортці ігнорується.

Рівняння показує, що для всіх просторових положень (y, x) застосовуються однакові фільтри для отримання виходу у ванільних згорткових шарах. Це має сенс для таких завдань, як класифікація зображень і виявлення об’єктів, де всі пікселі вхідного зображення дійсні, для вилучення локальних особливостей у ковзному вікні. Однак для зображення в живописі вхід складаються як з областей із дійсними пікселями/об’єктами поза отворами, так і з недійсними пікселями/об’єктами (у неглибоких шарах) або синтезованими пікселями/об’єктами (у глибоких шарах) у замаскованих областях. Це спричиняє неоднозначність під час навчання та призводить до візуальних артефактів, таких як невідповідність кольорів, розмитість і очевидні реакції країв під час тестування.

Нещодавно запропоновано часткову згортку, яка адаптує крок маскування та повторної нормалізації, щоб зробити згортку залежну лише від дійсних пікселів як (формула 1.10):

$$O_{y,x} = \begin{cases} \sum\sum W * \left(I \odot \frac{M}{sum(M)}\right), if\ sum(M) > 0 \\ 0, \qquad\qquad otherwise \end{cases} \tag{1.10}$$

Часткова згортка покращує якість малювання на неправильній масці, але все ще має проблеми:

- вона евристично класифікує всі просторові розташування як дійсні або недійсні. Маска в наступному шарі буде встановлена на одиниці незалежно від того, скільки пікселів охоплено діапазоном фільтра на попередньому шарі (наприклад, 1 дійсний піксель і 9 дійсних пікселів розглядаються як однакові для оновлення поточної маски);
- він несумісний із додатковими даними користувача. Вона націлені на керовану користувачем систему малювання зображень, де користувачі можуть додатково надати розріджений ескіз усередині маски як умовні канали. У цій ситуації чи слід вважати ці розташування пікселів дійсними чи недійсними? Як правильно оновити маску для наступного шару;
- для часткової згортки недійсні пікселі поступово зникатимуть у глибоких шарах, поступово перетворюючи всі значення маски на одиниці. Однак , якщо ми дозволяємо мережі автоматично вивчати оптимальну маску, мережа призначає значення м’якої маски для кожного просторового розташування навіть у глибоких шарах;
- усі канали в кожному шарі мають однакову маску, що обмежує гнучкість. По суті, часткову згортку можна розглядати як жорстке стробування одноканальної функції.

Стробована згортку для мережі малювання зображень, як показано на рисунку 1.11. Замість жорсткої маски стробування, оновленої правилами, стробована згортка автоматично вивчає м’яку маску з даних. Він формулюється так (формула 1.11):

$$Gating_{y,x} = \sum\sum W_g * I \tag{1.11}$$

$$Feature_{y,x} = \sum\sum W_f * I$$

$$O_{y,x} = \varphi(Feature_{y,x}) \odot \sigma(Gating_{y,x})$$

де σ є сигмоподібною функцією, таким чином вихідні значення стробування знаходяться між нулями та одиницями;

φ може бути будь-якою функцією активації (наприклад, ReLU, ELU та LeakyReLU);

W_g і W_f є двома різними згортковими фільтрами.

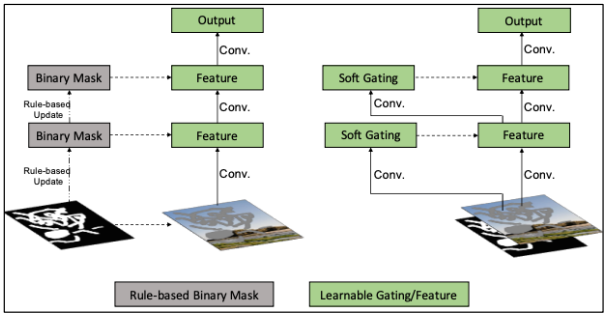

Рисунок 1.11 – Ілюстрація часткової згортки (ліворуч) і стробованої згортки (праворуч).

Стробована згортка вивчає динамічний механізм вибору ознак для кожного каналу та кожного просторового розташування. Цікаво, що візуалізація проміжних значень стробування показує, що він вчиться вибирати ознаку не тільки відповідно до фону, маски, ескізу, але також враховуючи семантичну сегментацію в деяких каналах. Навіть у глибоких шарах стробована згортка вчиться виділяти замасковані області та накреслювати інформацію в окремих каналах для кращого отримання результатів малювання.

Для попередніх мереж малювання, які намагаються заповнити один прямокутний отвір, додатковий локальний GAN використовується на маскованій прямокутній області для покращення результатів. Однак розглядаємо завдання малювання зображення вільної форми, де в будь-якому місці може бути кілька

отворів будь-якої форми. Спираючись на глобальні та локальні GAN, Markovian GAN, перцептивні втрати і недавню роботу над спектрально нормалізованими GAN, розглядаємо просту та ефективну GAN втрат, SN-PatchGAN, для безкоштовного навчання. Згорточна мережа використовується як дискримінатор, де вхідні дані складаються із зображення, маски та каналів наведення, а вихідні дані є тривимірним об'єктом форми $R^{h \times w \times c}$(h, w, c представляють висоту, ширину та число каналів відповідно). Як показано на рисунку 1.12, шість ступінчастих згорток із розміром ядра 5 і кроком 2 складено, щоб зафіксувати статистику функцій марковських патчів.

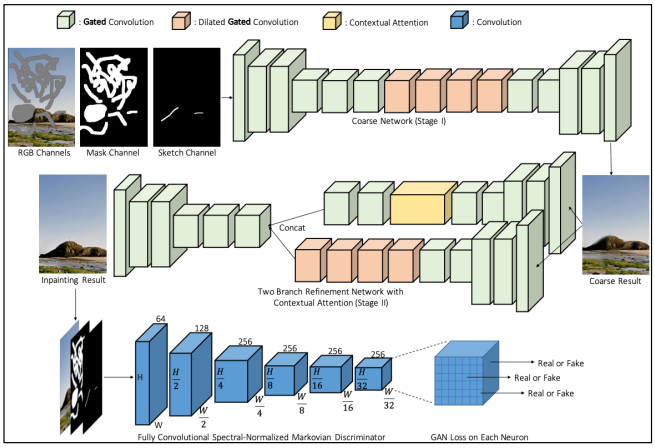

Рисунок 1.12 - Gated convolution та SN-PatchGAN

Порівняння результатів можна побачити в таблиці 1.2.

Таблиця 1.2 – Порівняння GatedConvolution з іншими методами.

Метод	Прямокутна маска		Вільної форми маск	
	l_1 помилка	l_2 помилка	l_1 помилка	l_2 помилка
PatchMatch	16.1%	3.9%	11.3%	2.4%
Global&Loca	9.3%	2.2%	21.6%	7.1%
ContextAttention	8.6%	2.1%	17.2%	4.7%
PartialConv	9.8%	2.3%	10.4%	1.9%
GatedConvolution	8.6%	2.0%	9.1%	1.6%

Приклади якісного порівняння на наборах перевірки Places2 і CelebA-HQ можна побачити на рисунку 1.13.

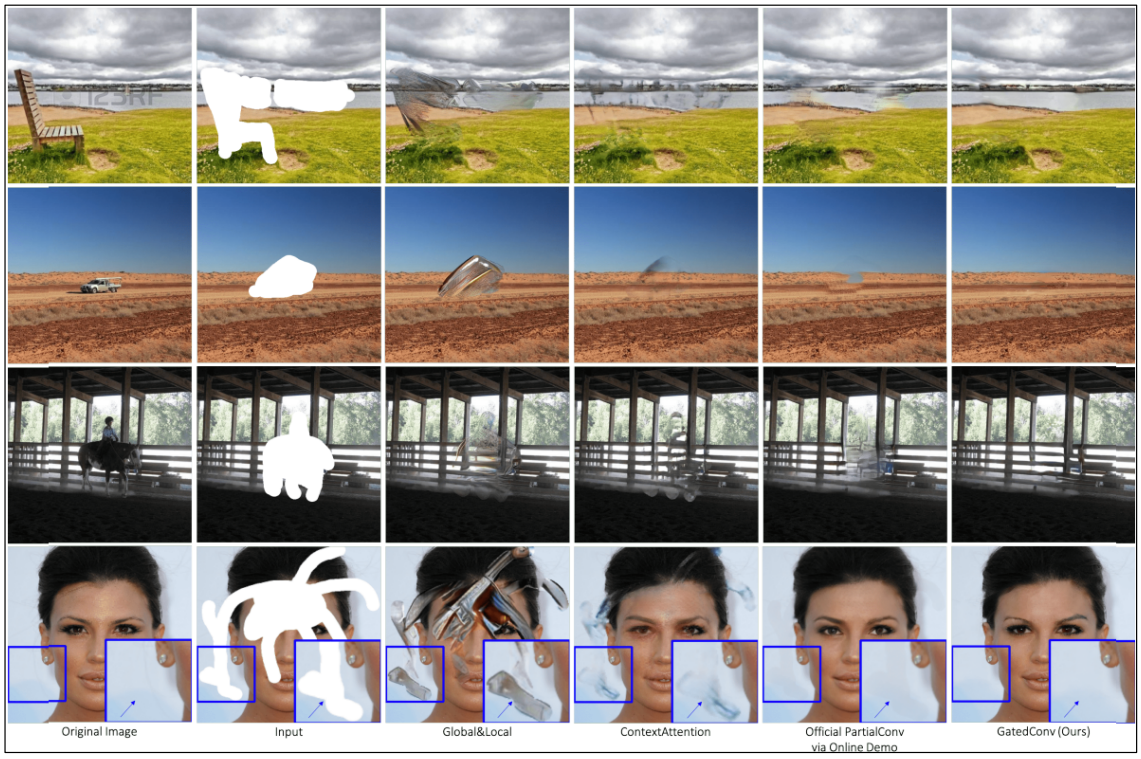

Рисунок 1.13 – Робота Gated Convolution

1.1.6 Stable Diffusion Inpainting

Stable Diffusion Inpainting – це техніка, яка поєднує можливості моделі Stable Diffusion з концепцією інпейнтингу. Ось детальний опис обох компонентів:

– Stable Diffusion – це AI модель, призначена для генерації високоякісних зображень з текстових описів. Вона побудована на основі передових технік глибокого навчання, що дозволяє створювати детальні та реалістичні зображення на основі широкого спектру запитів. Ця модель особливо відома своєю ефективністю та якістю генерованих зображень;

– інпейнтинг, у контексті AI та комп'ютерного зору, означає процес реконструкції втрачених або пошкоджених частин зображень. Це форма редагування зображень, що полягає у заповненні відсутніх або пошкоджених областей на фото чи мистецькому творі, використовуючи контекст, що оточує, як путівник. Це може включати все від ремонту

подряпин на старій фотографії до заповнення відсутніх частин зображення таким чином, щоб воно було безшовним і узгоджувалося з загальним зображенням.

У поєднанні Stable Diffusion Inpainting означає техніку, де модель Stable Diffusion використовується для виконання завдань інпейнтингу. Це означає, що модель не лише генерує зображення з нуля на основі текстових описів, але також може модифікувати існуючі зображення, заповнюючи відсутні частини або змінюючи певні аспекти, як вказано в запиті. Ця технологія має широкий спектр застосувань, включаючи створення цифрового мистецтва, відновлення фотографій і навіть у таких областях, як кіно та ігри, де потрібен реалістичний візуальний контент.

Інтеграція генеративних можливостей Stable Diffusion з інпейнтингом дозволяє створити більш гнучкий та потужний інструмент у генерації та маніпуляції зображень. Це яскравий приклад того, як AI розвиває область цифрової обробки та створення зображень. Приклад роботи цього методу можна побачити на рисунку 1.14. На сьогодні це найсучасніша технологія.

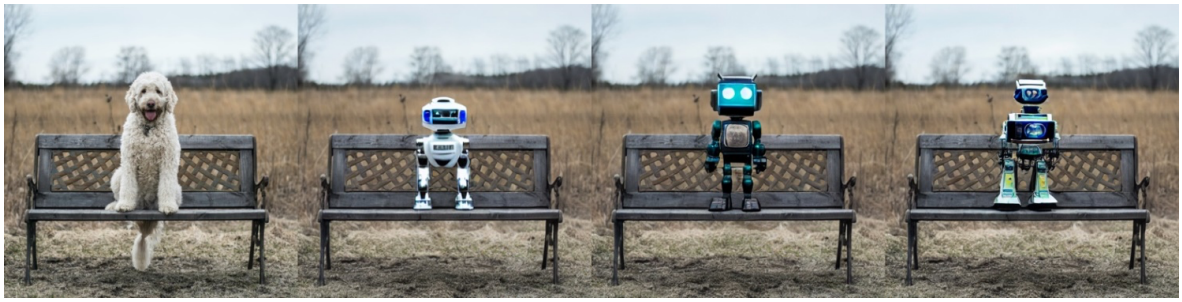

Рисунок 1.14 – Робота Stable Diffusion

1.2. Виявлення проблем та актуалізація рішень

Серед основних викликів – обмеження точності в заміщенні об'єктів та врахуванні контексту, що призводить до втрати якості та реалістичності кінцевих зображень. Важливо також зазначити високі вимоги до обчислювальних ресурсів, що обмежує широке використання складних алгоритмів.

У відповідь на ці виклики, розглядаємо застосування сучасних технологій, зокрема методів глибокого навчання, які можуть підвищити точність та якість

заміщення об'єктів. Залежно від конкретних вимог та умов, різні підходи можуть бути більш ефективними: деякі методи краще підходять для обробки складних зображень, інші – для швидкої обробки з меншими вимогами до ресурсів. Ця гнучкість у виборі методології дозволяє оптимізувати процес обробки зображень, враховуючи специфічні потреби кожного випадку.

Цей аналіз підкреслює важливість розвитку та імплементації інноваційних технологій в галузі обробки зображень, вказуючи на потенціал покращення якості та ефективності роботи з візуальними даними.

1.3 Постановка задачі

Метою цієї роботи є розробка системи для заміщення об'єктів на зображеннях, використовуючи різні методи обробки. Система повинна дозволяти вибирати найбільш підходящий метод в залежності від конкретних потреб та умов, таких як складність зображення, доступність ресурсів, та бажана швидкість обробки. Основні завдання включають аналіз існуючих методів, розробку та інтеграцію алгоритмів, а також оцінку ефективності та якості заміщення об'єктів. Ця система покликана вдосконалити процес обробки зображень, забезпечуючи більшу гнучкість та ефективність.

Програмна система повинна мати базовий функціонал такий як:

- завантаження власного зображення та маску;
- вибрати метод обробки зображення;
- можливість порівняти результати;
- розрахувати кількісні метрики.

Також система повинна надавати можливість власноруч намалювати маску на заданому зображенні, зберегти її та звантажити.

2 ПЛАНУВАННЯ ЕКСПЕРИМЕНТАЛЬНОЇ ЧАСТИНИ ДОСЛІДЖЕННЯ

2.1 Визначення об’єкта, предмета та мети дослідження

Об’єктом дослідження є методи обробки зображень, що використовуються для реалістичного заміщення об'єктів на зображеннях. Вони включають в себе різноманітні алгоритми та підходи, які дозволяють видалити, замінити або додати елементи в зображенні, зберігаючи його природний вигляд.

Предметом дослідження є аналіз і застосування сучасних технологій, таких як генеративні змагальні мережі (GANs), згорткові нейронні мережі (CNNs) та Stable Diffusion, для задачі заміщення об’єктів на зображеннях. Дослідження охоплює вивчення механізмів визначення та відновлення втрачених або пошкоджених частин зображень, а також створення методів для введення нових елементів у візуальний контент без порушення його цілісності.

Метою дослідження є розробка ефективних методів для реалістичного заміщення об’єктів на зображеннях, що включають визначення та аналіз найбільш дієвих підходів та алгоритмів у цій галузі. Основна увага приділяється не тільки покращенню якості заміщення та інтеграції об’єктів у зображення, але й збереженню природності та реалістичності вихідних зображень. Значну роль відіграє визначення критеріїв оцінки ефективності розроблених методів, що дозволить формалізувати процес вибору оптимального рішення для конкретних задач.

2.2 Вибір платформи проведення дослідження

Для реалізації методів обробки зображень, які фокусуються на реалістичному заміщенні об'єктів та відновленні зображень, зокрема використанні технології, відомої як DeepFill, було обрано фреймворк PyTorch. PyTorch є одним з провідних інструментів у галузі машинного навчання та глибокого навчання, що надає високий рівень гнучкості та ефективності для розвитку складних моделей нейронних мереж.

Для підвищення доступності та зручності користування розробленими методами перед кінцевими користувачами, демонстрація результатів дослідження

та інтерактивне тестування моделей буде виконано з використанням Next.js. Цей сучасний фреймворк для веб-розробки на основі React дозволить створити реактивний, високопродуктивний веб-інтерфейс, який забезпечить користувачам легкий доступ до інструментів редагування зображень, базованих на PyTorch і DeepFill, без необхідності занурення у технічні деталі використання самої моделі.

Вибір PyTorch як основи для реалізації DeepFill та Next.js для фронтенду забезпечує сильну основу для ефективної реалізації цілей дослідження, забезпечуючи одночасно гнучкість розробки моделі та зручність її використання..

2.3 Визначення алгоритмів

Дослідження включає в себе використання низки передових алгоритмів для обробки зображень, з акцентом на методах глибокого навчання. Серед ключових алгоритмів - DeepFill для відновлення зображень та Stable Diffusion для генерації зображень. Особливість Stable Diffusion полягає в здатності створювати високоякісні зображення з врахуванням контексту вхідних даних, що відкриває широкі можливості для заміщення об'єктів на зображеннях з високим рівнем реалістичності.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Проєктування архітектури програмної системи

Розроблена програмна система складається із двох основних компонентів: серверної частини (бекенд) та клієнтської частини (фронтенд). Така структура дозволяє чітко розділити логіку обробки даних та інтерфейс користувача, сприяючи кращій організації роботи системи.

Серверна частина (бекенд) відповідає за обробку запитів від клієнта, виконання операцій обробки зображень, тренування та використання моделей машинного навчання. Для розробки серверної частини використовується Python, зокрема, фреймворки та бібліотеки для глибокого навчання, як-от PyTorch. Комунікація з клієнтською частиною здійснюється через API, реалізоване з використанням FastAPI, що забезпечує швидку та ефективну обробку запитів.

Клієнтська частина (фронтенд) створено з використанням фреймворку Next.js і є інтерфейсом між користувачем та сервером. Ця частина забезпечує візуальне представлення даних, дозволяє користувачам взаємодіяти з системою через веб-інтерфейс: завантажувати зображення для обробки, налаштовувати параметри обробки та переглядати результати. Розробка клієнтської частини зорієнтована на створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу.

Обидві частини системи взаємодіють між собою через визначений API, що дозволяє забезпечити гнучкість та масштабованість проєкту. Використання FastAPI для реалізації API спрощує процес розробки та забезпечує високу продуктивність обробки запитів, а Next.js допомагає створити реактивний інтерфейс, який задовольнить потреби користувачів у зручності та швидкості роботи з системою.

3.2 UML-проєктування програмної системи

3.2.1 Use-Case діаграма

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених межею системи (прямокутник),

асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система під час діалогу з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів із системою.

На діаграмі (див. рис. 3.1) виокремлено основні сценарії використання системи для обробки зображень, які дозволяють користувачу виконати серію дій для досягнення бажаного результату. Кожен сценарій описує конкретну функцію системи і як користувач може її використовувати.

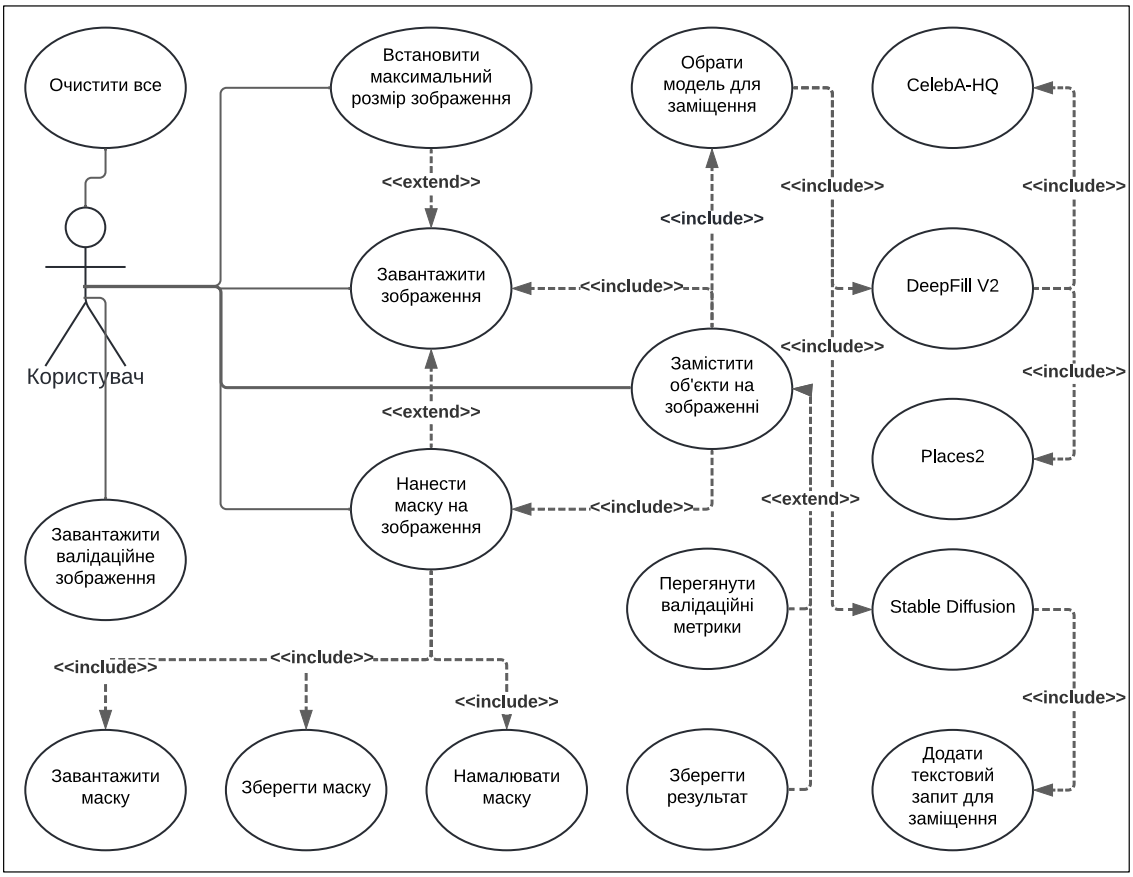

Рисунок 3.1 – Use Case діаграма

Діаграма включає наступні сценарії використання:

- завантаження зображення: користувач ініціює процес обробки зображень, вибираючи та завантажуючи зображення в систему. Тут також можливо встановлення максимального розміру зображення перед його завантаженням для відповідності технічним обмеженням системи;
- нанесення маски на зображення: користувач визначає область зображення, яка підлягає заміщенню або обробці, шляхом нанесення маски. Ця дія є важливою для подальшої точної роботи моделі;
- завантаження та збереження маски: програмна система надає можливість не тільки завантажувати зовнішні маски, але й зберігати створені маски в системі для подальшого використання;
- вибір моделі для заміщення: користувач обирає одну з доступних моделей, як-от CelebA-HQ, DeepFill V2, Places2 або Stable Diffusion, для виконання заміщення об'єктів на зображенні;
- додаткові опції для заміщення: у разі використання Stable Diffusion, користувачу надається додаткова можливість ввести текстовий опис, що буде використаний для генерації заміщення;
- завантажити зображення для перевірки та розрахунків метрик.

Кожен з цих сценаріїв описує певну взаємодію між користувачем та системою, вказуючи на потенційні дії, які користувач може виконувати. Ця діаграма є вихідним пунктом для розробки детальних вимог до системи та її функціональності. Use case діаграма слугує також основою для проектування інтерфейсу користувача та реалізації бізнес-логіки системи, забезпечуючи чітке та зрозуміле розуміння всіх взаємодій в системі.

3.2.2 Deployment діаграма

На Deployment діаграмі (рисунок 3.2) показано розгортання компонентів програмної системи для обробки зображень. Діаграма відображає взаємодію між різними частинами системи та описує середовище виконання для кожного з компонентів.

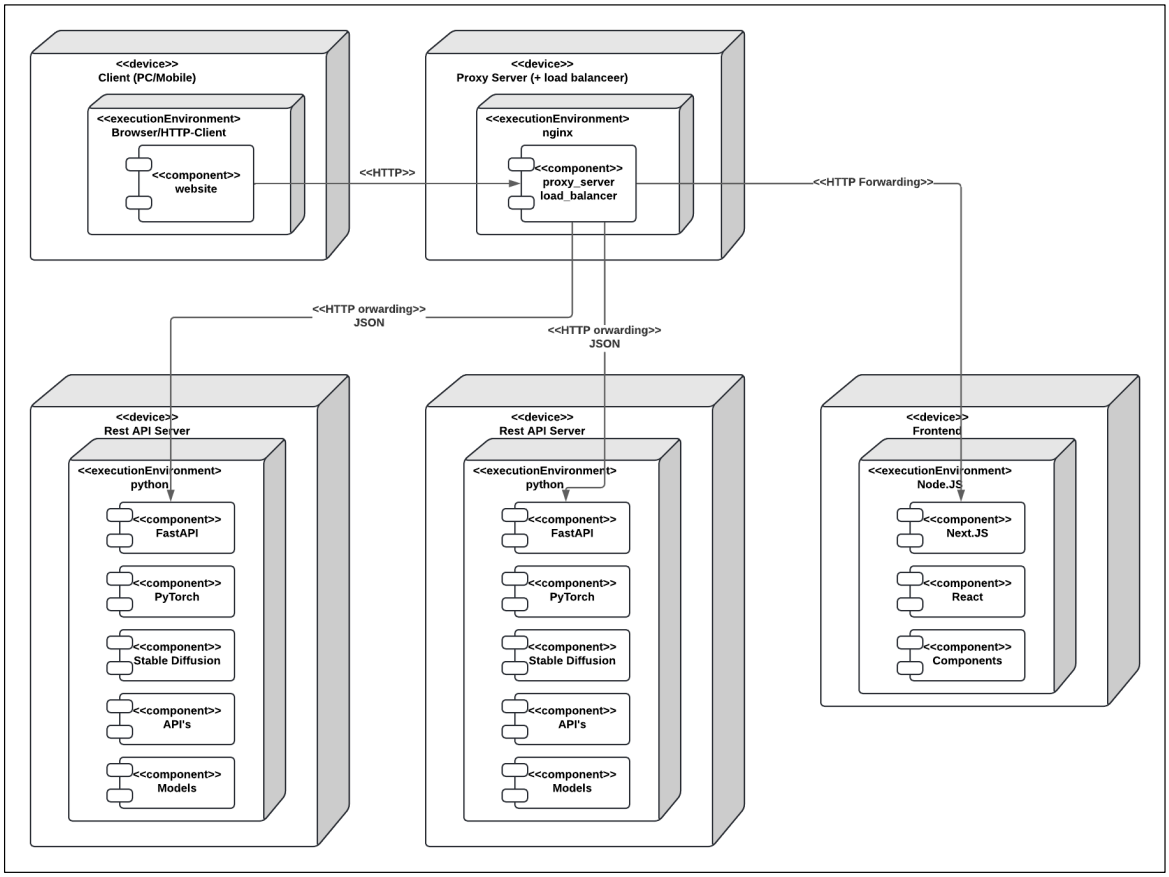

Рисунок 3.2 – Діаграма розгортання

На діаграмі зображені основні компоненти:

а) клієнт (PC/Mobile):

1) середовище виконання: Browser/HTTP Client;
2) компонент: Вебсайт, який є точкою входу для користувачів системи;

б) проксі-сервер (+ load balancer):

1) середовище виконання: nginx;
2) компонент: проксі-сервер та балансувальник навантаження, який відповідає за розподіл запитів до серверів та забезпечення високої доступності та швидкості обробки запитів;

в) REST API Server:

1) середовище виконання: Python;
2) компоненти:

- FastAPI, який забезпечує високопродуктивне API для обробки запитів;
- PyTorch, фреймворк для моделей глибокого навчання;
- Stable Diffusion, модель для генерації зображень;
- API's та Models, набір API для взаємодії з іншими сервісами та моделями;

г) Frontend:

1) середовище виконання: Node.JS;
2) компоненти:
   - Next.JS, серверний фреймворк для React додатків;
   - React, бібліотека для побудови інтерфейсу користувача;
   - Components, набір компонентів React, які використовуються для побудови клієнтської частини системи.

Запити від клієнтів перенаправляються через проксі-сервер, який використовує HTTP forwarding, для взаємодії з REST API сервером, що обробляє бізнес-логіку і повертає відповіді у форматі JSON. Ця структура розгортання сприяє ефективному масштабуванню системи та підвищенню її надійності та швидкості реакції на запити користувачів.

3.2.3 Діаграма пакетів серверної частини

Діаграма пакетів у UML (Unified Modeling Language) використовується для групування елементів моделі в пакети, що дозволяє організовувати великі системи на логічні кластери елементів, спрощує управління моделями і дає змогу виявити залежності між різними частинами системи.

На діаграмі пакетів (рисунок 3.3) елементи системи поділяються на пакети, що можуть включати класи, інтерфейси, компоненти, інші пакети, і так далі. Вони є контейнерами, що допомагають управляти елементами моделі, організовуючи їх у групи за певним критерієм, наприклад за функціональністю, за рівнем абстракції, за модулями чи за рівнями розподілу в мережевій інфраструктурі.

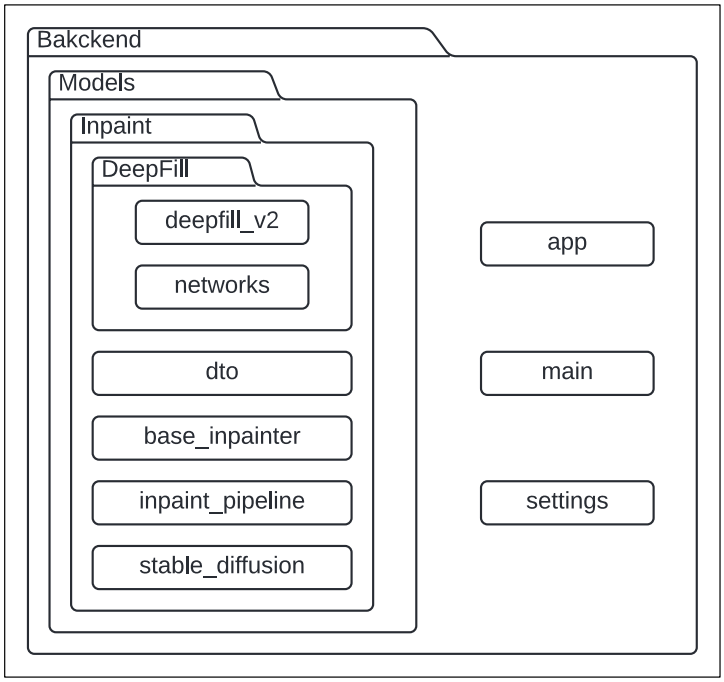

Рисунок 3.3 – Діаграма пакетів серверної частини

4 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

4.1 Розробка серверної частини

Серверна частина виконує ключову роль в обробці запитів від клієнтів, управлінні даними та обробці зображень. Для реалізації серверної частини було обрано мову програмування Python з наступних причин:

- простота та зручність: Python є одним з найпопулярніших мов програмування завдяки його простоті та читабельності коду. Це дозволяє розробникам швидко писати та підтримувати код, що є важливим фактором при створенні складних систем обробки зображень;
- широкий спектр бібліотек: Python має багатий набір бібліотек для машинного навчання та обробки зображень, таких як PyTorch. Ці бібліотеки забезпечують ефективну реалізацію та тренування моделей глибокого навчання, що є критичним для досягнення високої точності та якості обробки зображень;
- активна спільнота: Велика та активна спільнота розробників Python сприяє швидкому вирішенню проблем та доступності численних ресурсів для навчання та розвитку, що допомагає у швидкому впровадженні нових технологій та покращенні існуючих рішень.

Для створення API було обрано фреймворк FastAPI через такі переваги:

- висока продуктивність: FastAPI дозволяє створювати високопродуктивні API завдяки асинхронній обробці запитів. Це значно підвищує швидкість відповіді та зменшує затримки при обробці великої кількості запитів;
- швидкість розробки: FastAPI забезпечує автоматичну генерацію документації, що значно спрощує процес створення та тестування API. Це дозволяє розробникам зосередитися на логіці обробки даних, а не на написанні документації;
- підтримка сучасних стандартів: FastAPI базується на сучасних стандартах Python, таких як типізація та асинхронність, що дозволяє

створювати масштабовані та ефективні сервіси. Це забезпечує гнучкість та надійність у розробці складних систем;

– інтеграція з інструментами розробки: FastAPI легко інтегрується з іншими інструментами та бібліотеками, що забезпечує зручність у розробці та тестуванні. Це робить його ідеальним вибором для створення сучасних веб-сервісів.

4.1.1 Підготовчі кроки до інтеграції методів заміщення

У цьому підрозділі буде описано підготовчі кроки, які необхідно виконати перед інтеграцією методів заміщення об'єктів на зображеннях, а також використані патерни проектування.

Для реалізації методів заміщення використовується клас «InPaintPipeline» (див. Додаток Е), який забезпечує завантаження конфігурацій, управління моделями та здійснення заміщення об'єктів. Нижче наведено опис основних компонентів цього класу та підготовчих кроків до інтеграції.

Клас «InPaintPipeline» відповідає за завантаження конфігурацій, ініціалізацію моделей та здійснення заміщення об'єктів на зображеннях. Основні методи класу включають:

– ініціалізація: конструктор класу приймає шлях до конфігураційного файлу та ініціалізує порожні структури для зберігання конфігурацій та моделей;

– отримання доступних моделей: метод «get_available_models» повертає список доступних моделей, які можуть бути використані для заміщення об'єктів;

– завантаження моделей: метод «load» завантажує конфігураційний файл та ініціалізує моделі згідно з конфігурацією. Цей метод використовує патерн проектування «фабричний метод» для створення об'єктів моделей, що дозволяє гнучко додавати нові моделі без зміни основної логіки класу;

- заміщення об'єктів: метод «inpaint» здійснює заміщення об'єктів на зображенні за допомогою обраної моделі. Використання патерну проектування «Стратегія» дозволяє вибирати різні методи заміщення в залежності від обраної моделі;
- розрахунок метрик: метод «calculate_metrics» оцінює якість заміщення, порівнюючи результати з еталонними зображеннями та використовуючи відповідні метрики, такі як PSNR (англ. peak signal-to-noise ratio - пікове співвідношення сигналу до шуму), SSIM (англ. structure similarity), MSE (англ. Mean Square Error).

Клас «BaseInPainter» (див. Додаток Ж) є абстрактним базовим класом для конкретних реалізацій моделей заміщення. Основні методи класу включають:

- ініціалізація: конструктор класу задає ім'я, пристрій для обчислень та інші параметри. Якщо параметр «preload» встановлено, то модель завантажується автоматично;
- завантаження моделі: абстрактний метод «load» реалізує завантаження моделі. Використовується патерн проектування «шаблонний метод», який дозволяє конкретним реалізаціям визначати специфічні кроки завантаження;
- виконання обчислень: абстрактний метод «inference» відповідає за виконання обчислень по заміщенню об'єктів;
- попередня обробка: Метод «pre_process» конвертує зображення та маски у відповідні формати перед передачею їх до моделі;
- заміщення об'єктів: Метод «inpaint» виконує заміщення об'єктів, забезпечуючи правильний порядок обробки. Використовується структура «шаблонний метод», яка визначає послідовність кроків обробки;
- звільнення ресурсів: Метод «free» звільняє ресурси, використовувані моделлю;
- визначення доступного пристрою: Метод «get_available_gpu_or_cpu» визначає доступний пристрій для обчислень (CPU, GPU, MPS).

4.1.2 Інтеграція DeepFillV2

У цьому підрозділі буде детально описано процес інтеграції методу DeepFillV2 для заміщення об'єктів на зображеннях. DeepFillV2 є сучасним методом, який використовує генеративні змагальні мережі для виконання завдань інпейнтингу, забезпечуючи високу якість та реалістичність результатів.

Для інтеграції DeepFillV2 був розроблений клас «DeepFillV2InPainter», який успадковує основні властивості від абстрактного класу «BaseInPainter». Клас відповідає за завантаження моделі, підготовку даних, виконання обчислень та повернення результатів.

По-перше реалізуємо метод «load», котрий буде завантажувати модель:

```
def load(self):
    gen_sd = torch.load(self.model_path, map_location=self.device)['G']
    self.model = Generator(cnum_in=5, cnum=48, return_flow=False)
    self.model = self.model.to(self.device)
    self.model.eval()
    self.model.load_state_dict(gen_sd, strict=False)
```

Для цього ми завантажуємо вагу, а потім передаємо це у генеративну мережу. Наступним кроком потрібно додати метод, який буде відповідати за попередню обробку зображення:

```
def _pre_process(self, pil_image: Image.Image, pil_mask_image: Image.Image)
-> (torch.Tensor, torch.Tensor):
    mw, mh = pil_mask_image.size
    scale = self.max_size / max(mw, mh)

    pil_mask_image = pil_mask_image.resize(
        (self.max_size, int(scale * mh)) if mw > mh else (int(scale * mw),
self.max_size)
    )
    pil_image = pil_image.resize(pil_mask_image.size)

    image, mask = ToTensor()(pil_image), ToTensor()(pil_mask_image)
    image, mask = image.to(self.device), mask.to(self.device)
    return image, mask
```

У цьому методі ми масштабуємо вхідне зображення до максимально допустимого, а також переносимо на відповідний девайс.

Метод «_infer_deep_fill» виконує обчислення, необхідні для заміщення об'єктів на зображенні. Він нормалізує значення зображення, створює маски та передає їх у модель для отримання результату:

```
@torch.inference_mode()
def _infer_deep_fill(self, image: torch.Tensor, mask: torch.Tensor) ->
Image.Image:
    _, h, w = image.shape
    grid = 8

    image = image[:3, :h // grid * grid, :w // grid * grid].unsqueeze(0)
    mask = mask[0:1, :h // grid * grid, :w // grid * grid].unsqueeze(0)

    image = (image * 2 - 1.)  # map image values to [-1, 1] range
    # 1.: masked 0.: unmasked
    mask = (mask > 0.).to(dtype=torch.float32)

    image_masked = image * (1. - mask)  # mask image

    ones_x = torch.ones_like(image_masked)[:, 0:1, :, :]  # sketch channel
    x = torch.cat([image_masked, ones_x, ones_x * mask], dim=1)    #
concatenate channels

    x_stage1, x_stage2 = self.model(x, mask)

    output = image * (1. - mask) + x_stage2 * mask

    image_inpainted = self._output_to_img(output)
    return image_inpainted
```

Форматування результату займається метод «_output_to_img» перетворює тензор, отриманий від моделі, назад у зображення PIL для зручного відображення та подальшого використання:

```
@staticmethod
def _output_to_img(out) -> Image.Image:
    out = (out[0].cpu().permute(1, 2, 0) + 1.) * 127.5
    out = out.to(torch.uint8).numpy()
    out = Image.fromarray(out)
    return out
```

4.1.3 Інтеграція Stable Diffusion

Stable Diffusion є сучасним методом, який використовує дифузійні моделі для виконання завдань інпейнтингу, забезпечуючи високу якість та реалістичність результатів.

Для інтеграції Stable Diffusion був розроблений клас «StableDiffusionInPainter», який успадковує основні властивості від абстрактного класу «BaseInPainter». Клас відповідає за завантаження моделі, підготовку даних, виконання обчислень та повернення результатів.

Базовий конструктор був розширений параметром enable_attention_slicing для того, щоб забезпечити можливість оптимізації обчислень за допомогою поділу уваги. Ініціалізація також включає передачу додаткових параметрів до базового класу BaseInPainter:

```
class StableDiffusionInPainter(BaseInPainter):
    def __init__(self, enable_attention_slicing: bool = True, **kwargs):
        super().__init__(**kwargs)
        self.enable_attention_slicing = enable_attention_slicing
```

Метод «load» завантажує попередньо натреновану модель Stable Diffusion з використанням бібліотеки diffusers. Модель завантажується на відповідний обчислювальний пристрій (CPU або GPU), а також може бути оптимізована за допомогою поділу уваги:

```
def load(self) -> None:
    self.model = StableDiffusionInpaintPipeline.from_pretrained(
        "runwayml/stable-diffusion-inpainting",
        variant="fp16",
        torch_dtype=torch.float16,
    )
    self.model = self.model.to(self.device)
    if self.enable_attention_slicing:
        self.model.enable_attention_slicing()
    self.is_loaded = True
```

Метод «pre_process» змінює розмір зображення та маски, забезпечуючи їх відповідність розмірам, необхідним для обробки моделлю. Він викликає базовий метод «pre_process» з класу «BaseInPainter» для базової обробки та потім змінює розмір зображення до розміру маски:

```
def pre_process(self, image: Image, mask: Image, prompt: str = None) ->
(Image, Image, str):
```

```
    pil_image, pil_mask_image, prompt = super().pre_process(image, mask,
prompt)
    pil_image = pil_image.resize(mask.size)
    return pil_image, pil_mask_image, prompt
```

Метод «change_inpainted_area» забезпечує заміщення областей зображення, використовуючи маску. Він об'єднує вихідне зображення та інпейнтингове зображення на основі маски, створюючи кінцевий результат:

```
@staticmethod
def change_inpainted_area(image: Image, mask: Image, inpainted_image:
Image) -> Image:
    image, inpainted_image, mask = np.array(image),
np.array(inpainted_image), np.array(mask)
    mask_3ch = cv2.merge([mask, mask, mask])
    result = np.where(mask_3ch, inpainted_image, image)
    result = Image.fromarray(result)
    return result
```

Метод «inference» виконує основні обчислення для заміщення об'єктів на зображенні. Він використовує модель Stable Diffusion для генерації зображення з заміщеними об'єктами. Результати зберігаються у вигляді списку об'єктів «InPaintResult»:

```
def inference(self, image: Image, mask: Image, prompt: str = None) ->
list[InPaintResult]:
    model_result = self.model(prompt=prompt, image=image, mask_image=mask)
    result_image = model_result.images[0]
    results = [
        InPaintResult(name=self.name, image=result_image),
        InPaintResult(
            name=f"{self.name} Exchanged",
image=self.change_inpainted_area(image, mask, result_image)
        ),
    ]
    return results
```

4.2 Розробка клієнтської частини

У цьому підрозділі буде детально описано процес розробки клієнтської частини системи з використанням фреймворку Next.js. Next.js є популярним фреймворком для створення серверних рендерингів та статичних веб-додатків з

використанням React. Він забезпечує високу продуктивність, простоту в розробці та масштабованість, що робить його ідеальним вибором для нашого проекту.

Для реалізації клієнтської частини було обрано фреймворк Next.js з наступних причин:

- Next.js підтримує серверний рендеринг "із коробки", що забезпечує швидше завантаження сторінок та покращений SEO;
- можливість попереднього генерування сторінок під час збірки дозволяє створювати швидкі та ефективні веб-додатки;
- Next.js надає зручні інструменти для інтеграції з бекенд-сервісами через API маршрути;
- використання React як основи забезпечує високу модульність та можливість масштабування додатку.

Проект побудовано за підходом App Router, що забезпечує модульність та легкість у розробці. Структура проекту включає наступні основні компоненти:

- «app» - директорія для компонентів додатку та маршрутів. Кожен файл у цій директорії відповідає окремому маршруту;
- «components» - директорія для багаторазових React-компонентів;
- «styles» - директорія для файлів стилів, які використовуються для оформлення додатку;
- «api» - директорія для створення серверних API маршрутів, які можуть взаємодіяти з бекенд-сервісами.

Компонент «EditImage» створений для надання користувачам можливості завантаження, редагування зображень та створення масок для подальшої обробки інпейнтинговими моделями. Цей компонент грає ключову роль в інтерфейсі користувача, забезпечуючи простоту та зручність у взаємодії з системою. Нижче наведено кілька цікавих рішень, реалізованих у компоненті EditImage.

По-перше, компонент використовує два канваси:

- основний канвас відображає зображення та дозволяє користувачам малювати на ньому;

– прихований канвас використовується для створення та зберігання маски, не відображаючи зміни безпосередньо на основному канвасі. Це розділення дозволяє чітко відокремити логіку редагування маски від відображення зображення, забезпечуючи чистий та організований код.

Маска створюється на прихованому канвасі, що дозволяє відображати зміни на зображенні у реальному часі без зміни самого зображення. Це забезпечує точність редагування та зручність для користувача:

```
const draw = (e: React.MouseEvent<HTMLCanvasElement>) => {
    if (!isDrawing.current) return;
    const canvas = canvasRef.current;
    const maskCanvas = maskCanvasRef.current;
    if (!canvas || !maskCanvas) return;

    const rect = canvas.getBoundingClientRect();
    const x = e.clientX - rect.left;
    const y = e.clientY - rect.top;

    let ctx = canvas.getContext('2d');
    if (ctx) {
        ctx.fillStyle = 'white';
        ctx.beginPath();
        ctx.arc(x, y, brushRadius, 0, Math.PI * 2);
        ctx.fill();
    }

    let maskCtx = maskCanvas.getContext('2d');
    if (maskCtx) {
        maskCtx.fillStyle = 'white';
        maskCtx.beginPath();
        maskCtx.arc(x, y, brushRadius, 0, Math.PI * 2);
        maskCtx.fill();
        updateMask();
    }
};
```

Компонент забезпечує можливість збереження маски, створеної на прихованому канвасі, у форматі PNG. Це дозволяє користувачам зберігати результати своєї роботи для подальшого використання:

```
const saveMask = () => {
    if (maskCanvasRef.current) {
        const dataURL = maskCanvasRef.current.toDataURL('image/png');
        const link = document.createElement('a');
        link.download = 'mask.png';
        link.href = dataURL;
```

```
        link.click();
    }
};
```

Обробка завантаженої маски, використовуючи метод «preprocessMask», який перетворює зображення маски, видаляючи чорний колір і роблячи його прозорим. Це дозволяє користувачам завантажувати власні маски для редагування зображень:

```
const preprocessMask = (maskImage: HTMLImageElement, callback: (processedMask: HTMLCanvasElement) => void) => {
    const offscreenCanvas = document.createElement('canvas');
    offscreenCanvas.width = maskImage.width;
    offscreenCanvas.height = maskImage.height;
    const offscreenCtx = offscreenCanvas.getContext('2d');

    if (offscreenCtx) {
        offscreenCtx.drawImage(maskImage, 0, 0);
        const imageData = offscreenCtx.getImageData(0, 0, offscreenCanvas.width, offscreenCanvas.height);
        const data = imageData.data;

        for (let i = 0; i < data.length; i += 4) {
            if (data[i] === 0 && data[i + 1] === 0 && data[i + 2] === 0) {
                data[i + 3] = 0;
            }
        }

        offscreenCtx.putImageData(imageData, 0, 0);
        callback(offscreenCanvas);
    }
};
```

Компонент Inpaint є основним компонентом клієнтської частини додатку. Він відповідає за взаємодію користувача із системою, забезпечуючи завантаження зображень, створення та редагування масок, вибір моделей для інпейнтингу та відображення результатів. Також використовує бібліотеку SWR для асинхронного отримання даних. Основні функціональні можливості:

а) завантаження та збереження зображень і масок:
   1) користувач може завантажувати вихідне зображення, маску та додаткове зображення для перевірки коректності;
   2) використовується компонент EditImage для редагування зображень і створення масок;

б) вибір моделей для інпейнтингу:
   1) компонент динамічно завантажує доступні моделі інпейнтингу за допомогою SWR;
   2) користувач може обрати одну або декілька моделей для інпейнтингу зображення;
в) інпейнтинг зображень:
   1) при натисканні кнопки "Inpaint", обрані моделі застосовуються до завантаженого зображення;
   2) процес інпейнтингу виконується асинхронно для кожної моделі, і результати зберігаються у стані компонента;
г) відображення результатів:
   1) результати інпейнтингу відображаються у вигляді карток з зображеннями та метриками;
   2) користувач може завантажити результати у форматі PNG.

5 ОГЛЯД РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

Робота із програмною системою починається із головної сторінки веб-сайту, на якому ми можемо побачити (див. рисунок 5.1):

- список із доступних методів заміщення, такі як: StableDifusion, DeepFillV2Places2, DeepFillV2CelebaHQ;
- завантажити зображення (кнопка «Upload image»);
- завантажити маску (кнопка «Upload mask»);
- завантажити перевірочне зображення для розрахунку метрик (кнопка «Upload valid image»);
- змінити розмір канвасу;
- вказати розмір ластика для виділення маски;
- зберегти маску (кнопка «Save Mask»);
- видалити усе (кнопка «Clear All»;
- поле для вводу того, що ми хочемо добавити або просто видалити, але це має сенс лише для «StableDiffusion».

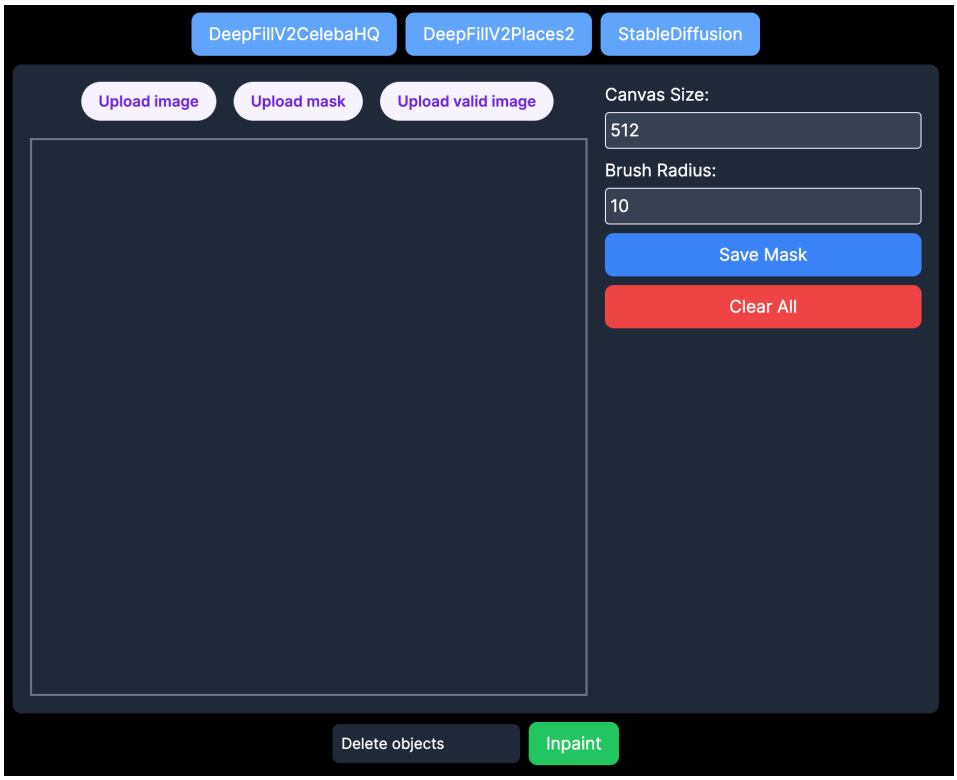

Рисунок 5.1 – Головна сторінка

Після вибору метода, він становиться більш темнішим (див рис. 5.2).

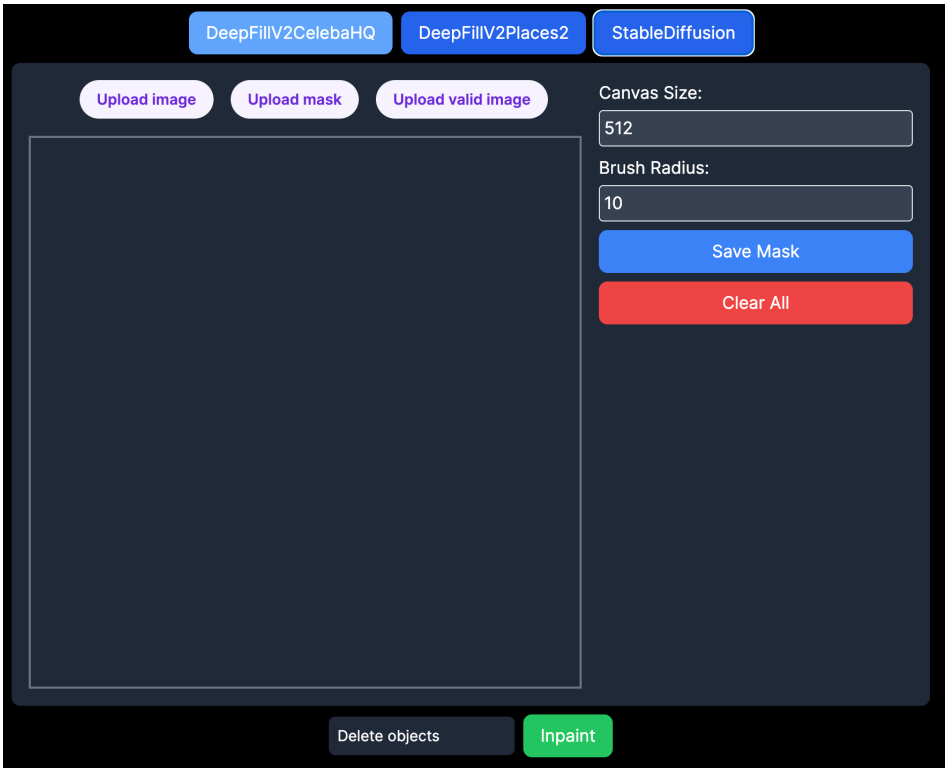

Рисунок № 5.2 – Вибрані методи

На рисунку 5.2 вибрані методи «DeepFillV2Places2» та «StableDiffusion». Після того як ми обрали методи, маємо можливість завантажити зобрження для обробки (див. рисунок 5.3).

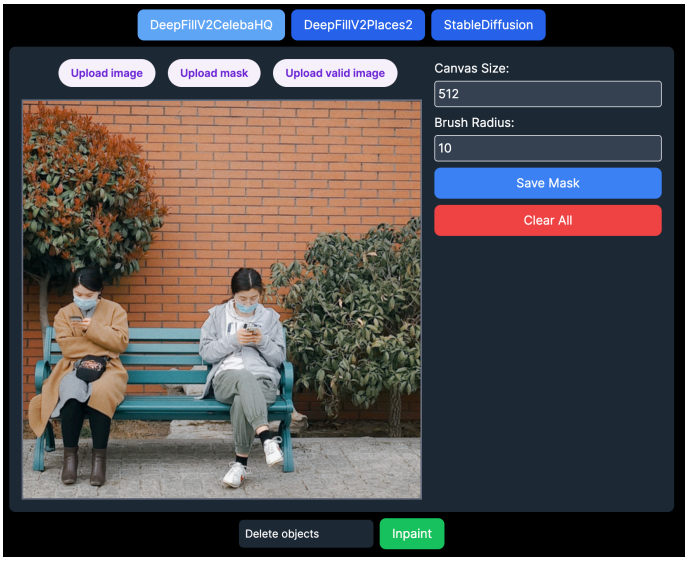

Рисунок 5.3 – Завантажене зображення для обробки

Далі ми можемо нанести маску з допомогою:

– завантажити маску;

– намалювати;

– завантажити та домлювати.

На рисунку 5.4 завантажили маску та домалювали.

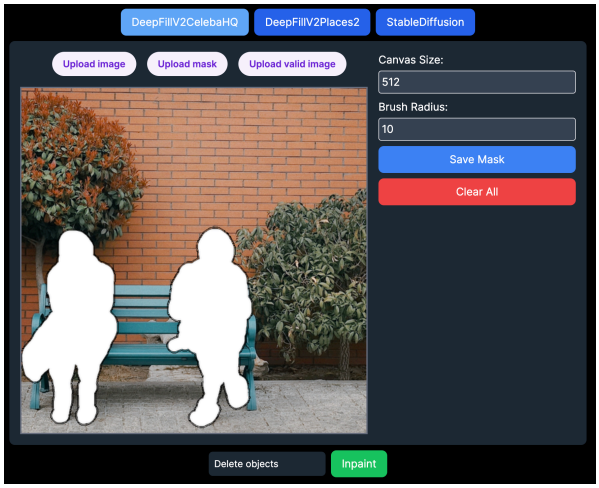

Рисунок 5.4 – Маска на зображенні

Тепер ми можемо спробувати замістити об’єкти, натиснувши кнопку «Inpaint» (див. рисунок 5.5).

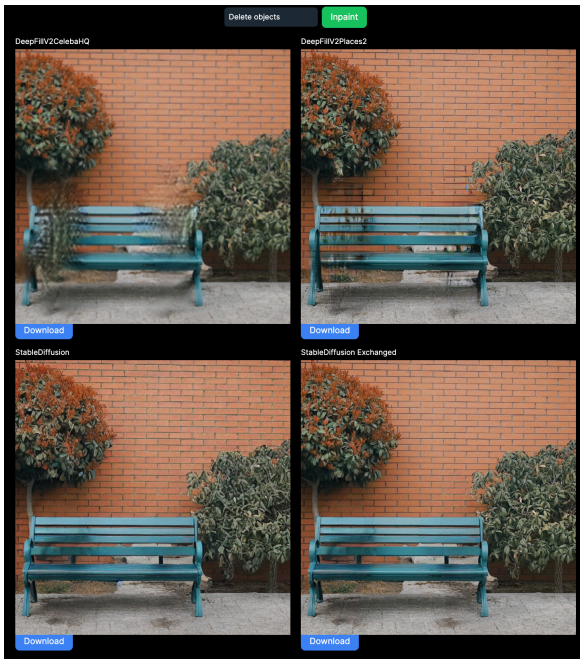

Рисунок 5.5 – Результат зміщення об’єктів

Можна побачити на рисунку 5.5 результат зміщення об’єкту, котрий складається з 4-х зображень, так як StableDiffusion метод має 2 варіанти результату.

6 ОПИС ПРОВЕДЕНИХ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

6.1 Мета експериментів

Метою експериментів було оцінити ефективність використання різних методів для реалістичного заміщення на зображені. Було важливо визначити, які моделі забезпечують найкращу якість відновлених зображень, враховуючи різні параметри та умови експериментів.

А також оцінити можливості StableDiffusion для заміщення одного об’єкта іншим.

6.2 Налаштування експериментів

Для проведення експериментів було обрано тестовий набір зображень, які включали різні типи пошкоджень. Моделі інпейнтингу, які використовувалися у дослідженні, включали DeepFillV2 та StableDiffusion, що були інтегровані в систему. Метрики для оцінки якості відновлення включали PSNR (Peak Signal-to-Noise Ratio) та SSIM (Structural Similarity Index).

6.3 Результати проведення експериментів

Для першого експерименту було взято вуличне зображення університету, намальована маска для шматочка даху, а потім спробували відновити частину (див. рисунок № 6.1). Як ми можемо побачити краще за все впорався «StableDiffusionExchange» зі власним методом заміни згенерованої частини.

Другий експеримент зробимо трохи простіше, де потрібно відновити горизонтальну частину. Як ми можемо побачити (див. рис. 6.1), то вже DeepFillV2Places показав себе краще ніж минулий раз.

Зведемо результати до таблиці 6.1.

Таблиця 6.1 – Результати двох експериментів

	PSNR	SSIM	MSE
DeepFillV2CelebaHQ	36.869	0.9892	1.1399

Кінець таблиці 6.1

	PSNR	SSIM	MSE
DeepFillV2Places2	35.7165	0.9919	1.1020
StableDiffusion	25.0433	0.7844	49.0216
StableDiffusion Exchanged	39.3201	0.9957	0.7679
Другий Експеримент			
DeepFillV2CelebaHQ	39.9344	0.9921	0.8854
DeepFillV2Places2	41.1606	0.9963	0.8657
StableDiffusion	25.1260	0.7868	49.2706
StableDiffusion Exchanged	42.2067	0.9976	0.5873

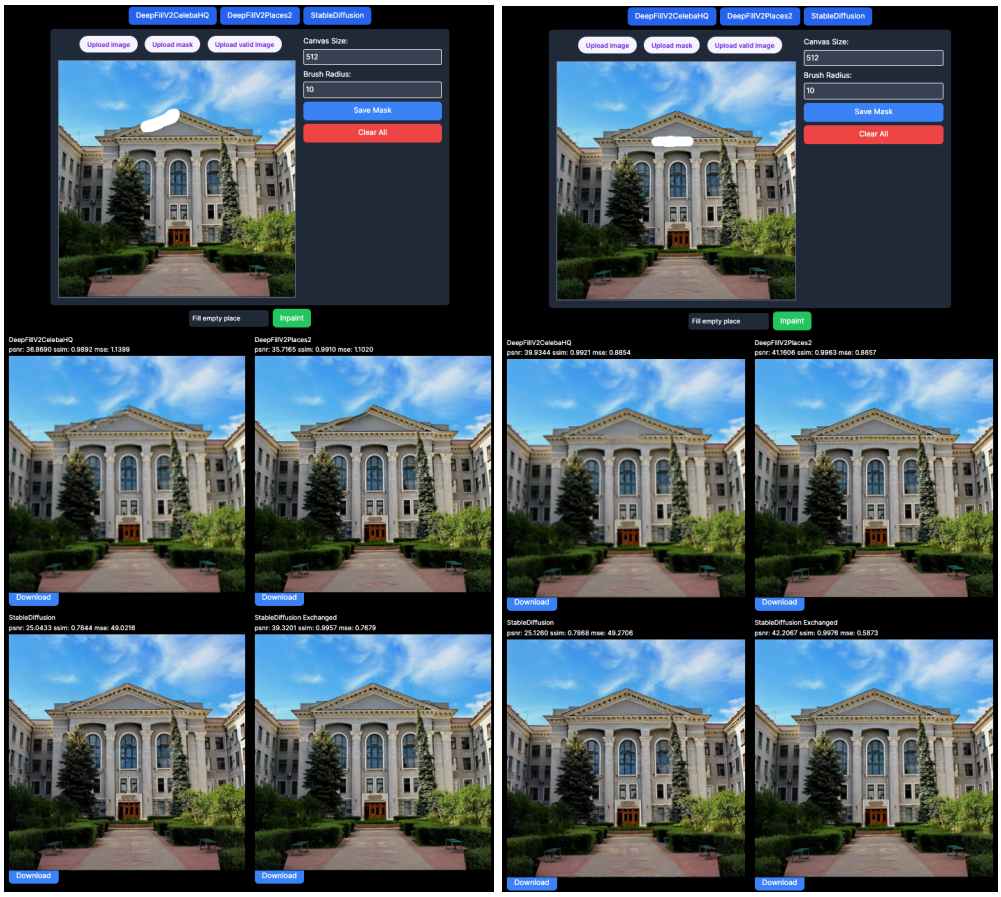

Рисунок 6.1 – Результат першого і другого експеремнта

Тепер змоделюємо трохи іншу ситуцію, припустимо, що нам потрібно видалити логотип. Візмемо зображення та маску (див. рисунок 6.2). Повторимо кроки ті що робили раніше та отримуємо результати роботи програмної системи (див. рисунок 6.3).

Рисунок 6.2 – Зображення з логотипом та маска його

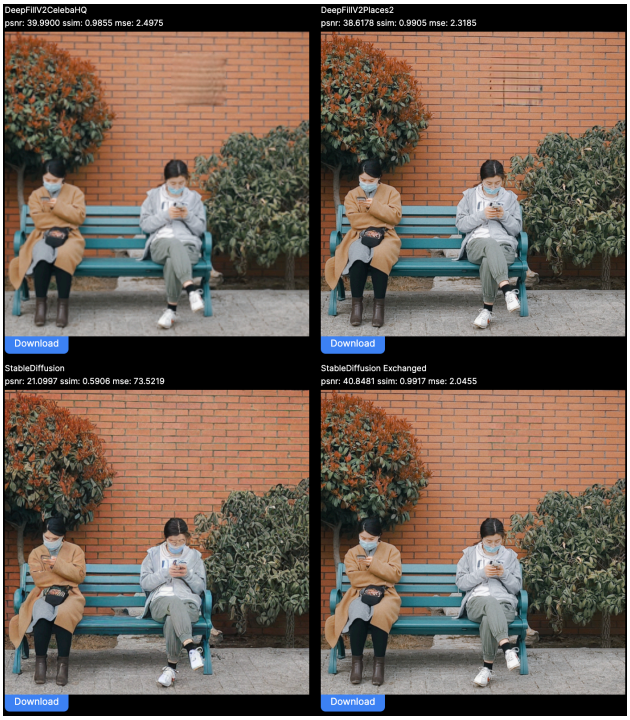

Рисунок 6.3 – Результат видалення логотипу

Зведемо результати до таблиці 6.2.

Таблиця 6.2 – Результат проведення 3-го експерименту

	PSNR	SSIM	MSE
DeepFillV2CelebaHQ	39.9900	0.9855	2.4975
DeepFillV2Places2	38.6178	0.9905	2.3185
StableDiffusion	21.0997	0.5906	73.5219
StableDiffusion Exchanged	40.8481	0.9917	2.0455

6.4 Результати проведення експериментів з заміною об’єктів

Метою даного експерименту було оцінити ефективність використання моделі Stable Diffusion для заміни об'єктів на зображеннях. Зокрема, завданням було замінити людей на інших об'єктів. Це дослідження мало на меті перевірити здатність моделі генерувати реалістичні зображення з новими об'єктами, зберігаючи загальний контекст та естетичну якість зображення.

На одному з тестових зображень було вирішено замінити двох людей на лавці на британьских котів. Модель Stable Diffusion продемонструвала здатність створювати реалістичні зображення з новими об'єктами, які гармонійно вписуються в контекст оригінального зображення. На рисунку 6.4 ми можемо побачити оригінальне зображення, маску та результати заміни.

Рисунок 6.4 – Результат заміни людей на котів

ВИСНОВКИ

У рамках магістерської роботи було проведено глибокий аналіз сучасних методів обробки зображень, з акцентом на технології реалістичного заміщення об'єктів. Основною метою дослідження була розробка та валідація програмної системи, яка демонструє ефективність використання комбінованих підходів у галузі генеративних змагальних мереж, інпейнтингу з використанням контекстуальної уваги, згорткових нейронних мереж та Stable Diffusion.

Дослідження виявило, що інтеграція методів штучного інтелекту значно покращує якість обробки зображень. Використання глибоких нейронних мереж та генеративних змагальних мереж дозволяє досягти високого рівня деталізації та реалізму при заміщенні об'єктів на зображеннях.

Проведені експерименти показали, що модель Stable Diffusion з заміною області виявилася найефективнішою для реставрації зображень, забезпечуючи високу якість та реалістичність відновлених областей. Модель DeepFillV2Places продемонструвала хороші результати у простих завданнях інпейнтингу, але не змогла досягти такого ж рівня деталізації, як Stable Diffusion. Модель DeepFillV2CelebA показала гірші результати, оскільки була навчена на зображеннях розміром 256x256 пікселів, що обмежує її здатність працювати з більшими зображеннями та складнішими сценаріями.

Крім того, модель Stable Diffusion показала себе дуже добре у завданнях заміни одних об'єктів на інші, що було продемонстровано в експериментальних дослідженнях. Це робить її особливо корисною для творчих та художніх застосувань.

Практичне застосування може бути в різноманітних областях, включаючи цифрову реставрацію зображень, редагування медіаконтенту та розробку інтерактивних мультимедійних додатків. Використання методів Stable Diffusion та DeepFillV2 відкриває широкі можливості для покращення якості та реалістичності оброблених зображень, що має значний потенціал для подальшого розвитку та комерційного впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Adaptive Image Enhancement Model for the Robot Vision System / K. Smelyakov, A. Chupryna. // Proceedings of the 14th International Scientific and Practical Conference. – 2023. – №3. – С. 246–251.

2. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing [Електронний ресурс] / C.Barnes, S. Eli, A. Finkelstein, D. Goldman // ACM Transactions on Graphics. – 2009. – Режим доступу до ресурсу: https://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR/index.php.

3. Image Completion using Planar Structure Guidance [Електронний ресурс] / J.Huang, S. Bing Kang, N. Ahuja, J. Kopf // ACM Transactions on Graphics. – 2014 – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/2601097.2601205.

4. Generative Image Inpainting with Contextual Attention [Електронний ресурс] / [J. Yu, Z. Lin, J. Yang та ін.]. – 2018. – Режим доступу до ресурсу: https://arxiv.org/abs/1801.07892.

5. Yu J. Free-Form Image Inpainting with Gated Convolution [Електронний ресурс] / J. Yu, Z. Lin, J. Yang. – 2019. – Режим доступу до ресурсу: https://arxiv.org/abs/1806.03589.

6. Newson A. Video Inpainting of Complex Scenes [Електронний ресурс] / A. Newson, A. Almansa, M. Fradet. – 2015. – Режим доступу до ресурсу: https://arxiv.org/abs/1503.05528.

7. Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play / David Foster., 2023. – (Oreilly & Associates Inc). – (2).

8. Rashid T. Make Your First GAN With PyTorch / Tariq Rashid., 2020. – 207 с.