Шифр: SYSPROXY

**НАУКОВА РОБОТА**

на тему:

“**СИСТЕМА ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ, ЗАСНОВАНА НА ВЕБ ТЕХНОЛОГІЯХ**”

***Анотація наукової роботи під шифром SYSPROXY***

Розроблено систему покращення якості зображень, що використовує клієнтський пристрій з використанням веб технологій, що є актуальним з точки зору економії системних ресурсів серверів. Використання авторської бібліотеки tracking, дозволяє зменшити кількість операцій при проведенні процедури обробки зображень. Досліджено ефективність, переваги та недоліки даної системи та її використання веб сервісами.

Метою дослідження є створення системи для покращення якості зображення на основі веб технологій мовою JavaScript для можливого її використання різними веб сервісами під різні задачі.

В роботі реалізовано наступні завдання:

1.Розроблено алгоритмічну та математичну моделі для покращення зображень, що може бути використаним для економії ресурсів та трафіку більшості веб сервісів на основі вже отриманих даних зображень.

2.Проаналізовано веб технології, що можуть бути використані при створені системи покращення зображень, їхні переваги та недоліки при експлуатації у веб застосуваннях.

3.Розроблено систему на основі запропонованого нового алгоритму та авторської бібліотеки tracking для покращення якості зображень та програмного інтерфейсу для використання його веб сервісами.

4.Протестовано та досліджено роботу розробленої системи (скрипту) з його відповідним інтерфейсом для передбачуваних поставлених для нього задач по покращенню якості зображень залежно від вхідних даних.

5.Отримано експериментальні оцінки залежності покращення візуального сприйняття зображення від часу, витраченого на його обробку.

ЗМІСТ:

[**ВСТУП** 4](#_Toc64383159)

[**1.** **ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ** 6](#_Toc64383160)

[**2.** **ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО КОРОТКИЙ ОПИС** 8](#_Toc64383161)

[**3.** **РОЗРОБКА СИСТЕМИ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ НА ОСНОВІ МЕТОДУ ПРОКСУВАННЯ ОБ’ЄКТІВ** 10](#_Toc64383162)

[**4. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ОБЧИСЛЕНЬ ТА ТЕСТУВАННЯ** 17](#_Toc64383163)

[**ВИСНОВКИ** 24](#_Toc64383164)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 25](#_Toc64383165)

# ВСТУП

Наразі щоденно збільшується кількість інформації в інтернеті, що в свою чергу збільшує кількість користувачів та сервісів що її надають [1]. Тому важливим питанням є використання меншого об’єму пам’яті та трафіку. Відповідно саме створення системи, що буде покращувати зображення саме на пристроях клієнта має великий потенціал.

Зараз дуже важко уявити сервіси в інтернеті без пошуку або категоризації, що означає можливість в певний момент часу за допомогою цих функцій користувачеві отримати досить велику частину даних, які мають спільні ознаки [2, 3]. Як, наприклад, при пошуку собаки в гугл картинках в результаті отримується сотні картинок з собаками. Також у системі Інстаграм при натисканні на картинку, що вас зацікавила, знизу погортавши видно значну кількість зображень схожих на попередню картинку. Саме під час таких дій сервіс надсилає нам значну кількість картинок чи відео, що зберігаються в певному місці, на певному або певних носіях та використовує трафік для передачі даних.

Також існує проблема, коли досить часто зображення є в поганій якості та виникає необхідність його покращення для певних цілей. У цьому випадку загалом всі дії по покращенню візуалізації зображення відбуваються на пристроях компанії, в той час, як ресурси пристроїв користувачів майже не використовуються.

Тому ідея моєї наукової роботи полягає у використанні цих ресурсів (ресурсів користувача) для потреб сервісу. Наприклад, на певний стрімінговий сервіс потрапляє відео, що певним чином розмите. Сервіс на серверній частині розбиває відео на кадри та передає ці дані на користувацький девайс, де певна система намагається покращити якість зображення. Також, можливе використання у випадку, коли користувач вже отримав певну подібну інформацію та намагається отримати новішу інформацію (наприклад, як у сервісі “ТікТок” [4]), де алгоритм аналізує, що подобається користувачеві та видає подібні результати. В такому разі сервісу немає необхідності передавати інформацію завжди в найкращій якості, достатньо відправити лише декілька зображень зі схожою інформацією у високій якості, а далі якість зображення можна покращувати на пристрої користувача.

**Метою роботи** є створення ефективної системи покращення якості зображення, що працює на клієнтських пристроях певного веб сервісу, з метою зменшення необхідних ресурсів певного сервісу.

Поставлені завдання для досягнення мети:

1.Розробка алгоритму для покращення зображень, що може бути використаним для економії ресурсів та трафіку більшості веб сервісів на основі вже отриманих даних зображень.

2.Аналіз веб технологій, що можуть бути використані при створені системи покращення зображень, їхні плюси та мінуси в експлуатації при роботі системи.

3.Розробка системи на основі запропонованого алгоритму для покращення зображень та зручного інтерфейсу для використання його веб сервісами.

4.Перевірка розробленого системи/скрипту з його інтерфейсом для передбачуваних поставлених на нього задач по покращенню якості зображень залежно від вхідних даних.

**Об’єкт дослідження:** система покращення якості зображень побудована на веб технологіях.

**Предмет дослідження:** покращення якості зображень використовуючи певні схожості в інших зображеннях використовуючи можливості веб технологій.

**Практичне значення** роботи полягає у використанні системи покращень зображень в залежності від вже отриманих даних у веб сервісах для економії коштів на ресурси певних компанії та збільшення її конкурентоспроможності. Наприклад, для загрузки половини зображень в високій якості при каталозі на сайті, а інші зображення завантажені в поганій якості та покращуються системою вже на клієнтській машині паралельно з його переглядом сторінки.

1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Система для покращення якості зображення на клієнтській частині сервісу – досить ефективний метод економії ресурсів компанії, та може використовуватись в різних цілях, тому необхідно, щоб скрипт було легко добавляти та змінювати під різні цілі.

Для цього в скрипті організовано завантаження файлів як окремий модуль, котрий дуже легко використовувати, для цього достатньо вказати посилання на файли та запрограмувати подію, при якій у випадку завершення завантаження будуть виконуватися наступні певні дії.

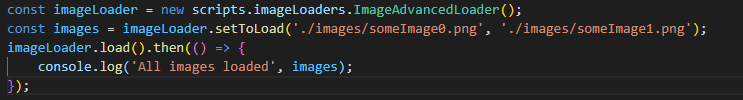


Рисунок 1.1 – Вигляд скрипту для завантаження зображень

По виходу із завантажувача ми отримуємо масив або один об’єкт, в якому міститься вся інформація про зображення, вона оновлюється, якщо зображення було завантажене.

Проте для зміни малюнка існує ще один клас, в який передається зображення як об’єкт. Саме в ньому ми можемо дізнатись інформацію про певний піксель зображення або змінити його.

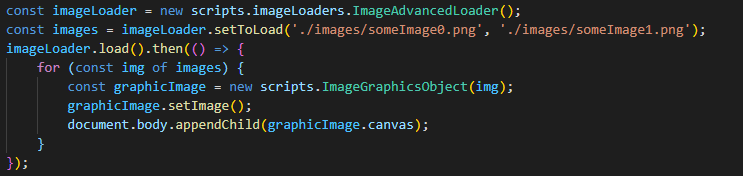


Рисунок 1.2 – Вигляд скрипту для зображення базових можливостей скрипту завантажувача та деяких дій над зображенням

Схожі скрипти використовуються в більшості вже існуючих веб сервісів. Використання скриптом для покращення якості зображення частин, що схожих на частини коду, як на рис. 1.1 та рис. 1.2, надає змогу користувачу не додавати побічні бібліотеки або дописувати код самотужки, а використовувати лише готовий функціонал. Незважаючи на це, у користувача залишається можливість й саму змінити/доробити певні частини коду, а потім використовувати їх в системі, це надає їй гнучкості. Найпростішим способом додавання до скрипту зображень для їх подальшої обробки є такий, що зображений на рис. 1.3.

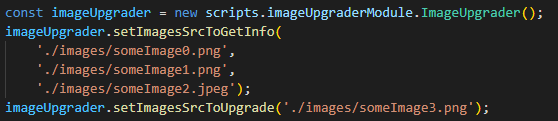


Рисунок 1.3 – Вигляд скрипту для покращення якості зображення з базовим функціоналом

Також є можливість налаштувань:

1. Допустимого часу для покращення зображення.
2. Налаштування вручну коефіцієнтів для обробки зображення, в іншому випадку вони будуть підлаштовуватись під пристрій та вхідні дані.

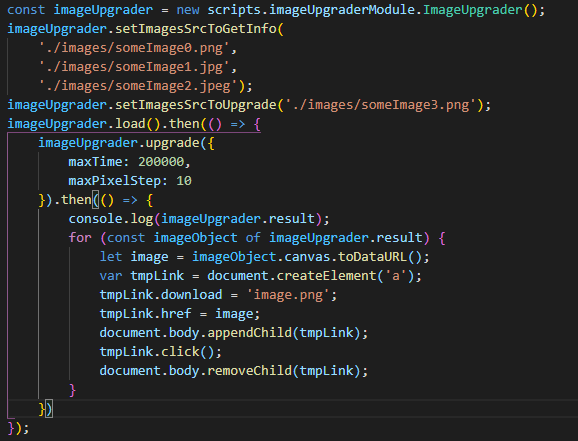


Рисунок 1.4 – Вигляд скрипту з додатковими налаштуваннями

# **2.** **ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО КОРОТКИЙ ОПИС**

Серед програмного забезпечення для створення цього скрипту можна виділити VScode, Chrome та Live server, git та gitHub.

VScode – це редактор кодів, що надає можливість писати код на javaScript [5, 6] та має низку функціоналу, з метою комфортного використання, такі як форматування коду для його кращого вигляду та дотримання певного стилю коду. Також він надає можливість тестувати код за допомогою дебагера з використанням браузера Chrome [7], та запускати певні плагіни, як наприклад Live server.

Саме за допомогою Live server було протестовано скрипт та його функціонал надав змогу використовувати модульність в javaScript скриптах [8]. Live server розгортає http сервер на певному порті і таким чином можна тестувати систему в ілюзії реального веб сервісу, де система може використовуватись.

За допомогою git відбувався контроль версій, що допомогло побачити зміни, що вже відбулись та додати певні зміни або відслідковувати і правити певні частини коду.

GitHub в свою чергу використовувався, як хмарне сховище коду для забезпечення стабільності в випадку поломки девайсу де зберігався код, тощо.

За мову програмування було взято javaScript без використання додаткових фреймворків для забезпечення швидкості роботи скрипту. Ця мова використовується в багатьох різноманітних сферах починаючи від створення сайтів і розробка програмних застосунків до серверного програмного забезпечення [9, 10]. Наразі ця мова дуже затребувана і це означає, що більшість програмного забезпечення її використовує, а тому додавання системи для покращення якості зображень не створить багато проблем. Також це основна мова програмування в веб технологіях, для яких скрипт і розроблений.

Серед бібліотек, взята бібліотека під назвою tracking [11, 12, 13] яка розроблена Ткаченко Романом (мною) для забезпечення зв’язків об’єктів за допомогою класу Proxy [14].

# **3.** РОЗРОБКА СИСТЕМИ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ НА ОСНОВІ МЕТОДУ ПРОКСУВАННЯ ОБ’ЄКТІВ

В основі системи лежить розроблена автором бібліотека tracking що допомагає робити з’єднання між декількома prediction – це об'єкт класу Prediction таким чином створюються рейтинги зв’язків на основі котрих відбувається подальше порівняння наскільки та чи інша зв’язка підходить під конкретні умови.

Використання саме такого методу дозволяє скоротити час виконання програми, адже прохід по масиву для знаходження зв’язка, до якого можна додати рейтинг займає надто багато часу, як і додавання такого зв’язка до масиву. Також це надає компактності коду, але вимагає високої технічної компетентності в розумінні коду (за рахунок високорівневої організації коду) та у відловлюванні помилок, якщо такі є.

TrackingProxy – це клас, який виловлює зміни в об’єкті та зміни в його властивостях. Якщо властивістю є об’єкт, який зроблений за классом TrackingProxy, то відновленні зміни в тому об’єкті передаються «нагору», до наступного об’єкта TrackingProxy, якщо він є його властивістю. Там присутні також певні виключення при створенні такого об’єкта. Наприклад, при створенні об’єкта, якщо в ньому вже був об’єкт класу TtrackingProxy, то його зміни не будуть відловлюватись. При відновленні такого виключення виконуються певні дії, що зазначені в функції.

Також TrackingProxy створюється для знаходження найбільш підходящого зв’язка в певній ситуації, а згодом і самого об’єкта Prediction, що краще всього підходить на певне місце в ієрархії.

Об'єктом Prediction в даній науковій роботі я називаю певний об’єкт, що зберігає інформацію про певний піксель, який згодом використовується в цілях підстановки на деяке місце. У випадку, якщо він підходить за певним коефіцієнтом та має максимальний вирахуваний рейтинг, то маємо право припустити, що для покращення зображення має бути використаний цей піксель (розроблені алгоритмічні моделі представлені на рис. 3.4, рис. 3.5).

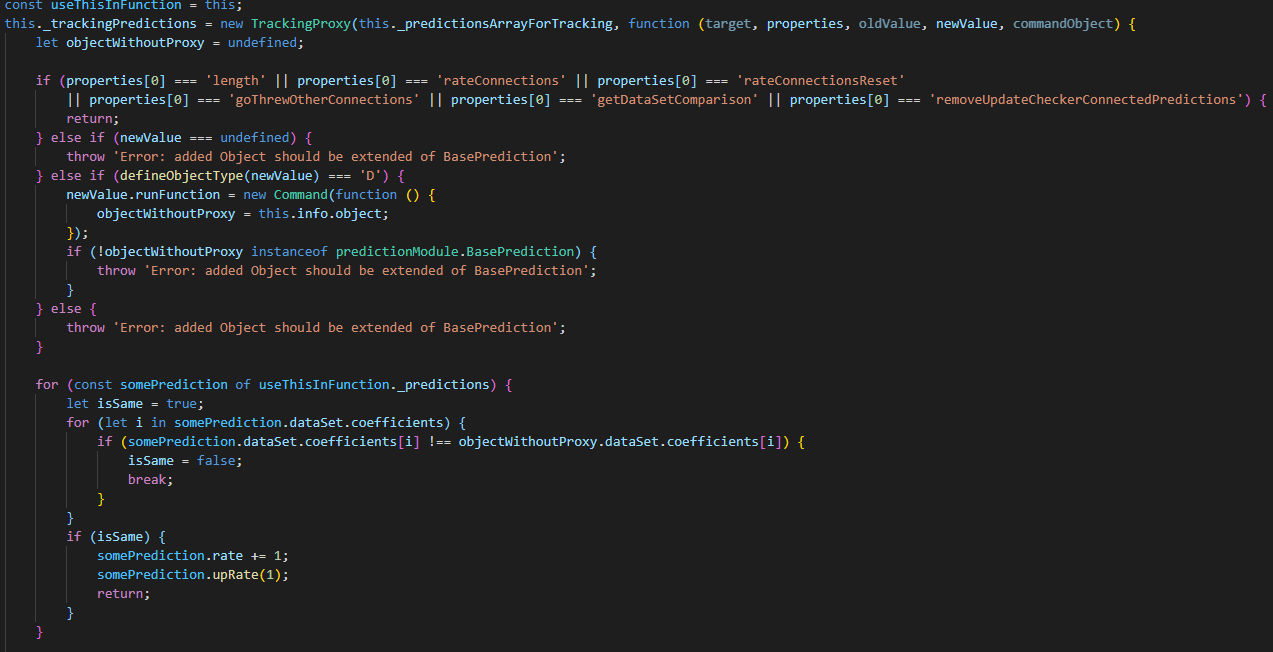


Рисунок 3.1 – Зображення частини створення об’єкту класу Tracking Proxy, що відповідає за підрахунок рейтингу з’єднання між об’єктами Prediction.

Ми також можемо виділити певні класи, що використані в цій системі. Файлова структура має пряме відношення до класів, адже кожна з папок відповідає за свій клас окрім папки tracking, що є певною бібліотекою.

1. PixelDataSet i DataSetLabel класи відповідають про збереження інформації щодо певного пікселя зображення;
2. ImageGraphicsObject клас відповідає за можливість роботи з зображенням, як з графічним об’єктом;
3. ImageLoader класи відповідають за загрузку файлів;
4. ImageObject клас зберігає об’єкт Image, що э DOM елементом;
5. ImageUpgrader клас є певною оболонкою і невід’ємною частиною системи по покращенню зображень;
6. Prediction клас містить в собі DataSet та рейтинг, ну а також для нього є оболочка створена класом TrackingProxy, що використовується для підрахунків рейтингів та зв'язків між іншими об’єктами класу Prediction;
7. PixelsPredictionChecker клас використовується, для створення додаткових об’єктів для зв’язків між об’єктами Prediction і ImageUpgrader.

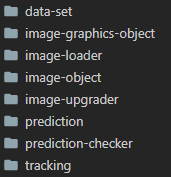


Рисунок 3.2 – Файлова структура, що є прямим відображенням короткого змісту класів, що використовуються в проекті

Певні об’єкти класів мають застосовують механізм успадкування класів, при якому додається певний функціонал. Наприклад, можливості завантажувати декілька файлів однією функцією, відслідковування моменту завантаження усіх файлів тощо. Успадкування у класах PixelDataSet і DataSetLabel відіграє функцію гнучкого (agile architecture) покращення системи, тобто можливості в майбутньому змінювати систему лише фрагментами, що робить цей процес простішим [15, 16].

В системі має бути також розглянута швидкість обробки картинки, адже вона залежати має від цілей покладених на користувача, наприклад, якщо це зображення, яке за короткий проміжок часу необхідно буде продемонструвати користувачеві, то швидкість обробки в цьому випадку має бути високою. Якщо це певний стрімінговий сервіс та в ньому розраховується, що користувач буде дивитись певний фільм протягом деякого часу, а оброблена картинка має бути відправлена на серверну частину і не буде повністю показана користувачу, або, наприклад, через годину, то час можна збільшити. На рисунку 3.3 показана діаграма, що відображає залежність точності, якості покращення зображень та час від певних коефіцієнтів.

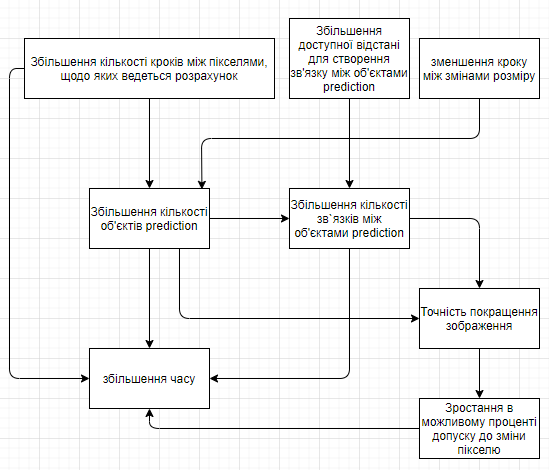


Рисунок 3.3 – Діаграма залежності часу та точності/якості покращення від коефіцієнтів

Якщо в систему не додати всі повністю коефіцієнти, то система буде намагатись отримати результат з заданими стартовими коефіцієнтами та тими, що внесли методом підбору, тобто якщо якесь з рівнянь не виконується, то перервати виконання, потім зменшити, або збільшити коефіцієнт і спробувати знову з моменту, в якому можна було перервати та змінити певний коефіцієнт.

Рейтинг, що рахується для об'єкта Prediction вираховується за формулою (1):

, (1)

де – деякий об'єкт prediction;

– об'єкта prediction, що має бути доданий;

– рейтинг деякого об'єкта prediction.

Рейтинг з’єднання між prediction вираховується відповідно (2):

, (2)

де – рейтинг зв'язка між деякими двома об'єктами prediction;

– рейтинг першого об'єкта prediction;

– рейтинг іншого об'єкта prediction.

Рейтинг для перевірки, наскільки цей чи інший prediction підходить на певне місце вираховується згідно (3), (4):

, (3)

(4)

де – значення що буде доданий до рейтингу що відображає на скільки вірогідність певної інформації про піксель, що зберігається в певному об'єкті prediction підходить на певне місце в зображені, що має бути покращено;

– cхожість інформації про певний піксель, що зберігається в об'єкті prediction на який можлива заміна в порівнянні з інформацією про певний піксель який може бути змінений;

– cхожість інформації про певний піксель, що зберігається в певному об'єкті prediction, що міститься в певному об'єкті зв'язку з об'єктом prediction на який можлива заміна в порівнянні з інформацією про певний піксель який може бути змінений;

– схожість коефіцієнтів зв'язку відносно певних двох пікселів і коефіцієнтів зв'язку між двома об'єктами prediction на які ті відносно можуть бути замінені;

– рейтинг об'єкту prediction що містить інформацію про певний піксель яка може використовуватись для заміни вже існуючого пікселю;

– рейтинг іншого об'єкту prediction(що міститься в об'єкті зв'язку) що містить інформацію про певний піксель яка може використовуватись для замінити вже існуючого пікселю;

– рейтинг зв'язку відносно певних двох пікселів і коефіцієнтів зв'язку між двома об'єктами prediction інформація з яких може бути використана для заміни певних пікселів.

Коефіцієнти – це певні 18 значень від 0 до 1 що відображають залежність від максимальних значень, так і одного від іншого.

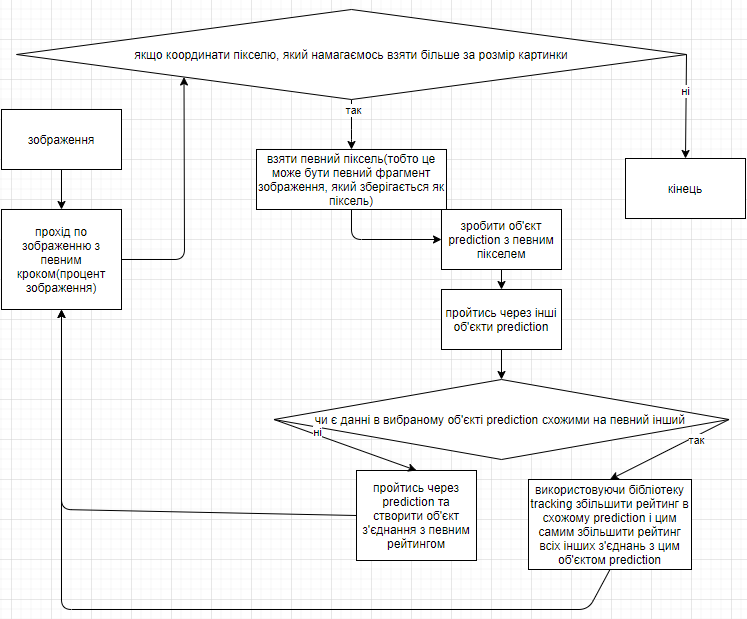


Рисунок 3.4 – Алгоритмічна модель створення зв’язків між пікселями

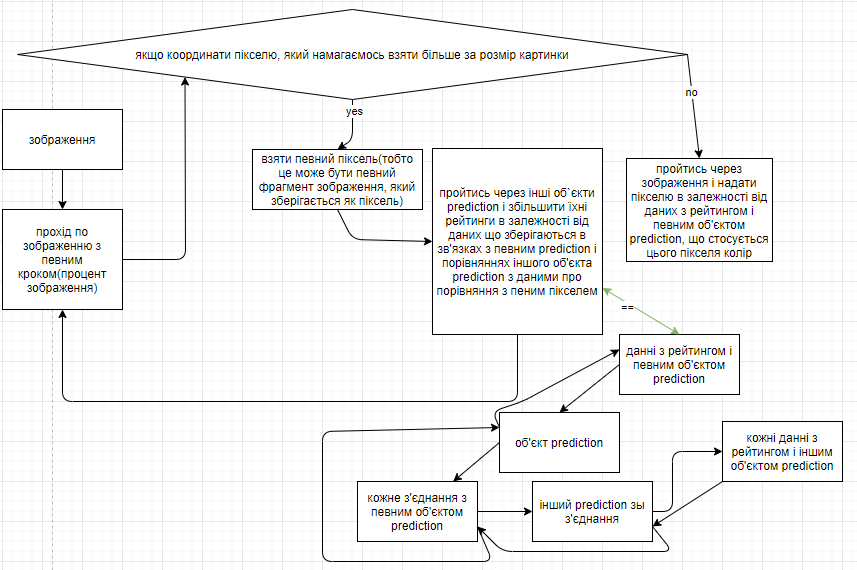


Рисунок 3.5 – Алгоритмічна модель для встановлення на піксель певного кольору

# **4. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ОБЧИСЛЕНЬ ТА ТЕСТУВАННЯ**

Візьмемо для порівняння два зображення букви А, де перше – зображення якості, якої ми намагаємось досягнути (рис. 4.1), а інше – гіршої якості (рис. 4.2).



Рисунок 4.1 – Приклад зображення, якого ми хочемо досягти



Рисунок 4.2 – Зображення гіршої якості

Проведемо процедуру візуального покращення зображення, визначивши як максимальний час 200000 мілісекунд, для розрахунку наскільки зображення подібні буде використано сервіси imgonline.com.ua [17] для перевірки за кожним пікселем, цей сервіс проходиться по кожному пікселю двох зображень і якщо вони не є однаковим – рейтинг зменшується. Також застосуємо сервіс diffchecker.com [18] для візуальної перевірки, який візуально відображає два зображення таким чином, що одне зображення знаходиться над іншим і є декілька варіантів виведення результатів (в нашому випадку відбувається повне накладання, хоча можна водити і переглядати слайдером). Результати, отримані при тестуванні є такими: ми отримали 91.75% в imgonline.com.ua у той час, як diffchecker.com видав результат, зображений на рис. 4.3. В нашому випадку, чим темнішим кольором позначена певна частина зображення, тим сильніше частина на одному зображені сходиться з іншим зображенням. Аналізуючи такі результати можна припустити, що чим більше таких місць, тим більше зображення на візуальному рівні покращене, адже певні зміни можуть бути не надто помітні для користувача, тому imgonline.com.ua може показувати менші результати, хоча було виконане певне покращення на візуальному рівні.



Рисунок 4.3 – Результат diffchecker.com при порівнянні ідеального зображення з картинкою в поганій якості

Після проведення процедури покращення зображення результат в imgonline.com.ua був менший і він дорівнює 91.58%. Проте при порівнянні в diffchecker.com отриманий результатом є рис. 4.4.

За результатами котрого можна побачити, що зображення стало більш чітким і такі елементи, як верхня частина букви стала більш чітка також по краям букви були виправленні частини.

З цього можна зробити висновок, що система також певним чином покращує якість зображення методом більш чіткого виділення певних частин зображення.



Рисунок 4.4 – Результат diffchecker.com при порівнянні ідеального зображення з картинкою отриманою в результаті обробки системою покращення зображення

Беручи ті ж вхідні дані при покращенні зображення що зображене на рис. 4.5 можна провести таке ж розширення з зображенням в хорошій якості для того, щоб відобразити зміни. Ці зміни позначені на рис. 4.6. Як можна побачити, ті частини зображення, які були покращені програмою раніше, візуально виглядають чіткіше при тому-ж заданому часі. При збільшенні часу візуальне покращення відбувається ефективніше.



Рисунок 4.5 – Зображення, що має сторонні графічні елементи (рамку)



Рисунок 4.6 – Зміни при іншому розташуванні символу на зображенні

Також проведено перевірку з дещо зміненим додатково зображенням (Рисунок 4.9). Перевірка за допомогою imgonline.com.ua показав результат 91.69% при порівнянні з ідеальним зображенням, перевірка з зміненим зображенням дав результат 91.59%

Перевірки на diffchecker.com дали результати зображені на рис. 4.7 та рис. 4.8.



Рисунок 4.7 – Результати перевірки на diffchecker.com зображення в поганій якості з певним чином зміненим зображенням



Рисунок 4.8 – Результати перевірки на diffchecker.com зображення в поганій якості з певним чином зміненим зображенням після покращення через систему



Рисунок 4.9 – Зображення в поганій якості з певними змінами

Тобто, при покращенні зображення відбуваються певні зміни і можуть бути змінені певні пікселі, що призводить до зменшення процента схожості при попіксельному порівнянні. Однак, відбувається покращення зображення при його візуальному перегляді.

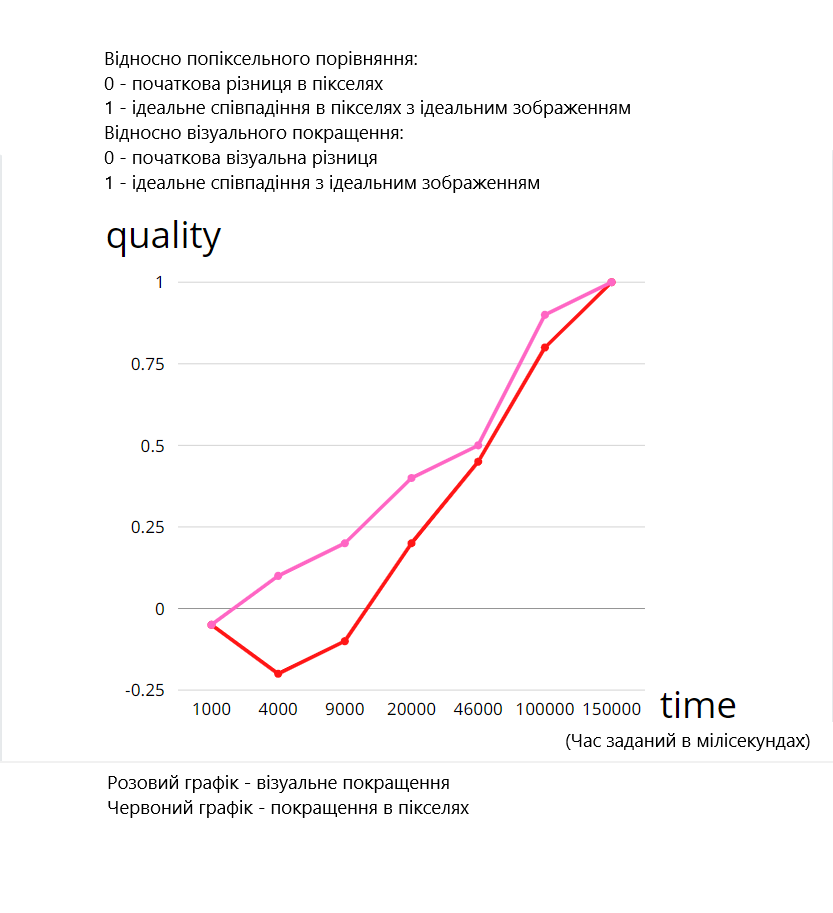


Рисунок 4.10 – Графік залежності покращення від виділеного часу

Графік залежності виглядає таким чином (рис. 4.10), він також залежить від зображення та можливості пристрою, для наступного якісного покращення потрібно більше часу з кожним досить якісним зображенням, тобто час для покращення росте експоненціально.

Проте, можна відображати готові результати до завершення процедури покращення зображення, воно може відбуватись паралельно до відображення. Це означає, що користувач потрібен мати можливість добре користуватись сервісом та не перевантажувати пристрій.

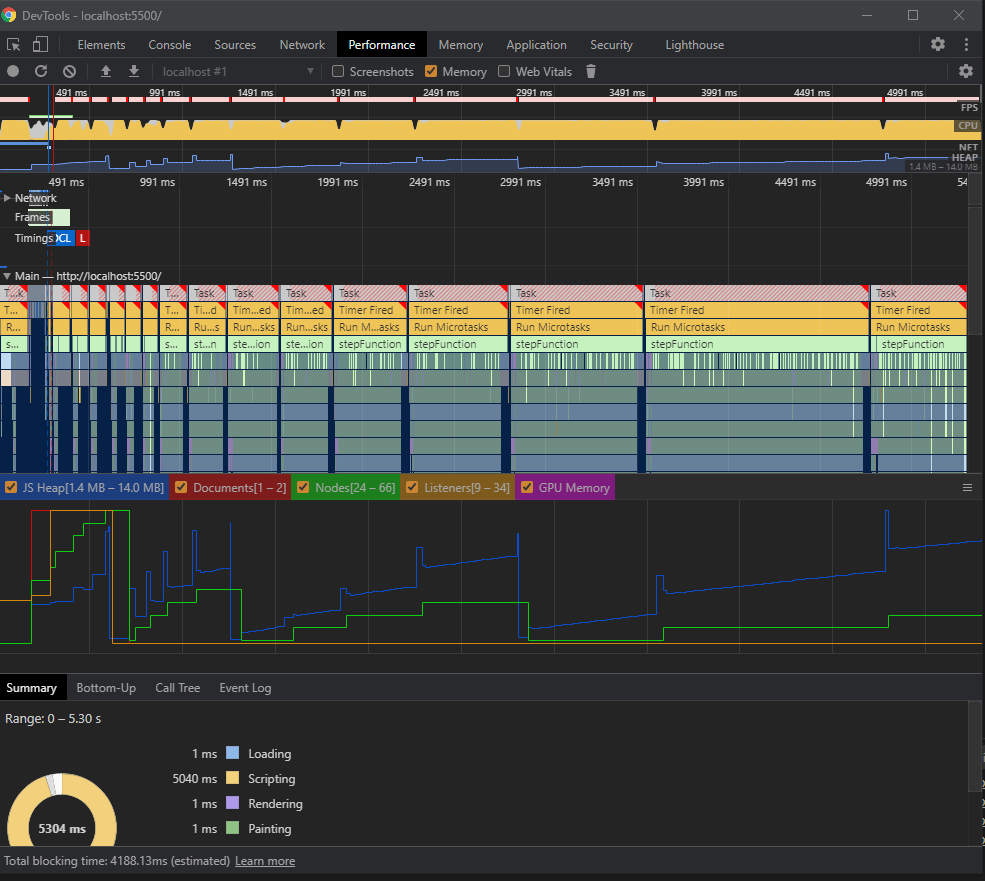


Рисунок 4.11 – Зображення вкладки performance

За графіком у вкладці «продуктивність» можна зробити висновок, що є етапи, коли відбувається певне навантаження на систему, проте воно досить швидко припиняє відпрацювання.

# **ВИСНОВКИ**

Обробка зображень на серверній частині дозволяє отримати кращі результати через можливість збереження більшого об’єму даних та можливостей в часі. Проте розподіл навантаження на користувачів сервісу, або повна передача в деяких випадках надають можливість веб сервісу краще конкурувати, адже не має потреби платити за додаткові ресурси.

С точки зору розробки забезпечення для клієнтської частини у веб технологіях, то можливості клієнтських ресурсі менші ніж серверних, адже об’єм пам’яті до використання надається малий, а також хочу відмітити, що використовувати повністю ресурси клієнтської машини досить складно, адже це погіршує роботу користувача, тому для цього потрібно знаходити компроміс.

Беручи до уваги фактори, наведені вище, слід зазначити, що система покращення візуального сприйняття зображень на клієнтській машині може бути використана і надає переваги. Створена система надає можливості обробки зображення паралельно до користувацьких дій, тобто користувач може споглядати на зображення навіть в момент його обробки, що надає змогу збільшити час покращення, адже не створює дискомфорту клієнту. Система працює на основі розробленої автором бібліотеки traсking, що використовує механізм проксування об’єктів proxy, що певним чином створюючи швидкодію та гнучкість коду, допомагає при подальшій зміні скрипту під особливі задачі. Також зазначимо, що дана система працює неоднаково на всіх пристроях, адже ресурси пристроїв відрізняються і система має підлаштовуватися під пристрій для знаходження компромісу між можливостями пристрою та якістю покращення зображення.

Наразі існують занадто примітивні системи покращень зображень на клієнтських пристроях, в той час як всі інші запропоновані системи роблять обчислення на серверних пристроях. Ця ж система надає більш інтелектуальне покращення зображень та не потребує серверних ресурсів під час обчислень.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. About Data [Електронний ресурс]. – Режим доступу: <https://aboutdata.ru/2017/04/27/volume-of-data-by-2025/>
2. Штучний інтелект для покращення якості фото [Електронний ресурс]. – Режим доступу: <https://evergreens.com.ua/ua/articles/image-enhancement-solutions.html>
3. AKVIS Enhancer. Программа для обработки фотографий, предназначенная для улучшения детализации снимка, коррекции затемненных областей, повышения резкости границ и контрастности изображения [Електронний ресурс]. – Режим доступу: <https://akvis.com/ru/enhancer/index.php>
4. How TikTok recommends videos #ForYou [Електронний ресурс]. – Режим доступу: <https://newsroom.tiktok.com/en-us/how-tiktok-recommends-videos-for-you>
5. JavaScript modules. Описання модулів js [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>
6. Ткаченко О.М., Каплун В.А. Об'єктно-орієнтоване програмування мовою Java. Навчальний посібник. - Вінниця: ВНТУ, 2006. - 106 с.
7. Стаття по використання Chrome DevTools [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/company/simbirsoft/blog/337116/>
8. ECMAScript® 2021 Language Specification [Електронний ресурс]. – Режим доступу: <https://tc39.es/ecma262/>
9. Інформація про елемент canvas в веб браузерному програмуванні [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial>
10. Описання використання Promise в мові програмування javaScript та підтримка його відносно браузерів [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>
11. Бібліотека tracking [Електронний ресурс]. – Режим доступу: <https://github.com/RomaTk/tracking>
12. R. Ponomarenko, R. Tkachenko. Method of Processing Complex Objects Based on ObjectOriented Proxy System. Satellite conference processing: “VII International conference “Information Technology and Interactions” (IT&I-2020) December 02-04”, 2020, pp. 371–373.
13. Пономаренко Р.М., Ткаченко Р.А. Об’єктно-орієнтована Proxy-система для обробки складних об’єктів із неоднорідною структурою// Матер. VII Міжнар. наук.-техн. Internet-конф. «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами», 26 листоп. 2020 р. – К: НУХТ, 2020. – 264-266 с. ISBN 978-966-612-244-8.
14. Proxy и Reflect. Опис системи використання Proxy в js [Електронний ресурс]. – Режим доступу: <https://learn.javascript.ru/proxy>
15. Гамма Э./ Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2004. – 366 с.
16. Lea D. Concurrent programming in Java: design principles and patterns / D. Lea – Addison-Wesley Professional. 2000. – 411p.
17. Сервіс для отримання інформації про подібність зображень imgonline [Електронний ресурс]. – Режим доступу: https://www.imgonline.com.ua/eng/
18. Сервіс для отримання інформації про подібність зображень DiffChecker [Електронний ресурс]. – Режим доступу: <https://www.diffchecker.com/image-diff/>

**ДОДАТОК. КОД ПРОГРАМИ**

index.js

import scripts from './srcipts/index.js';

const imageUpgrader = new scripts.imageUpgraderModule.ImageUpgrader();

imageUpgrader.setImagesSrcToGetInfo(

'./images/someImage0.png',

'./images/someImage1.jpg',

'./images/someImage2.jpeg');

imageUpgrader.setImagesSrcToUpgrade('./images/someImage3.png');

imageUpgrader.load().then(() => {

imageUpgrader.upgrade({

maxTime: 200000,

maxPixelStep: 10

}).then(() => {

console.log(imageUpgrader.result);

for (const imageObject of imageUpgrader.result) {

let image = imageObject.canvas.toDataURL();

var tmpLink = document.createElement('a');

tmpLink.download = 'image.png';

tmpLink.href = image;

document.body.appendChild(tmpLink);

tmpLink.click();

document.body.removeChild(tmpLink);

}

})

});

srcipts\index.js

import imageLoaders from './image-loader/index.js';

import imageObjects from './image-object/index.js';

import ImageGraphicsObject from './image-graphics-object/index.js';

import dataSets from './data-set/index.js';

import predictionModule from './prediction/index.js';

import predictionCheckerModule from './prediction-checker/index.js';

import imageUpgraderModule from './image-upgrader/index.js';

export default {

imageLoaders: imageLoaders,

imageObjects: imageObjects,

ImageGraphicsObject: ImageGraphicsObject,

dataSets: dataSets,

predictionModule: predictionModule,

predictionCheckerModule: predictionCheckerModule,

imageUpgraderModule: imageUpgraderModule

}

srcipts\prediction-checker\index.js

import PixelsPredictionChecker from './pixel-prediction-checker-optimaze.js';

export default {

PixelsPredictionChecker: PixelsPredictionChecker

}

srcipts\prediction-checker\pixel-prediction-checker-optimaze.js

import predictionModule from '../prediction/index.js';

import {TrackingProxy, defineObjectType, Command} from '../tracking/index.js';

export default class PixelsPredictionChecker {

constructor() {

this.\_predictions = [];

this.\_predictionsArrayForTracking = [];

const useThisInFunction = this;

this.\_trackingPredictions = new TrackingProxy(this.\_predictionsArrayForTracking, function (target, properties, oldValue, newValue, commandObject) {

let objectWithoutProxy = undefined;

if (properties[0] === 'length' || properties[0] === 'rateConnections' || properties[0] === 'rateConnectionsReset'

|| properties[0] === 'goThrewOtherConnections' || properties[0] === 'getDataSetComparison' || properties[0] === 'removeUpdateCheckerConnectedPredictions') {

return;

} else if (newValue === undefined) {

throw 'Error: added Object should be extended of BasePrediction';

} else if (defineObjectType(newValue) === 'D') {

newValue.runFunction = new Command(function () {

objectWithoutProxy = this.info.object;

});

if (!objectWithoutProxy instanceof predictionModule.BasePrediction) {

throw 'Error: added Object should be extended of BasePrediction';

}

} else {

throw 'Error: added Object should be extended of BasePrediction';

}

for (const somePrediction of useThisInFunction.\_predictions) {

let isSame = true;

for (let i in somePrediction.dataSet.coefficients) {

if (somePrediction.dataSet.coefficients[i] !== objectWithoutProxy.dataSet.coefficients[i]) {

isSame = false;

break;

}

}

if (isSame) {

somePrediction.rate += 1;

somePrediction.upRate(1);

return;

}

}

const coefficientsLength = objectWithoutProxy.dataSet.coefficients.length;

for (const somePrediction of useThisInFunction.\_predictions) {

let isNear = false;

for (let i = 1; i < 5; i += 1) {

if (Math.abs(somePrediction.dataSet.coefficients[coefficientsLength - i] - objectWithoutProxy.dataSet.coefficients[coefficientsLength - i]) <= 0) {

isNear = true

break;

}

}

if (!isNear) {

return;

}

const compareCoefficients = [];

for (const data0 of somePrediction.dataSet.coefficients) {

for (const data1 of objectWithoutProxy.dataSet.coefficients) {

compareCoefficients.push((data0 - data1) / 2);

}

}

const connectedPredictions = {

predictions: new TrackingProxy([], function (target, properties, oldValue, newValue, commandObject) {

if (properties[0] === 'rateConnections') {

connectedPredictions.rate += newValue;

} else if (properties[0] === 'rateConnectionsReset') {

if (properties[1] == '0') {

connectedPredictions.rate = connectedPredictions.predictions[0].rate;

} else {

connectedPredictions.rate += connectedPredictions.predictions[1].rate;

}

} else if (properties[0] === 'goThrewOtherConnections') {

newValue.connectedPredictionsRate = connectedPredictions.rate;

newValue.compareCoefficients = connectedPredictions.compareCoefficients;

if (newValue.predictions[0].prediction.trackingProxy == connectedPredictions.predictions[0]) {

if (connectedPredictions.predictions[1].trackingProxyForSomeReason) {

connectedPredictions.predictions[1].trackingProxyForSomeReason.getDataSetComparison = newValue;

}

} else if (newValue.predictions[0].prediction.trackingProxy == connectedPredictions.predictions[1]) {

if (connectedPredictions.predictions[0].trackingProxyForSomeReason) {

connectedPredictions.predictions[0].trackingProxyForSomeReason.getDataSetComparison = newValue;

}

}

}

}),

compareCoefficients: compareCoefficients,

rate: 0

}

connectedPredictions.predictions.push(somePrediction.trackingProxy);

connectedPredictions.predictions.push(objectWithoutProxy.trackingProxy);

}

useThisInFunction.\_predictions.push(objectWithoutProxy);

});

}

addDataSet(dataSet) {

const newPrediction = new predictionModule.BasePrediction(dataSet);

newPrediction.addPixelDataSet(dataSet);

this.\_trackingPredictions.push(newPrediction.trackingProxy);

if (this.\_predictions.length != this.\_predictionsArrayForTracking.length) {

this.\_predictionsArrayForTracking.pop();

}

}

upgradeRate() {

for (let i = 0; i < this.\_predictions.length; i += 1) {

for (let j = i + 1; j < this.\_predictions.length; j += 1) {

const somePrediction0 = this.\_predictions[i];

const somePrediction1 = this.\_predictions[j];

let samePercentage = 0;

for (let i in somePrediction0.dataSet.coefficients) {

samePercentage += (1 - Math.abs(somePrediction0.dataSet.coefficients[i] - somePrediction1.dataSet.coefficients[i]) / 2);

}

somePrediction0.upRate(samePercentage / somePrediction0.dataSet.coefficients.length);

}

}

}

resetRate() {

for (const somePrediction of this.\_predictions) {

somePrediction.trackingProxy.resetRate();

}

}

set predictions(value) {

this.\_predictions = value;

}

get predictions() {

return this.\_predictions;

}

}

srcipts\prediction\index.js

import BasePrediction from './base-prediction.js';

export default {

BasePrediction: BasePrediction

}

srcipts\image-upgrader\image-upgrader.js

import imageLoaders from '../image-loader/index.js';

import predictionCheckerModule from '../prediction-checker/index.js';

import ImageGraphicsObject from '../image-graphics-object/index.js';

import dataSetModule from '../data-set/index.js';

import predictionModule from '../prediction/index.js';

import {TrackingProxy} from '../tracking/index.js';

export default class ImageUpgrader {

constructor() {

this.imageLoader = new imageLoaders.ImageAdvancedLoader();

this.predictionChecker = new predictionCheckerModule.PixelsPredictionChecker();

this.imagesToGetInfo = [];

this.imagesToUpgrade = [];

this.images = [];

this.\_step = 2;

this.\_result = [];

this.maxTime = 10000;

}

setImagesSrcToGetInfo(...args) {

const images = this.imageLoader.setToLoad(...args);

if (Array.isArray(images)) {

this.images = [...this.images, ...images];

this.imagesToGetInfo = [...this.imagesToGetInfo, ...images];

} else {

this.images.push(images);

this.imagesToGetInfo.push(images);

}

}

resetRate() {

for (const prediction of this.predictionChecker.predictions) {

prediction.removeUpdateCheckerConnectedPredictions = undefined;

}

}

setImagesSrcToUpgrade(...args) {

const images = this.imageLoader.setToLoad(...args);

if (Array.isArray(images)) {

this.images = [...this.images, ...images];

this.imagesToUpgrade = [...this.imagesToUpgrade, ...images];

} else {

this.images.push(images);

this.imagesToUpgrade.push(images);

}

this.predictionChecker.resetRate();

this.resetRate();

}

upgrade(config) {

if (!config) {

config = {};

}

const startTime = Date.now();

const maxTime = config.maxTime || this.maxTime;

let maxWidth = 0;

let maxHeight = 0;

for (const image of this.images) {

if (maxWidth < image.width) {

maxWidth = image.width;

}

if (maxHeight < image.height) {

maxHeight = image.height;

}

}

const originalGraphicsToUpdate = [];

for (const image of this.imagesToUpgrade) {

const graphicImage = new ImageGraphicsObject(image);

originalGraphicsToUpdate.push(graphicImage);

}

const updatedGraphics = [];

for (const image of this.imagesToUpgrade) {

const graphicImage = new ImageGraphicsObject(image);

updatedGraphics.push(graphicImage);

}

this.result = updatedGraphics;

let resolveWhenAllDone = undefined;

const promiseToReturn = new Promise((resolve, reject) => {

resolveWhenAllDone = resolve;

});

const maxPixelSize = Math.max(maxWidth, maxHeight);

let stepDoneResolve = undefined;

let stepDonePromise = new Promise((resolve, reject) => {

stepDoneResolve = resolve;

stepDoneResolve();

})

if (maxTime < Date.now() - startTime) {

resolveWhenAllDone();

return stepDonePromise;

}

const maxPixelStep = config.maxPixelStep || maxPixelSize;

let stepInPixels = 1;

let pixelSize = maxPixelSize;

let stepFunction = () => {

this.\_step = Math.ceil(stepInPixels);

this.predictionChecker = new predictionCheckerModule.PixelsPredictionChecker();

const graphicImagesToGetInfo = [];

for (const image of this.imagesToGetInfo) {

const scaleX = 1 - (pixelSize / image.width);

const scaleY = 1 - (pixelSize / image.height);

const graphicImage = new ImageGraphicsObject(image);

if (scaleX > scaleY) {

graphicImage.scale(scaleX, scaleX);

} else {

graphicImage.scale(scaleY, scaleY);

}

graphicImagesToGetInfo.push(graphicImage);

}

const graphicImagesToUpgrade = [];

for (const image of this.imagesToUpgrade) {

const scaleX = 1 - (pixelSize / image.width);

const scaleY = 1 - (pixelSize / image.height);

const graphicImage = new ImageGraphicsObject(image);

if (scaleX > scaleY) {

graphicImage.scale(scaleX, scaleX);

} else {

graphicImage.scale(scaleY, scaleY);

}

graphicImagesToUpgrade.push(graphicImage);

}

for (let i in graphicImagesToGetInfo) {

const imageGraphics = graphicImagesToGetInfo[i];

const stepW = this.\_step;

const stepH = this.\_step;

for (let w = 0; w < imageGraphics.width; w += stepW) {

for (let h = 0; h < imageGraphics.height; h += stepH) {

this.predictionChecker.addDataSet(new dataSetModule.PixelDataSet(...imageGraphics.getPixel(w, h), w, h, imageGraphics.width, imageGraphics.height));

}

}

}

if (maxTime < Date.now() - startTime || this.\_step > pixelSize) {

resolveWhenAllDone();

return;

}

this.predictionChecker.upgradeRate();

for (let i in graphicImagesToUpgrade) {

const resultPredictions = {};

const imageGraphics = graphicImagesToUpgrade[i];

const stepW = this.\_step;

const stepH = this.\_step;

for (let w = 0; w < imageGraphics.width; w += stepW) {

for (let h = 0; h < imageGraphics.height; h += stepH) {

const startTimeForFor = Date.now();

const lineOfPredictions = [];

if (!resultPredictions[w + '-' + h]) {

resultPredictions[w + '-' + h] = lineOfPredictions;

}

const dataSetOfThisPixel = new dataSetModule.PixelDataSet(...imageGraphics.getPixel(w, h), w, h, imageGraphics.width, imageGraphics.height);

for (const prediction of this.predictionChecker.predictions) {

const predictionDataSetCoefficients = prediction.dataSet.coefficients;

let sameColorPersentage = 0;

for (let i = 0; i < predictionDataSetCoefficients.length - 4; i += 1) {

sameColorPersentage += 1 - Math.abs(predictionDataSetCoefficients[i] - dataSetOfThisPixel.coefficients[i]) / 2;

}

sameColorPersentage = sameColorPersentage / (predictionDataSetCoefficients.length - 4);

if (sameColorPersentage > 0.5) {

lineOfPredictions.push(this.compare(prediction, dataSetOfThisPixel));

}

if (maxTime / (pixelSize - stepInPixels) < (Date.now() - startTimeForFor) \* w \* h) {

stepInPixels += 1;

stepDonePromise = stepDonePromise.then(stepFunction);

return;

}

}

}

}

for (let w = 0; w < imageGraphics.width; w += stepW) {

for (let h = 0; h < imageGraphics.height; h += stepH) {

let connectedPredictionsWithMaxRate = null;

for (const connectedPredictions of resultPredictions[w + '-' + h]) {

if (connectedPredictionsWithMaxRate === null) {

connectedPredictionsWithMaxRate = connectedPredictions;

} else if (connectedPredictionsWithMaxRate.rate < connectedPredictions.rate) {

connectedPredictionsWithMaxRate = connectedPredictions;

}

}

resultPredictions[w + '-' + h] = connectedPredictionsWithMaxRate;

}

}

const thisUpdatedGraphics = updatedGraphics[i];

const thisOriginalGraphicsToUpdate = originalGraphicsToUpdate[i];

const stepInW = thisUpdatedGraphics.width / imageGraphics.width;

const stepInH = thisUpdatedGraphics.height / imageGraphics.height;

for (let w = 0; w < imageGraphics.width; w += stepW) {

const realW = w \* stepInW;

for (let h = 0; h < imageGraphics.height; h += stepH) {

if (resultPredictions[w + '-' + h]) {

const startTimeForFor = Date.now();

const realH = h \* stepInH;

const colorList = resultPredictions[w + '-' + h].predictions[0].prediction.dataSet.colorList;

let changed = false;

for (let i = 0; i < 4; i += 1) {

if (imageGraphics.getPixel(w, h)[i] != colorList[i]) {

changed = true;

}

}

if (changed) {

for (let wToAdd = realW; wToAdd < realW + stepInW; wToAdd += 1) {

for (let hToAdd = realH; hToAdd < realH + stepInH; hToAdd += 1) {

let sameColorPersentage = 0;

for (let i = 0; i < colorList.length; i += 1) {

sameColorPersentage += 1 - Math.abs(thisOriginalGraphicsToUpdate.getPixel(wToAdd, hToAdd)[i] / 255 - colorList[i] / 255) / 2;

}

sameColorPersentage = sameColorPersentage / (colorList.length);

if (sameColorPersentage > 0.9) {

const colorListToAdd = [0, 0, 0, 0];

for (let i = 0; i < 4; i += 1) {

colorListToAdd[i] = Math.floor(thisUpdatedGraphics.getPixel(wToAdd, hToAdd)[i] \* (1 - sameColorPersentage) + colorList[i] \* sameColorPersentage);

}

thisUpdatedGraphics.putPixel(wToAdd, hToAdd, ...colorListToAdd);

}

}

}

}

if (maxTime / (pixelSize - stepInPixels) < (Date.now() - startTimeForFor) \* w \* h) {

stepInPixels += 1;

stepDonePromise = stepDonePromise.then(stepFunction);

return;

}

}

}

}

//document.body.appendChild(thisUpdatedGraphics.canvas);

pixelSize -= 1;

if (pixelSize > 0) {

setTimeout(() => {

stepDonePromise = stepDonePromise.then(stepFunction);

}, 10);

}

}

};

stepDonePromise = stepDonePromise.then(stepFunction);

return promiseToReturn;

}

compare(prediction, dataSet) {

let samePercentage = 0;

for (let i in prediction.dataSet.coefficients) {

samePercentage += (1 - Math.abs(prediction.dataSet.coefficients[i] - dataSet.coefficients[i]) / 2);

}

samePercentage = samePercentage / dataSet.coefficients.length;

const connectedPredictions = {

predictions: new TrackingProxy([], function (target, properties, oldValue, newValue, commandObject) {

if (properties[0] === 'goThrewOtherConnections') {

} else if (properties[0] === 'getDataSetComparison') {

const compareCoefficients = [];

for (const data0 of newValue.dataSet.coefficients) {

for (const data1 of dataSet.coefficients) {

compareCoefficients.push((data0 - data1) / 2);

}

}

let samePercentageCompareCoefficients = 0;

for (let i in compareCoefficients) {

samePercentageCompareCoefficients += (1 - Math.abs(compareCoefficients[i] - connectedPredictions.compareCoefficients[i]) / 2);

}

samePercentageCompareCoefficients = samePercentageCompareCoefficients / compareCoefficients.length;

let toRate = samePercentage \* newValue.samePercentage \* samePercentageCompareCoefficients \* (connectedPredictions.connectedPredictionsRate + prediction.rate + newValue.predictions[0].prediction.rate);

newValue.rate += toRate / 3;

} else if (properties[0] === 'removeUpdateCheckerConnectedPredictions') {

connectedPredictions.predictions[0] = undefined;

}

}),

secondPredictionRate: 0,

connectedPredictionsRate: 0,

compareCoefficients: undefined,

dataSet: dataSet,

samePercentage: samePercentage,

rate: 0

}

prediction.trackingProxyForSomeReason = new TrackingProxy({prediction});

connectedPredictions.predictions.push(prediction.trackingProxyForSomeReason);

prediction.trackingProxy.goThrewOtherConnections = connectedPredictions;

return connectedPredictions;

}

load() {

return this.imageLoader.load();

}

set result(value) {

this.\_result = value;

}

get result() {

return this.\_result;

}

}

srcipts\image-upgrader\index.js

import ImageUpgrader from './image-upgrader.js';

export default {

ImageUpgrader: ImageUpgrader

}

srcipts\image-object\index.js

import ImageObject from './image-class.js';

import ImageObjectWithLoader from './image-class-loader.js';

export default {

ImageObject: ImageObject,

ImageObjectWithLoader: ImageObjectWithLoader

};

srcipts\image-object\image-class.js

export default class ImageObject {

constructor() {

this.\_src = undefined;

this.loaded = false;

this.\_width = undefined;

this.\_height = undefined;

this.\_domElement = new Image();

this.\_domElement.onload = ImageObject.onloadImage.bind(this);

this.\_domElement.onerror = ImageObject.onerrorImage.bind(this);

}

static onloadImage() {

this.width = this.\_domElement.naturalWidth;

this.height = this.\_domElement.naturalHeight;

this.loaded = true;

console.log('The ' + this.src + ' loaded');

}

static onerrorImage(error) {

this.\_width = undefined;

this.\_height = undefined;

this.loaded = false;

console.error('The ' + this.src + ' not loaded');

}

set src(value) {

if (this.\_src !== value) {

this.loaded = false;

this.\_src = value;

this.\_domElement.src = this.\_src;

}

}

get src() {

return this.\_src;

}

set width(value) {

this.\_width = value;

}

get width() {

return this.\_width;

}

set height(value) {

this.\_heigth = value;

}

get height() {

return this.\_heigth;

}

set loaded(value) {

this.\_loaded = value;

}

get loaded() {

return this.\_loaded;

}

set domElement(value) {

this.\_domElement = value;

}

get domElement() {

return this.\_domElement;

}

}

srcipts\image-object\image-class-loader.js

import ImageObjectToExtend from './image-class.js';

export default class ImageObject extends ImageObjectToExtend {

constructor() {

super();

this.\_resolveLoading = undefined;

this.\_rejectLoading = undefined;

this.\_domElement.onload = ImageObject.onloadImage.bind(this);

this.\_domElement.onerror = ImageObject.onerrorImage.bind(this);

}

static onloadImage() {

ImageObjectToExtend.onloadImage.bind(this)();

this.\_resolveLoading();

}

static onerrorImage() {

ImageObjectToExtend.onerrorImage.bind(this)();

this.\_rejectLoading();

}

set src(value) {

if (this.\_src !== value) {

this.loaded = false;

this.\_src = value;

}

}

get src() {

return this.\_src;

}

load() {

const loadPromise = new Promise((resolve, reject) => {

this.\_resolveLoading = resolve;

this.\_rejectLoading = reject;

this.\_domElement.src = this.\_src;

});

return loadPromise;

}

}

srcipts\image-loader\index.js

import ImageLoader from './image-loader.js';

import ImageAdvancedLoader from './image-advanced-loader.js';

export default {

ImageLoader: ImageLoader,

ImageAdvancedLoader: ImageAdvancedLoader

}

srcipts\image-loader\image-loader.js

import imageObjects from '../image-object/index.js'

export default class ImageLoader {

constructor() {

this.\_notLoadedImages = {};

this.\_loadedImages = {};

this.\_cannotLoadImages = {};

}

uploadImage(imageSrc = '') {

const image = new imageObjects.ImageObjectWithLoader();

image.src = imageSrc;

this.\_notLoadedImages[image.src] = (this.\_notLoadedImages[image.src] || 0) + 1;

const loadImagePomise = image.load();

loadImagePomise.then(() => {

this.\_notLoadedImages[image.src] = (this.\_notLoadedImages[image.src] || 0) - 1;

this.\_loadedImages[image.src] = (this.\_loadedImages[image.src] || 0) + 1;

}).catch(() => {

this.\_cannotLoadImages[image.src] = (this.\_cannotLoadImages[image.src] || 0) + 1;

});

return image;

}

}

srcipts\image-loader\image-advanced-loader.js

import ImageLoaderToExtend from './image-loader.js'

import imageObjects from '../image-object/index.js'

export default class ImageLoader extends ImageLoaderToExtend {

constructor() {

super();

this.\_notStartedLoading = [];

this.\_resolveLoaded = undefined;

this.\_rejectLoaded = undefined;

this.countImages = 0;

}

setToLoad(...args) {

const arrayOfImageSrcs = [];

const arrayToReturn = [];

if (args.length === 1) {

if (Array.isArray(args[0])) {

Object.assign(arrayOfImageSrcs, args[0]);

} else {

arrayOfImageSrcs.push(args[0]);

}

} else if (args.length > 1) {

Object.assign(arrayOfImageSrcs, args);

} else {

throw 'Error: upload image function should have at least one parameter';

}

for (const imageSrc of arrayOfImageSrcs) {

if (!(typeof (args[0]) === 'number' || typeof (args[0]) === 'string' || typeof (args[0]) === 'symbol' || typeof (args[0]) === 'bigint')) {

throw 'Error: image src can not be that type';

}

const image = new imageObjects.ImageObjectWithLoader();

image.src = imageSrc;

this.\_notLoadedImages[image.src] = (this.\_notLoadedImages[image.src] || 0) + 1;

this.\_notStartedLoading.push(image);

this.countImages += 1;

arrayToReturn.push(image);

}

if (arrayToReturn.length > 1) {

return arrayToReturn;

} else {

return arrayToReturn[0];

}

}

uploadImage(...args) {

const arrayOfImageSrcs = [];

const arrayToReturn = [];

if (args.length === 1) {

if (Array.isArray(args[0])) {

Object.assign(arrayOfImageSrcs, args[0]);

} else {

arrayOfImageSrcs.push(args[0]);

}

} else if (args.length > 1) {

Object.assign(arrayOfImageSrcs, args);

} else {

throw 'Error: upload image function should have at least one parameter';

}

for (const imageSrc of arrayOfImageSrcs) {

if (!(typeof (args[0]) === 'number' || typeof (args[0]) === 'string' || typeof (args[0]) === 'symbol' || typeof (args[0]) === 'bigint')) {

throw 'Error: image src can not be that type';

}

const image = new imageObjects.ImageObjectWithLoader();

image.src = imageSrc;

this.\_notLoadedImages[image.src] = (this.\_notLoadedImages[image.src] || 0) + 1;

this.countImages += 1;

const loadImagePomise = image.load();

loadImagePomise.then(() => {

this.countImages -= 1;

if (this.\_resolveLoaded && this.countImages === 0) {

this.\_resolveLoaded();

}

this.\_notLoadedImages[image.src] = (this.\_notLoadedImages[image.src] || 0) - 1;

this.\_loadedImages[image.src] = (this.\_loadedImages[image.src] || 0) + 1;

}).catch(() => {

if (this.\_rejectLoaded) {

this.\_rejectLoaded();

}

this.countImages -= 1;

this.\_cannotLoadImages[image.src] = (this.\_cannotLoadImages[image.src] || 0) + 1;

});

arrayToReturn.push(image);

}

if (arrayToReturn.length > 1) {

return arrayToReturn;

} else {

return arrayToReturn[0];

}

}

uploadImages(...args) {

return this.uploadImage(args);

}

load() {

for (const notStartedLoadingImage of this.\_notStartedLoading) {

const loadImagePomise = notStartedLoadingImage.load();

loadImagePomise.then(() => {

this.countImages -= 1;

if (this.\_resolveLoaded && this.countImages === 0) {

this.\_resolveLoaded();

}

this.\_notLoadedImages[notStartedLoadingImage.src] = (this.\_notLoadedImages[notStartedLoadingImage.src] || 0) - 1;

this.\_loadedImages[notStartedLoadingImage.src] = (this.\_loadedImages[notStartedLoadingImage.src] || 0) + 1;

}).catch(() => {

if (this.\_rejectLoaded) {

this.\_rejectLoaded();

}

this.countImages -= 1;

this.\_cannotLoadImages[notStartedLoadingImage.src] = (this.\_cannotLoadImages[notStartedLoadingImage.src] || 0) + 1;

});

}

this.\_notStartedLoading = [];

const promiseToReturn = new Promise((resolve, reject) => {

this.\_resolveLoaded = resolve;

this.\_rejectLoaded = reject;

});

promiseToReturn.finally(() => {

this.\_resolveLoaded = undefined;

this.\_rejectLoaded = undefined;

});

if (this.countImages === 0) {

if (this.\_resolveLoaded) {

this.\_resolveLoaded();

}

}

return promiseToReturn;

}

}

srcipts\image-graphics-object\index.js

import imageObjects from '../image-object/index.js';

export default class ImageGraphicsObject {

constructor(imageObject) {

if (typeof (imageObject) === 'object' && imageObject instanceof imageObjects.ImageObject) {

this.\_imageObject = imageObject;

} else {

throw 'Error: imageObject should extends imageObjects.ImageObject';

}

this.\_canvas = undefined;

this.\_canvasContext = undefined;

this.\_imageSetted = false;

this.\_scaleX = 1;

this.\_scaleY = 1;

}

static setCanvas() {

if (this.\_imageObject.loaded) {

if (!this.\_canvasContext) {

this.\_canvas = document.createElement('canvas');

this.\_canvas.width = Math.ceil(this.\_imageObject.width \* this.\_scaleX) || 1;

this.\_canvas.height = Math.ceil(this.\_imageObject.height \* this.\_scaleY) || 1;

this.\_canvasContext = this.\_canvas.getContext("2d"); // not necessary to use advanced contexts

} else {

this.\_canvasContext.clearRect(0, 0, Math.ceil(this.\_canvas.width \* this.\_scaleX) || 1, Math.ceil(this.\_canvas.height \* this.\_scaleY) || 1);

this.\_canvas.width = Math.ceil(this.\_imageObject.width \* this.\_scaleX) || 1;

this.\_canvas.height = Math.ceil(this.\_imageObject.height \* this.\_scaleY) || 1;

}

} else {

throw 'Error: image is not loaded';

}

}

setImage() {

ImageGraphicsObject.setCanvas.bind(this)();

this.\_canvasContext.drawImage(this.\_imageObject.domElement, 0, 0, this.\_imageObject.width \* this.\_scaleX, this.\_imageObject.height \* this.\_scaleY);

this.\_imageSetted = true;

}

scale(x, y) {

this.\_scaleX = x;

this.\_scaleY = y;

this.setImage();

}

getPixel(x, y) {

if (!this.\_imageSetted) {

this.setImage();

}

return this.\_canvasContext.getImageData(x, y, 1, 1).data;

}

putPixel(x, y, color0 = 255, color1 = 255, color2 = 255, color3 = 255) {

const image = this.\_canvasContext.getImageData(x, y, 1, 1);

const pixelData = image.data;

pixelData[0] = color0;

pixelData[1] = color1;

pixelData[2] = color2;

pixelData[3] = color3;

this.\_canvasContext.putImageData(image, x, y);

}

set canvas(object) {

this.\_canvas = object;

}

get canvas() {

if (!this.\_imageSetted) {

this.setImage();

}

return this.\_canvas;

}

set width(value) {

throw 'Error: width should be taken from canvas';

}

get width() {

if (!this.\_imageSetted) {

this.setImage();

}

return this.\_canvas.width;

}

set height(value) {

throw 'Error: width should be taken from canvas';

}

get height() {

if (!this.\_imageSetted) {

this.setImage();

}

return this.\_canvas.height;

}

}

srcipts\data-set\data-set-label.js

export default class DataSetLabel { };

srcipts\data-set\index.js

import PixelDataSet from './pixel-data-set.js';

import DataSetLabel from './data-set-label.js';

export default {

DataSetLabel: DataSetLabel,

PixelDataSet: PixelDataSet

}

srcipts\data-set\pixel-data-set.js

import DataSetLabel from './data-set-label.js';

export default class PixelDataSet extends DataSetLabel {

constructor(color0, color1, color2, color3, x, y, imageWidth, imageHeight) {

super();

this.colorList = [color0, color1, color2, color3];

this.\_x = x;

this.\_y = y;

this.\_imageWidth = imageWidth;

this.\_imageHeight = imageHeight;

this.coefficients = PixelDataSet.getKofInsideDataSet.bind(this)();

}

static minColor = 0;

static maxColor = 255;

static minX = 0;

static minY = 0;

static getKofInsideDataSet() {

const returnArray = [];

for (let i = 0; i < 4; i += 1) {

returnArray.push((this['\_color' + i] - PixelDataSet.minColor) / (PixelDataSet.maxColor - PixelDataSet.minColor));

//added for more accuracy

returnArray.push((PixelDataSet.maxColor - this['\_color' + i]) / (PixelDataSet.maxColor - PixelDataSet.minColor));

}

for (let i = 0; i < 4; i += 1) {

for (let j = i + 1; j < 4; j += 1) {

returnArray.push((this['\_color' + i] - this['\_color' + j]) / (PixelDataSet.maxColor - PixelDataSet.minColor));

}

}

returnArray.push((this.\_x - PixelDataSet.minX) / (this.\_imageWidth - PixelDataSet.minX));

//added for more accuracy

returnArray.push((this.\_imageWidth - this.\_x) / (this.\_imageWidth - PixelDataSet.minX));

returnArray.push((this.\_y - PixelDataSet.minY) / (this.\_imageHeight - PixelDataSet.minY));

//added for more accuracy

returnArray.push((this.\_imageHeight - this.\_y) / (this.\_imageHeight - PixelDataSet.minY));

return returnArray;

}

set coefficients(value) {

this.\_kofInsideDataSet = value;

}

get coefficients() {

return this.\_kofInsideDataSet;

}

set colorList(array) {

this.\_color0 = array[0];

this.\_color1 = array[1];

this.\_color2 = array[2];

this.\_color3 = array[3];

}

get colorList() {

return [this.\_color0, this.\_color1, this.\_color2, this.\_color3];

}

}