УДК 004.05
DOI https://doi.org/10.32851/tnv-tech.2023.1.4

МОДИФІКОВАНИЙ АЛГОРИТМ СТИСКАННЯ ПОСЛІДОВНОСТІ ЗОБРАЖЕНЬ

Панков Т. С. *– студент магістратури факультету прикладної математики Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»*
ORCID ID: 0000-0003-0230-5062

Потапова К. Р. *– кандидат технічних наук, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»*
ORCID ID: 0000-0002-3347-6350

Радченко К. О. *– асистент кафедри системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»*
ORCID ID: 0000-0002-1282-6307

З поширенням в сучасному світі як цифрових носіїв, так і мережі Інтернет, все частіше постає питання про зберігання та передачу даних у цифровому форматі. Потреби користувачів зростають швидше, ніж розвиваються апаратні технології зберігання та передачі даних, тому все частіше постає питання програмної оптимізації даних процесів. Крім механізмів балансування навантаження, оптимізації в мережевій передачі на транспортному рівні, важливу роль відіграють механізми стискання даних. Стискання даних має на увазі під собою зменшення розміру аналізованої інформації за рахунок використання того факту, що здебільшого дані не є випадковим набором біт, а підпорядковуються певному закону. Іншими словами, використовується той факт, що дані є залежними або випадковими величинами, або випадковими величинами, що підпорядковуються певній, нерівномірній функції розподілу.

Більшість даних, що передаються по мережі – фото і відео файли. Для швидкої передачі цих файлів та компактного їх зберігання потрібні ефективні алгоритми стискання, які здатні швидко та якісно стискати окремі зображення та послідовності кадрів відео. Проблемою при стисканні послідовності кадрів відео є обчислювальна складність пошуку областей кореляції зображень, яка проводиться з метою підвищення коефіцієнта стискання шляхом кодування посилання на схожу область зображення та різниці між попереднім та наступним зображенням замість кодування повністю вихідного зображення. Проблемою при стисканні набору зображень є виділення загального контексту даних зображень з метою підвищення якості стиснення.

У цій роботі ми розглянемо особливості стискання зображень, дослідимо підходи знаходження подібних областей, оптичних потоків та реалізуємо модель, що дозволяє підвищити ефективність стискання послідовності зображень.

Ключові слова: *стискання, квантування, оптичний потік, ключові точки, піксель, дескриптор, кадр, ентропія, фрагмент, декодування.*

Pankov T. S., Potapova K. R., Radchenko K. O. Modified image sequence compression algorithm

With the spread of both digital media and the Internet in the modern world, the question of storing and transferring data in digital format is becoming increasingly common. The needs of users are growing faster than the hardware technologies of data storage and transmission are

developing, so the question of software optimization of data processing processes is becoming more and more common. In addition to load balancing mechanisms, optimization of network transmissions at the transport level, data compression mechanisms play an important role. Data compression implies a reduction in the size of analyzed information due to the fact that the larger size is not a random set of bits, but obeys a certain law. In other words, it exploits the fact that the data are dependent or random variables, or random variables subject to a certain, non-uniform distribution function.

More data transmitted on the network - photo and video files. For fast transfer of these files and their compact storage, efficient compression algorithms are needed, which are able to quickly and efficiently compress individual images and a system of video frames. The problem with compressing video frame positions is the computational complexity of finding image correlation regions, which is designed to improve compression by encoding the image references of the image region and the difference between the previous and next image instead of encoding the full source image. The challenge when compressing a set of images is to remove the overall context of the image data in order to improve compression quality.

In this work, we will consider the features of image compression, explore approaches for finding similar areas, optical flows, and implement a model that allows increasing the efficiency of image sequence compression.

Key words: *compression, quantization, optical flow, key points, pixel, descriptor, frame, entropy, fragment, decoding.*

Постановка задачі. Задача полягає в аналізі існуючих алгоритмів стискання даних та зображень, пошуку кращих складових для нового алгоритму (дескриптор, алгоритми пошуку та опису ключових точок) та побудові алгоритму стискання зображень, що потенційно перевершує за ефективністю існуючі алгоритми стискання за певних умов.

Виклад основного матеріалу.

Особливості стискання зображень. І нарешті, під час стискання зображень часто користуються особливостями людського зору, саме тим фактом, що людське око більш сприйнятливе до кількості світла на зображенні, ніж до його кольоровості. Тому перед кодуванням зображення без втрат його зазвичай переводять у YCrCb формат і частина Y піддають меншому квантуванню, ніж Cr і Cb частини. Також при кодуванні зображень окрема увага приділяється переходам між кольорами сусідніх пікселів.

Практично всі з сучасних найбільш поширених алгоритмів стискання зображень базуються на описаних вище техніках, а саме – є лише поєднанням тої чи іншої підмножини з безлічі технік (рис. 1).

Говорячи про алгоритми стискання відео, варто сказати, що найчастіше вони

Рис. 1. Складові найбільш популярних алгоритмів

базуються на алгоритмах стискання зображень. Єдина відмінність – алгоритми відео використовують інформацію про раніше стислі кадри і покладаються на факт, що зображення у відео потоці відрізняються в більшості лише зміщенням деяких пікселів на кілька позицій. Звідси з'являється дві додаткові техніки: компенсація руху та кодування різниці. Перша використовується для знаходження переміщення областей та компенсації цього переміщення, друга – для кодування не самого кадру, а різниці між поточним кадром та попереднім кадром після компенсації руху (рис. 2).

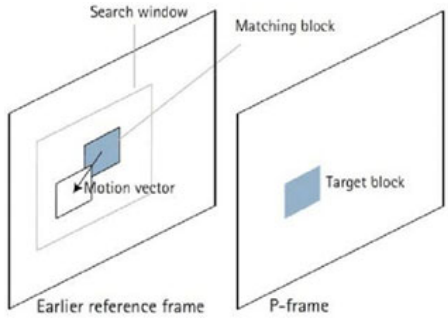

Рис. 2. Схема роботи компенсації руху

Окрім алгоритмів стискання зображень та алгоритмів стискання відео є також алгоритми, що спеціалізуються на стисканні послідовності подібних зображень. Зазвичай вони будуються так само, як алгоритми стискання відео, однак без етапу компенсації руху, так як на послідовності зображень об'єкти зазвичай сильно зсунуті відносно один одного і немає можливості застосувати алгоритми компенсації руху. Ця робота орієнтована на написання алгоритму, що є оптимальним для стискання послідовності зображень, для яких немає можливості ефективно застосувати сучасні види компенсації руху.

Дослідження підходів до вирішення задачі знаходження подібних об'єктів на зображеннях. Вивчивши принцип роботи алгоритмів стискання і знаючи, що підвищити рівень стискання можна за допомогою контексту попередніх зображень, реалізуємо алгоритм, що дозволяє на високому рівні отримувати, зберігати та шукати контекст зображень, а саме алгоритм, який знаходитиме схожі ділянки на різних зображеннях. цей алгоритм відмінно підійде для стискання ключових кадрів відео (рис. 3).

Зазвичай ключові кадри відео стискаються за допомогою звичайних алгоритмів стискання зображень без урахування контексту попередніх стиснутих ключових кадрів. Так відбувається, бо зазвичай на різних ключових кадрах об'єкти зазвичай сильно змінюють своє розташування і стандартні алгоритми компенсації руху не можуть впоратися з такими відхиленнями. Тому в даному випадку потрібно побудувати алгоритм, що дозволяє знайти не точні збіги фрагментів зображення незалежно від їх розташування.

Алгоритми знаходження оптичного потоку. Для знаходження оптичного потоку і розпізнавання об'єктів використовується така методика дій: спочатку виділяємо

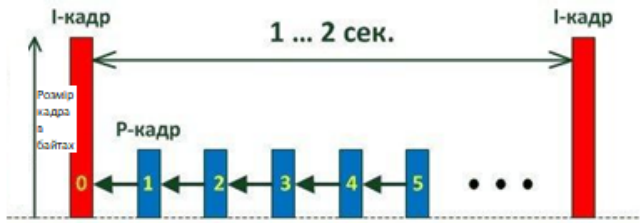

Рис. 3. Ключові (I) та різностні (P) кадри

на зображенні ключові точки, потім будуємо для них дескриптори (опис області навколо), так робимо для обох зображень і після шукаємо збіг між дескрипторами. Чим кращий збіг між дескрипторами – тим більше схожі області навколо ключових точок, що розглядаються.

Одні з найпопулярніших методів знаходження та опису ключових точок – ORB, SIFT, SURF. У SIFT і SURF серед особливостей інваріант за розміром дескриптора і по поворотах. Однак ми не маємо на меті знайти опис ключових точок інваріантний до поворотів і, тим більше, скейлінгу (масштабна інваріантність). Тому SIFT та SURF нам одразу не підходять. Спробуємо ORB на простому прикладі. Візьмемо зображення X.png, виріжемо частину і перевіримо, наскільки добре ORB зіставить дескриптори ключових точок. Для пошуку однакових дескрипторів на двох фрагментах будемо використовувати BFMatch (рис. 4, 5).

Як бачимо, навіть на такому простому прикладі ORB не готовий гарантувати необхідну точність. Спробуємо знайти окремо алгоритм пошуку ключових точок та алгоритм їх опису. Як частина ORB використовується алгоритм FAST. Розглянемо спершу його окремо. Після підбору параметрів можна дійти до наступного результату (рис. 6, 7).

Помітно, що ключові точки обираються однаково на обох фрагментах, отже при правильному їхньому описі між ними знайдеться відповідність.

```python
import cv2

def orb(file1, file2, count=50):
    img1 = cv2.imread(file1, 0)
    img2 = cv2.imread(file2, 0)

    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    if des1 is None or des2 is None:
        return

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)
    match_img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:count], None)
    cv2.imwrite('Matches.png',match_img)
    return kp1, des1, kp2, des2, matches
```

Рис. 4. Отримання ключових точок та дескрипторів через ORB

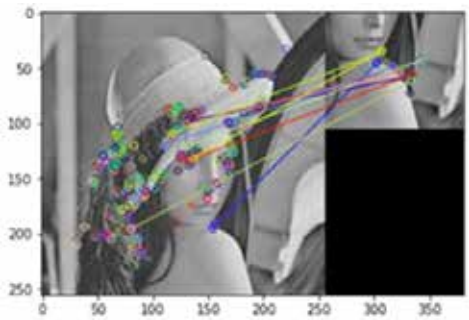

Рис. 5. Результат (пошук на цілому зображенні його частини)

```
def keypoints(file):
    img = cv2.imread(file,0)

    fast = cv2.FastFeatureDetector_create(threshold=15)
    kp = fast.detect(img,None)

    img2 = cv2.drawKeypoints(img, kp, None,color=(255,0,0))
    cv2.imwrite('fast_true.png',img2)
    plt.imshow(imread("fast_true.png"))
```

Рис. 6. Отримання ключових точок через FAST

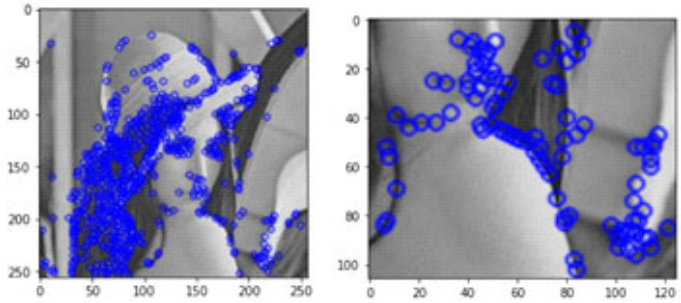

Рис. 7. Отримання ключових точок через FAST візуально

Кращим кандидатом на роль дескриптора був знайдений і обраний HOG з огляду на те, що він зберігає інформацію про палітру кольорів навколо заданої області, причому враховує розташування розглянутих пікселів, а тому краще за інших знаходитиме відповідність між однаковими фрагментами (рис. 8, 9).

```
def fast_hog(file1, file2, count=50, threshold=10):
    img1 = cv2.imread(file1, 0)
    img2 = cv2.imread(file2, 0)

    fast = cv2.FastFeatureDetector_create(threshold=threshold)
    winSize = (32,32)
    blockSize = (16,16)
    blockStride = (8,8)
    cellSize = (8,8)
    nbins = 9
    derivAperture = 1
    winSigma = 4.
    histogramNormType = 0
    L2HysThreshold = 2.0000000000000001e-01
    gammaCorrection = 0
    nlevels = 64
    hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                            histogramNormType,L2HysThreshold,gammaCorrection,nlevels)

    winStride = (32,32)
    padding = (32,32)
    kp1 = fast.detect(img1, None)
    kp2 = fast.detect(img2, None)
    des1 = hog.compute(img1,winStride,padding,[k.pt for k in kp1])
    des2 = hog.compute(img2,winStride,padding,[k.pt for k in kp2])

    if des1 is None or des2 is None:
        return

    des1, des2 = np.array(des1).reshape((len(kp1),-1)), np.array(des2).reshape((len(kp2),-1))

    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)
    match_img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:count], None)
    cv2.imwrite('Matches.png',match_img)
    return kp1, des1, kp2, des2, matches[:count]
```

Рис. 8. Отримання ключових точок через FAST та отримання дескрипторів через HOG

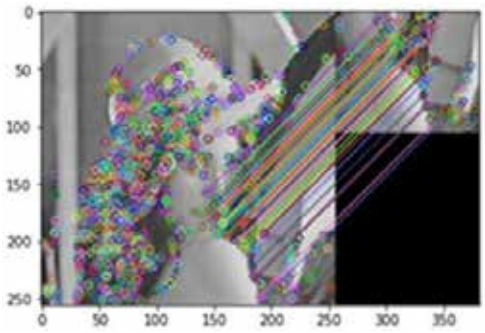

Рис. 9. Отримання ключових точок через FAST та отримання дескрипторів через HOG візуально

В результаті, кількість правильних відповідностей близька до максимуму. Розглянемо дещо інший приклад (рис. 10).

Майже всі відповідності вірні. Може здатися, що це два однакові кадри, тому наведемо модуль попіксельної різниці між цими кадрами (рис. 11).

Підсумкова комплектація – FAST + HOG + BFMatch.

Реалізація алгоритму стискання на основі пошуку та перевикористання областей високої ентропії. Опишемо алгоритм стискання зображення виходячи з отриманих збігів. Зауважимо, що алгоритм вибору ключових точок позитивно впливає на ефективність одержуваного алгоритму, оскільки ключові точки вибираються в областях з високою ентропією, а саме ці області найскладніше стискаються будь-яким алгоритмом стискання, у тому числі й JPEG. У кращому випадку нам

Рис. 10. Результат FAST+HOG на тесті із двох ключових кадрів відеофрагменту

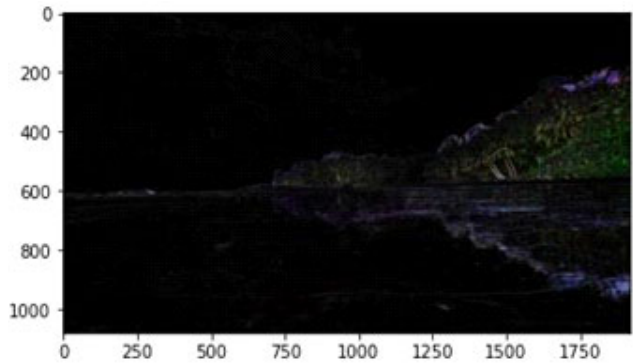

Рис. 11. Модуль різниці тестових ключових кадрів відеофрагменту

доведеться стискати лише одноколірні області та різниці між кешованими частинами та вихідним зображенням.

Отже, обираємо кращі збіги дескрипторів, відповідні ключові точки і замінюємо області навколо цих ключових точок на зображенні, що розглядається, на різницю між знайденою областю і аналізованою. Ділянки, що використовуються, і різниця, відповідно (розмір блоку навколо ключової точки – 64 x 64 пікселів) (рис. 12).

Спробуємо стиснути тільки отримане зображення. Вихідне зображення у форматі JPEG важить 159.1 KB, отримане – 147.9 KB. Краще, але трохи.

Згадаймо, що насправді JPEG розбиває зображення на фрагменти по 8 x 8, тому спробуємо вирівняти наші частини, що отримуються, по кратній сітці 8 x 8. Відобразимо отримуване зображення (рис. 13).

Отриманий розмір – 141.1 KB.

Тепер згадаємо ще про те, що всі блоки 8 x 8 все ж таки під кінець обробляються разом, тому давайте об'єднаємо різниці в одне зображення, а все інше – в інше, тим самим розташувавши поруч схожі блоки 8 x 8. Пропорції зображення зберегти не вдасться, тому зберігати все будемо у зображення у вигляді рядка шириною в ширину блоку, що розглядається навколо ключової точки. У поточному випадку – 64.

Отримані розміри – для різниць: 37KB, для частин зображення, що залишилися: 105KB. Сумарно вийшло більше, ніж було, тому що JPEG для кожного зображення зберігає додаткову інформацію. Таку як таблиці Хаффмана, наприклад. Однак тепер ми маємо можливість варіювати якість окремо зображення та окремо – різниць. Так як різниці незначні, їх можна стискати досить без втрати якості. Збережемо зображення з різницею із параметром quality = 10. Отримаємо

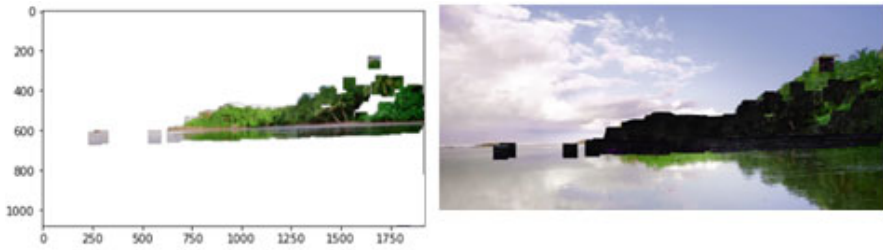

Рис. 12. Виділення та видалення областей схожих із областями з попереднього ключового кадру

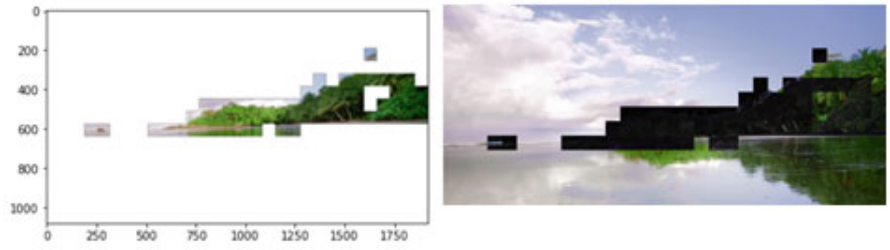

Рис. 13. Виділення та видалення областей схожих із областями з попереднього ключового кадру з вирівнюванням по сітці 8x8

розмір різниць 7.5KB. Разом сумарний розмір 7.5 KB+105 KB = 112.5 KB проти вихідних 159.1 KB. Подивимося на вихідне та декодоване після нашого стискання зображення відповідно (рис. 14).

Отже, розглянемо загальну схему кодування та декодування за допомогою наведеного вище алгоритму. Кодування:

1. Зчитуємо чергове зображення.
2. Знаходимо ключові точки за допомогою алгоритму FAST.
3. Знаходимо дескриптори ключових точок за допомогою алгоритму HOG.
4. Знаходимо всі відповідності між дескрипторами даного зображення та попередніми зображеннями.
5. Виділяємо до найбільш відповідних відповідностей.
6. Для кожної ключової точки, для якої знайшлася відповідність, знаходимо найближчу точку з сітки align x align і зсовуємо розглянуту ключову точку до знайденої, зсовуючи відповідно і точку, і відповідності.
7. Проходимо циклом по всіх блоках align x align зображення та виконуємо наступне.
8. Якщо даному блоку знайшлася відповідність на попередніх зображеннях, зберігаємо в масив блоків-різниць модуль різниці між блоком, що розглядається, і йому відповідним. Також окремо зберігаємо у масив знаків – знаки різниць. Це робиться через те, що алгоритми стискання зображень працюють з позитивними числами, тому знак різниці переважно зберігати і стискати окремо.
9. Якщо даному блоку не знайшлося відповідності, зберігаємо до масиву блоків-вихідників.
10. Отримані масиви блоків-вихідників і блоків-різниць зберігаємо відповідно у два зображення, причому блоки-різниці зберігаємо в низькій якості.
11. Отримані знаки та словник відповідностей зберігаємо в окремий файл з lossless стисканням.

Декодування відбувається аналогічно у зворотному порядку.

Зображення, які погано стискаються звичайним алгоритмом стискання, тобто зображення, у яких високий коефіцієнт ентропії, стискаються розробленим алгоритмом. Проте алгоритм не справляється із зображеннями, в яких немає великих подібних областей. Таким чином, він може бути корисний у випадках, коли маємо справу із зображеннями різних розмірів, в яких спостерігаються схожі фрагменти, або із зображеннями однакових розмірів, в яких спостерігаються схожі фрагменти, наприклад, відео з малим FPS.

Висновки. На основі принципів стискання даних з втратами та без втрат було сформовано концепцію універсального алгоритму стиснення даних без втрат. Було проведено дослідження в області стиснення зображень, кадрів відео,

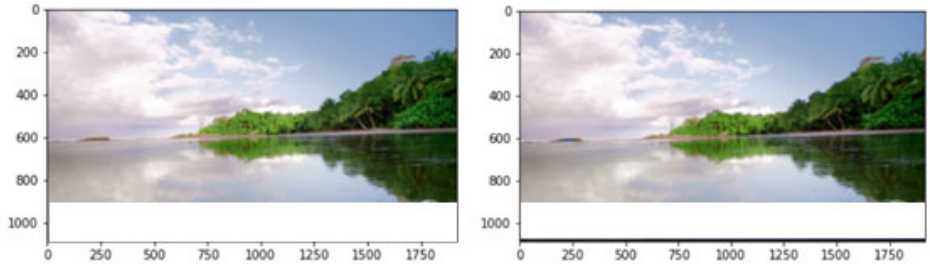

Рис. 14. Зліва – вихідне зображення, праворуч – збережене у новому форматі

в основних підходах до пошуку оптичного потоку, компенсації руху відео, підходах до завдання розпізнавання об'єктів. На підставі отриманих знань, було проведено ряд експериментів націлених на пошук найбільш відповідного методу для пошуку подібних областей високої ентропії на кількох зображеннях без поворотів та зміни розмірів.

Після цього був розроблений алгоритм стискання послідовності подібних зображень, що оптимізує зберігання областей високої ентропії у зображеннях, тим самим скорочуючи розмір підсумкового зображення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Р. Вудс. Цифровая обработка изображений. Москва : Техносфера, 2005.
2. Стискання на основі передбачення значень пікселей. [Електронний ресурс]. https://www.csd.uwo.ca/~melsakka/publications/journals/pdfs/2007_jvcir_Nathan ael.pdf
3. Стискання на базі передбачення збігу значень довколишніх в двумірному сенсі пікселей зображення. [Електроний ресурс]. https://naun.org/main/NAUN/computers/17-679.pdf
4. Вейвлет перетворення для стискання зображень. [Електроний ресурс]. https://www.researchgate.net/publication/266018963_Wavelet_image_compressi on
5. Стандарт JPEG. [Електроний ресурс]. https://web.stanford.edu/class/ee398a/handouts/lectures/08-JPEG.pdf.
6. Гістограми орієнтованих градієнтів для виявлення людини [Електронний ресурс]. http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
7. Зіставлення зображень за допомогою SIFT, SURF, BRIEF і ORB: порівняння продуктивності для спотворених зображень [Електронний ресурс]. https://arxiv.org/pdf/1710.02726.pdf
8. FFD: швидкий детектор функцій [Електронний ресурс]. https://ieeexplore.ieee.org/document/9292438
9. Основи Brute-Force Matcher [Електронний ресурс]. http://man.hubwiz.com/docset/OpenCV.docset/Contents/Resources/Documents/dc/dc3/tutorial_py_matcher.html

REFERENCES:

1. R. Vuds (2005) Tsifrovaya obrabotka izobrazheniy [Digital Image Processing]. Moskow: Tekhnosfera.
2. Compression based on pixel value prediction. Retrieved from: https://www.csd.uwo.ca/~melsakka/publications/journals/pdfs/2007_jvcir_Nathan ael.pdf
3. Dictionary Based Compression for Images. Retrieved from: https://naun.org/main/NAUN/computers/17-679.pdf
4. Wavelet transform for image compression. Retrieved from: https://www.researchgate.net/publication/266018963_Wavelet_image_compressi on
5. JPEG standard. Retrieved from: https://web.stanford.edu/class/ee398a/handouts/lectures/08-JPEG.pdf.
6. Histograms of Oriented Gradients for Human Detection. Retrieved from: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
7. Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. Retrieved from: https://arxiv.org/pdf/1710.02726.pdf
8. FFD: Fast Feature Detector. Retrieved from: https://ieeexplore.ieee.org/document/9292438
9. Basics of Brute-Force Matcher. Retrieved from: http://man.hubwiz.com/docset/OpenCV.docset/Contents/Resources/Documents/dc/dc3/tutorial_py_matcher.html