НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
з напряму підготовки 6.050101 «Комп'ютерні науки»
на тему: «Методи масштабування зображень без втрати якості з використанням нейронних мереж»

Виконав:
студент IV курсу, групи КА-55
Фоменко Нікіта Андрійович

_____
Керівник:
доцент, д.т.н. Недашківська Н.І.

_____
Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А. Є. _____

Рецензент:
проф., д. т. н. Теленик С.Ф. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019 року

РЕФЕРАТ

Дипломна робота: __ с., 55 рис., 9 табл., 2 дод., 15 джерел.

МАСШТАБУВАННЯ ЗОБРАЖЕНЬ, ГЕНЕРАТИВНО ЗМАГАЛЬНІ НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, КРИТЕРІЇ ЯКОСТІ МАСШТАБУВАННЯ, МЕТОДИ ІНТЕРПОЛЯЦІЇ ЗОБРАЖЕННЯ

В цій роботі розглядається SRGAN – генеративно змагальна нейронна мережа для масштабування зображень. Це один з найперших фреймворків, що використовується на збільшення зображень у 4 рази, при цьому майже без втрат якості. Для цього було запропоновано перцептивну функцію втрат. Також було проведе порівняння з іншими методами масштабування, такими як білінійна та бікубічна інтерполяції, метод Ланцоша та метод найближчого сусіда.

Об’єкт дослідження: зображення малого розміру, зображення з низькою роздільною здатністю.

Предмет дослідження: методи нейронних мереж, інтерполяційні для збільшення зображення та покращення їх роздільної здатності.

Мета роботи: дослідити моделі нейронних мереж для покращення якості зображення з малою роздільною здатністю.

Методи дослідження: чисельні експерименти з порівнянням результатів, що імплементовані за допомогою мови програмування Python та бібліотеки Tensorflow.

ABSTRACT

The work consists of __ pages, 55 images, 9 tables, 2 appendices, 15 sources.

IMAGE SCALING, GENERATIVE ADVERSARIAL NETWORKS, CONVOLUTIONAL NEURAL NETWORKS, CRITERIA OF IMAGE SCALING QUALITY, IMAGE INTERPOLATION METHODS

The SRGAN is a Generative Adversarial Neural Network for Image Scaling discussed in this paper. This is one of the very first frameworks used to scale-up images up to 4 times, with almost no loss of quality. For this purpose, a perceptive function of losses was proposed. A comparison was also made with other scaling methods, such as bilinear and bicubic interpolation, the Lanczos method, and the nearest neighbor method.

Object of research: small image, low resolution image.

Subject of research: methods of neural networks, interpolation to scale-up the image and improve their resolution.

Objective: to explore models of neural networks to improve the quality of images with low resolution.

Research methods: numerous comparative experiments implemented using Python programming language and Tensorflow library.

Зміст

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі, що розглядається

Останнім часом мультимедіа розповсюджується дуже швидко і проникає в кожен аспект повсякденного життя. Споживачі переглядають картинки, відео і інші візуальні данні на широкому спектрі продуктів, починаючи з телевізорів і комп‘ютерних моніторів до великої кількості різних портативних апаратів. Виробники контенту не мають змоги прогнозувати і створювати таку кількість продукту, яка б могла задовольнити потреби кінцевих користувачів з різноманітними дисплеями. Тому з‘явилася необхідність в алгоритмах, які можуть обробляти вхідний сигнал відповідно таким чином, щоб він належно пристосовувався до вихідного формату. З кожним днем все більше і більше пристроїв приєднуються до інтернету і додаткова обробка даних може стати необхідною, аби подолати проблеми, що можуть виникнути з обмеженнями на швидкість передачі і пропускну здатність.

Масштабування (або інтерполяція) зображень, і зменшення, і збільшення, є необхідною під час зміни розмірів даних, щоб задовільнити вимоги каналу зв‘язку або вихідного дисплею. В той час як передавати версію з низькою роздільною здатністю до клієнта є більш ефективним, то апроксимація до оригіналу з високою роздільною здатністю може бути необхідною, щоб зобразити її кінцевому користувачу. Тому зміна розмірів візуальних даних зі збереженням пропорцій це важливий крок в багатьох прикладних задачах, починаючи з різних версій споживчих товарів до критичних завдань у медичній, оборонній та багатьох інших сферах. Новою тенденцією є масштабування старих відеофільмів та мультфільмів до більш

високої роздільної здатності, щоб вони були пристосовані до телевізорів та моніторів з високим розширенням. Аналогічним чином покращують якість текстур старих відеоігор або ігор без підтримок моніторів з високої роздільною здатністю. Особливо часто це використовується в ігрових приставках.

1.2 Аналіз існуючих підходів до її вирішення

Ідеальний алгоритм інтерполяції зображення має зберігати якісні характеристики вхідного зображення, тому що масштабоване зображення зазнає втрат від таких артефактів як розмивання, переривання по краях і ефекти різкої зміни картинки. Крім того, методи, що орієнтовані на додатки, які працюють у реальному часі на мобільних телефонах чи цифрових камерах, мають уважно обробляти якісні атрибути. Ці алгоритми повинні задовольняти вимоги у невеликих обчислювальних витратах і маленькій кількості пам‘яті у роботі на мобільних пристроях у реальному часі.

Алгоритми, що широко застосовуються, такі як метод найближчого сусіда і білінійна інтерполяція задовольняють вимоги до простоти в обчислюваннях, але страждають від серйозних проблем із розмиванням, особливо в регіонах з краями. Лінійні підходи застосовуються частіше від нелінійних, таких як бікубічна інтерполяція і інтерполяція сплайнами, у зв‘язку з тим фактом, що останні вимагають більше обчислювальне навантаження і додають розмивання. Інтерполяція, основана на роботі з ребрами, повинна була подолати такі недоліки застосовуючи різні оператори відповідно до напрямку країв. Головною проблемою цієї категорії це потреба у додатковому кроці, який витягає ребра з картинки та їх здібність до розпізнавання ребр лише під деякими кутами. Марковська мережа, що

базується на моделюванні напрямку ребр, запропонована у. Вона пов'язує інтерпольовані зображення з мінімальним енергетичним станом випадкового 2D поля. Нейронні мережі також використовують у масштабуванні зображень. Цей метод потребує велику кількість шарів і нейронів для інтерполяції зображень, раніше це було дуже важко в обчислювальному плані, проте зараз з ростом технологій і впровадженням нових архітектур нейромереж це вже є реальним. Нечіткі підходи до інтерполяції були запропоновані у для двовимірної передискретизації сигналу. Ці методи показують гарні результати, але вони страждають від складності і також іноді потребують додаткову обробку аби виявити ребра. Інтерполяція на основі площ обчислює кожен піксель пропорційно до площі покриття фільтруючого вікна, яке застосовують до вхідного зображення. Методи інтерполяції Мітчелла і Ланцоша використовують вагову функцію обмеженої просторової підтримки, щоб зменшити спектральні витоки і втрати оптичної роздільної здатності, які однак потребують великої кількості апаратних ресурсів. Аналогічно, квадратична інтерполяція зображень показала чудові візуальні результати, але обчислювальні витрати для кодування є її найбільшим недоліком.

Базуючись на тому, як алгоритм обробляє зміст, алгоритми масштабування зображення можна класифікувати на адаптивне та неадаптивне масштабування.

Алгоритми адаптивного масштабування зображення модифікують свою техніку інтерполяції в залежності від того чи візуальний зміст являє собою гладку текстуру, чи гострі краї. Оскільки методи інтерполяції змінюються у реальному часі, то ці алгоритми досить складні і обчислювально затратні. Цей клас алгоритмів широко застосовується у програмному забезпеченні для редагування зображень, бо вони забезпечують високу якість масштабованого зображення.

Алгоритмами неадаптивного масштабування є вищезгадані, такі як метод найближчого сусіда, білінійна та бікубічна інтерполяції. В них метод інтерполяції не змінюється незалежності від контенту зображення. В залежності від складності алгоритму цей клас використовує від 0 до 256 суміжних пікселів для інтерполяції. Чим більше пікселів використовується тим краще і точніше відбувається масштабування зображення, проте і більше часу вимагається для інтерполяції.

1.3 Особливості предметної області

Всі неадаптивні методи намагаються знайти баланс між трьома небажаними ефектами: розмивання, згладжування і освітлення країв. Ці ефекти зображені на рисунках 1.1 - 1.4.

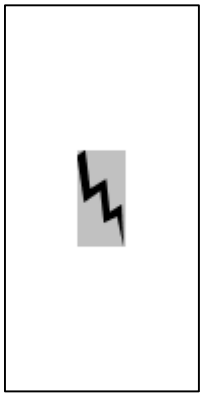


Рисунок 1.1 – Оригінал зображення

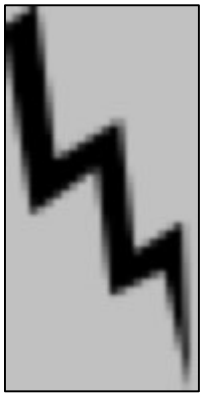


Рисунок 1.2 – Ефект згладжування

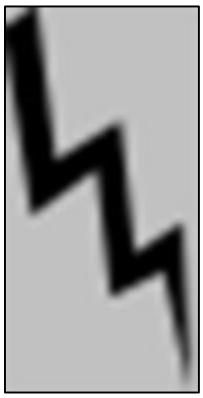


Рисунок 1.3 – Ефект розмиття.

Рисунок 1.4 – Ефект освітлення країв.

Навіть найбільш досконалі неадаптивні алгоритми завжди збільшують або зменшують наявність одного з вище вказаних артефактів за рахунок інших двох — тому принаймні один з них залишиться наявним.

Адаптивні інтерполятори можуть створювати або ні, наведені ефекти, проте вони також можуть спричиняти появу неіснуючих текстур або дивних пікселів у малих частинах зображення, як показано на рисунках 1.5 та 1.6.

Рисунок 1.5 – Оригінал

Рисунок 1.6 – Збільшене на 200% за допомогою адаптивного алгоритму

З іншої сторони, деякі з цих артефактів у адаптивному масштабуванні можна розглядати як переваги. Оскільки око очікує, що зможе розпізнавати деталі до найменших масштабів у дрібних текстурах, як у листі, то цей шаблон повинен обманювати око з деякої дистанції (для деяких задач).

Існує процес, який допомагає мінімізувати появу зубчастих країв, під назвою згладжування. Він дає тексту або зображенню грубий цифровий вигляд, що показано на рисунку 1.7 та 1.8.

Рисунок 1.7 – Збільшено на 300% із згладжуванням

Рисунок 1.8 – Збільшено на 300% без згладжування.

Згладжування прибирає гостроту і створює ефект більш гладких країв і високої роздільної здатності. Цей процес бере до уваги кількість пікселів, які перекриває ідеальне ребро. Грубі краї просто округляються вгору або вниз без проміжного значення, а згладжуванні краї, в свою чергу отримують значення пропорційне до кількості пікселів, що належать даному краю, я показано на рисунку 1.9 - 1.11.

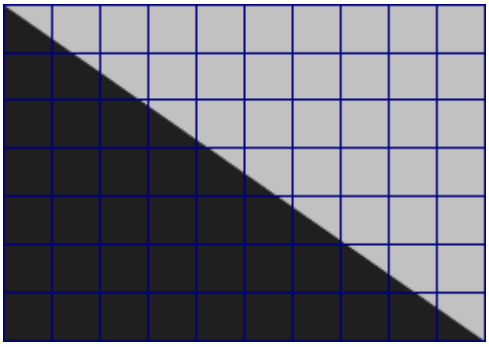


Рисунок 1.9 – Ідеальна діагональ

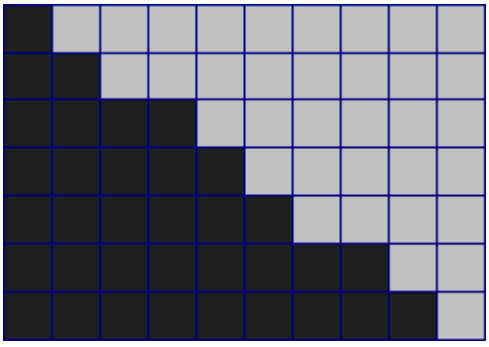


Рисунок 1.10 – Згладжена діагональ

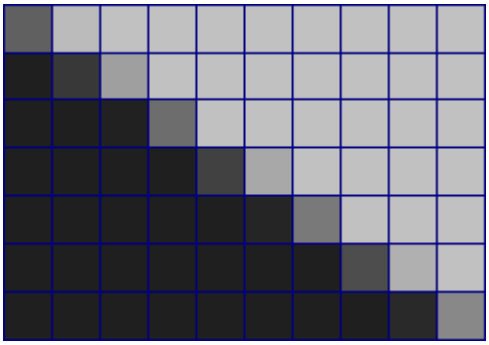


Рисунок 1.11 – Діагональ із антизгладжуванням

Основною перепоною під час збільшення зображення є запобігання загострення або згладжування. Багато адаптивних алгоритмів розпізнають наявність країв і змінюються, щоб мінімізувати загострення, але в той же час залишаючи гостроту ребр. Оскільки згладжений край містить інформацію про місцезнаходження цього краю на більш високій роздільній здатності, то стає зрозуміло, що потужний адаптивний (який може розпізнавати ребра) інтерполятор може хоча б частково реконструювати ці краї під час збільшення зображення.

1.4 Постановка задачі дослідження

Масштабування зображення можна описати як процес використання відомих даних для оцінки значень в невідомих координатах. Інтерпольоване значення $f(\mathrm{x})$ в координаті x в просторі розмірності q можна визначити як лінійну комбінацію зразків f_{k} обчислених в цілочисельних координатах $\mathrm{k} = (k_1, k_2, \ldots, k_k) \in Z^q$:

$$f(\mathrm{x}) = \sum_{\mathrm{k} \in z^q} f_{\mathrm{k}} \varphi_{int}(\mathrm{x} - k) \forall \mathrm{x} = (x_1, x_2, \ldots, x_q) \in R^q.$$

Ваги зразків обчислюються базисною функцією $\varphi_{int}(\mathrm{x} - \mathrm{k})$. Головна ідея інтерполяції в тому, щоб визначити базисну функцію, яка апроксимує початкове вхідне зображення, аби отримати вихідне інтерпольоване зображення. Оскільки базисна функція інтерполяції визначена на параметрі, що приймає неперервні значення, то вона може бути обчислена в довільній точці. Отримані значення складають інтерпольоване зображення. Важливою вимогою до інтерполюючою функції є її можливість проходити через відомі точки зображення. Порядок інтерполяції вказує на максимальну кількість неперервних похідних, які інтерполююча функція має мати. Через це, інтерполяція зображень B-сплайнами високого порядку призводить до більш гладкого і неперервного вигляду вихідного зображення. Тим не менш, разом із якістю росте і складність обчислень.

Найпростішим інтерполятором B-сплайнами є методи нульового і першого порядку. Вони відомі як реплікація пікселів (метод найближчого сусіда) і білінійна інтерполяція відповідно. В методі найближчого сусіда кожен вихідний піксель просто присвоює значення найближчого вхідного пікселя. В білінійній інтерполяції базисна функція є лінійно-кусковою. Це значить, що кожен вихідний піксель може бути обчислений як лінійна комбінація до чотирьох вхідних пікселів. Обидва методи дуже стандартні, і задовільно інтерполюють гладкі текстури. З іншої сторони, в них є проблеми з обробкою країв і маленьких деталей. Інтерполяція B-сплайнами третього порядку показує кращі результати, ніж метод найближчого сусіда і білінійна інтерполяція. Проте, один недолік кубічної інтерполяції B-сплайнами - це великі обчислювальні вимоги, що вимагаються, аби віднайти інтерполюючу функцію. Інша проблема цього методу виникає після того як ми отримали функцію інтерполяції — великий об‘єм обчислень все ще необхідний для відновлення бажаного вихідного зразка.

Тому в роботі ставляться задачі:

- Провести порівняльний аналіз існуючих методів та підходів масштабування зображень.
- Проаналізувати критерії оцінки якості масштабування зображень.
- Вивчити різні моделі нейронних мереж і віднайти модель, яка покаже добрий результат і перевершить результати інших відомих методів.
- Провести аналіз результатів масштабування зображень різних типів, отриманих методами найближчого сусіда, білінійної та бікубічної інтерполяції, Ланцоша та моделі нейронної змагальної мережі SRGAN.

1.5 Висновки

Кожен з цих алгоритмів описаних вище застосовує єдиний лінійний фільтр. Оскільки один і той же фільтр використовується, щоб отримати кожен вихідний фільтр, то ці методи схильні усереднювати вміст зображення біля границь з великим контрастом, таких як ребра. Результат такого масштабування блоковий і розмитий. Це особливо помітно у методі найближчого сусіда та білінійній інтерполяції. Аби подолати цю проблеми запропонували велику кількість нелінійних підходів, що раніше було описано.

Як ми бачимо, алгоритми масштабування відрізняються функцією інтерполяції, що робить їх придатними в різних ситуаціях і різним обчислювальним вартостям. Визначення методу, який може давати досить якісне вихідне зображення і в той же час не вимагає великих обчислень — дуже важливий, але важкий процес.

Головною задачею цієї роботи є дослідження роботи нейронних мереж для задачі масштабування. Необхідно віднайти архітектуру мережі, яка показувала дуже добрий результат і могла перевершити результати інших методів і підходів.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1.1 Дослідження існуючих методів для вирішення задачі

В цій частині роботі розглядаються неадаптивні методи масштабування зображення: найближчого сусіда, білінійна, бікубічна, модифікована бікубічна і Ланцоша. Нехай $P(x, y)$ - це піксель в точці (x, y) вхідного зображення розміром $M \times N$, а $U(i, j)$ - це піксель масштабованого вихідного зображення розмірністю $M' \times N'$. T - оператор трансформації, що використовується для масштабування. Кожен піксель вихідного зображення обчислюється шляхом застосування оператора T до вхідного зображення як показано на рисунку 2.1.

$$\begin{bmatrix} P_{11} & \cdots & P_{1N} \\ \vdots & \ddots & \vdots \\ P_{M1} & \cdots & P_{MN} \end{bmatrix} \xrightarrow{U(i,j)=T(P(x,y))} \begin{bmatrix} U_{11} & \cdots & U_{1N'} \\ \vdots & \ddots & \vdots \\ U_{M'1} & \cdots & U_{M'N'} \end{bmatrix}$$

Рисунок 2.1 – Застосування оператора трансформації до вхідного зображення

2.1.2 Метод найближчого сусіда

Найпростіший метод інтерполяції зображення - це обирати значення найближчого пікселя вхідного зображення і присвоювати те саме значення пікселю масштабованого зображення. Алгоритм найближчого сусіда обирає значення найближчого відомого пікселя, не беручи до уваги значення інших сусідніх пікселів, що дає нам кусково-константну функцію.

Алгоритм найближчого сусіда має низьку складність, що призводить до досить швидких обчислень. Проте якість масштабованого зображення низька і спостерігаються великий ефект згладжування. Алгоритм методу найближчого сусіда зображено на рисунку 2.2.

```
float tx = width_source/width_dst;
float ty = height_source/ height_dst;

for(i=0; i<height_dst; i++)
 for(j=0; j<width_dst; j++)
  {
      x = ceil(j*tx);
      y = ceil(i*ty);
      U(i,j) = P(y,x);
 }
```

Рисунок 2.2 – Алгоритм найближчого сусіда

2.1.3 Білінійне масштабування зображення

Ще один простий метод інтерполяції - це лінійна інтерполяція, незначне покращення алгоритму найближчого сусіда. В лінійній інтерполяції, значення, що інтерполюємо, обчислюється на основі середньозваженого двох точок. Ваги обернено пропорційні до відстані від вхідних точок до точки, яку інтерполюємо.

Алгоритм білінійної інтерполяції використовує лінійну інтерполяцію в одному напрямку, за якою слідує лінійна інтерполяція інтерпольованих значень в іншому напрямку. Алгоритм використовує 4 діагональних пікселя, що оточують відповідний піксель вхідного зображення для інтерполяції. Він обраховує середньозважене, враховуючи близькість і яскравість 4 діагональних пікселів і присвоює це значення пікселю у вихідному зображенні. Алгоритм показано на рисунку 2.3.

```
float tx = (width_source)/width_dst;
float ty = (height_source)/ height_dst;
float x_diff, y_diff;

for(i=0; i<height_dst; i++)
 for(j=0; j<width_dst; j++)
  {
    x = (int)(tx * j);
    y = (int)(ty * i);

    x_diff = ((tx * j) -x);
    y_diff = ((ty * i) -y );

   U(i,j) = P(y,x) *(1-x_diff)*(1-y_diff) +
            P(y,x+1)*(1-y_diff)*(x_diff) +
            P(y+1,x)*(y_diff)*(1-x_diff) +
            P(y+1,x+1)*(y_diff)*(x_diff);
 }
```

Рисунок 2.3 – Алгоритм білінійної інтерполяції

Порівняно з методом найближчого сусіда, метод білінійної інтерполяції надає відчутні покращення якості зображення з невеликим збільшенням складності. Ефект згладжування зникає, але з‘являється ефект розмиття.

2.1.4 Бікубічне масштабування зображення

Алгоритм бікубічної інтерполяції виконує кубічну інтерполяцію в одному напрямку за якою слідує кубічна інтерполяція інтерпольованих значень в іншому напрямку.

Бікубічна інтерполяція апроксимує sinc інтерполяцію використовуючи кубічні форми поліномів замість лінійних форм під час обрахунку значення вихідного пікселя. Для інтерполяції алгоритм використовує 16 найближчих пікселів, що оточують відповідно найближчий піксель вхідного зображення. Алгоритм зображено на рисунку 2.4.

```
float tx = (width_source)/width_dst;
float ty = (height_source)/ height_dst;

for(i=0; i<height_dst; i++)
 for(j=0; j<width_dst; j++)
  {
        x =(int)(tx*j);
        y =(int)(ty*i);

        dx=(float)(tx*j-x);
        dy=(float)(ty*i-y);

        for(jj=0;jj<=3;jj++)
          {
            d0 = P(y-1+jj,x-1) - P(y-1+jj,x);
            d2 = P(y-1+jj,x+1) - P(y-1+jj,x);
            d3 = P(y-1+jj,x+2) - P(y-1+jj,x);
            a0 = P(y-1+jj,x);
            a1 = (-1.0/3*d0 + d2 -1.0/6*d3);
            a2 = (1.0/2*d0 + 1.0/2*d2);
            a3 = (-1.0/6*d0 - 1.0/2*d2 + 1.0/6*d3);
            C[jj]=(a0 + a1*dx + a2*dx*dx + a3*dx*dx*dx);
          }
            d0 = (C[0]-C[1]);
            d2 = (C[2]-C[1]);
            d3 = (C[3]-C[1]);
            a0 = C[1];
            a1 = ( -1.0/3*d0 + d2 -1.0/6*d3);
            a2 = (1.0/2*d0 + 1.0/2*d2);
            a3 = (-1.0/6*d0 - 1.0/2*d2 + 1.0/6*d3);
            Cc = (a0 + a1*dy + a2*dy*dy + a3*dy*dy*dy);
            if(Cc>255) Cc=255;
            if(Cc<0) Cc=0;
            U(i,j) = Cc
  }
```

Рисунок 2.4 – Алгоритм бікубічної інтерполяції

Бікубічний метод масштабування показує високу якість масштабованого зображення, але з відносно високою складністю. З‘являється ефект сяйва країв, проте зникає розмиття та згладжування зображення.

2.1.5 Масштабування зображення методом Ланцоша

Алгоритм інтерполяції Ланцоша використовує зважену форму sinc фільтра, щоб зробити інтерполяцію на 2-D сітці. Sinc функцію однієї змінною можна математично отримати шляхом ділення sine функції однієї змінної на саму себе. Sinc функція в теорії ніколи не досягає нуля. Тому на практиці можна отримати потрібний фільтр скінченного розміру множенням sinc функції на оператор вікна такий як Хеммінга і Ганна.

Фільтр Ланцоша можна добути множенням sinc функції з вікном Ланцоша, який треба відсікати до нуля зовні головної частини. Розмір вікна Ланцоша визначається порядком згорткового ядра.

Кількість сусідніх пікселів, які ми розглядаємо змінюються в залежності від порядку ядра. Якщо порядок ядра дорівнює 2, то розглядається 16 пікселів, а коли порядок ядра - 3, то 36 сусідніх пікселів використовується для інтерполяції. Для достатньої якості зображення, у реалізації зазвичай обирають порядок рівним 3. Алгоритм продемонстровано на рисунку 2.5.

```
float tx = (width_source)/width_dst;
float ty = (height_source)/ height_dst;

for(i=0; i<height_dst; i++)
 for(j=0; j<width_dst; j++)
  {
    x =(int)(tx*j);     y =(int)(ty*i);
    dx = tx*j - x;      dy = ty*i - y;

    a0 = sin(pi*(dx+2))*sin(pi*(dx+2)/3)/(pi*pi*(dx+2)*(dx+2)/3);
    a1 = sin(pi*(dx+1))*sin(pi*(dx+1)/3)/(pi*pi*(dx+1)*(dx+1)/3);
    a2 = sin(pi*(dx))*sin(pi*(dx)/3)/(pi*pi*(dx)*(dx)/3);
    a3 = sin(pi*(dx-1))*sin(pi*(dx-1)/3)/(pi*pi*(dx-1)*(dx-1)/3);
    a4 = sin(pi*(dx-2))*sin(pi*(dx-2)/3)/(pi*pi*(dx-2)*(dx-2)/3);
    a5 = sin(pi*(dx-3))*sin(pi*(dx-3)/3)/(pi*pi*(dx-3)*(dx-3)/3);
    b0 = sin(pi*(dy+2))*sin(pi*(dy+2)/3)/(pi*pi*(dy+2)*(dy+2)/3);
    b1 = sin(pi*(dy+1))*sin(pi*(dy+1)/3)/(pi*pi*(dy+1)*(dy+1)/3);
    b2 = sin(pi*(dy))*sin(pi*(dy)/3)/(pi*pi*(dy)*(dy)/3);
    b3 = sin(pi*(dy-1))*sin(pi*(dy-1)/3)/(pi*pi*(dy-1)*(dy-1)/3);
    b4 = sin(pi*(dy-2))*sin(pi*(dy-2)/3)/(pi*pi*(dy-2)*(dy-2)/3);
    b5 = sin(pi*(dy-3))*sin(pi*(dy-3)/3)/(pi*pi*(dy-3)*(dy-3)/3);

    for(jj=0;jj<=5;jj++)
     {
       d0 = P(y-2+jj,x-2);        d1 = P(y-2+jj,x-1);
       d2 = P(y-2+jj,x);          d3 = P(y-2+jj,x+1);
       d4 = P(y-2+jj,x+2);        d5 = P(y-2+jj,x+3);
       C[jj]= a0*d0 + a1*d1 + a2*d2 + a3*d3 + a4*d4 +a5*d5;
     }

     Cc = a0*C[0]+a1*C[1]+a2*C[2] +a3*C[3]+a4*C[4]+a5*C[5];
     if(Cc>255) Cc=255;    if(Cc<0) Cc=0;
     U(i,j) = Cc;
}
```

Рисунок 2.5 – Алгоритм інтерполяції Ланцоша

Алгоритм Ланцоша видає вихідне масштабоване зображення такої ж якості як і бікубічне масштабування, але ціна за це - більша кількість обчислень, оскільки для інтерполяції значення кожного вихідного пікселя використовується 36 пікселів, в порівнянні з 16 пікселями у бікубічній інтерполяції. Алгоритм Ланцоша порівняно швидший при багаторазовому масштабуванні.

2.1.6 Модифікований алгоритм масштабування зображення

Згідно з рівнянням бікубічного згорткового ядра, найважчі обчислення, що алгоритм бікубічної інтерполяція робить - це операції піднесення до степені трійки і множення чисел з плаваючою точкою. Алгоритм бікубічної інтерполяції, розглянутий вище, може бути дещо модифікований як показано нижче рисунку 2.6.

```
float tx = (width_source)/width_dst;
float ty = (height_source)/ height_dst;

for(i=0; i<height_dst; i++)
 for(j=0; j<width_dst; j++)
  {
    x =(int)(tx*j);
    y =(int)(ty*i);

    dx=(float)(tx*j-x);
    dy=(float)(ty*i-y);

    select a0,a1,a2,a3 based on dx and dy
    from pre-computed coefficient  values

    select b0,b1,b2,b3 based on dx and dy
    from pre-computed coefficient  values

    for(jj=0;jj<=3;jj++)
      {
        d0 = P(y-1+jj,x-1);
        d1 = P(y-1+jj,x);
        d2 = P(y-1+jj,x+1);
        d3 = P(y-1+jj,x+2);
        C[jj]= -a0*d0 + a1*d1 + a2*d2 - a3*d3;
      }

   Cc = (-b0*C[0] + b1*C[1] + b2*C[2] - b3*C[3]);
   if(Cc>255) Cc=255;
   if(Cc<0) Cc=0;
   U(i,j) = Cc;
}
```

Рисунок 2.6 – Модифікований алгоритм бікубічної інтерполяції

Коефіцієнти $a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3$ тепер є функціями від dx і dy (різниця між інтерпольованими координатами і найближчими цілими

координатами є меншою), замість того аби залежати від пікселів. Цей трюк дозволяє нам обчислювати і зберігати значення коефіцієнтів для детермінованих значень dx і dy. Таким чином ми уникаємо велику кількість степенів трійки і операцій з числами з плаваючою точкою, які були в звичайній бікубічній інтерполяції. Даний процес зображено на рисунку 2.7.

$$C[jj] = a0 * d0 + a1 * d1 + a2 * d2 + a3 * d3$$
$$Cc = b0 * C[0] + b1 * C[1] + b2 * C[2] + b3 * C[3]$$
$$Where,$$
$$a0 = - dx /3 + dx * dx /2 - dx * dx * dx/6;$$
$$a1 = 1 - dx/2 - dx * dx/2 + dx * dx * dx/2;$$
$$a2 = dx + dx * dx/2 - dx * dx * dx/2;$$
$$a3 = - dx/6 + dx * dx * dx/6;$$
$$b0 = - dy /3 + dy * dy /2 - dy * dy * dy/6;$$
$$b1 = 1 - dy/2 - dy * dy/2 + dy * dy * dy/2;$$
$$b2 = dy + dy * dy/2 - dy * dy * dy/2;$$
$$b3 = - dy/6 + dy * dy * dy/6;$$

Рисунок 2.7 – Процес обчислення коефіцієнтів

Хоча результат алгоритму не сильно змінився (дещо погіршився), обчислювальна складність досить сильно спростилася. Це в основному завдяки тому, що ця модифікація обходить недолік важкої апаратної реалізації алгоритму, яка лежить у великій кількості операцій з числами з плаваючою точкою.

Модифікація бікубічного алгоритму інтерполяції з таблицею пошуку. Дистанція, на якій зображення дорівнює одиничній видістані, між двома сусідніми пікселями в початковому зображенні ділиться порівно на 16

інтервалів. Тобто такі: $[0, \frac{1}{16}), [\frac{1}{16}, \frac{2}{16}), [\frac{2}{16}, \frac{3}{16}), \ldots, [\frac{14}{16}, \frac{15}{16}), [\frac{15}{16}, 1)$. Коефіцієнти для всіх 16 значень dx і dy можна обчислити завчасно і зберігати у пам‘яті. Це зменшує обчислювальну складність процесу: визначення коефіцієнтів у реальному часі. Значення dx і dy порівнюються з одним із 16 інтервалів і значення коефіцієнтів $a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3$ обираються відповідно до завчасно обчислених значень.

2.1.7 Порівняльний аналіз неадаптивних алгоритмів інтерполяції

Неадаптивні алгоритми завжди зустрічаються з проблемою знаходження компромісу між трьома артефактами, що породжує масштабування — освітлення країв, розмиття та згладжування. Інтерполяція методом найближчого сусіда породжує ефект сильного згладжування, що призводить до зубчастих країв. Білінійна інтерполяція прибирає ефект згладжування, але призводить до помірного розмивання біля країв. Бікубічна інтерполяція створює невеликий ефект згладжування, розмиття та освітлення ребр. Інтерполяція Ланцоша дає якість зображення дуже близьку до бікубічної. Модифікована бікубічна інтерполяція має якість масштабованого зображення дещо гіршою, порівняно з бікубічною, тому що коефіцієнти апроксимують, аби зменшити обчислювальну складність.

В плані обчислень, метод найближчого сусіда є найменш витратним, оскільки він розглядає лише один піксель для інтерполяції. Білінійна використовує 4 діагональні пікселі для інтерполяції, тому вимагає більше обчислень ніж метод найближчого сусіда. Бікубічна і Ланцоша інтерполяції дуже складні в плані обчислень, тому що вони вимагають 16 і 32 пікселі відповідно для інтерполяції. Функція Ланцоша використовує sin функцію повторно, яка сама по собі вимагає велику кількість операцій додавання і

множення згідно з апроксимацією ряду Тейлора, тому при багаторазовому використанні може мати кращу швидкість, ніж бікубічна інтерполяція. Модифікована бікубічна інтерполяція дуже сильно зменшує кількість обчислювань, тому що коефіцієнти обчислюються використовуючи таблицю пошуку. На рисунках 2.8 та 2.9 результати масштабування.

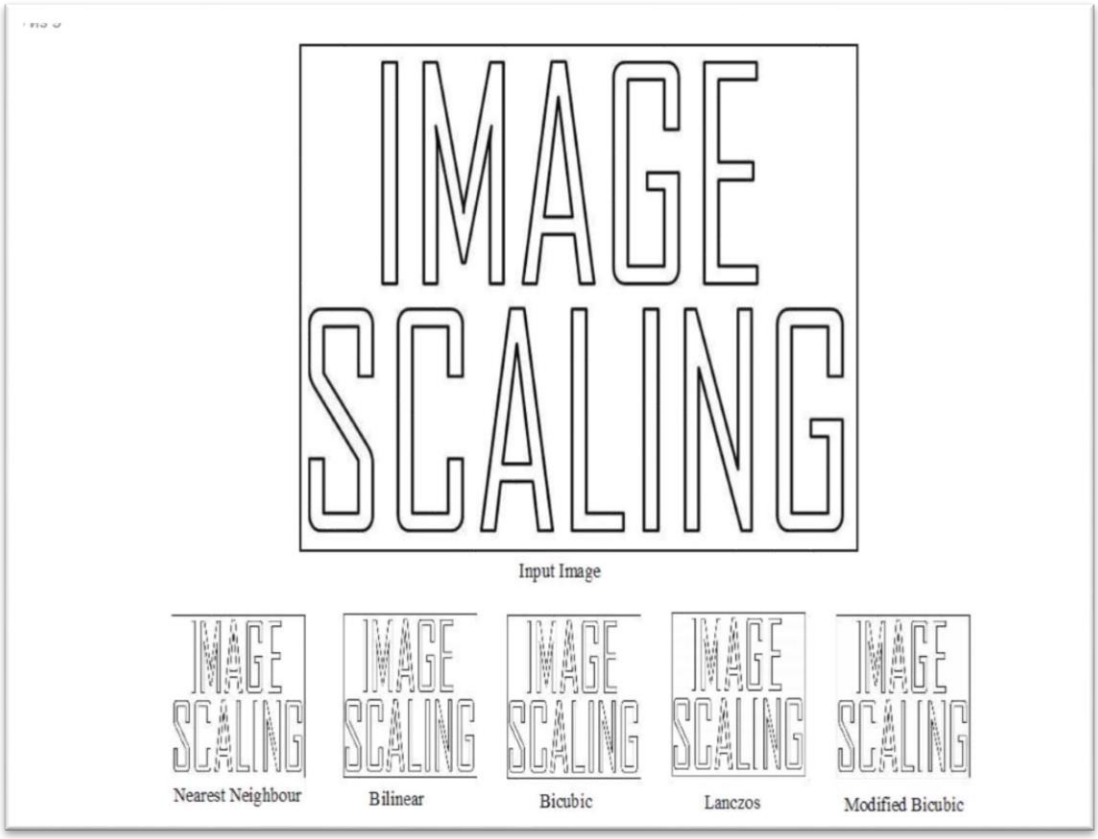


Рисунок 2.8 – Масштабоване зображення тексту

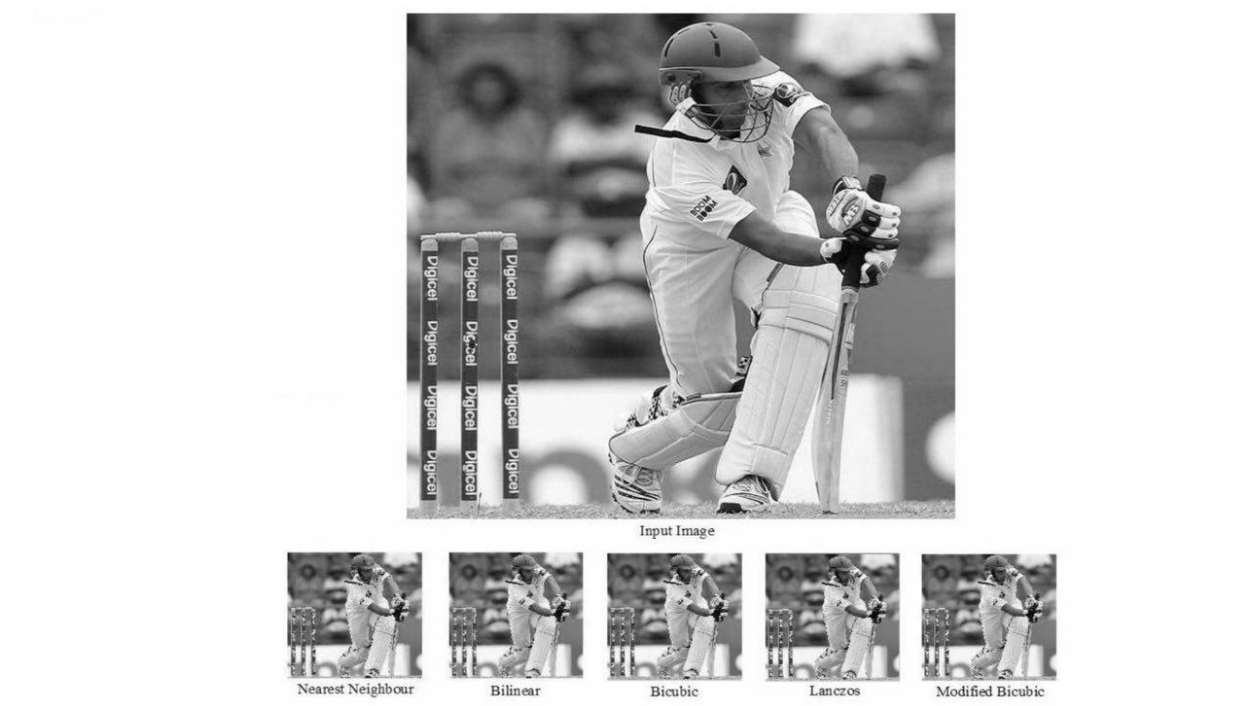


Рисунок 2.9 – Масштабоване реальне зображення

2.2 Критерії якості рішення задачі

Розробці методів інтерполяції зображення приділяють велику увагу, але набагато менше звертають на техніки оцінки цих методів. Декілька різних технік оцінки можна знайти у літературі зв‘язаною з інтерполяцією зображень. Тим не менш, через унікальні характеристики і широке застосування інтерполяції зображень, оцінка методів є дуже важливим аспектом. Більше того, різниця між евклідовими, афінними та трансформаціями проекції вимагає різні підходи до оцінки, в порівнянні із загальноприйнятими метричними методами.

В цій роботі розглянута схема класифікації критеріїв якості по двом категоріям. Перша категорія включає всі методи, які пов‘язані з критерієм точності алгоритму. Представлені методи можуть оцінювати функціональні

можливості інтерпольованого вихідного зображення з точки зору артефактів: розмиття, неперервність в ребрах, згладжування та ефекту освітлення країв. Нові методи оцінки використовують реальне оптично зняте зображення для порівняння. Детальніше, запропоновані методи вимірюють ефективність алгоритму через набір інтерпольованих зображень. Перше зображення інтерполюють цифровим шляхом тим методом, що ми оцінюємо, а друге отримують оптичним шляхом відповідно до початкового зображення. Друга категорія оцінки якості включає методи, що вимірюють функцію витрат самого алгоритму. Обчислювальні навантаження алгоритму і його вимоги до пам’яті можна оцінити цими методами. В кінці розглянемо приклади застосування і недоліки кожного з методів оцінки якості. Доречне використання обох категорій є важливим аспектом у визначенні того, який алгоритм інтерполяції справляється краще в конкретних умовах.

З попередніх розділів видно, що алгоритми інтерполяції відрізняються функцією інтерполяції, що робить їх чутливими в певних умовах, також змінюючи обчислювальну вартість. Вирішувати який алгоритм справляється краще з точки зору продуктивності і обчислювальної складності є важким процесом. Тому методи оцінки якості повинні бути чітко визначені, категоризовані і уважно використані.

Методи оцінки

Методи оцінки класифіковані на дві категорії. Перша категорія вивчає продуктивність алгоритмів базуючись на критерію точності. І алгоритм інтерполяції, і вихідне інтерпольоване зображення піддаються аналізу, що уточнює продуктивність кожного алгоритму. Тим не менш, оцінка вихідного зображення вимагає конкретного застосування алгоритму у визначеній трансформації зображення. В цій роботі використовується операція масштабування. Крім того, для більш ретельного аналізу, візьмемо до уваги не тільки процес збільшення зображення, який частіше розглядають, але й процес зменшення.

Раніше, всі методи характеризували різні алгоритми інтерполяції просто шляхом обчислення точності на основі порівняння інтерпольованих зображень без використання будь-яких попередніх знань про існування ідеально інтерпольованого зображення. В цій категорії, пропонується розглянути оціночні методи, що використовують оптично трансформовані зображення для порівняння з інтерпольованими. Цей метод покладається на той факт, що оптично трансформоване зображення відповідає ідеальному для інтерпольованих.

Тим не менш, не всі характеристики алгоритму інтерполяції можна оцінити аналізом продуктивності. Друга категорія працює напряму з самим алгоритмом, розглядаючи його складність і використання пам’яті. Ці властивості можуть допомогти обрати зручний алгоритм під конкретну задачу. Наприклад, мобільні телефони мають обмежені апаратні ресурси, такі як пам’ять і потужність обробки; тому реалізаціє витончених і складних алгоритмів неефективна. Аналіз споживання енергії алгоритмом також є важливим етапом, особливо для мобільних девайсів.

В цій частині використовуються 3 найбільш популярних алгоритми інтерполяції зображення для демонстрування роботі методів оцінки якості: метод найближчого сусіда, білінійна та бікубічна інтерполяції, що були описані раніше.

2.2.1 Аналіз продуктивності, метрика PSNR

Тести для порівняння методів масштабування в цій категорії мають включати як синтетичні, так і реальні зображення. Коефіцієнт масштабування треба обирати і з цілих, і нецілих, щоб показати різницю і слабкості алгоритмів. Крім того, повна процедура оцінки повинна включати

аналіз декількох різних зображень, а кінцевий результат має бути усереднений. В деяких випадках, можна аналізувати мінімальну або максимальну помилку. В наступних методах, всі використані зображення є чорно-білими і використовують однакові рівняння для оцінки якості. Тим не менш, всі рівняння можна розширити до будь-якого простору кольорів.

Наприклад, пікове відношення сигнал/шум (PSNR) інтерпольованого кольорового зображення, позначеного як $PSNR_p$:

$$PSNR_p = (4 \times PSNR_Y + PSNR_U + PSNR_V)/6,$$

де $PSNR_Y, PSNR_U, PSNR_V$ — це відповідні значення PSNR (4:1:1) компонент Y, U і V відповідно інтерпольованого зображення.

2.2.2 Аналіз Фур‘є

Аналіз Фур‘є порівнює ядро інтерполяції з sinc функцією, що є ідеальним реконструктором обмеженого сигналу. Рисунок 2.10 зображує величину трансформації Фур‘є для кожного ядра інтерполяції. Вертикальна переривчаста лінія визначає точку відсічення для $f = \frac{1}{2}$.

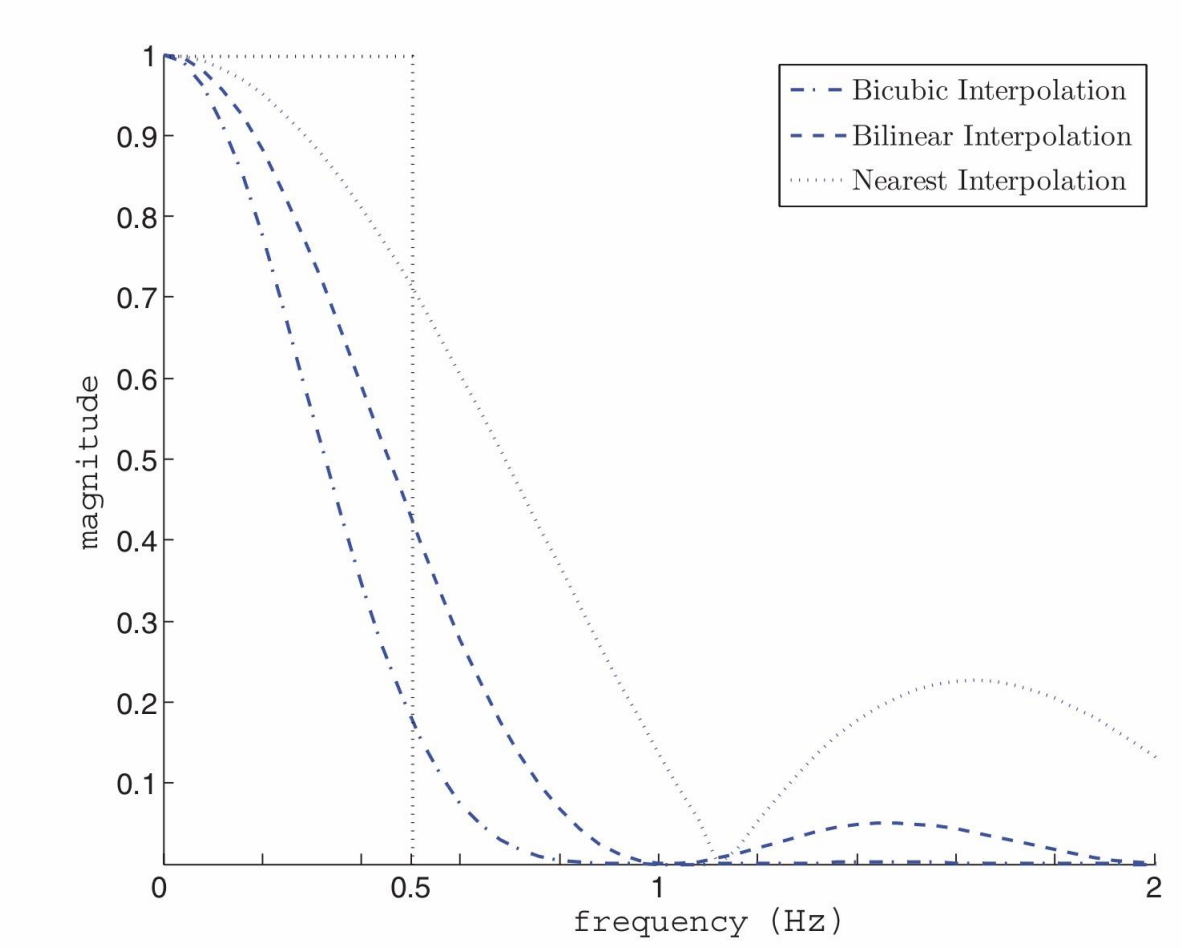


Рисунок 2.10 – Величина трансформації Фур'є у лінійному масштабі для оцінки продуктивності смуги пропуску

Величина трансформації Фур'є для кожного ядра побудована в проміжку $0 \leq f \leq 2$ або $0 \leq \omega \leq 4\pi$, де $\omega = 2\pi f$. Інтервал $0 \leq f \leq 1/2$ називають смуга пропуску, а $f = 1/2$ чи $\omega = \pi$ — точка відсічення. Для оцінки якості, у смузі пропуску нас цікавить дві характеристики, які тісно зв'язані з відхиленням від ідеального реконструктора: як швидко функція починає зменшуватися і значення при якому, падіння починається. У смузі подавлення, найкраща продуктивність виражається діапазоном, що є найближчим до нуля, який, крім цього, представляє найменші схили. Відхилення у смузі пропуску спричиняє розмивання, великі діапазони хвилястості і бокові пелюстки у смузі подавлення перетворюються у

згладжування і смуги. У аналізі Фур'є, вибірка інтерпольованого (неперервного) зображення еквівалентна інтерполяції (дискретного) зображення створеного інтерполюючою функцією.

2.2.3 Метод оцінювання якості інтерполяції

Цей метод оцінки вимагає повторного застосування алгоритму інтерполяції зображення. Використовуючи початкове вхідне зображення $I(x, y)$, повторюючи виконуємо алгоритм інтерполяції, що оцінюється, для того, щоб отримати кінцеве інтерпольоване зображення $\hat{I}(x, y)$. Фінальне зображення має мати такий же розмір і кількість каналів, як і початкове. Потім якість інтерполяції оцінюється шляхом різниці між пікселями до і після вдалої інтерполяції. Оскільки всі види трансформації зображення можна використовувати для цього методу, то цей критерій є ідеальним для візуалізації помилки інтерполяції, і тому його використовують найчастіше як метод оцінки.

Далі, застосуємо методи оцінки для трансформації масштабування, використовуючи корінь середньоквадратичної помилки (RMSE) і пікове відношення сигнал/шум (PSNR) для вимірювання помилки інтерполяції. При даному зображенні $I(x, y)$, процедура тестування для оцінки алгоритму інтерполяції у трансформації масштабування має наступний вигляд. Метод вимагає дві трансформації масштабування застосованих одним і тим же алгоритмом інтерполяції, як показано на рисунку 2.11.

Рисунок 2.11 – Процес збільшення зображення, з наступним його зменшенням, що використовується в оцінки якості інтерполяції

Перша трансформація - це збільшення зображення коефіцієнтом масштабування f. Потім збільшене зображення зменшують тим самим коефіцієнтом f. Кінцеве зображення $\hat{I}(x,y)$ має ту саму кількість каналів і роздільну здатність, що і початкове. Крім того, для ще більш точної оцінки, користувач може використовувати декілька коефіцієнтів масштабування, щоб отримати середній результат. Це може стати корисним при оцінці алгоритму інтерполяції, який показує оптимальні результати лише при певних умовах. Формула для обчислення $RMSE$ між початковим зображенням $I(x,y)$ і інтерпольованим $\hat{I}(x,y)$ наступна:

$$RMSE = \left(\frac{1}{m \times n} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} (\hat{I}(x,y) - I(x,y))^2\right)^{1/2},$$

де зображення має розмір $m \times n$.

Метрику $PSNR$ можна тісно зв‘язати з $RMSE$. $PSNR$, що вимірюється в децибелах (dB). Формула для її обчислення між початковим зображенням $I(x,y)$ і інтерпольованим $\hat{I}(x,y)$ має вигляд:

$$PSNR = 20 \times \log_{10}\left(\frac{MAX_I}{RMSE}\right),$$

де MAX_I - це максимальне значення пікселля в зображенні.

Коли пікселі виражаються 8 бітам, то це значення дорівнює 255. Аби уникнути ефекту границі, рекомендується витягти центроване субзображення перед обчисленням. Тим не менш, актуальне значення не дуже важливе, і тільки порівнюючи з іншими результатами інтерполювання дає міру якості. Результати представлені у таблиці 2.1 демонструють метрики $RMSE$ і $PSNR$ для зображення фотографа після трансформацій збільшення і зменшення.

Таблиця 2.1 – Порівняння збільшеного, а потім зменшеного зображення фотографа і оригіналу

	Метод інтерполяції		
Метрика	Сусіда	Білінійна	Бікубічна
PSNR (dB)	70.50	73.22	74.43
RMSE	0.0761	0.0556	0.0484

Обернена процедура, спочатку зменшення, а потім збільшення, показана на рисунку 2.12.

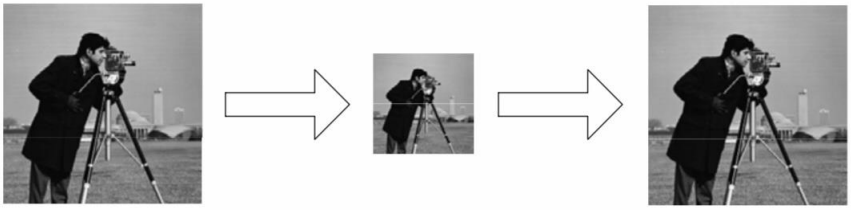


Рисунок 2.12 – Процес зменшення зображення, з наступним його збільшенням, що використовується в оцінки якості інтерполяції

Якщо спочатку може здатися, що обернена операція має дати таку саму помилку, як попередня процедура, то виявляється, що помилки

відрізняються, оскільки багато алгоритмів інтерполяції мають різну поведінку при збільшенні та зменшенні зображення. Особливо у випадку збільшення зображення після зменшення, значення $RMSE$ для масштабованих зображень методом найближчого сусіда дорівнює нулю, тому що точки із вибірки повертаються назад до початкового зображення.

Головна особливість цього методу - це те, що багато різних помилок інтерполяції можна оцінити у результатах. Наприклад, алгоритм, що має неперевершену продуктивність у процесі масштабування, але неадекватні результати у повороті зображення, будуть представляти середні результати якості.

2.2.4 Оцінка якості інтерполяції зміни інтенсивності при ребрах

Наступний метод оцінки пов‘язаний з якістю зображення і особливо з поведінкою інтерполяції відносно ребр. Аналогічні вимірювання можна отримати з аналізу Фур‘є, що був описаний раніше. Крім того, цей метод можна використовувати до алгоритмів інтерполяції, які не виражені базисною функцією. Наприклад для оцінки алгоритмів інтерполяції основаних на площі, або інтерполяція нейронними мережами і інтерполяція основана на локальних особливостях. Хоча існують різні типи ребр у зображенні, ребра, при яких змінюється інтенсивність пікселів, візуально більше вирізняються серед інших. Тому збереження гостроти і неперервності цих країв є головною метою. Ціллю цього методу є оцінка того, як алгоритм зберігає гостроту і неперервність початкових перехідних ребр. Щоб оцінити це, синтетичне вхідне зображення, що містить лише дві різні області, інтенсивністю 225 і 25, геометрично трансформують. У випадку масштабування, зображення збільшують і в залежно від алгоритму, що був

використаний, небажана зона може вирости прогресивним переходом з 225 до 25. В загальному випадку, швидший перехід дає кращу продуктивність. Рисунок 2.13, ілюструє зміну інтенсивності з 255 до 25.

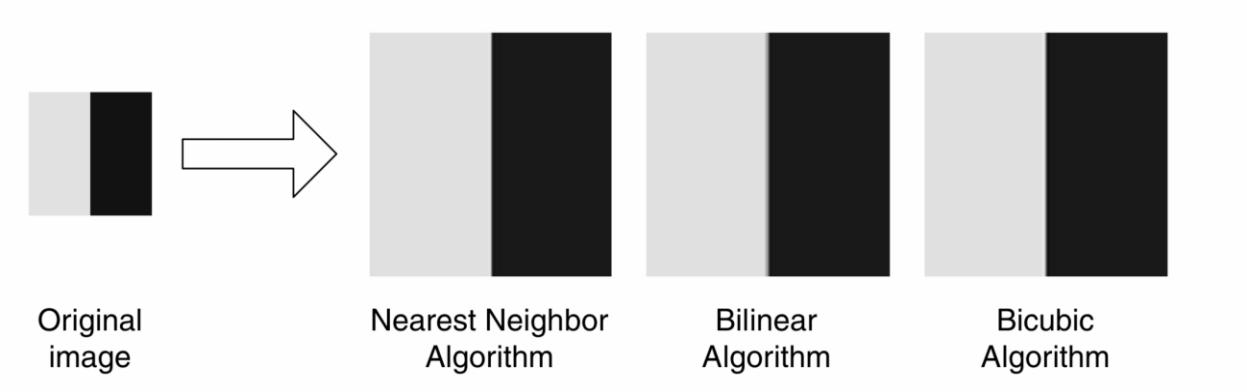


Рисунок 2.13 – Порівняння продуктивності інтерполювання синтетичного зображення з інтенсивністю 225 і 25

Результати краще видно на рисунку 2.14 - діаграмі продуктивності.

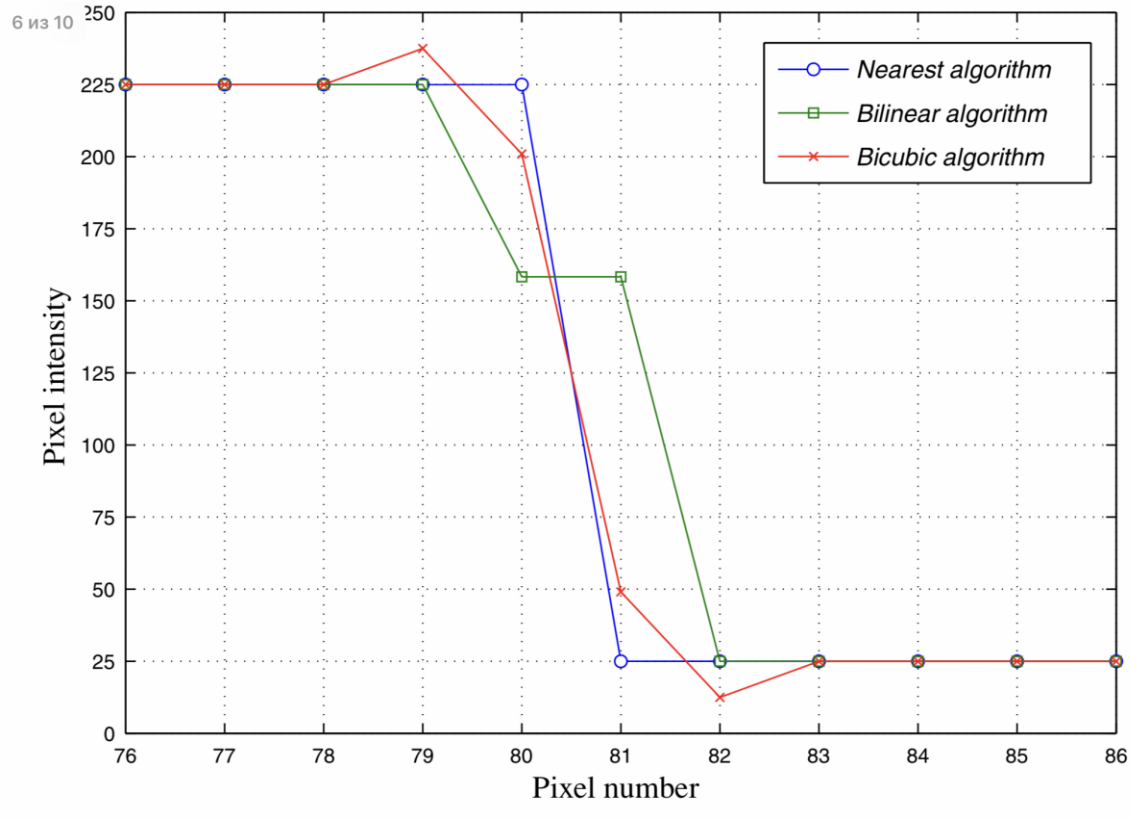


Рисунок 2.14 – Діаграма продуктивності

Для білінійної інтерполяції, з‘являється єдина нова роздільна зона, спричинена скромним фільтром низьким частот, який використовує білінійна інтерполяція. Нові роздільна зона ще більше вирізняється у бікубічної інтерполяції через згладжування у смузі пропуску. В цьому завданні алгоритм найближчого сусіда показав найкращі результати, оскільки він відтворює якраз інтерпольовані пікселі. Проте, ця властивість також приводить до сильного ефекту згладжування, що пов‘язаний в інтерполяцією методом найближчого сусіда.

Цей метод оцінки також широко використовують у методах інтерполяції текстів, де вивчаються велико частотні вхідні зображення.

Для розрахунку помилки ядра можна порівнювати $f(\mathrm{x})$ і $f_h(\mathrm{x})$, використовуючи наступне рівняння:

$$\epsilon^2(h) = \int_{-\infty}^{\infty}(f(\mathrm{x}) - f_h(\mathrm{x}))^2 dx_1 dx_2 \ldots dx_q \forall \mathrm{x} = (x_1, x_2, \ldots, x_q) \in R^q,$$

де $h > 0$ – крок вибірки;

$f_h(\mathrm{x})$ - функція, що інтерполюємо в певному інтервал, позначеному як h.

Ця різниця між $f(\mathrm{x})$ і $f_h(\mathrm{x})$ описує наскільки швидко функія, що інтерполюємо, f_h сходиться до реальної функції f, коли крок стає все менше і менше.

2.2.5 Оцінка з реальними зображеннями

Зараз розглянемо метод, який полегшує атрибути необробленого фото, оскільки сучасні пристрої обробки зображень можуть виконувати бажані трансформації без використання цифрової обробки зображення. Більш точно, можна відзняти фотографію ближче, повернути фотографію і багато хого інше. Використовуючи це як апріорне знання, ми в кінці кінців можемо отримати набір трансформованих оптичним та цифровим чином зображень і застосувати методи оцінки на них. В залежності від трансформації, ми повинні слідувати певним процедурам. Для трансформації зображення необхідно використовувати оптичний зум пристрою обробки зображення.

Фокусна відстань об‘єктивна визначається як дистанція в міліметрах від оптичного центру об‘єктива до точки фокусу, яка знаходиться на твердотілому датчику. Зміна фокусної відстані дозволяє користувачеві наближатися ближче до суб‘єкта або відходити від нього. Коефіцієнт оптичного масштабування обчислюється за наступною формулою:

$$Opticalzoomingfactor = \frac{f_{max}}{f_{min}},$$

де f_{max} відповідає за максимальну фокусну відстань;

f_{min} - за мінімальну.

Сучасні пристрої обробки зображення забезпечують точну оцінку оптичного коефіцієнту масштабування і виводять його на екран користувача. Використовуючи попередню властивість, можна отримати зображення з різними налаштуваннями оптичного масштабування як показано на рисунку 2.15.

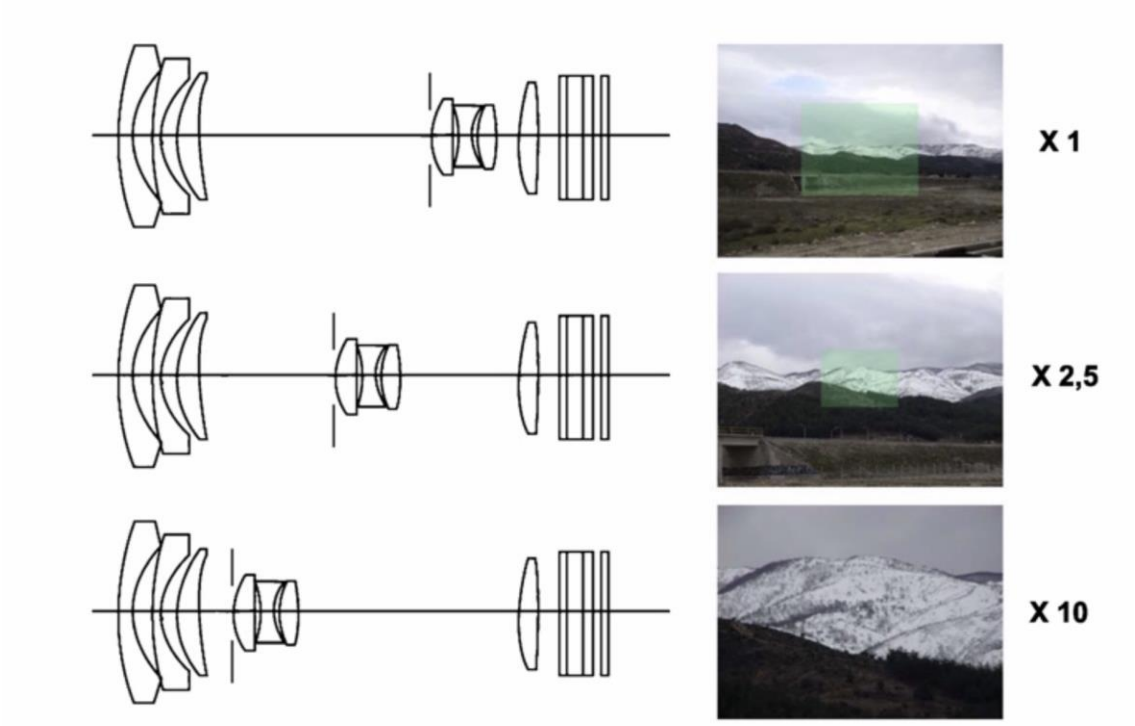


Рисунок 2.15 – Різні настройки системи оптичного зуму: x1; x2; x10

Більш детально, з цією процедурою можна створити базу даних, яка містить у собі оригінальне зображення фігури 2.16(a) і оптично збільшених зображень у 2.16(b) з відомим коефіцієнтом масштабування. Потім, за

допомогою алгоритму, що ми вивчаємо, отримуємо вже масштабоване зображення з відомим коефіцієнтом, як продемонстровано на малюнку 2.16(c).

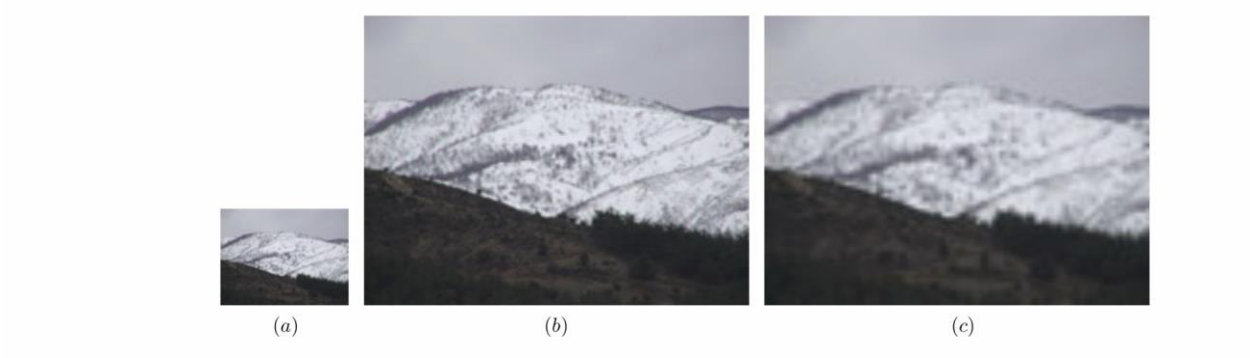


Рисунок 2.16 – Оригінальне зображення і два зображення, які порівнюються: (a) оригінальне зображення; (b) оптично масштабоване зображення (ідеальне); (c) зображення масштабоване цифровим шляхом

Описаний метод створює ідеальне зображення для порівняння з алгоритмами, що ми оцінюємо. Всі майбутні тестові зображення будуть інтерпольовані цифровим шляхом із оригінального зображення, як показано раніше. Можна створити онлайн базу даних із схожими наборами зображень. Якісно оцінити їх можна за допомогою таких метрик, як $RMSE$ чи $PSNR$, що були розглянуті раніше. Результати порівнянь для масштабування зображення з попереднього прикладу продемонстровані у таблиці 2.2.

Таблица 2.2 – Порівняння між оптичного масштабованим зображенням гори (ідеальним) і масштабованим цифровим шляхом різними алгоритмами інтерполяції

	Метод інтерполяції		
Метрика	Сусіда	Білінійна	Бікубічна
PSNR (dB)	67.28	67.39	67.56
RMSE	0.1102	0.1089	0.1067

Базуючись на результатах, бікубічна інтерполяція перевершує білінійну інтерполяцію та метод найближчого сусіда.

2.2.6 Суб’єктивні спостереження

Судження, що базуються на суб’єктивних спостереження приймаються і широко використовуються в літературі. Досвідчені спостерігачі можуть надавати корисні вимірювання, особливо коли кількість зображень, що оцінюємо і результати, обробляють за допомогою аналітичних методів. На рисунку 2.17, продемонстровані результати візуальних спостережень масштабування зображення фотографа з коефіцієнтом масштабування 1.5, для того щоб надати суб’єктивного порівняння методу найближчого сусіда, білінійної інтерполяції і бікубічної.

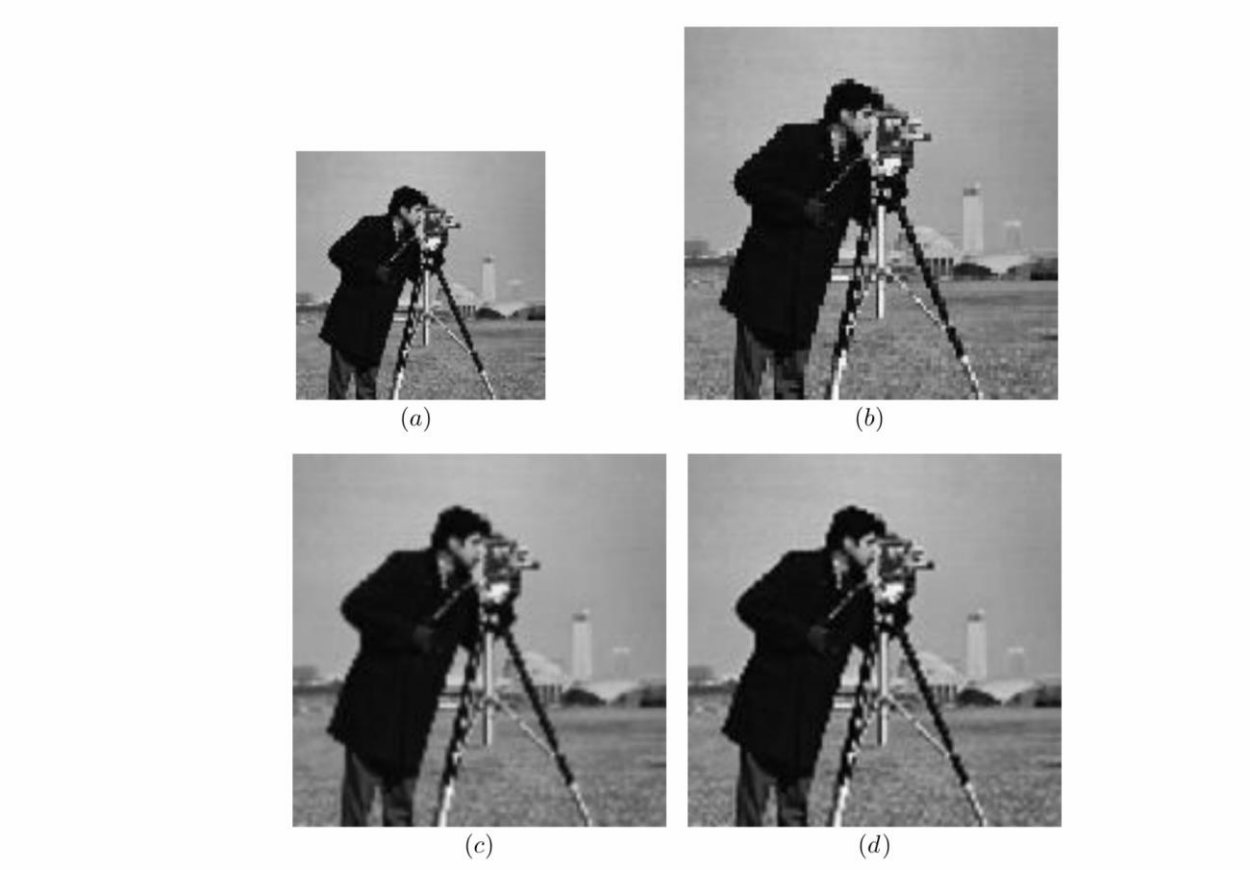


Рисунок 2.17 – Візуальні результати інтерполяції для зображення фотографа: (a) оригінал; (b) метож найближчого сусіда; (c) білінійна інтерполяція; (d) бікубічна інтерполяція

Аналіз витрат

Друга категорія якості включає методи аналізу вартості самого алгоритму інтерполяції. Обчислювальна складність і вимоги до пам‘яті алгоритму оцінюються у цій категорії. Такі методи забезпечують загальний критерій і головним чином вказують на застосування кожного алгоритму інтерполяції в певних додатках, таких як реалізація у реальному часі.

2.2.7 Обчислювальна складність методу масштабування

Визначення складності алгоритму є важливою і важкою задачею у літературі. Тим не менш, деякі методи були застосовані в роботах зв‘язаних з інтерполяцією зображень. Кількість операцій необхідна для інтерполяції пікселів є широко прийнятним методом. Це число включає як і операції, що вимагає згортка, так і ті, що необхідні для обчислення базисної функції. Згортка $N \times N$ потребує N^2 операцій множення і $N^2 - 1$ операцій додавання. Таблиця 2.3 вказує на кількість операцій на піксель для методу найближчого сусіда, білінійної та бікубічної інтерполяцій.

Таблиця 2.3 – Кількість операцій на піксель

	Метод інтерполяції		
Метрика	Сусіда	Білінійна	Бікубічна
Додавання	2	16	22
Множення	0	18	29

Тим не менш, цей метод призначений лише для простих і зрозумілих реалізацій алгоритмів і його застосування обмежене у складних алгоритмах. Додаткові операції вимагають реалізації, які використовують попереднє фільтрування і базисні функції, що включають такі математичні функції як $sine$ або $cosine$.

2.2.8 Порівняння часу виконання

Час виконання використовували у для оцінки обчислювальної вартості алгоритму. Порівняльні тести виконують для декількох трансформацій на комп’ютері з відомими характеристиками. Порівняний час виконання надає міру техніки з найменшою складністю. Для ще кращого порівняння, детальні статистики, такі як цикли внутрішнього ядра, кількість непослідовних циклів і кількість програмних інструкцій, що були виконані можна зібрати із спеціально виділеного емулятора процесора. Підсумкові діаграми та графіки, що включають і продуктивність, і метрики вартості також дуже корисні, оскільки вони надають загальний аналіз для кожного алгоритму в порівнянні з іншими. На рисунку 2.18 зображено наступне: горизонтальна вісь відповідає за час виконання трансформації зображення, а вертикальна вісь - якість інтерполяції, за допомогою метрики $PSNR$.

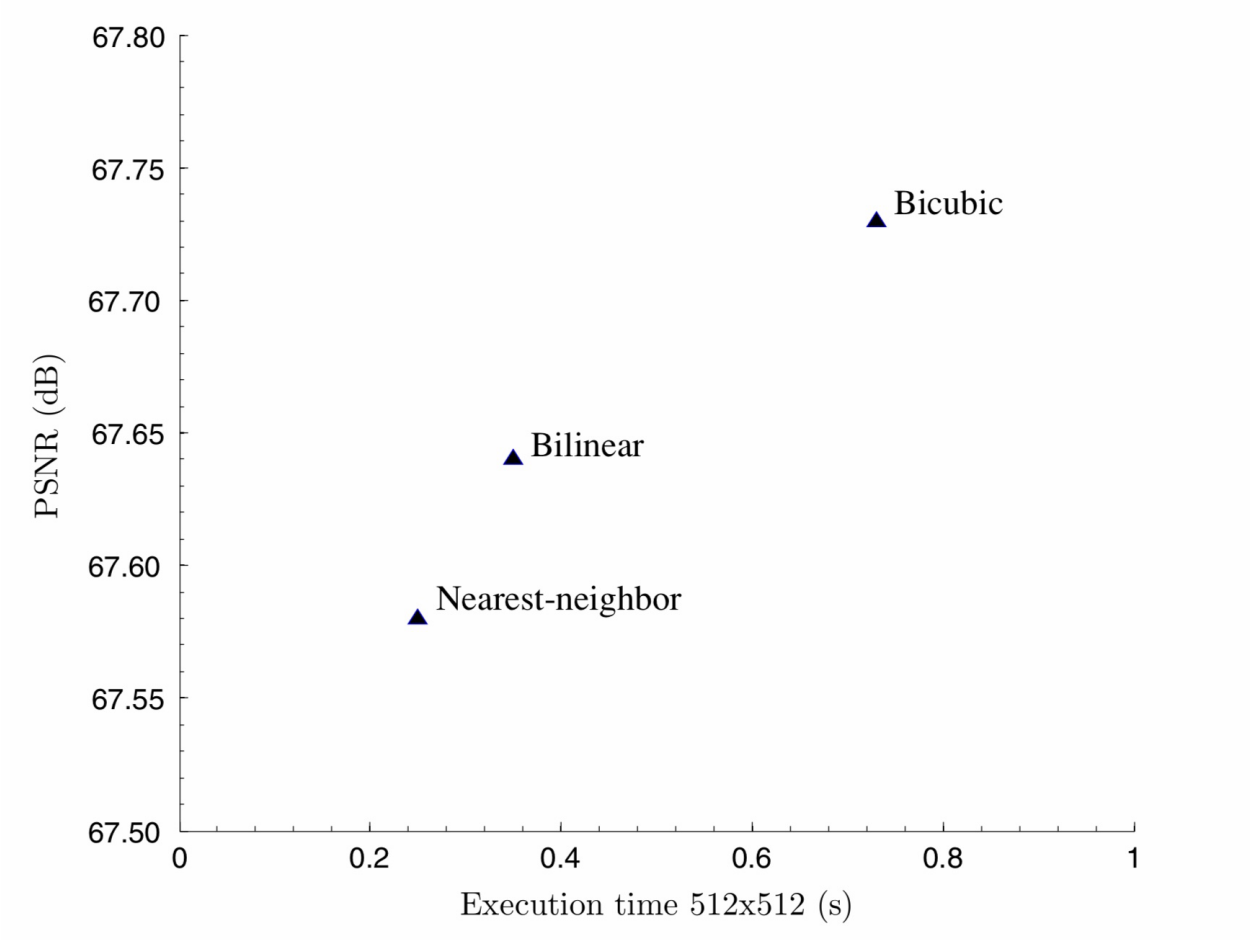


Рисунок 2.18 – Графік продуктивності

З графіку зрозуміло, що бікубічна інтерполяція демонструє перевагу по метриці $PSNR$ відносно інших методів, разом з цим і має найбільший час виконання. Крім того, необхідно зауважити, що за останні роки збільшився попит в дослідженнях у сфері використання елементу обробки графіки (GPU) для алгоритмів обробки зображень замість обчислень загального призначення. GPUs розроблені спеціально аби виконувати обмежену кількість операцій на дуже великих масивах даних. Вони зазвичай мають більше одного обробляючого потоку працюючи паралельно один з одним. За цієї причини, деталізоване порівняння повинно бути точно визначеним в залежності від типу процесору, що використовується.

2.2.9 Вимоги до використання пам‘яті

В той час як обчислювальні потреби позначаються більше за все на навантаженні процесору, то потребу в ресурсах пам‘яті не можна оцінювати такою ж метрикою. Апарати обробки зображень такі як мобільні телефони, цифрові камери і планшети мають обмежену кількість вбудованої пам’яті для обробки зображень. Тому, алгоритми інтерполяції мають бути розроблені і оптимізовані для такого застосування, щоб виконувати обмеження на продуктивність у реальному часі. Оскільки більшість технік використовують оператори згортки, то вимоги до використання пам’яті жорстко зв’язані з кількістю вхідних пікселів. Згортка $N \times N$ потребує $N^2 + 1$ одиниць пам’яті. Крім того, кількість вбудованої пам’яті, що потребується, тісно зв’язана з базисною функцією. Наприклад, білінійна інтерполяція використовує чотири сусідніх пікселі для кожного обрахунку. Це значить, що незалежно від трансформації зображення, операцію можна виконати лише з дворівневим буфером. Тому, 8-бітне кольорове вхідне зображення з роздільною якістю 640×480 пікселів вимагає $8 \times 3 \times 640 \times 2 = 3.75KB$ внутрішньої пам’яті. Однак, бікубічна інтерполяція, яка використовує 16 вхідних пікселів, потребу чотирирівневого буфера для кожної операції, що дорівнює $8 \times 3 \times 640 \times 4 = 7.5KB$ внутрішньої пам’яті.

Тести на використання пам‘яті у алгоритмах інтерполяції можна реалізувати з допомогою пакету розробників процесору. Пакет дозволяє користувачеві назначати відповідний шар пам’яті, що має спеціальні характеристики. Ця можливість може допомогти користувачам симулювати певні моделі пам’яті навіть для завдань з обмеженою кількістю пам’яті. Алгоритми, написані на мові С, компілюються в процесорний код і відображаються у пам’яті. Ретельно симульований процесор зчитує і виконує машинний код для тестування пам’яті. Процес можна легко модифікувати

для різних цілей і кількох процесорів. Вимірювання симуляції дають всі деталі розподілу пам’яті алгоритму, що оцінюється, такі як кількість звернень до пам’яті поза чипом, кількість звернень до пам’яті на чипі і час очікування. Алгоритм з з меншою кількістю звернень і часом очікування показує найкращу продуктивність.

2.2.10 Оцінка потужності на основі метрики енергоспоживання

Метод оцінки потужності надає метрику енергоспоживання, якого потребує алгоритм інтерполяції. В системах, що впроваджують багатовимірні потоки сигналів, такі як зображення або відео послідовність, було показано, що більшість енерговитрат зв’язані з взаємодією із пам’яттю поза чипом. Ці взаємодії включають використання пам’яті для інструкцій і даних. Очевидно, що споживання енергії пам’яті для даних тісно пов’язана з вимогами до пам’яті кожного алгоритму інтерполяції. І використання енергії пам’яті для інструкцій еквівалентна обчислювальній складності алгоритму. Відносно малі вимоги використання пам’яті алгоритмів інтерполяції зображення також показують малий вплив на загальне використання енергії. Головним споживачем є пам’ять з інструкціями.

Початковою точкою цього методу оцінки є опис інтерполяції на мові високого рівня. Потім, використовуючи пакет розробника процесора, можна отримати кількість інструкцій і циклів ядра, які потребує алгоритм. Після цього, використовуючи оцінювач енергії і продуктивності моделі можна отримати прогноз енерговитрат алгоритму, що оцінюється. Моделі, що описують характеристики енергії і продуктивності шарів пам‘яті мають бути реальними моделями пам‘яті, які базуються на властивостях моделей промислових постачальників. Підсумовуючи, в задачах, де

енергоспоживання є критичним, алгоритми інтерполяції зображень мають мати стратегії з малої кількості звернень до пам’яті, оскільки вони зменшують використання енергії через трафік пам’яті.

Хоча інтерполювання зображень широко використовується в обробці зображень, кожне завдання має свої особливі вимоги. Тому, в залежності від застосування, необхідно використовувати відповідні критерії оцінки. Крім того, чи метод оцінки є оптимальний чи ні пов’язано з технікою інтерполювання, змістом зображення і геометричною трансформацією. Описана категоризація методів оцінки розділяє їх на дві частини: вимоги до продуктивності і витрат.

Для таких задач як медична візуалізація, де найбільш важливою особливістю є точність вихідного зображення, найкращими методами будуть ті, що показують найкращі результати у категорії продуктивності. Більш точно, оскільки змістом є низькочастотні зображення, то необхідно використовувати критерій якості інтерполяції, тому що він надає загальну міру якості. Для задач, що займаються високочастотними зображеннями, такі як обробка тексту чи документу, аналіз Фур’є чи якість обробки зміни інтенсивності при ребрах метрикам треба надавати перевагу. Однак, складні алгоритми інтерполяції, які не можна виразити єдиною базисною функцією такі як нейронні мереж чи нечіткі підходи можна оцінити лише методом оцінки якості інтерполювання переходів на краях.

В порівнянні з якістю виконання, що вимагає задача, аналіз витрат необхідний у задачах з обмеженими ресурсами. В залежності від критичного ресурсу задачі, необхідно застосовувати відповідні методи оцінки. Перед впровадженням алгоритму інтерполяції в мобільний девайс, необхідно провести аналіз енергоспоживання, тому що в цій задачі найбільш критичним фактором є споживання енергії, яка дорівнює збільшенню використання пам’яті. Аналіз пам’яті даних і обчислювальної складності необхідна у

задачах, що інтерполюють зображення з величезною роздільною здатністю, таких як GIS задачі.

2.3 Згорткові нейронні мережі

Згорткові нейронні мережі – це клас відносно нових архітектур, їх основна ідея полягає в тому, щоб повторно використовувати одні й ті ж частини нейронної мережі для роботи з різними малими, локальними частинами входів. Ідея згорткових нейронних мереж мотивована дослідженнями про частину кори головного мозку, яка відповідає за зір. Згорткові нейронні мережі знайшли багато практичних використань. Основним застосуванням залишається обробка зображень. Першою згортковою мережею можна вважати неокогнітрон, яка запропонована К.Фукусімою в 1979 – 1980 роках. В ній не використовувався ані градієнтний спуск, ані навчання з учителем. У сучасній формі ці мережі з’явилися в роботах Яна ЛеКуна в кінці 1980-х років.

2.3.1 Поняття згортки в згорткових нейронних мережах

У всіх нейронних мережах присутні афінні перетворення. В кожному шарі повнозв’язної мережі повторюється одна й та ж операція, а саме, вхідний вектор множиться на матрицю ваг, до результату додається вектор вільних членів; до результату застосовується деяка функція активації.

Однак, багато типів даних мають свою власну заздалегідь відому внутрішню структуру. Наприклад, зображення, яке часто представляють у

вигляді масиву векторів чисел. У випадку чорно-білого зображення це буде масив інтенсивностей, у випадку кольорового – масив векторів з чисел, які означають інтенсивності основних трьох кольорів, а саме зеленого, червоного і чорного. Узагальнення такої внутрішньої структури виглядає наступним чином:

- початкові дані представляють багатовимірний масив, «тензор»;
- присутня одна або більше осей серед розмірностей цього масиву, порядок вздовж яких грає важливу роль; наприклад, це може бути розташування пікселів у зображенні, порядок слів або символів у тексті;
- інші осі означають «канали», якими описуються властивості елементів за вказаною вище підмножиною осей; наприклад, три компоненти для зображень.

Ці додаткові знання про структуру даних безпосередньо враховуються при навчанні згорткових нейронних мереж.

Основна ідея згорткової мережі в тому, що обробка частини зображення часто може відбуватися незалежно від конкретного розташування цієї частини. Розпізнати зображення обличчя, наприклад, можна на сильно обрізаній фотографії, де немає нічого, крім самого обличчя – це локальна задача, її розв’язок не залежить від конкретного положення ділянки з об’єктом всередині великого зображення.

Згорткова мережа працює наступним чином: вхідне зображення покривається невеликими вікнами (наприклад 5x5 пікселів) і ознаки отримуються в кожному такому вікні невеликою нейронною мережею. Ключовим моментом є те, що в кожному вікні отримуються одні й ті ж ознаки, тобто маленька нейромережа буде одна, входів в неї буде 5x5, а з кожного зображення для неї буде дуже багато різних входів. Далі результати цієї нейронної мережі знову представляються у вигляді «зображення», замінюючи вікна 5x5 їх центральними пікселями, і до неї застосовується

другий згортковий шар з іншою маленькою нейронною мережею і т.д. В кожному згортковому шарі буде небагато параметрів порівняно з повнозв'язними мережами.

Зазвичай кольорові зображення, які є вхідними для нейронної мережі, представлені у вигляді декількох прямокутних матриць, кожна з яких задає рівень одного з кольорових каналів в кожному пікселі зображення. Припустимо, що кожному пікселю вхідного зображення ставиться у відповідність деякий тензор, а його компоненти називаються каналами.

Такі ж матриці отримуються і після згорткового шару: вони мають просторову структуру, яка відповідає початковому зображенню, однак каналів може бути більше. Значення кожної ознаки, які отримано з вікон в початковому зображенні, тепер представляють цілу матрицю. Кожна така матриця називається *картою ознак* (feature map).

Під згорткою розуміють лінійне перетворення вхідних даних особливого виду. Нехай x^l – карта ознак в шарі з номером l. Тоді результат на наступному шарі для двовимірної згортки з ядром розмірності $2d+1$ і матрицею ваг W розмірністю $(2d+1)\times(2d+1)$ наступний:

$$y_{i,j}^l = \sum_{-d\le a,b\le d} W_{a,b} x_{i+a,j+b}^l,$$

де $y_{i,j}^l$ – результат згортки в шарі з номером l;

$x_{i,j}^l$ - її вхід, тобто результат попереднього шару.

Це означає, що для отримання (i,j) компоненти наступного шару застосовується лінійне перетворення до квадратного вікна попереднього шару, пікселі з вікна скалярно множаться на вектор згортки.

Властивості згортки:

- згортка зберігає структуру вхідних даних, оскільки застосовується окремо до кожної їх ділянки;
- має властивість розрідженості, оскільки значення кожного нейрону шару залежить лише від невеликої долі вхідних нейронів. В повнозв'язній мережі, наприклад, кожний нейрон залежить від усіх нейронів попереднього шару;
- згортка багаторазово використовує одні й ті ж ваги, оскільки вони повторно застосовуються до різних ділянок вхідного зображення.

Після згортки в нейронній мережі майже завжди йде нелінійність:

$$z_{i,j}^{l} = h\left(y_{i,j}^{l}\right),$$

де в якості функції h часто використовують ReLU.

2.3.2 Субдискретизація в згорткових мережах

Ідея субдискретизації (pooling, subsampling) базується на тому, що в згорткових мережах зазвичай виходять з припущення, що факт наявності або відсутності тієї чи іншої ознаки є набагато важливішим за точні її координати. Наприклад, при розпізнаванні обличчя згортковою нейронною мережею нам важливіше дізнатися, чи присутнє на фотографії обличчя і кому воно належить, ніж знати з якого пікселя воно починається і на якому закінчується. Тому здійснюють «узагальнення» шуканих ознак, втративши частину інформації щодо їх місця розташування, при цьому зменшивши розмірність.

Зазвичай в якості операції субдискретизації в кожній локальній групі нейронів застосовується операція взяття максимума (max-pooling). Така субдискретизація запропонована в роботах Хьюбела і Візеля, і базується на тому, що нейрони в корі головного мозку, яка відповідає за зір, ведуть себе саме таким чином. Зустрічаються й інші операції субдискретизації, зокрема в згорткових нейромережах ЛеКуна використовувалося взяття середнього значення. Дослідження, в яких проводилося порівняння, зазвичай виступали за операцію max-pooling. Застосовується також проміжний варіант між максимумом і середнім значенням, заснований на тому, що максимум береться на за всім вікном, а за деякою його випадковою підмножиною.

Однак, саме максимум зустрічається на практиці найчастіше і для більшості практичних задач дає гарні результати. Формально операція max-pooling визначена наступним чином (в тих самих позначеннях, що і раніше):

$$x_{i,j}^{l+1} = \max_{-d \le a,b \le d} z_{i+a,j+b}^{l},$$

де d – розмір вікна субдискретизації.

Як правило, крок дискретизації вибирають рівним розміру вікна, тобто, вхідна матриця ділиться на вікна, які не перетинаються, і у кожному вікні вибирається максимальне значення. Для $d = 2$ цей випадок показано на рисунку 2.19, де штриховка в початковій матриці а відповідає вікнам, за якими береться максимум з кроком 2.

0	1	2	1
4	1	0	1
2	0	1	1
1	2	3	1

А

4	2	2
4	1	1
2	3	3

б

4	2
2	3

в

Рисунок 2.19 – Приклад субдискретизації з вікном розміру 2x2: а – початкова матриця; б – матриця після субдискретизації з кроком 1; в – матриця після субдискретизації з кроком 2.

В результаті субдискретизації втрачається частина інформації, проте мережа більш стійка до незначних трансформацій зображення, таких як зсув і поворот.

Згорткові нейронні мережі забезпечують часткову стійкість до масштабування, зміщення, повороту, зміни ракурсу і інших змін. Згорткові нейронні мережі об'єднують три архітектурні ідеї, для забезпечення інваріантності до зміни масштабу, повороту і зміщенню:

- локальні рецепторні поля (забезпечують локальний двовимірний зв'язок нейронів);
- загальні синаптичні коефіцієнти (забезпечують виявлення деяких характеристик у будь якому місці зображення та зменшують загальне число вагових коефіцієнтів);
- ієрархічна організація с просторовими підвибірками.

Згорткова нейромережа складається з різних видів шарів: згорткові шари, субдискретизуючі (агрегація) шари і шари «звичайної» нейронної мережі - перцептрона, у відповідності до рисунку 2.20.

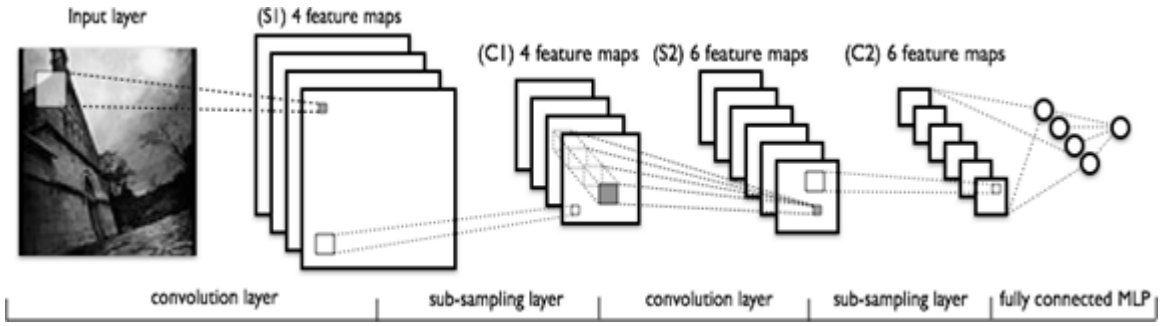


Рисунок 2.20 – Топологія згорткової нейронної мережі

Перші два типи шарів (згорткові, агрегація), по черзі, формують вхідний вектор ознак для багатошарового перцептрона.

Згорткові мережі можуть швидко працювати на послідовній машині і швидко навчатися за рахунок чистого розпалалелювання процесу згортки по кожній карті, а також оберненої згортки при поширенні помилок по мережі.

На рисунку 2.21 продемонстрована візуалізація згортки і агрегації:

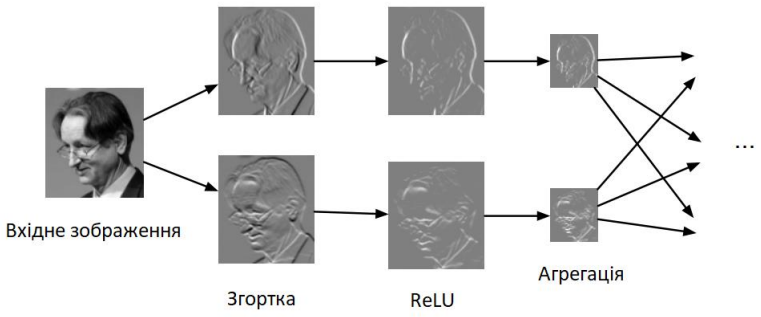


Рисунок 2.21 – Операція згортки + ReLU + Агрегація

2.3.3 Топологія згорткової нейронної мережі

Можна виділити наступні елементи, що впливають на вибір топології:

– визначити задачу, яку нейромережа має розв’язувати (класифікація, прогнозування, модифікація);
– визначити обмеження в розглянутій задачі (швидкість, точність відповіді);
– визначити вхідні (тип: зображення, звук, розмір: 100x100, 30x30, формат: RGB, в градаціях сірого) і вихідні дані (кількість класів).

В моїй роботі завдання, яку розв’язує нейромережа - модифікація зображень. Обмеження залежать від пристрою, на якому розгортають нейромережу.

Вхідні дані представляють з себе кольорові зображення. Кожне зображення розбивається на 3 канали: червоний, синій, зелений. Таким чином отримується 3 зображення з однаковою роздільною здатністю. Вхідний шар враховує дворівневу топологію зображень і складається з декількох карт (матриць). Карта може бути одна, в тому випадку, якщо зображення представлено у чорно-білому форматі, в іншому випадку їх 3, де кожна карта відповідає зображенню з відповідним каналом (червоним, зеленим і синім).

Вхідні дані кожного конкретного значення пікселя нормалізуються в діапазоні від 0 до 1, за формулою:

$$f(p, min, \max) = \frac{p-min}{max-min},$$

де f – функція нормалізації;
p – значення конкретного кольору пікселя від 0 до 255;
min – мінімальне значення пікселя – 0;
max – максимальне значення пікселя – 255.

Згортковий шар являє собою набір карт (інша назва - карти ознак, в побуті це звичайні матриці), у кожної карти є синаптичне ядро (в різних джерелах його називають по-різному: скануюче ядро або фільтр).

Кількість карт визначається вимогами до задачі, якщо взяти велику кількість карт, то підвищиться якість розпізнавання, але збільшиться обчислювальна складність. Виходячи з аналізу наукових статей, в більшості випадків пропонується брати співвідношення один до двох, тобто кожна карта попереднього шару (наприклад, у першого згорткового шару, попереднім є вхідний) пов'язана з двома картами згорткового шару, відповідно до малюнком 2.22.

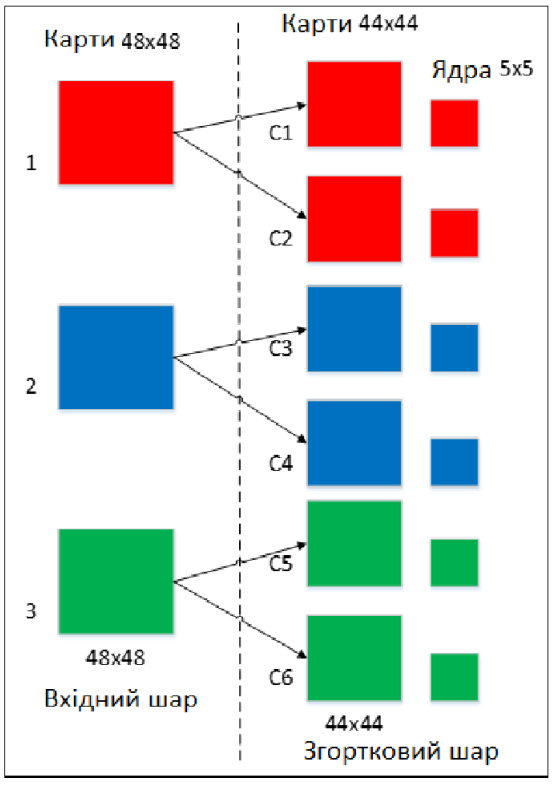


Рисунок 2.22 – Організаця зв’язків між картками згорткового шара і попереднього

Розмір у всіх карт згорткового шару – однаковий і обчислюється за наступною формулою:

$$(w, h) = (mW - kW + 1, mH - kH + 1),$$

де (w, h) – обчислюємий розмір згорткової карти;

mW – ширина попередньої карти;

mH – висота попередньої карти;

kW – ширина ядра;

kH – висота ядра.

Ядро являє собою фільтр або вікно, яке ковзає по всій області попередньої карти і знаходить певні ознаки об'єктів. Наприклад, якщо мережу навчали на множині осіб, то одне з ядр могло б в процесі навчання видавати найбільший сигнал в області очей, рота, брів або носа, інше ядро могло б виявляти інші ознаки. Розмір ядра зазвичай беруть в межах від 3x3 до 7x7. Якщо розмір ядра маленький, то воно не зможе виділити хоча б якісь ознаки, якщо занадто велике, то збільшується кількість зв'язків між нейронами. Також розмір ядра вибирається таким, щоб розмір карт згорткового шару був парний, це дозволяє не втрачати інформацію при зменшенні розмірності в субдискретизуючому шарі, описаному нижче.

Ядро являє собою систему поділюваних ваг або синапсів, це одна з головних особливостей згорткових нейромереж. У звичайної багатошарової мережі дуже багато зв'язків між нейронами, тобто синапсів, що вельми уповільнює процес виявлення ознак. У згорткової мережі - навпаки, загальні ваги дозволяє скоротити число зв'язків і дозволити знаходити одну й ту саму ознаку по всій області зображення.

З самого початку значення кожної карти згорткового шару дорівнює 0. Значення ваг ядр задаються випадковим чином в діапазоні від -0.5 до 0.5.

Ядро проходить по попередній карті і виконує операцію згортки, яка часто використовується для обробки зображень і має наступний вигляд:

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l],$$

де f – вхідна матриця зображень,

g – ядро згортки.

Неформально цю операцію можна описати таким чином - вікном розміру ядра g проходимо з заданим кроком (зазвичай 1) все зображення f, на кожному кроці поелементно множимо вміст вікна на ядро g, результат підсумовується і записується в матрицю результату, як на рисунку 2.24.

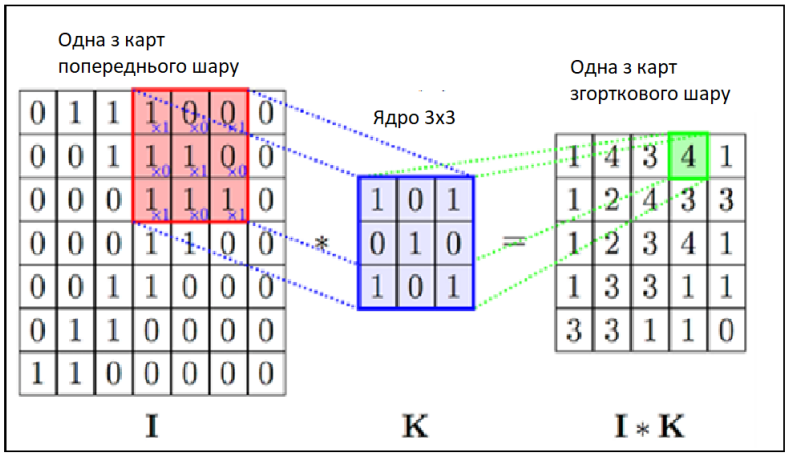


Рисунок 2.23 – Операція згортки і отримання значень згорткової карти

При цьому в залежності від методу обробки країв вихідної матриці результат може бути менше вихідного зображення такого ж розміру або більшого розміру, відповідно до рисунка 2.24.

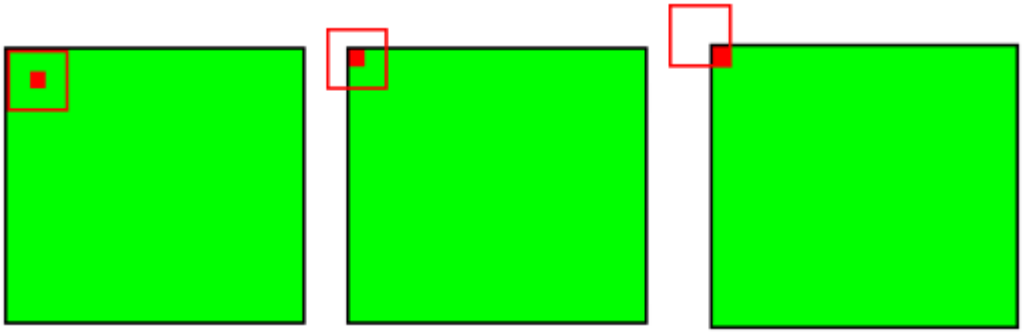


Рисунок 2.24 – Три види згортки початкової матриці

У спрощеному вигляді цей шар можна описати наступною формулою:

$$x^l = f(x^{l-1} * k^l + b^l),$$

де x^l – вихід шару l;

f() – функція активації;

b^l – коефіцієнт здвигу шару l;

* - операція згортки входу x з ядром k.

При цьому за рахунок крайових ефектів розмір вихідних матриць зменшується:

$$x_j^l = f(\sum_i x_i^{l-1} * k_j^l + b_j^l),$$

де x_j^l – карта ознак j (вихід шару l);

$f()$ – функція активації;

b_j^l – коефіцієнт здвигу шару l для карти ознак j;

k_j^l – ядро згортки j карти, шару l;

$*$ - операція згортки входу x з ядром k.

2.3.4 Шар агрегації

Шар агрегації також, як і згортковий, має карти, але їх кількість співпадає з попереднім (згортковим) шаром. Мета шару - зменшення розмірності карт попереднього шару. Якщо на попередній операції згортки вже були виявлені деякі ознаки, то для подальшої обробки настільки докладне зображення вже не потрібно, і воно ущільняється до менш докладного. До того ж фільтрація вже непотрібних деталей допомагає з проблемою перенавчання.

У процесі сканування ядром субдискретизуючого шару (фільтром) карти попереднього шару, скануюче ядро не перетинається на відміну від згорткового шару. Зазвичай, кожна карта має ядро розміром 2x2, що дозволяє зменшити попередні карти згорткового шару в 2 рази. Вся карта ознак поділяється на блоки 2x2 елемента, з яких вибираються максимальні за значенням.

Зазвичай в агрегуючому шарі застосовується функція активації ReLU. Операція підвибірки (або MaxPooling - вибір максимального) відповідно до рисунка 2.26.

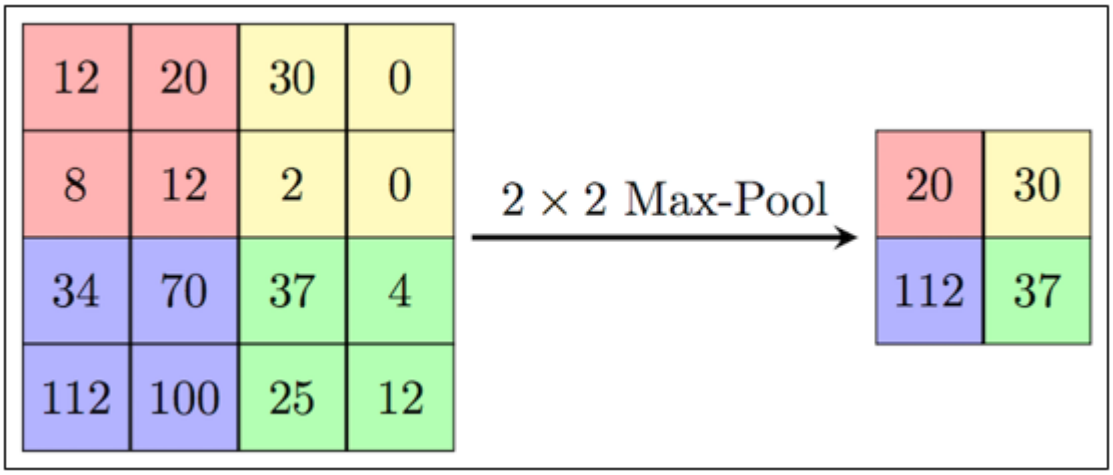


Рисунок 2.25 – Формування нової карти агрегуючого шару на основі попередньої карти згорткового шару. Операція підвибірки (Max Pooling)

Формально шар можна описати за формулою:

$$x^l = f(a^l * subsample(x^{l-1}) + b^l),$$

де x^l – вихід шару l;

$f()$ – функція активації;

a^l, b^l – коефіцієнти здвигу шару l;

$subsample()$ – операція вибірки локальних максимальних значень.

Останній з типів шарів це шар звичайного багатошарового перцептрона. Мета шару - класифікація, моделює складну нелінійну функцію, оптимізуючи яку, поліпшується якість розпізнавання.

Нейрони кожної карти попереднього агрегаційного шару пов'язані з одним нейроном прихованого шару. Таким чином число нейронів прихованого шару дорівнює числу карт агрегаційного шару, але зв'язки можуть бути не обов'язково такими, наприклад, тільки частина нейронів будь-якої з карт агрегаційного шару може бути пов'язана з першим нейроном

прихованого шару, а частина, що залишилася з другим, або всі нейрони першої карти пов'язані з нейронами 1 і 2 прихованого шару. Обчислення значень нейрона можна описати формулою:

$$x_j^l = f\left(\sum_i x_i^{l-1} * w_{i,j}^{l-1} + b_j^{l-1}\right),$$

де x_j^l – карта ознак j (вихід шару l);

$f()$ – функція активації;

b^l – коефіцієнт здвигу шару l;

$w_{i,j}^l$ – матриця вагових коефіцієнтів шару l;

Вихідний шар пов'язаний з усіма нейронами попереднього шару. Кількість нейронів відповідає кількості розпізнаваних класів. Але для зменшення кількості зв'язків і обчислень для бінарного випадку можна використовувати один нейрон і при використанні в якості функції активації гіперболічний тангенс, вихід нейрона зі значенням -1 означає приналежність до класу "не осіб", навпаки вихід нейрона із значенням 1 - означає приналежність до класу осіб.

2.3.5 Функція активації

Одним з етапів розробки нейронної мережі є вибір функції активації нейронів. Вид функції активації багато в чому визначає функціональні можливості нейронної мережі і метод навчання цієї мережі. Класичний алгоритм зворотного поширення помилки добре працює на двошарових і тришарових нейронних мережах, але при подальшому збільшенні глибини

починає відчувати проблеми. Одна з причин - так зване загасання градієнтів. У міру поширення помилки від вихідного шару до вхідного на кожному шарі відбувається множення поточного результату на похідну функції активації. Похідна у традиційній сігмоїдной функції активації менше одиниці на всій області визначення, тому після декількох шарів помилка стане близькою до нуля. Якщо ж, навпаки, функція активації має необмежену похідну (як, наприклад, гіперболічний тангенс), то може статися вибухове збільшення помилки у міру поширення, що призведе до нестійкості процедури навчання.

Функція активації сигмоїди належить до класу неперервних функцій і приймає на вході довільне дійсне число, а на виході дає дійсне число в інтервалі від 0 до 1. Зокрема, великі (по модулю) негативні числа перетворюються в нуль, а великі позитивні - в одиницю. Історично сигмоїда знаходила широке застосування, оскільки її вихід добре інтерпретується, як рівень активації нейрона: від відсутності активації (0) до повністю насиченою активації (1). Сигмоїда (sigmoid) виражається формулою:

$$f(s) = \frac{1}{1+e^{-s}}.$$

Вкрай небажаною властивістю сигмоїд полягає в тому, що при насиченні функції з тієї чи іншої сторони (0 або 1), градієнт на цих ділянках стає близький до нуля.

Нагадаємо, що в процесі зворотного поширення помилки даний (локальний) градієнт множиться на загальний градієнт. Отже, якщо локальний градієнт дуже малий, він фактично обнуляє загальний градієнт. В результаті, сигнал майже не буде проходити через нейрон до його ваг і рекурсивно до його данних. Крім того, слід бути дуже обережним при ініціалізації ваг сігмоїдних нейронів, щоб запобігти насичення. Наприклад, якщо вихідні ваги мають занадто великі значення, більшість нейронів перейде в стан насичення, в результаті чого мережа буде погано навчатися.

Часто у вигляді функції активації для прихованих і вихідних шарів обирають гіперболічний тангенс. Це зумовлено наступними причинами:

– симетричні активаційні функції, типу гіперболічного тангенса забезпечують більш швидку збіжність, ніж стандартна логістична функція;
– функція має неперервну першу похідну;
– функція має просту похідну, яка може бути обчислена через її значення, що дає економію обчислень.

Відомо, що нейронні мережі здатні наблизити як завгодно складну функцію, якщо в них досить шарів і функція активації є нелінійної. Функції активації типу сігмоїди або тангенсу є нелінійними, але призводять до проблем із загасанням або збільшенням градієнтів. Однак можна використовувати і набагато простіший варіант - випрямлену лінійну функцію активації (rectified linear unit, ReLU), яка виражається формулою:

$$f(s) = \max(0, s).$$

Переваги використання ReLU:

– її похідна дорівнює або одиниці, або нулю, і тому не може статися розростання або загасання градієнтів, тому що помноживши одиницю на дельту помилки ми отримаємо дельту помилки, якщо ж ми б використовували іншу функцію, наприклад, гіперболічний тангенс, то дельта помилки могла, або зменшитися, або зрости, або залишитися такою ж, тобто, похідна гіперболічного тангенса повертає число з різним знаком і величиною, що может сильно вплинути на загасання або розростання градієнта. Більш того, використання даної функції приводить до проріджування ваг;

– обчислення сигмоїди і гіперболічного тангенса вимагає виконання ресурсномістких операцій, таких як піднесення до степеня, в той час як ReLU може бути реалізований за допомогою простого порогового перетворення матриці активацій в нулі;
– відсікає непотрібні деталі в каналі при негативному виході.

З недоліків можна відзначити, що ReLU не завжди достатньо надійна і в процесі навчання може виходити з ладу («вмирати»). Наприклад, великий градієнт, що проходить через ReLU, може привести до такого оновлення ваг, що даний нейрон ніколи більше не активується. Якщо це станеться, то, починаючи з даного моменту, градієнт, що проходить через цей нейрон, завжди буде дорівнювати нулю. Відповідно, даний нейрон буде необоротно виведений з ладу. Наприклад, при дуже великій швидкості навчання (великому коефіцієнту навчання), може виявитися, що до 40% ReLU «мертві» (тобто, ніколи не активуються). Ця проблема вирішується за допомогою вибору належної швидкості навчання.

2.4 Змагальні нейронні мережі для масштабування зображень

Основна ідея породжувальних змагальних мереж (Generative Adversarial Networks, GANs) з’явилася в 2014 році в роботах Ієн Гудфеллоу. Зараз ці мережі дуже швидко розвиваються і, як пише Ян ЛеКун, активно використовуються в Facebook для обробки зображень і відеороликів.

В базовому варіанті модель породжуючих змагальних мереж складається з двох нейронних мереж, які змагаються одна з одною: генератор породжує об’єкти у просторі даних; дискримінатор навчається

відрізняти породжені генератором об’єкти від реальних прикладів з навчальної вибірки. Модель GAN складається з двох частин з протилежними цілями:

- дискримінатор розв’язує задачу бінарної класифікації: для заданого прикладу вирішити, чи є він елементом навчальної вибірки, чи породженим генератором;
- ціль генератора – «обманути» дискримінатор, зробити так, що дискримінатор не зможе відрізнити розподіл даних p_{data} і розподіл p_{gen}, який породжує генератор.

По мірі навчання ці дві мережі поступово роблять одна одну кращою. Мета полягає в тому, щоб генератор переміг і навчився робити p_{gen} настільки схожим на p_{data}, щоб ніхто не міг відрізнити.

Нехай I^{LR} - це версія з низькою роздільною здатністю, а I^{SR} - навпаки. Зображення з високою роздільною якістю доступні лише під час навчання. Під час навчання, I^{LR} отримують шляхом застосування Гаусового фільтра до I^{HR}, а слідом за цим операцію зменшення з коефіцієнтом r. Для зображення з C кольоровими каналами, можна описати I^{LR} за допомогою тензору з дійсними значеннями розміру $W \times H \times C$, а I^{HR}, I^{SR} - $rW \times rH \times C$ відповідно.

Головною метою є натренувати функцію генерації G, що наближає вхідне LR зображення до його відповідного HR варіанту. Щоб досягти цього, ми тренуємо генеративну мережу як згорткову нейронну мережу прямого поширення G_{θ_G} параметризованою θ_G . $\theta_G = \{W_{1:L}; b_{1:L}\}$ позначає ваги і зміщення шару L глибокої мережі і його одержують шляхом оптимізації відповідною функцією витрат SR l^{SR}. Для тренувальних зображень $I_n^{HR}, n = 1, \ldots, N$ з відповідними $I_n^{LR}, n = 1, \ldots, N$ розв’язуємо наступну задачу:

$$\hat{\theta}_G = \arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}\left(G_{\theta_G}(l_n^{LR}), I_n^{HR}\right).$$

В цій роботі розглянуто спеціально розроблену перцептивну функцію втрат l^{SR} . Вона являє собою зважену комбінацію декількох компонентів.

2.4.1 Архітектура змагальної мережі

Нехай маємо дискримінантну мережу D_{θ_D}, яку треба оптимізувати разом з генеративною мережею G_{θ_G}. Для цього треба вирішити змагальну мінімаксну задачу:

$$\min_{\theta_G}\max_{\theta_D} \mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[log(1 - D_{\theta_D}(G_{\theta_G}(I^L R))].$$

Загальна ідея цього формулювання наступна: необхідно натренувати генеративну модель G таким чином, щоб вона могла обманювати дискримінатор D , який у свою чергу тренується відрізняти реальні зображення від масштабованиї За допомогою цього підходу, генератор вчиться створювати зображення максимально близькі до реальних і тому дискримінатору D стає досить важко справлятися зі своєю задачею. Цей підхід наближає простір згенерованих зображень лежати максимально близько до простору реальних зображень.

У ядрі цієї дуже глибокої генераторної мережі G, що проілюстрована на малюнку 4, існує B залишкових блоків. Зокрема, використовується два згорткових шари з малим 3×3 ядром і 64 відображення характеристик, після яких прямує шар нормалізації шарів і параметрична ReLU у ролі

активаційної функції. В мережі збільшується роздільна здатність вхідного зображення за допомогою двох натренованих субпіксельних згорткових шарів, що було запропоновано у.

Щоб відрізнити реальне HR зображення від згенерованих SR зразків необхідно натренувати дискримінантну мережу. Архітектура показана на рисунку 2.26. Використовувалася Leaky ReLU у ролі функції активації ($\alpha = 0.2$), і уникалася max-агрегація по всій мережі. Дискримінантну мережу тренували, щоб розв'язати проблему максимізації в рівнянні 2. Вона містить вісім згорткових шарів із фільтром 3×3, що збільшуються у 2 рази із 64 до 512 ядер як у VGG мережі. Згортка із кроком використовується, щоб зменшити роздільну здатність картинки, кожного разу збільшуючи вдвічі кількість характеристик. Після 512 відображень характеристик, слідує два щільних шари і фінальна сигмоїдальна функція активації, яка надає вірогідність того, що зображення є реальним.

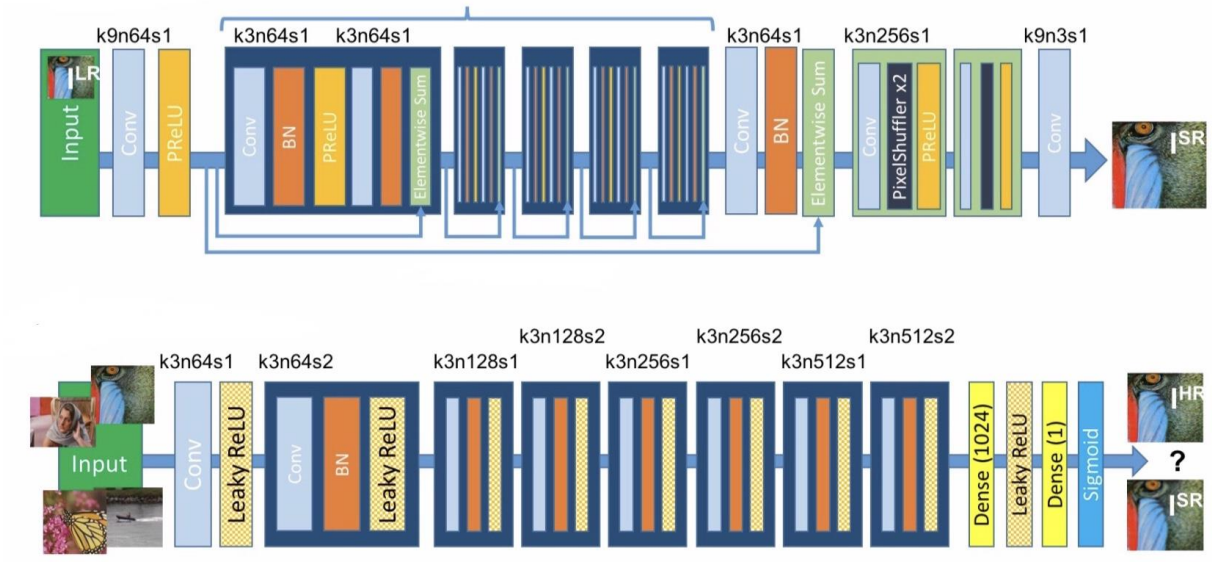


Рисунок 2.26 – Архітектура генеративної і дискримінатної мереж з відповідним розміром ядра (k), кількістю відображень характеристик (n) і кроком (s) позначеними для кожного згорткового шару

2.4.2 Перцептивна функція втрат

Означення перцептивної функції втрат l^{SR} є критичним для продуктивності генеративної мережі. В той час, коли l^{SR} зазвичай моделюється на MSE, вона була розроблена, щоб оцінювати розв'язок відносно сприйняття пов'язаних характеристик. Perceptual loss був сформульований як зважена сума функції втрат змісту (l_X^{SR}) і змагальної функціїї втрат наступним чином:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR},$$

де l_X^{SR} - content loss, а інший доданок - adversarial loss.

Попіксельна функція витрат MSE розраховується наступним чином:

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

Ця функція втрат є досить поширеною у задачі збільшення роздільної здатності SR. Багато новітніх підходів покладаються саме на неї. Проте у випадку, коли значення PSNR досягає досить високого рівня, може виникнути наступна проблема: частина зображення, що містить високодеталізовані текстури починає згладжуватися. Перш за все це виникає через функцію втрат MSE, оскільки вона не бере до уваги загальний зміст зображення.

Щоб замінити попіксельну функцію витрат, часто використовують функцію втрат, що покладається сприйняття людини. Однією з таких функції є функція втрат VGG, що базується на завчасно натренованої мережі VGG19.

Нехай $\phi_{i,j}$ – це відображення характеристик, що отримуємо j-ою згорткою (після активації) перед і-м max-агрегативним шаром у мережі VGG19. Тоді функція втрат VGG визначається як евклідова відстань між представленнями характеристик реконструйованого зображення $G_{\theta_G}(I^{LR})$ і відповідним оригіналом зображення I^{HR}:

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2.$$

Тут $W_{i,j}$ і $H_{i,j}$ описують розмірності відповідних відображень характеристик у мережі VGG.

2.4.3 Змагальна функція втрат

До перцептивної функції втрат, окрім функції втрат змісту, також входить генеративна функція втрат. Це допомагає мережі створювати максимально реальні зображення, шляхом постійних спроб обманути дискримінантну мережу. Генеративна функція витрат l^{SR}_{Gen} визначена на основі ймовірностей дискримінатора $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ серед всіх навчальних зразків:

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})).$$

Тут, $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ - це вірогідність, що реконструйоване зображення $G_{\theta_G}(I^{LR})$ є натуральним HR зображенням. Для кращої поведінки градієнту мінімізують $-\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$ замість $\log[1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))]$.

2.4 Висновки

Я описав деякі обмеження PSNR-метрики. Також було детально описано архітектуру генеративно змагальної мережі, що використовується у цій роботі – SRGAN.

Також було розглянуто деякі метрики, що використовуються при оцінці алгоритмів інтерполяції зображення. Деякі з них мають свої особливості і обмеження. Також було показано проблему метрики, що найчастіше використовують у даному аналізі – PSNR. Її проблема полягає у тому, що воно бере до уваги попіксельну різницю між реальним і масштабованим зображенням, в той же час страчаючи загальні ознаки зображення.

Крім цього було описано методи масштабування зображення, що зазвичай використовуються. Було проведено порівняльна характеристика і описано їх алгоритм дії. Не обійшлось і без описання недоліків і переваг кожного з методів, що робить їх використання у різних видів зображень по-своєму корисними.

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ МАСШТАБУВАННЯ ЗОБРАЖЕНЬ РІЗНИХ ТИПІВ

3.1 Обґрунтування вибору платформи та мови програмування

Для реалізації програмного продукту у моїй роботі було використано мову програмування Python. Ця мова програмування є найбільш популярною серед розробників штучного інтелекту і архітекторів нейронних мереж. Тому для реалізації свого програмного продукту було обрано саме її. Детальніше про особливості вибору описано нижче

1. Велика екосистема бібліотек

Великий вибір бібліотек є однією з головних причин, чому Python є найпопулярнішою мовою програмування для штучного інтелекту. Бібліотека є модулем або групою модулів, опублікованих різними джерелами, такими як PyPi, які включають попередньо написаний фрагмент коду, який дозволяє користувачам досягати певної функціональності або виконувати різні дії. Бібліотеки Python надають елементи базового рівня, тому розробники не повинні кодувати їх з самого початку.

Машинне навчання вимагає неперервної обробки даних, а бібліотеки Python дозволяють отримувати доступ, обробляти та перетворювати дані. Ось деякі з найпоширеніших бібліотек, які можна використовувати для машинного навчання та штучного інтелекту:

- Scikit-learn для використання основних алгоритмів машинного навчання, таких як кластеризація, лінійна і логістична регресії, регресія, класифікація та інші.
- Pandas для високорівневих структур даних і їх аналізу. Це дозволяє об'єднувати і фільтрувати дані, а також збирати їх з інших зовнішніх джерел, наприклад, Excel.

- Keras для глибокого навчання. Вона дозволяє швидко обчислювати і створювати прототипи, оскільки використовує GPU як додаток до процесора комп'ютера.
- TensorFlow для роботи з глибоким навчанням шляхом створення, навчання та використання штучних нейронних мереж з масивними наборами даних.
- Matplotlib для створення 2D графіків, гістограм, діаграм та інших форм візуалізації.
- NLTK для роботи з обчислювальною лінгвістикою, розпізнаванням природних мов і обробкою.
- Scikit-image для обробки зображень.
- PyBrain для нейронних мереж, навчання без вчителя та навчання з підкріпленням.
- Caffe для глибокого навчання, що дозволяє перемикатися між процесором і графічним процесором і обробляти 60 млн. Зображень в день за допомогою єдиного GPU NVIDIA K40.
- StatsModels для статистичних алгоритмів і дослідження даних.

У репозиторії PyPI можна знайти та порівняти більше бібліотек Python.

2. Низький бар’єр входу

Робота в індустрії машинного навчання та штучного інтелекту означає роботу з великою кількістю даних, які потрібно обробляти найбільш зручним і ефективним способом. Низький вхідний бар'єр дозволяє більш швидко вивчити Python і почати використовувати його для розвитку штучного інтелекту, не витрачаючи занадто багато зусиль на вивчення мови.

Мова програмування Python нагадує повсякденну англійську мову, що полегшує процес навчання. Його простий синтаксис дозволяє комфортно працювати зі складними системами, забезпечуючи чіткі зв'язки між елементами системи.

На додаток до цього є багато доступної документації, і спільнота Python, яка завжди рада допомогти і дати поради.

3. Гнучкість

Python для машинного навчання є прекрасним вибором, оскільки ця мова дуже гнучка:

- Він пропонує можливість вибору або використання ООП або скриптів.
- Також немає необхідності перекомпілювати вихідний код, розробники можуть вносити будь-які зміни і швидко переглядати результати.
- Програмісти можуть поєднувати Python та інші мови для досягнення своїх цілей.

Крім того, гнучкість дозволяє розробникам обирати стилі програмування, якими вони повністю задовольняються або навіть комбінувати ці стилі для вирішення різних типів проблем найбільш ефективним способом.

- Імперативний стиль складається з команд, які описують, як комп'ютер повинен виконувати ці команди. За допомогою цього стилю ви визначаєте послідовність обчислень, яка відбувається як зміна стану програми.
- Функціональний стиль також називається декларативним, оскільки він декларує, які операції слід виконувати. Він не враховує стан програми, у порівнянні з імперативним стилем, він декларує висловлювання у вигляді математичних рівнянь.
- Об'єктно-орієнтований стиль заснований на двох поняттях: клас і об'єкт, де подібні об'єкти формують класи. Цей стиль не повністю підтримується Python, оскільки він не може повністю виконати інкапсуляцію, але розробники все ще можуть використовувати цей стиль до доступного рівня.

– Процедурний стиль є найбільш поширеним серед початківців, оскільки він виконує завдання в покроковому форматі. Він часто використовується для послідовування, ітерації, модуляції та вибору.

Фактор гнучкості зменшує можливість помилок, оскільки програмісти мають можливість взяти ситуацію під контроль і працювати в комфортних умовах.

4. Незалежність від платформи

Python не тільки зручна у використанні мова, але й проста в освоєнні, і також дуже універсальна. Мається на увазі, що Python для розробки машинного навчання може працювати на будь-якій платформі, включаючи Windows, MacOS, Linux, Unix і інших. Щоб перенести процес з однієї платформи на іншу, розробникам необхідно реалізувати кілька дрібномасштабних змін і змінити деякі рядки коду для створення виконуваної форми коду для обраної платформи. Розробники можуть використовувати пакунки, такі як PyInstaller, щоб підготувати свій код для роботи на різних платформах.

Знову ж таки, це заощаджує час і гроші для тестів на різних платформах і робить процес більш простим і зручним.

5. Читаємість

Python дуже легко читати, тому кожен розробник Python може зрозуміти код своїх колег і змінити, скопіювати або поділитися ним. Немає ніякої плутанини, помилок або суперечливих парадигм, і це призводить до більш ефективного обміну алгоритмами, ідеями та інструментами між професіоналами штучного інтелекту та машинного навчання.

Є також такі інструменти, як IPython, що є інтерактивною оболонкою, яка надає додаткові функції, такі як тестування, налагодження, завершення табуляції та інші, і полегшує процес роботи.

6. Хороша візуалізація

Я вже згадував, що Python пропонує різноманітні бібліотеки, а деякі з них є чудовими засобами візуалізації. Проте для розробників штучного інтелекту важливо підкреслити, що у штучного інтелекту, глибокого навчання і машинного навчання, життєво важливим є представлення даних у зручному для людини форматі.

Бібліотеки, такі як Matplotlib, дозволяють дослідникам даних створювати діаграми, гістограми та графіки для кращого розуміння даних, ефективної презентації та візуалізації. Різні інтерфейси прикладного програмування також спрощують процес візуалізації та спрощують створення чітких звітів.

7. Підтримка спільноти

Це завжди дуже корисно, коли існує сильна підтримка спільноти навколо мови програмування. Python - це мова з відкритим вихідним кодом, що означає, що для програмістів відкривається багато ресурсів, починаючи від початківців і закінчуючи професіоналами.

Багато документів Python доступні в Інтернеті, а також у спільнотах і на форумах Python, де програмісти і розробники машинного навчання обговорюють помилки, вирішують проблеми і допомагають один одному.

Мова програмування Python абсолютно безкоштовна, як і різноманітність корисних бібліотек і інструментів.

Серед бібліотек машинного навчання було обрано дві наступні: Tensorflow та Keras. Основна частина програми, а особливо архітектура нейронної мережі, розроблялась за допомогою Keras. Вибір впав на неї, оскільки вона побудована на більш низькорівневої бібліотеці Tensorflow і має досить зрозумілий для людини інтерфейс. Більш специфічні функції, які мають працювати на низькому рівні і їх важко реалізувати на Keras, розроблювалися на Tensorflow. Це стає можливим якраз через те, що Keras тісно інтегрований у Tensorflow.

Іншим аспектом при виборі описаних бібліотек стала можливість навчати моделі як на CPU, так і на GPU. Це надає можливість балансувати між швидкістю навчання і кількістю доступною пам’яті.

Також надані бібліотеки з легкістю можуть інтегрувати в різні системи, програмні модулі на мобільних телефонах та різних систем.

3.2 Навчання моделі

В даній роботі для навчання нейромережі було використано COCO data set 2017. Він містить близько 18 гігабайт зображень різної розмірності. Для навчання було обрано 800 зображень. Можна обрати більшу кількість при наявності потужного GPU, оскільки із збільшенням кількості навчальних зображень зростає як і час необхідний для тренування, так і кількість пам’яті. В цьому випадку треба обирати між оперативною пам’яттю і CPU, що надає досить малу швидкість навчання, але з великою кількістю доступної пам’яті та пам’яттю GPU і ним самим як блоком, що виконує всі операції. Остання конфігурація збільшує в рази швидкість навчання, але з досить великими втратами доступної пам’яті. Тому для виконання цього завдання було обрано відносно невелику кількість зображень, аби вистачило пам’яті, та велику швидкість навчання.

Перед навчанням всі зображення необхідно було завчасно обробити. Для цього було зроблено обрізання, щоб всі зображення мали однакову розмірність. В роботі було обрано зображення з розмірністю 384 на 384. Після цього кроку ми отримуємо дані для порівняння, які вважаються ідеальними для після інтерполяції. Наступним кроком необхідно отримати зображення, які ми будемо збільшувати. Для цього кожне ідеальне зображення було зменшено з коефіцієнтом масштабування 4 методом

бікубічної інтерполяції – отримано зображення з роздільною якістю 96 на 96 пікселів. Отримані зображення в подальшому збільшувалися і порівнювалися з оригіналом під час навчання.

Архітектура мережі, що була описана раніше складається з наступних блоків:

- 16 залишкових блоків.
- PixelShuffler x2: для збільшення відображення характеристик.
- PReLU (Parameterized ReLU): Використовувалася ця функція активації замість ReLU та LeakyReLU. Вона надає можливість навчання функції активації, що допомагає адаптуватися у випадки негативних коефіцієнтів.
- k3n64s1 означає 3 ядра, 64 канали і крок 1.
- Використано перцептивну функцію втрат. Вона складається з функції втрат змісту і змагальної функції втрат.

Мережа працє наступним чином. Спочатку обробляється HR зображення, щоб отримати зменшене LR зображення. Після цього ми отримуємо HR і LR зображення для навчальних даних. Після цього ми подаємо зображення з низькою роздільною здатністю до генератора. Він у свою чергу масштабує його і видає відповідне масштабоване зображення. Надалі використовується дискримінатор, щоб відрізнити реальне зображення від масштабованого і методом оберненого поширення помилки функції втрат генеративно змагальної мережі навчають дискримінатор і генератор. В результаті, генератор вчиться створювати все більш реалістичні зображення, а дискримінатор вчиться ще більш точно визначати реальне зображення від масштабованого під час навчання.

Під час навчання було використано наступні параметри:

- Метод оптимізації: Adam з $\beta_1 = 0.9$ і параметром швидкості навчання, що дорівнює 0.0001.
- Кількість ітерацій: 3000.

– Розмір пакетів даних: 64.

3.3 Аналіз результатів масштабування зображень різних типів

Темою роботи було масштабування зображень методами нейронних мереж. Для цього було побудовано і навчено генеративно змагальну нейронну мережу, яка виконувала це завдання. Також було розглянуто інші алгоритми масштабування запропоновані раніше. В програмному продукті за допомогою мови Python було побудовано нейромережу, що генерувала збільшене в 4 рази зображення. В тестовому режимі можна перевірити роботу програми для наступних методів:

– Метод найближчого сусіда.
– Білінійна інтерполяція.
– Бікубічна інтерполяція.
– Метод Ланцоша.
– Інтерполяція за допомогою нейронних мереж (SRGAN).

У тестовому режимі обирається зона яку необхідно масштабувати. Після цього все навколо неї обрізається і вона залишається сама. Аби можна було адекватно порівнювати із якимось ідеальним зображенням, то обрізана частина залишається за ідеал. Потім створюють копію цієї частини і зменшують у 4 рази. Вже зменшену інтерполюють до тієї роздільної здатності, що у оригінала. Потім вже збільшене різними методами зображення разом із оригіналом виводиться на екран.

Далі буде продемонстровано роботу нейромережі і порівняння з оригіналом та іншими алгоритмами. Також на екран виводяться відповідні метрики якості.

На рисунку 3.1 зображено чорно-білий малюнок, у якого є досить невелика кількість якихось дрібних деталей. На малюнку 3.2 відповідно показано вже обрізана частина з його зменшеними, а потім збільшеними частинами.

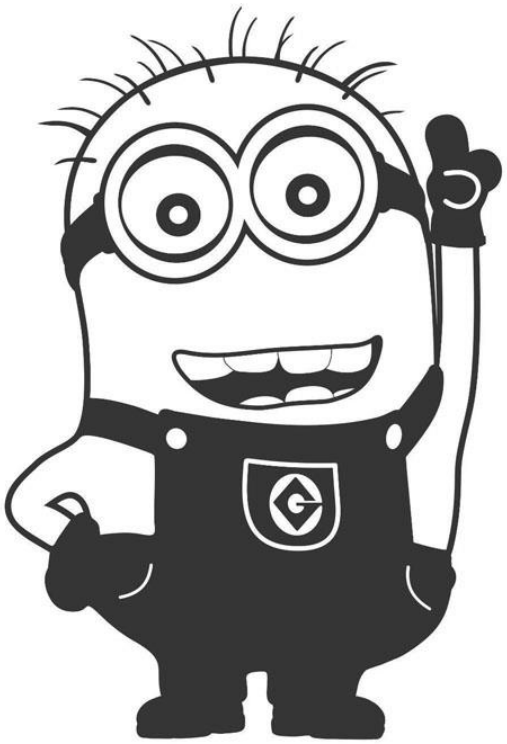


Рисунок 3.1 – Чорно-біле зображення оригіналу

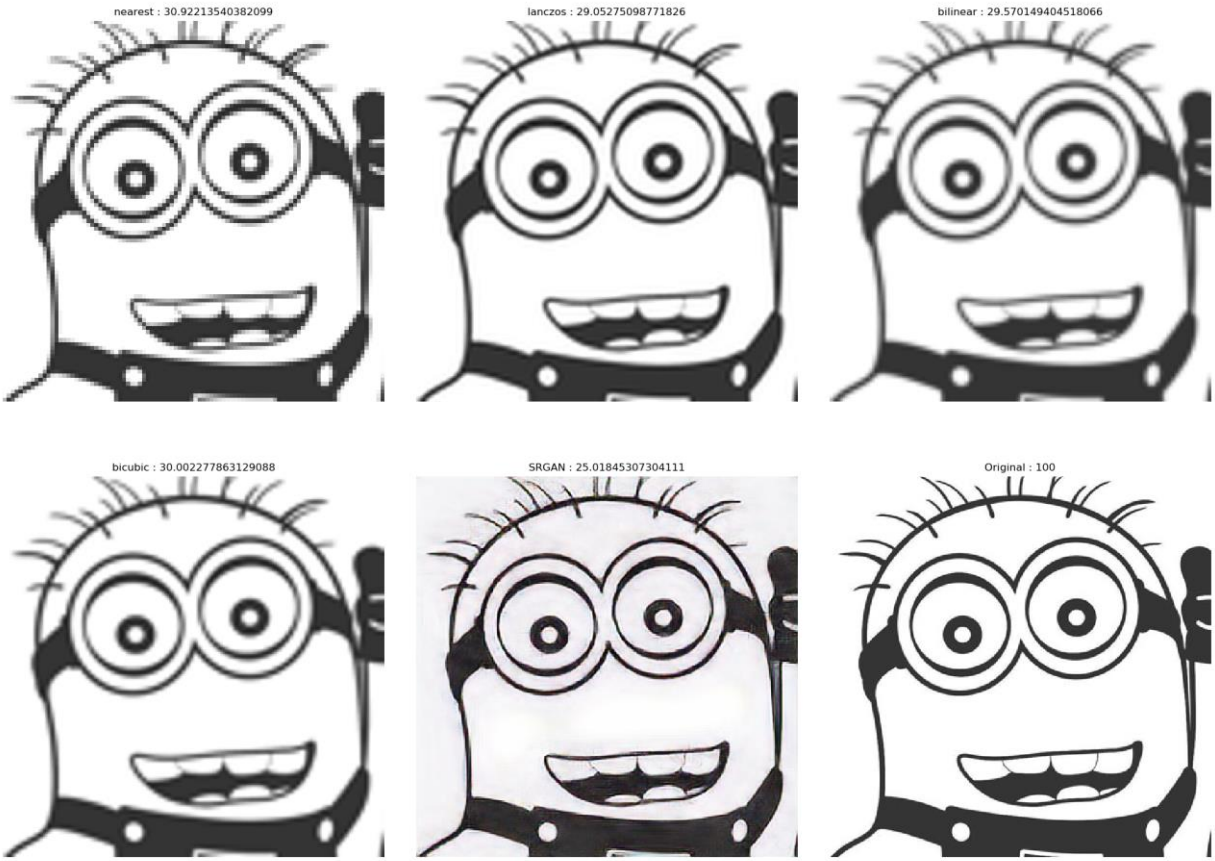


Рисунок 3.2 – Масштабоване зображення з відповідними метриками якості

На наступному прикладі вже зображено чорно-біла фотографія з дуже багатьма деталями. Оригінал можна побачити на рисунку 3.3, а їх масштабування на рисунку 3.4.

Рисунок 3.3 – Зображення ока, яке має досить багато дрібних деталей

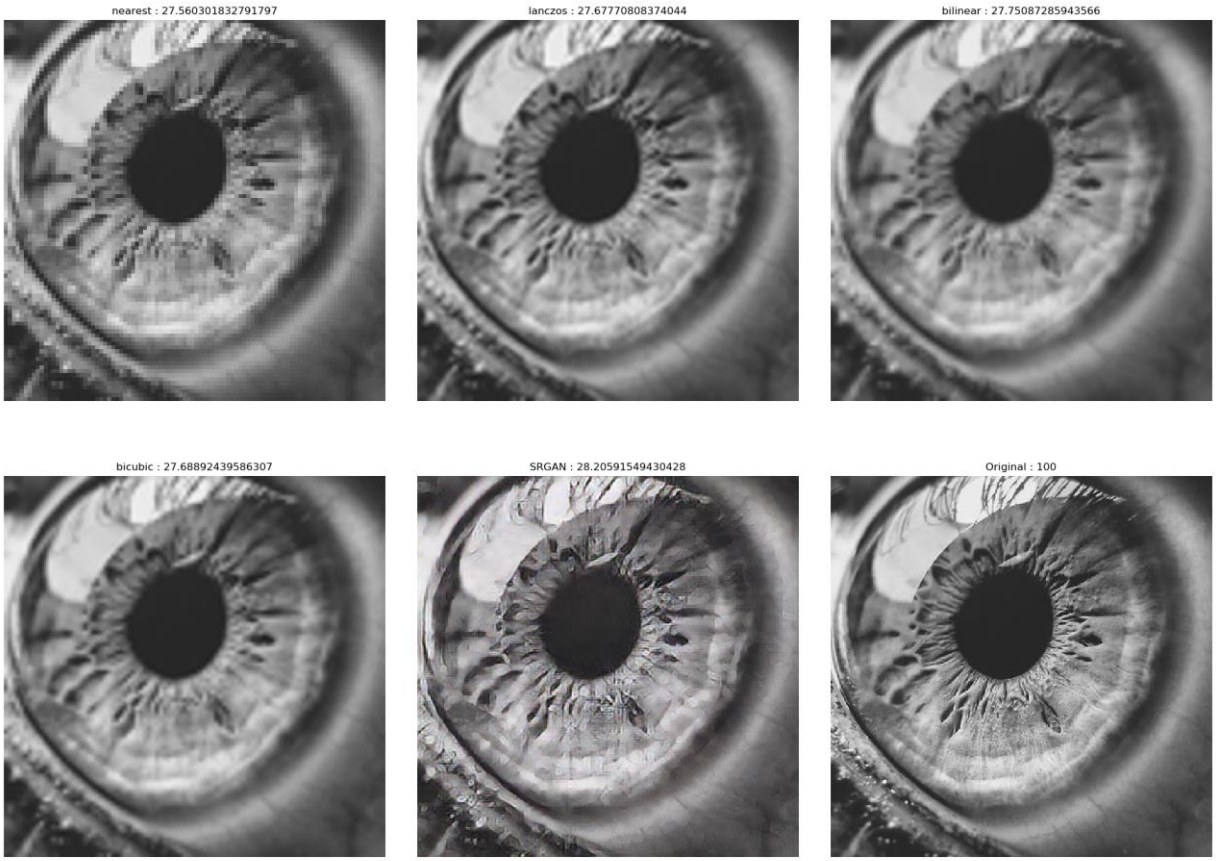


Рисунок 3.4 – Масштабоване зображення ока

Надалі розглядається чорно-білий малюнок від руки. На рисунку 3.5 зображено оригінал, а на рисунку 3.6 відповідно масштабована частина.

Рисунок 3.5 – Оригінал чорно-білого малюнка

Рисунок 3.6 – Масштабований чорно-білий малюнок дівчини

Проаналізуємо випадок чорно-білих зображень. У всіх трьох випадках зображення, масштабоване за допомогою генеративно змагальної мережі для ока людини сприймається приємніше. Проте у двох з трьох випадків значення PSNR у нейромережі є найменшим. Це можна пояснити тим, що нейронна мережа розглядає ознаки зображення і різні лінії та ребра. У випадку інших алгоритмів, то частина пікселів просто переноситься, а всі інші просто інтерполюються відповідними методами, що створює такий собі ефект розмиття, що відразу кидається в око. Також той факт, що значення PSNR у випадку з першим та третім малюнком найменший можна пояснити тим, що з’явився новий артефакт, який не спостерігався у інших алгоритмах. Навколо контуру дівчини та персонажу з мультфільму з’явився майже непомітний світло-сірий фон. Він займає досить велику частину зображення і

тому досить послабив значення метрики PSNR. На малюнку ж ока, відповідно, цього фону немає, що й призвело до більш високого значення PSNR, ніж у інших прикладах, але й крім того досить приємну для сприйняття картинку.

Наступним типом зображень, що розглянемо стануть реальні кольорові фотографії.

На рисунку 3.7 зображена дівчина. На рисунку 3.8 показано роботу різних методів масштабування відповідно обрізаної частини рисунка.

Рисунок 3.7 – Зображення дівчини

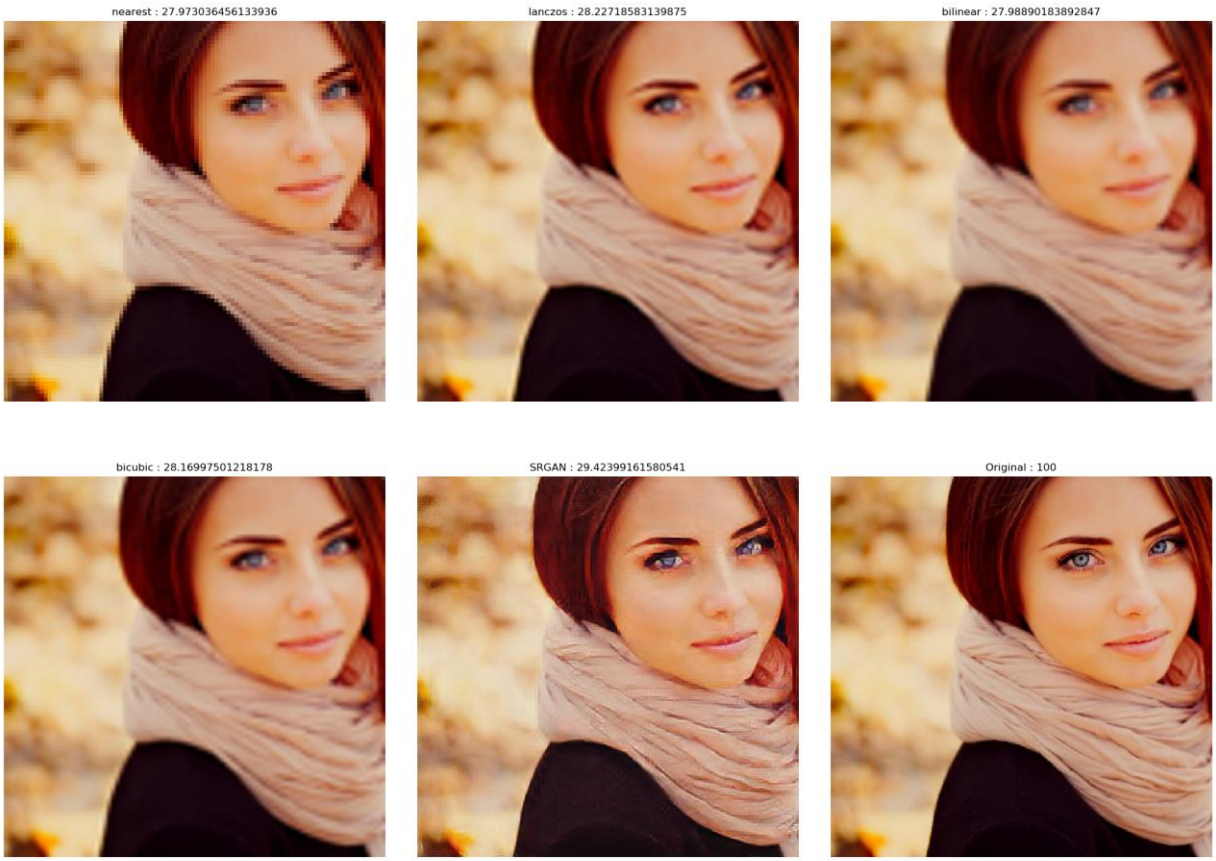


Рисунок 3.8 – Масштабовані кольорові фотографії дівчини

Наступний приклад розглядає мою фотографію. На рисунку 3.9 показано оригінальне фото. На рисунку 3.10 вже масштабовану частину. Ця частина примітна тим, що було масштабовано текстуру неба і хмар.

Рисунок 3.9 – Фотографія моя

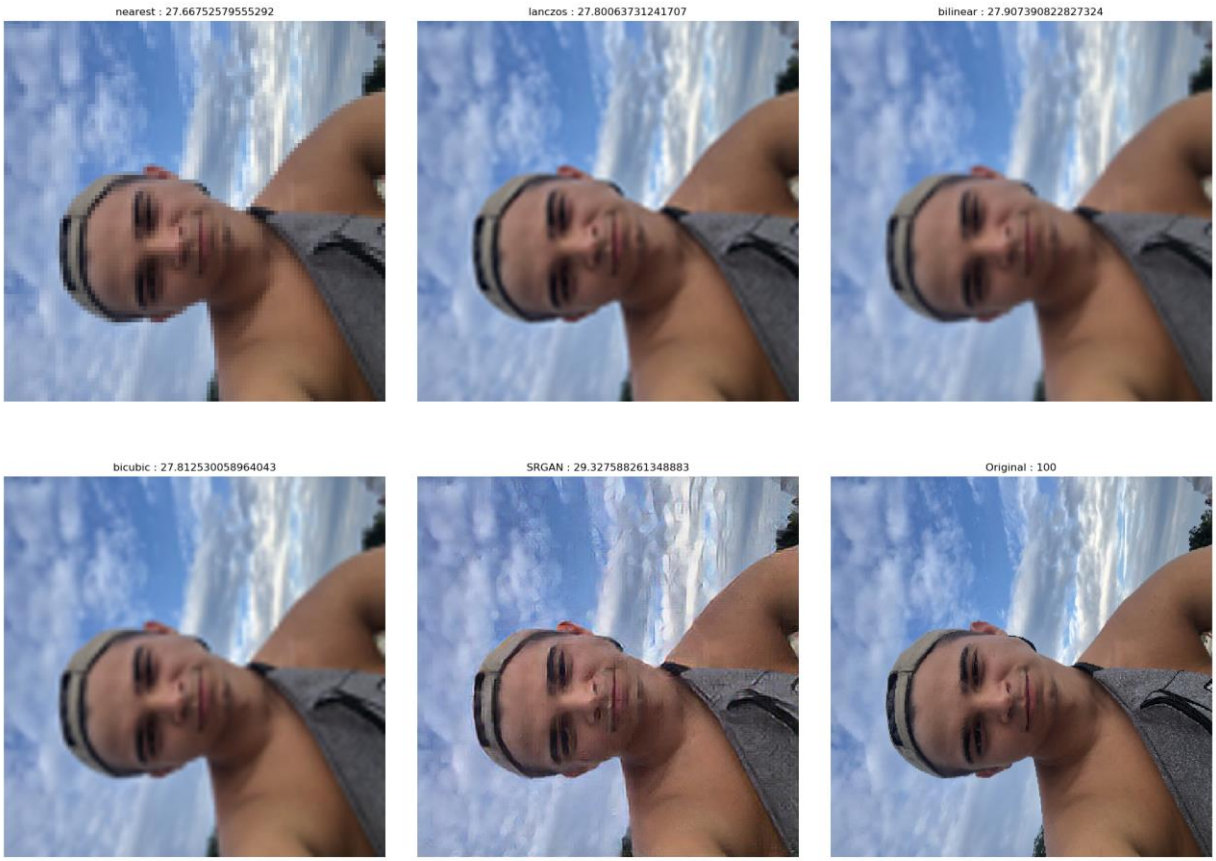


Рисунок 3.10 – Масштабована фотографія

У цьому прикладі відсутній білий фон, тому нейромережа не додає зайвого сірого кольору і значення PSNR не падає через це. Тому у цих двох випадках масштабування за допомогою генеративно змагальної мережі випереджає інші за метрикою PSNR. Також на фотографії людини помітно, що зникли нерівності по краях і немає ефекту згладжування. Це є досить гарним результатом, що допомагає побороти основні проблеми масштабування. Також на моїй фотогрфії досить важко розгледіти велику різницю, проте є невелике спостереження щодо зменшення згладжування текстур.

Наступним прикладом стануть масштабовані зображення з мультфільмів.

На рисунку 3.11 показано зображення герою мультфільма, а на рисунку 3.12 відповідно його масштабована версія.

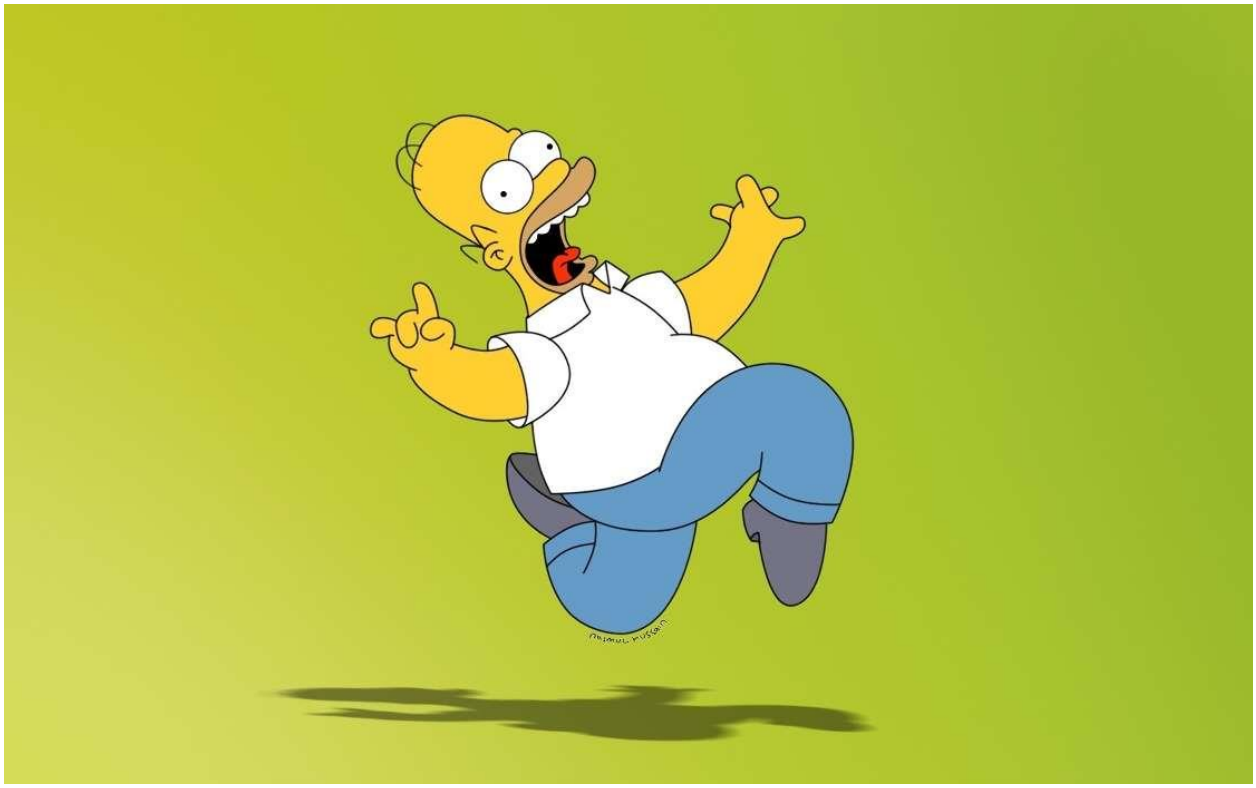


Рисунок 3.11 – Зображення герою мальтфільма

Рисунок 3.12 – Масштабована частина герою мультфільма

На рисунку 3.13 зображено малюнок з мультфільма з досить малою кількістю деталей відносно попереднього прикладу, а на рисунку 3.14 вже масштабована частина.

Рисунок 3.13 – Малюнок з мультфільма

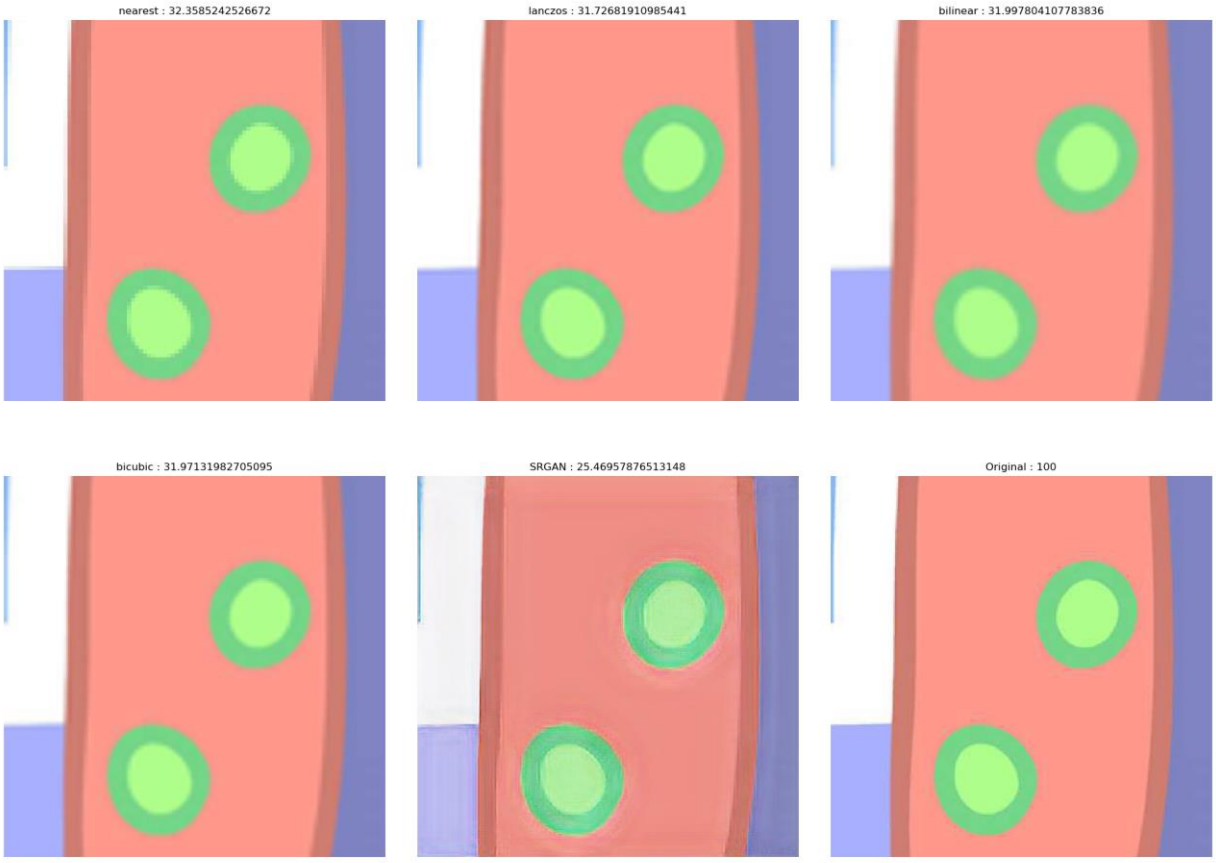


Рисунок 3.14 – Масштабована частина малюнка з мульфільму

Як видно з цього прикладу, то ось тут підхід з нейромережою показав не дуже гарний результат. Метрика PSNR набагато менша у масштабуванні за допомогою нейромереж, відносно інших методів. Також візуально картинка краще сприймається у звичайних методах інтерполяції. Це можна пояснити тим, що нейромережа навчалася інтерполювати зображення на реальних фотографіях, а цей приклад порівнює роботу з мультиплікативними зображеннями. Проте не дивлячись на ці недоліки нейромережа показала гарний результат з гарним відновленням ребер картинки. Зображення є більш чіткими, хоча навколо країв з’являється дивний ореол, якого немає у оригіналі.

І останні приклади масштабування відображення на різних картинах

На рисунках 3.15 – 3.18 показані різні стилі у художньому мистецтві і відповідні масштабовані частини.

Рисунок 3.15 – Оригінал картини

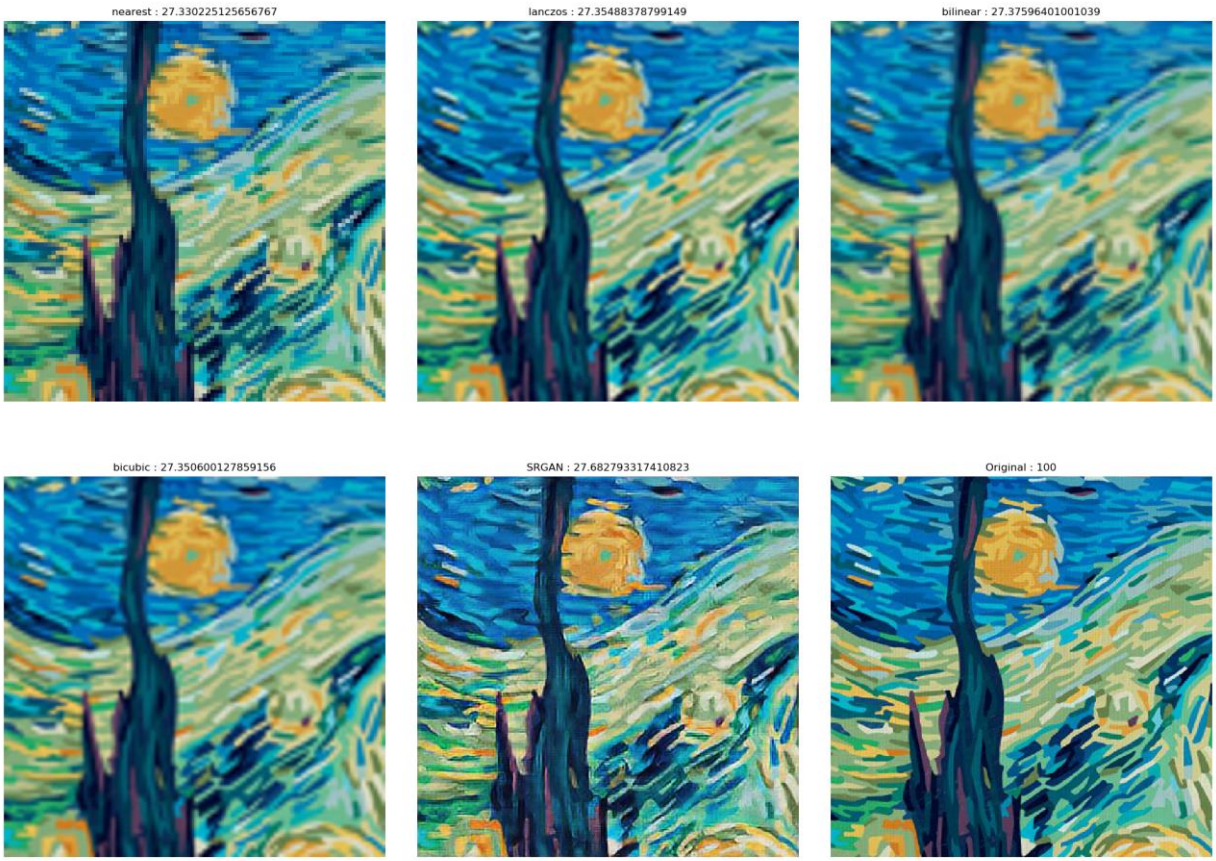


Рисунок 3.16 – Масштабована картина

Рисунок 3.17 – Оригінал картини

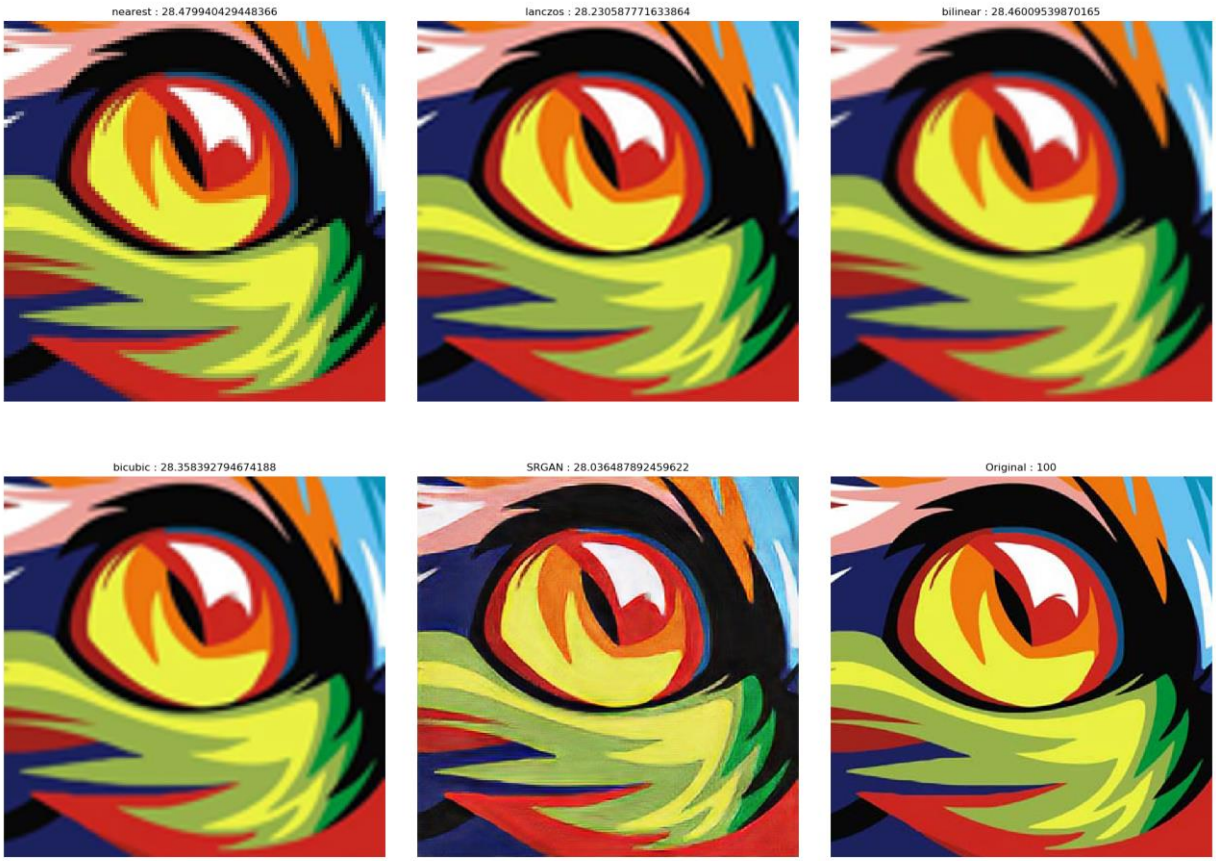


Рисунок 3.18 – Масштабована картина

Масштабування за допомогою розглянутої нейронної мережі показало найбільш приємні для сприйняття ока людини результати. Не дивлячись не те, що метрика PSNR майже не змінюється серед методів, сам візуальний результат набагато перцептивно приємніший у нейронної мережі. Проте у другому прикладі дещо проявився артефакт із прикладом з мультфільмами, а саме – невідомі ореоли навколо країв.

3.4 Висновки результатів роботи

В результатах роботи було продемонстровано різні типи зображень, і у кожному з типів існували свої особливості. Запропонований у цій роботі метод показав досить гарні результати у всіх випадках. Тільки у випадку з мультфільмами виявились деякі проблемі, проте і там він показав кращий результат по деяким частинам.

Головними особливостями, що виникали у генеративно змагальної нейронної мережі, запропонованої у цій роботі стали наступні артефкти: дивні ареоли навколо ребер і контурів зображення, зміна білого фону на сірий. Саме ці два ефекти досить сильно занижували метрику якості PSNR, яка є основною у цій сфері обробки зображень. Проте, це також показало той факт, що метрика PSNR не є ідеальною, бо вона базується на попіксельній різниці між фотографіями, в той час як сприйняття масштабованого зображення досить сильно падає. В плані сприйняття інтерпольованих зображень нейронна мережа показала найкращі результати і це можна пояснити тим, що функцією втрат, яку необхідно було оптимізувати було обрано саме перцептивну функцію втрат, яка звертає увагу на людське сприйняття картинок. Також можна вказати той факт, що нейронна мережа була натренована на досить невеликій кількості зображень і невеликій кількості ітерацій. При їх збільшенні можливо і якість масштабованих зображень виросла і зникли різні артефакти.

Найгірше нейромережа показала себе у випадку з мультиплікативними зображеннями. Там досить сильно впала метрика PSNR відносно інших методів і з’явилися артефакти, що відразу кидаються у очі. Це можна пояснити тим фактом, що у тренуванні зовсім не використовувалися зображення з мультфільмів, а це є важливим фактором, оскільки вони мають зовсім іншу структуру.

Найкращий результат показала інтерполяція фотографії чорно-білого ока. У цьому прикладі людина майже не може відрізнити реальне зображення від масштабованого, хоча метрика PSNR не дуже й висока.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задачі масштабування зображень.

Програмний продукт призначено для використання на персональних комп’ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, щовпливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
- забезпечувати високу швидкість обробки даних та відклик користувачеві у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем;
- забезпечувати можливість зручного масштабування та обслуговування;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу масштабування та будує відповідну модель.

Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір фреймворка машинного навчання;

F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

a) Python;

b) C++;

Функція F_2:

a) Tensorflow

b) Caffe

Функція F_3:

a) Jupyter Notebook

b) Visual Studio

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи, що зображено на рисунку 5.1. На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій, яку можна побачити у таблиці 5.1.

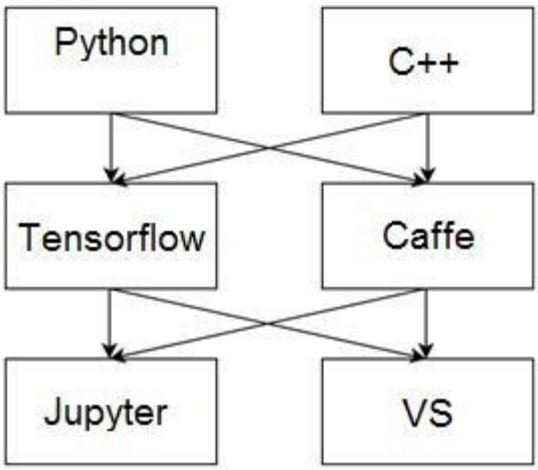


Рисунок 5.1 Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 5.1 Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
		Кросплатформенний,	Низька швидкодія,

F1	А	Займає менше часу при написанні коду	більший час на виконання операцій
	Б	Висока швидкодія, оптимізація пам’яті	Займає більше часу при написанні коду

Продовження таблиці 5.1

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F2	А	Безкоштовність, чудова документація	Відсутність нативної реалізації Windows

	Б	Надійність, швидкість, безкоштовність	Необхідність додаткової інсталяціїї, низький рівень користувацької підтримки
F3	А	Не займає багато пам’яті, можна виконувати окремі діляньки програмного коду	Повільний
	Б	Висока інтегрованість з сервісами MS, дуже багато додаткових інструментів	Займає дуже багато пам’яті

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки реалізація програмного коду для вирішення поставленої задачі є концептуально складною, необхідно обрати мову, що спрощує написання коду, тому варіант Б має бути відкинутий.

Функція *F2:*

Вибір фреймворку машинного навчання не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Функція *F3:*

Оскільки, програмний продукт реалізується мовою Python, використовуємо варіант А як найбільш сумісний з цією мовою.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2б – F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри: На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для збереження даних;
- X3 – час обробки даних;
- X4 – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій мови програмування залежно від обраної серверної технології.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.2.

Таблиця 5.2 Основні параметри ПП

Назва параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			Гірші	**Середні**	**Кращі**

Швидкодія мови програмування	*X1*	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	*X2*	Мб	32	16	8
Час обробки запитів користувача	*X3*	Мс	200	100	50
Потенційний об’єм	*X4*	Кількість строк коду	2000	1500	1000

За даними, наведеними у таблиці 5.2, будуються графічні характеристики параметрів – рисунки 5.2 - 5.5.

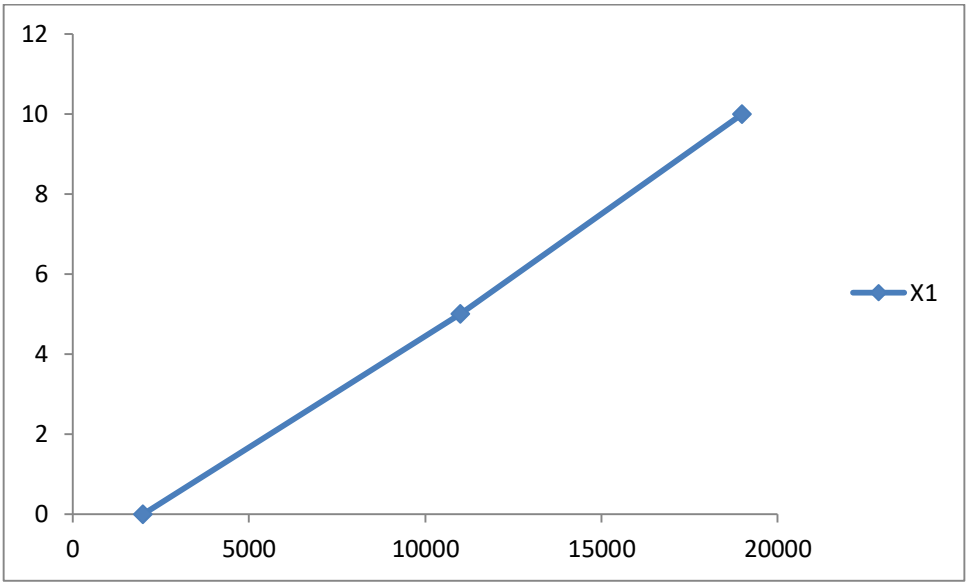


Рисунок 5.2 Швидкодія мови програмування

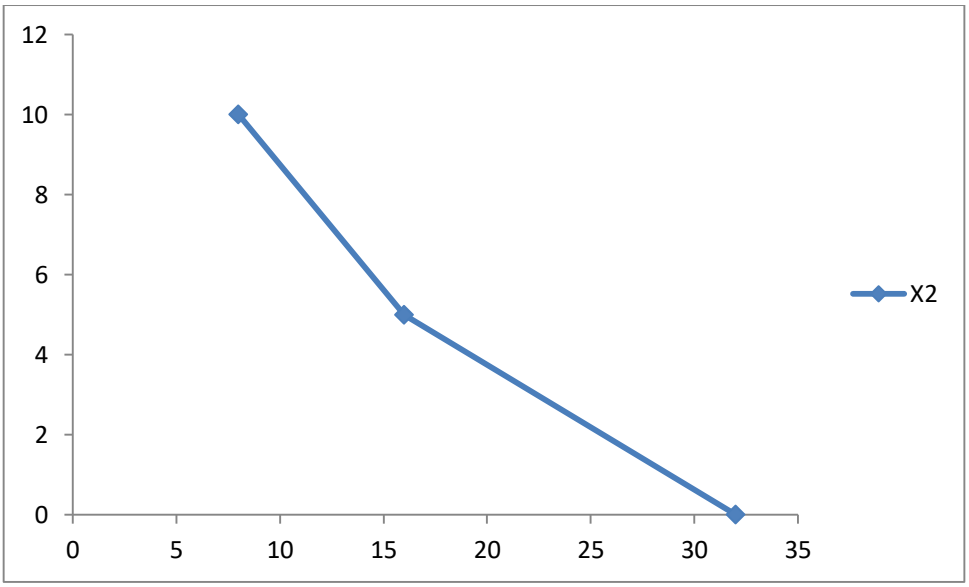


Рисунок 5.3 Об’єм пам’яті збереження данних

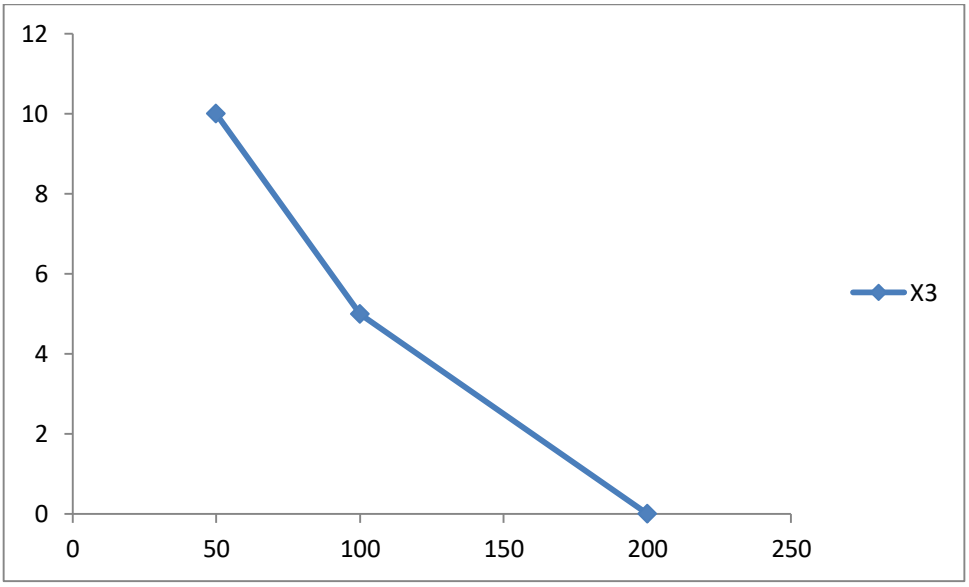


Рисунок 5.4 Час обробки запитів користувача

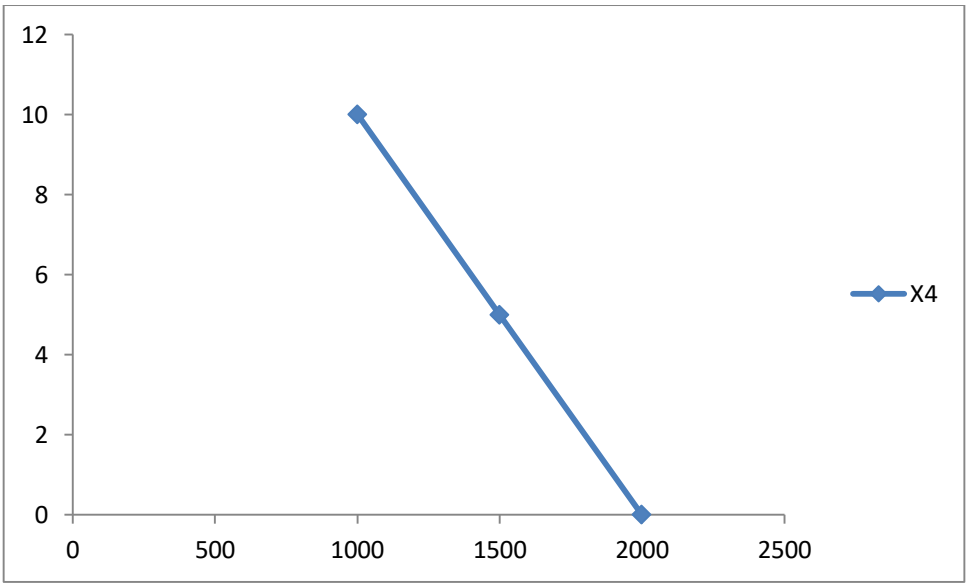


Рисунок 5.4 Потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого
- використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.2.

Таблиця 5.3 Результати ранжування параметрів

Позначення параметра	**Назва параметра**	**Одиниці вимірру**	**Ранг параметра за оцінкою експерта**							**Сума рангів R_i**	**Відхилення Δ_i**	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	3	4	3	1	3	3	19	1,5	2,25
X2	Об’єм пам’яті для збереження даних	Мб	3	2	2	1	3	2	2	15	-2,5	6,25
X3	Час обробки запитів користувача	Мс	1	1	1	2	2	1	1	9	-8,5	72,25
X4	Потенційний об’єм коду	Кількість строк коду	4	4	3	4	4	4	4	27	9,5	90,25
	Разом		10	10	10	10	10	10	10	70	0	171

Для перевірки ступені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де $\boldsymbol{N}$ – число експертів,

$\boldsymbol{n}$ – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 171}{7^2(4^3 - 4)} = 0.98 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 Попарне порівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	2	3	4	5	6	7		
X1 і X2	<	>	>	>	<	>	>	>	1.5
X1 і X3	>	>	>	>	<	>	>	>	1.5
X1 і X4	<	<	>	<	<	<	<	<	0.5
X2 і X3	>	>	<	<	>	>	>	>	1.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a = \begin{cases} 1.5, \text{якщо } X_i > X_j \\ 1, \text{якщо } X_i = X_j \\ 0.5, \text{якщо } X_i < X_j. \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$. Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i{}'}{\sum_{i=1}^{n} b_i{}'},$$

де $b_i{}' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 Розрахунок вагомості параметрів

Параметри x_i	**Параметри x_j**				**Перша ітерація**		**Друга ітерація**		**Третя ітерація**	
	X1	*X2*	*X3*	*X4*	b_i	$К_{ві}$	b_i^1	$К_{ві}^1$	b_i^2	$К_{ві}^2$
X1	1.0	1.5	1.5	0.5	4.5	0.281	16.25	0.265	62,5	0.274
X2	0.5	1.0	1.5	0.5	3.5	0.219	12.25	0.2	48.25	0.211
X3	0.5	0.5	1.0	0.5	2.5	0.156	11.5	0.188	36.375	0.159
X4	1.5	1.5	1.5	1.0	5.5	0.344	21.25	0.347	81.25	0.356
Всього:					16	1	61.25	1	228.375	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так, що можна побачити у таблиці 5.6:

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості і–го параметра;

B_i – оцінка і–го параметра в балах.

Таблиця 5.6 Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	**Варіант реалізації функції**	**Абсолютне значення параметра**	**Бальна оцінка параметра**	**Коефіцієнт вагомості параметра**	**Коефіцієнт рівня якості**
F1(X1)	А	11000	5	0.274	1.37
F2(X3)	А, Б	3600	2.8	0.159	0.45
F2(X4)	А	9000	4	0.356	1.42
	Б	12000	2	0.356	0.71
F3(X2)	А	16	5	0.211	1.1

За даними з таблиці К за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1.37 + 0.45 + 1.42 + 1.1 = 4.34$$

$$K_{K2} = 1.37 + 0.45 + 0.71 + 1.1 = 3.63$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P * K_П * K_{СК} * K_M * K_{СТ} * K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 * 1.7 * 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ = 27 людино-днів, $К_П$ = 0.9,$К_{СК}$ = 1,$К_{СТ}$ = 0.8:

$$T_2 = 27 * 0.9 * 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44)\ 8 = 1328{,}64 \text{ людино-годин;}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44)\ 8 = 1345.52 \text{ людино-годин;}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 7000 грн., один фінансовий аналітик з окладом 9500грн. Визначимо зарплату за годину за формулою:

$$\text{СЧ} = \frac{\text{М}}{T_m \cdot t} \text{грн.},$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$\text{СЧ} = \frac{7000 + 7000 + 9500}{3 \cdot 21 \cdot 8} = 46{,}62 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{СЗП} = \text{С}_\text{ч} \cdot \text{Т}_i \cdot \text{КД},$$

де $\text{С}_\text{ч}$– величина погодинної оплати праці програміста;

$\text{Т}_\text{і}$ – трудомісткість відповідного завдання;

$\text{К}_\text{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $\text{С}_\text{ЗП}$ = 46,62 * 1328.64 * 1.2 = 74340,57 грн.

II. $\text{С}_\text{ЗП}$ = 46,62 * 1345.52 * 1.2 = 75285,04 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} * 0.22 = 74340,57 * 0.22 = 16354.93$ грн.

II. $С_{ВІД} = С_{ЗП} * 0.22 = 75285,04 * 0.22 = 16562.71$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 * М * К_З = 12 * 7000 * 0,2 = 16800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г * (1 + К_З) = 16800 * (1 + 0.2) = 20160 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} * 0.22 = 20160 * 0,22 = 4435.20 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1.15 * 0.25 * 10000 = 2875 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ}\ Ц_{ПР}\ К_Р = 1.15 * 10000 * 0.05 = 575 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – (Д_В + Д_С) – Д_Р) * t_З * К_В = (365 – 116 – 4) * 8 * 0.9 = 1764 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} * N_С * Ц_{ЕН} = 1764 * 0,156 * 2.7515 = 757.17 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} * 0.67 = 10000 * 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$СЕКС = СЗП + СВІД + СА + СР + СЕЛ + СН$$

$$С_{ЕКС} = 20160 + 4435.20 + 2875 + 575 + 757.17 + 6700 = 35502.37 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 35502.37 / 1764 = 20.13 \text{ грн/час.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} * Т$$

I. $С_М = 20,13 * 1328,64 = 26745.52$ грн.;

II. $С_М = 20,13 * 1345.52 = 27085.32$ грн.;

Накладні витрати складають 67% від заробітної плати:

.

$$С_Н = С_{ЗП} * 0,67$$

I. $С_Н = 74340,57 * 0,67 = 49808,18$ грн.;

II. $С_Н = 75285,04 * 0,67 = 50440,98$ грн.;

Отже, вартість розробки ПП за варіантами становить:

СПП = СЗП+ СВІД+ СМ +СН

I. $С_{ПП}$ = 74340,57 + 16354.93 + 26559.51 + 49808,18 = 167063.19 грн.;

II. $С_{ПП}$ = 75285,04 + 16562.71 + 26896.94 50440,98 = 169185.67 грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$КТЕРj = ККj / СФj,$$

$$К_{ТЕР1} = 3.98 / 167063.19 = 23,82 *10^{-6};$$

$$К_{ТЕР2} = 3.39 / 169185.67 = 20,04 * 10^{-6};$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 23,82 $*10^{-6}$.

4.6 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 15,38 * 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- фреймворк машинного навчання - Tensorflow;
- середовище розробки – Jupyter Notebook.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, функціонал і швидкодію.

Розділ 5 Висновки

В цій роботі було детально досліджено предметну область сфери масштабування зображень. Було проведено дослідження щодо актуальності цієї задачі і показано, що останнім часом використання методів масштабування зображень набуває все більше і більше прикладних задач. За останні часи розширена здатність телевізорів і моніторів досить швидко почалася збільшуватися, що призвело до великої потреби масштабувати зображення.

Також було поставлено задачу дослідження і розглянуто різні підходи, що були винайдені: білінійна та бікубічна інтерполяції, метод Ланцоша, метод найближчого сусіда, модель нейронних мереж SRGAN.

В основній частині роботи було описано алгоритми роботи основних методів масштабування зображення, їх особливості і недоліки. Найкраще себе показувала бікубічна інтерполяція, проте вона страждала від швидкості роботи, що у деяких випадках створювало велику перешкоду. Також було чітко описано особливості предметної області, а саме артефакти, що виникають під час масштабування. Цими артефактами стали згладжування, розмиття та сяйво навколо країв. Було розглянуто метод, який допомагає подолати згладжування, проте й він мав свої недоліки.

Кожен з наведених алгоритмів - метод найближчого сусіда, білінійна та бікубічна інтерполяції і метод Ланцоша – мали свої переваги і недоліки. В основному це проявлялося у збільшенні чи зменшенні того чи іншого артефакту. Тому необхідно знаходити баланс між цими методами і опиратися на поставлену задачу, що стоїть.

Ще одною особливістю, а точніше, обмеженням, являлося час роботи і кількість пам’яті. У той час як на досить сучасних і потужних комп’ютерах всі методи можуть працювати досить швидко, то на деяких смартфонах

виникають обмеження на час роботи і кількість необхідної пам’яті. У цьому випадку бікубічна інтерполяція або метод Ланцоша можуть не задовільняти ці обмеження і зазвичай обирають більш прості методи, такі як метод найближчого сусіда або білінійної інтерполяції. Також можна вносити деякі модифікації у складні методи, аби збільшити їх швидкість роботи або кількість необхідної пам’яті. Це було показано у модифікованому алгоритмі бікубічної інтерполяції. Він дещо втрачав якість масштабування, проте за рахунок цього досить сильно зменшувалось використання пам’яті.

Головною частиною роботи є опис та дослідження методів масштабування зображень за допомогою нейромереж. Для цього було обрано генеративно змагальну нейронну мережу SRGAN. Вона, як і всі генеративно змагальні мережі, складається з двох частин: генератора та дискримінатора. Перший намагається згенерувати масштабовано зображення таким, чином, щоб це зображення було неможливо відрізнити від оригіналу, в той час як дискримінатор вже відрізняє реальні зображення від згенерованих. Таким чином, у постійному змаганні, обидві мережі навчаються, одна згенерувати максимально правдоподібне зображення, а інша відрізнити його від справжнього. Також у роботі було описано перцептивну функцію втрат, що є однією з найголовніших особливостей цієї нейромережі. Оскільки звичайна функція втрат MSE, що зазвичай використовується, базується на попіксельній різниці і не бере до уваги загальних концептів зображення, його характеристик, ознак, ребер і інше, то було введено перцептивну функцію втрат. Вона, у свою чергу, вже бере до уваги той факт, як людське око сприймає зображення у цілому. Це надає нейронній мережі показувати візуально більш сприйнятливі і приємні результати, ніж описані раніше методи інтерполяції.

Результати SRGAN були продемонстровані поряд з відповідними результатами інших методів масштабування, а також відповідними показниками якості PSNR. Нейронна мережа не завжди генерувала

зображення, у яких значення PSNR було найбільш високим. Зазвичай вона мала менше за середнє значення цього показника. Проте це не заважало їй показувати найкращі результати за візуальним представленням. Це зумовлюється тим, що навчання даної мережі базується на перцептивній функції втрат. Якби вона навчалася на MSE, то можливо результат був би кращий з точки зору PSNR, тому що PSNR базується на MSE. Також було виявлено появу деяких артефактів у роботі нейронної мережі. Першим було виявлено появу сірого фону у зображенні, що в свою чергу додатково зменшувало показник PSNR, бо фон займає більшу частину зображення і попіксельня різниця на цій ділянці показує поганий результат, хоча візуально досить важко відрізнити від оригіналу. Наступним артефактом, що спостерігався став орел навколо країв. При чому він з’явився при роботі мережі з мультиплікативними зображеннями і картиною, яка теж була виконана у цьому стилі. Можливо це викликано тим фактом, що мережа не навчалася на мультиплікативних зображень.

Нейронна мережа показала чудові результати у масштабуванні зображень. Візуальний результат обігнав описані у роботі методи при збільшенні зображення з коефіцієнтом масштабування, що дорівнює 4.

Література

1. Jonathan Hui GAN—RSGAN & RaGAN (A new generation of cost function.). URL: https://medium.com/@jonathan_hui/gan-rsgan-ragan-a-new-generation-of-cost-function-84c5374d3c6e
2. Улучшаем текстуры с помощью ESRGAN. URL: https://dtf.ru/science/37841-uluchshaem-tekstury-s-pomoshchyu-esrgan
3. Christian Ledig, Lucas Theis, Ferenc Husz´ar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi Photo-RealisticSingleImageSuper-ResolutionUsingaGenerativeAdversarial Network. URL: https://arxiv.org/pdf/1609.04802.pdf
4. Ethan E. Danahy, Sos S. Agaian, Karen A. Panetta Algorithms for the resizing of binary and grayscale images using a logical transform. URL: https://sisu.ut.ee/sites/default/files/imageprocessing/files/interpolation2.pdf
5. Angelos Amanatiadis and Ioannis Andreadis A survey on evaluation methods for image interpolation. URL: https://sisu.ut.ee/sites/default/files/imageprocessing/files/interpolation3.pdf
6. Digital Image Interpolation. URL: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm
7. Digital Photo Enlargement. URL: https://www.cambridgeincolour.com/tutorials/digital-photo-enlargement.htm
8. Chetan Suresh, Sanjay Singh, Ravi Saini, Anil K Saini. URL: https://www.researchgate.net/profile/Amarjot_Singh/publication/272092207_A_Novel_Visual_Cryptographic_Method_for_Color_Images/links/57f3d0fe08ae91deaa5acb81/A-Novel-Visual-Cryptographic-Method-for-Color-Images.pdf?origin=publication_detail
9. Нейронні мережі – шлях до глибинного навчання. URL: https://codeguida.com/post/739

10. PSNR и SSIM или как работать с изображениями под C. URL: https://habr.com/ru/post/126848/
11. Generative adversarial networks. URL: https://habr.com/ru/post/352794/
12. Пашин В.П. Функционально-стоимостный анализ конструкторско-технологических решений. - К.: РДЭНТП «Знание» УССР, 1989. - 22с.
13. Пашін В.П. Оцінка конкурентоспроможності електронних пристроїв на стадії проектування. - К. Економічний вісник НТУУ „КПІ", 2006. - №3. с. 252-255.
14. Пашин В.П. Управление качеством изделий на основе функционально-стоимостного анализа. - К.: «Технология и организация производства», 1989. - №1. с. 17-19.
15. Пашін В. П. Методичні вказівки до виконання економіко-організаційного розділу дипломних проектів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу: Навч. посібник / Пашін В. П., Романов В. В., Єгорова Н. В – К.: НТУУ “КПІ”, 2011. – 118 с.

Додаток А

Дипломна робота

- **Об‘єкт дослідження** — зображення малого розміру, зображення з низькою роздільною здатністю
- **Предмет дослідження** — методи нейронних мереж, інтерполяційні для збільшення забраження та покращення їх роздільної здатності
- **Мета роботи** — дослідити моделі нейронних мереж для покращення якості зображення з малою роздільною здатністю

Актуальність задачі

- Пристосування зображення відповідно до цільового дисплею
- Покращення якості текстур у старих відеоіграх
- Збільшення роздільної здатності мультфільмів
- Додаткова обробка зображень

Постановка задачі

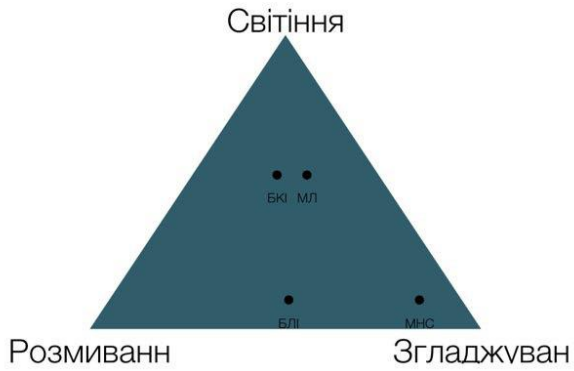
- Провести порівняльний аналіз існуючих методів та підходів
- Дослідити методи покращення якості зображення з малою роздільною здатністю на основі глибокої нейронної змагальної мережі
- Розробити програмний продукт для відновлення зображення з високою роздільною здатністю з аналогу з малою роздільною здатністю і аналізу якості методів масштабування зображення
- Дослідити критерії якості масштабування зображень

Особливості предметної області

Оригінал Згладжування Розмивання Сяйво країв

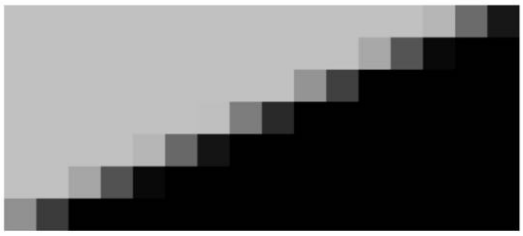
Існуючі методи масштабування зображення

- Метод найближчого сусіда (МНС)
- Білінійна інтерполяція (БЛІ)
- Бікубічна інтерполяція (БКІ)
- Метод Ланкоша (МЛ)



Порівняння існуючих методів

Оригінал



Метод найближчого сусіда

Методи розв‘язання задачі, що використовують нейромережі

- **Модифіковані згорткові мережі SRCNN**

Згорткові нейронні мережі, що складаються лише з трьох згорткових шарів: вилучення характеристик і їх представлення, нелінійне відображення і реконструкція.

- **Мережі глибинного навчання VDSR**

Нейронна мережа, що за структурою схожа на SRCNN, проте має набагато глибші шари для досягнення більшої точності

- **Генеративні змагальні мережі SRGAN**

Критерії якості

- Аналіз Фур‘є
- Використання метрик якості інтерполяції
- Аналіз інтерполяції ребер
- Порівняння з реальними зображеннями
- Обчислювальна складність
- Використання пам‘яті

Метрики якості інтерполяції

Дано:

$I(x, y)$ — вхідне зображення, розміром m x n

$\hat{I}(x, y)$ — вихідне зображення, розміром m x n

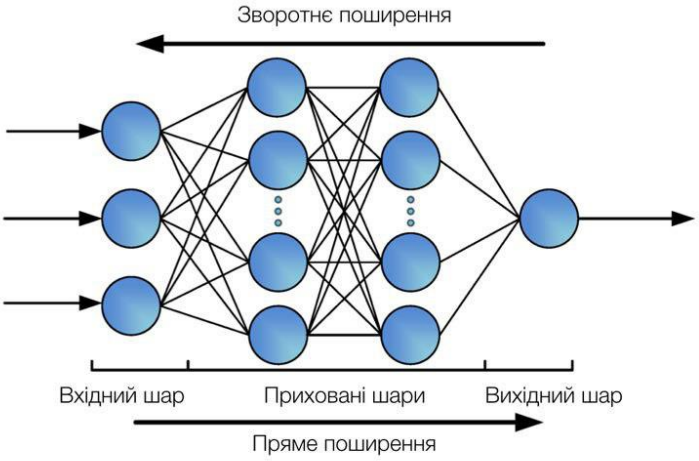
MAX_I — максимальне значення пікселя в зображенні

Метрики:

$$RMSE = \left(\frac{1}{m \times n} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} (\hat{I}(x, y) - I(x, y))^2 \right)^{1/2}$$

$$PSNR = 20 \times \log_{10} \left(\frac{MAX_I}{RMSE} \right)$$

Нейронна мережа

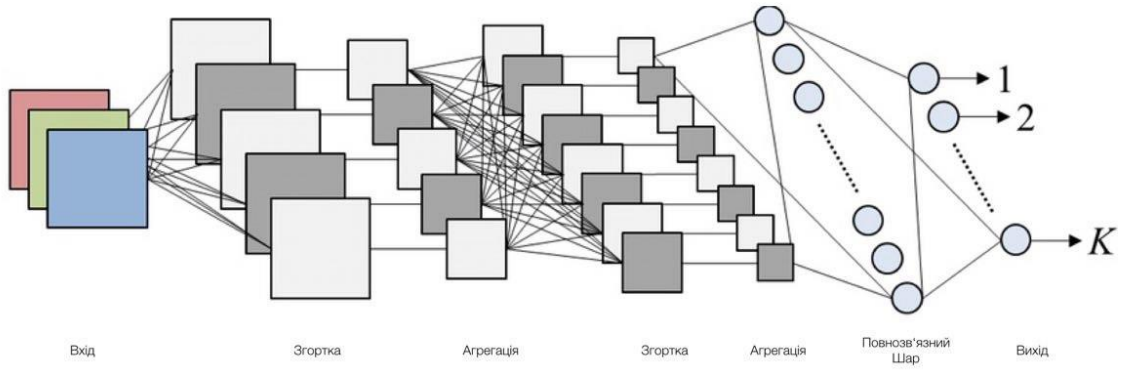


$$z^{(l+1)} = W^{(l)} h^{(l)} + b^{(l)}$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

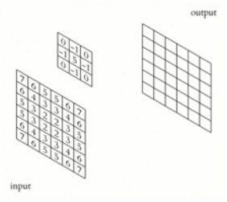
$$J(w, b) = \frac{1}{m} \sum_{i=0}^{m} J(w, b, x^{(i)}, y^{(i)})$$

Згорткові нейронні мережі



Стандартний шар згорткової нейронної мережі

- Згортка у вигляді лінійного відображення, яка знаходить локальні ознаки
- Нелінійна функція, що покомпонентно застосовується до результатів згортки
- Субдискретизація, яка зазвичай скорочує геометричний розмір отриманих тензорів



Анімація згортки

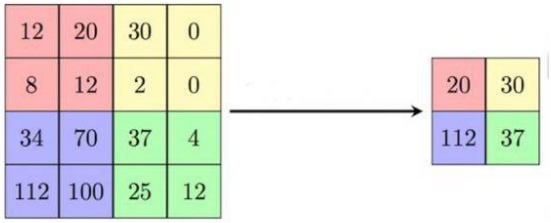
$W_{a,b}$ — ваги

$y_{i,j}^{l}$ — результат згортки

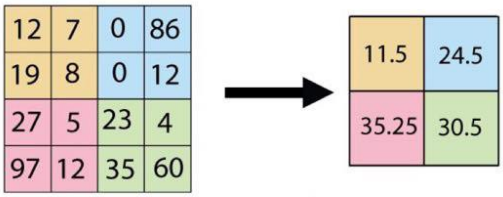
$x_{i,j}^{l}$ — результат попереднього шару

$$y_{i,j}l = \sum_{-d \le a,b \le d} W_{a,b} x_{i+a,j+b}^{l}$$

Субдискретизація



Max-Pooling



Average-Pooling

$$x^l = f(a^l \times subsample(x^{l-1}) + b^l)$$

x^l — вихід слою l

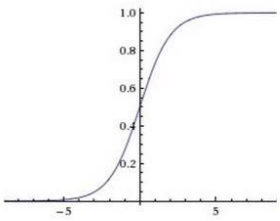
$f()$ — функція активації

a^l, b^l — коефіцієнти зсуву шару l

$subsample()$ — операція вибірки локальних значень

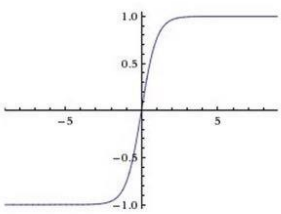
Скорочує геометричний розмір отриманих тензорів

Функція активації



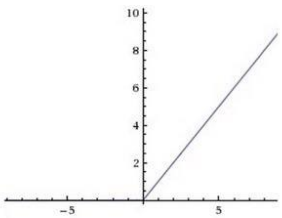
Сигмоїда

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



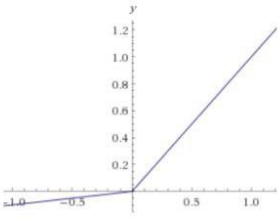
Гіперболічний тангенс

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



ReLU

$$\sigma(x) = \max(0,x)$$



Leaky ReLU

$$\sigma(x) = \max(0.01x, x)$$

Методи навчання

Nesterov Accelerated Gradient:

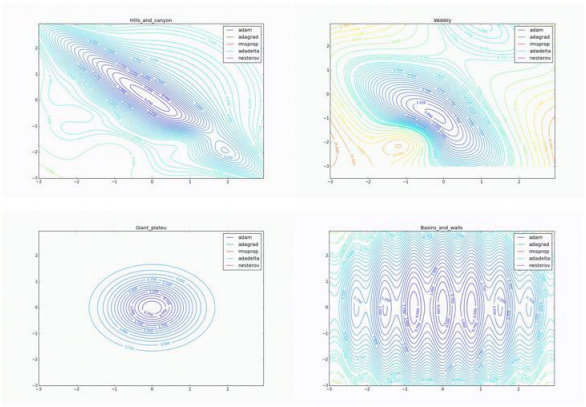
$$\upsilon_t = \gamma \upsilon_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - \upsilon_t$$

Adagrad:

$$G_t = G_t + g_t^2$$
$$\theta_{t+1} = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

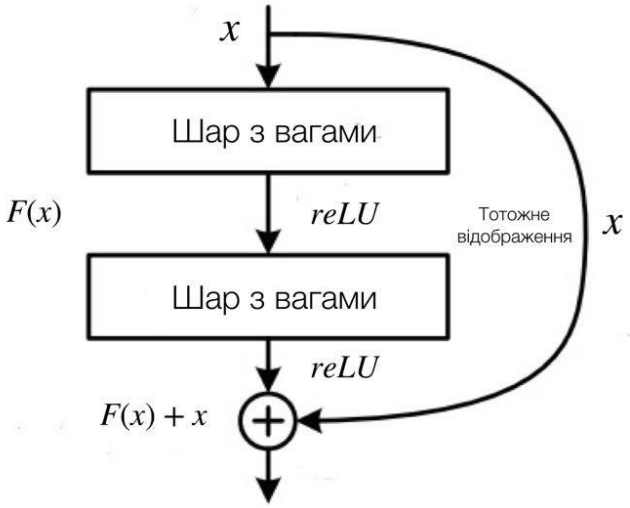
Adam:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{\upsilon}_t = \frac{\upsilon_t}{1 - \beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t + \epsilon}} \hat{m}_t$$

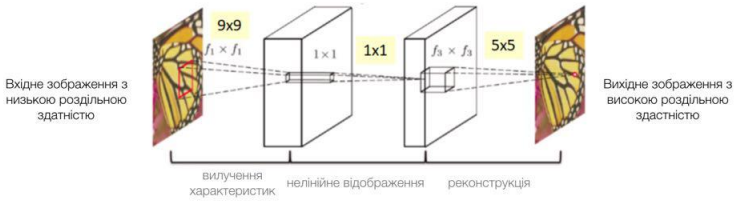


Обхідне з‘єднання

- Швидке навчання
- Тотожне відображення
- Уникнення зникаючих градієнтів
- Навчання відбувається за принципом ансамблю моделей
- Динамічна модель



Модифіковані згорткові мережі SRCNN

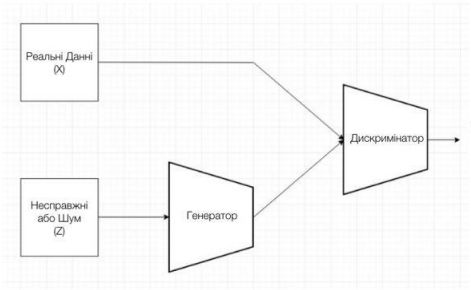


Вилучення характеристик: $F_1(Y) = \max(0, W_1 \times Y + B_1)$

Нелінійне відображення: $F_2(Y) = \max(0, W_2 \times F_1(Y) + B_2)$

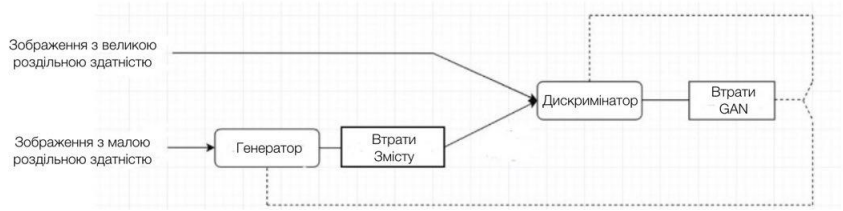
Реконструкція: $F(Y) = W_3 \times F_2(Y) + B_3$

Генеративні змагальні мережі (GANs)



Генератор створює дані з розподілу вірогідності, а дискримінатор вирішує чи вхідні дані є реальними чи згенерваними. Генератор намагається оптимізувати свій вивід, щоб він був якнайправдоподінішим. А дискримінатор, у свою, навчається все краще відрізняти згенеровані дані від реальних.

Основна ідея SRGAN



Головною проблемою завжди залишалося деталізоване відновлення текстур. Останні роботи мінімізували MSE, що давало високий результат PSNR. Проте використання цих метрик призводуть до того, що візуально зображення не сприймаються оком людини. Тому виникла ідея у створенні моделі , яка змогла б розуміти перцептивну різницю між реальними зображеннями і створеними моделлю.

Перцептивна функція втрат

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR}$$

$$l_X^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

$G_{\theta_G}(I)$ — результат роботи генератора

$D_{\theta_D}(I)$ — вірогідність того, що I реальне зображення

I^{LR} — зображення з низькою роздільною здатністю

I^{HR} — зображення з високою роздільною здатністю

Навчання SRGAN

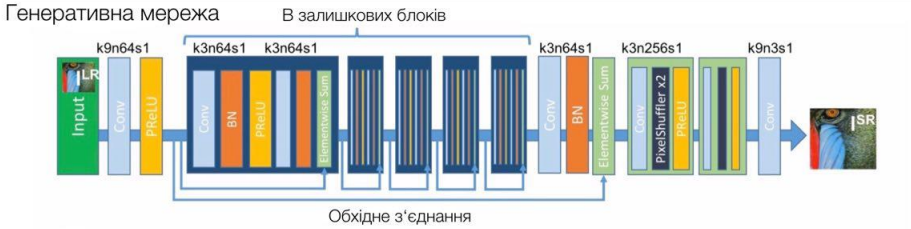
SRGAN поєднує мережу глибинного навчання і змагальну мережу

Процедура навчання має наступний вигляд:

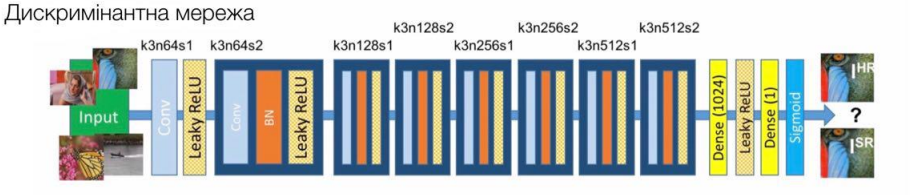
- Зображення з високою роздільною здатністю обробляють, щоб отримати його аналог з низькою роздільною здатністю. Після цього кроку ми маємо обидва варіанти зображення у навчальній вибірці.
- Зображення з низькою роздільною здатністю подають до генератора, який масштабує його і відтворює аналог з великою роздільною здатністю
- Використовують дискримінатор, щоб відрізнити реальне зображення від згенерованого і за допомогою оберненого розподілу помилки витрат GAN тренують дискримінатор і генератор

Архітектура SRGAN

Генеративна мережа



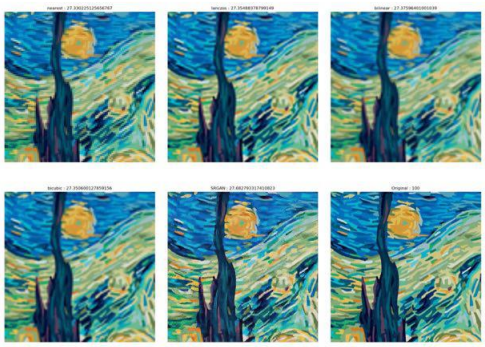
Дискримінантна мережа



Результати роботи

- Високе значення PSNR
- Немає згладжування
- Зникли нерівності по краях
- Менше згладжування текстур

Результати роботи

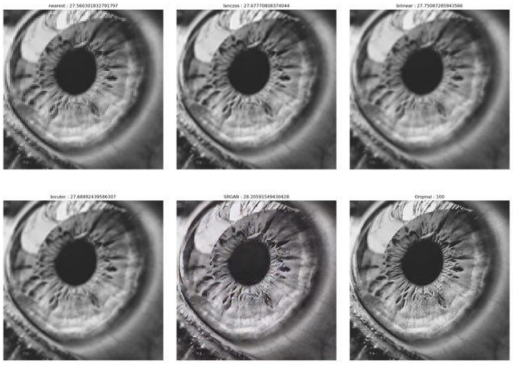


- Однакова метрика PSNR
- Сприйняття оком набагато краще

Результати роботи

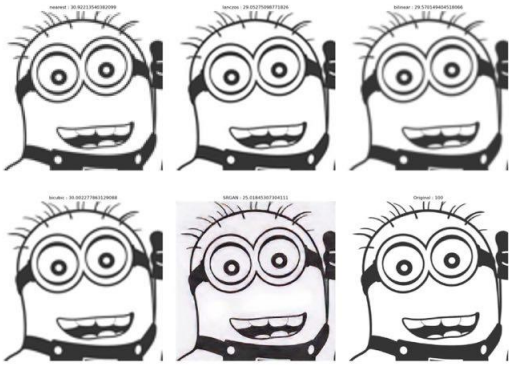
- Низьке значення PSNR
- Ореоли навколо країв
- Висока чіткість

Результати роботи



- Висока деталізація текстур
- Немає згладжування
- Чіткі краї

Результати роботи



- Сірий фон
- Висока деталізація
- Немає згладжування
- Немає розмиття
- Низьке значення PSNR

Результати

Глибокі нейронні змагальні мережі SRGAN показують кращі результати за інтерполяційні методи, а саме:

- Масштабоване зображення є менш згладженим
- Менш розмитим
- Критерієм якості зображення є перцептивна функція втрат
- Зображення краще сприймаються людиною
- Більш чіткі ребра

Висновки

- Проведено порівняльний аналіз інтерполяційних методів: білінійної, бікубічної інтерполяції, Ланцоша
- Вивчено теоретичні основи згорткових та змагальних нейронних мереж
- Досліджено можелі нейронних мереж для покращення якості зображення з малою роздільною здатністю
- Проаналізовано можливості моделей нейронних для реконструювання зображення з великою роздільною здатністю шляхом масштабування.

Дякую за увагу!

Додаток Б

```
# Modules
from keras.layers import Dense
from keras.layers.core import Activation
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import UpSampling2D
from keras.layers.core import Flatten
from keras.layers import Input
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.models import Model
from keras.layers.advanced_activations import LeakyReLU, PReLU
from keras.layers import add


# Residual block
def res_block_gen(model, kernal_size, filters, strides):

    gen = model

    model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides, padding = "same")(model)
    model = BatchNormalization(momentum = 0.5)(model)
    # Using Parametric ReLU
    model = PReLU(alpha_initializer='zeros', alpha_regularizer=None, alpha_constraint=None,
shared_axes=[1,2])(model)
    model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides, padding = "same")(model)
    model = BatchNormalization(momentum = 0.5)(model)

    model = add([gen, model])

    return model


def up_sampling_block(model, kernal_size, filters, strides):

    # In place of Conv2D and UpSampling2D we can also use Conv2DTranspose (Both are used for Deconvolution)
    # Even we can have our own function for deconvolution (i.e one made in Utils.py)
    #model = Conv2DTranspose(filters = filters, kernel_size = kernal_size, strides = strides, padding =
"same")(model)
    model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides, padding = "same")(model)
    model = UpSampling2D(size = 2)(model)
    model = LeakyReLU(alpha = 0.2)(model)

    return model


def discriminator_block(model, filters, kernel_size, strides):

    model = Conv2D(filters = filters, kernel_size = kernel_size, strides = strides, padding = "same")(model)
    model = BatchNormalization(momentum = 0.5)(model)
    model = LeakyReLU(alpha = 0.2)(model)

    return model

# Network Architecture is same as given in Paper https://arxiv.org/pdf/1609.04802.pdf
class Generator(object):

    def __init__(self, noise_shape):
```

```
        self.noise_shape = noise_shape

    def generator(self):

                gen_input = Input(shape = self.noise_shape)

                model = Conv2D(filters = 64, kernel_size = 9, strides = 1, padding = "same")(gen_input)
                model = PReLU(alpha_initializer='zeros', alpha_regularizer=None, alpha_constraint=None,
shared_axes=[1,2])(model)

                gen_model = model

        # Using 16 Residual Blocks
                for index in range(16):
                    model = res_block_gen(model, 3, 64, 1)

                model = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = "same")(model)
                model = BatchNormalization(momentum = 0.5)(model)
                model = add([gen_model, model])

                # Using 2 UpSampling Blocks
                for index in range(2):
                    model = up_sampling_block(model, 3, 256, 1)

                model = Conv2D(filters = 3, kernel_size = 9, strides = 1, padding = "same")(model)
                model = Activation('tanh')(model)

                generator_model = Model(inputs = gen_input, outputs = model)

                return generator_model

# Network Architecture is same as given in Paper https://arxiv.org/pdf/1609.04802.pdf
class Discriminator(object):

    def __init__(self, image_shape):

        self.image_shape = image_shape

    def discriminator(self):

        dis_input = Input(shape = self.image_shape)

        model = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = "same")(dis_input)
        model = LeakyReLU(alpha = 0.2)(model)

        model = discriminator_block(model, 64, 3, 2)
        model = discriminator_block(model, 128, 3, 1)
        model = discriminator_block(model, 128, 3, 2)
        model = discriminator_block(model, 256, 3, 1)
        model = discriminator_block(model, 256, 3, 2)
        model = discriminator_block(model, 512, 3, 1)
        model = discriminator_block(model, 512, 3, 2)

        model = Flatten()(model)
        model = Dense(1024)(model)
        model = LeakyReLU(alpha = 0.2)(model)

        model = Dense(1)(model)
        model = Activation('sigmoid')(model)

        discriminator_model = Model(inputs = dis_input, outputs = model)
```

```
    return discriminator_model
```

```
    from Network import Generator, Discriminator
    import Utils_model, Utils
    from Utils_model import VGG_LOSS

    from keras.models import Model
    from keras.layers import Input
    from tqdm import tqdm
    import numpy as np
    import argparse

    np.random.seed(10)
    # Better to use downscale factor as 4
    downscale_factor = 4
    # Remember to change image shape if you are having different size of images
    image_shape = (384,384,3)

    # Combined network
    def get_gan_network(discriminator, shape, generator, optimizer, vgg_loss):
        discriminator.trainable = False
        gan_input = Input(shape=shape)
        x = generator(gan_input)
        gan_output = discriminator(x)
        gan = Model(inputs=gan_input, outputs=[x,gan_output])
        gan.compile(loss=[vgg_loss, "binary_crossentropy"],
                    loss_weights=[1., 1e-3],
                    optimizer=optimizer)

        return gan

    # default values for all parameters are given, if want defferent values you can give via commandline
    # for more info use $python train.py -h
    def train(epochs, batch_size, input_dir, output_dir, model_save_dir, number_of_images, train_test_ratio):

        x_train_lr, x_train_hr, x_test_lr, x_test_hr = Utils.load_training_data(input_dir, '.jpg', number_of_images,
train_test_ratio)
        loss = VGG_LOSS(image_shape)

        batch_count = int(x_train_hr.shape[0] / batch_size)
        shape = (image_shape[0]//downscale_factor, image_shape[1]//downscale_factor, image_shape[2])

        generator = Generator(shape).generator()
        discriminator = Discriminator(image_shape).discriminator()

        optimizer = Utils_model.get_optimizer()
        generator.compile(loss=loss.vgg_loss, optimizer=optimizer)
        discriminator.compile(loss="binary_crossentropy", optimizer=optimizer)

        gan = get_gan_network(discriminator, shape, generator, optimizer, loss.vgg_loss)

        loss_file = open(model_save_dir + 'losses.txt' , 'w+')
        loss_file.close()
```

```
        for e in range(1, epochs+1):
            print ('-'*15, 'Epoch %d' % e, '-'*15)
            for _ in tqdm(range(batch_count)):

                rand_nums = np.random.randint(0, x_train_hr.shape[0], size=batch_size)

                image_batch_hr = x_train_hr[rand_nums]
                image_batch_lr = x_train_lr[rand_nums]
                generated_images_sr = generator.predict(image_batch_lr)

                real_data_Y = np.ones(batch_size) - np.random.random_sample(batch_size)*0.2
                fake_data_Y = np.random.random_sample(batch_size)*0.2

                discriminator.trainable = True

                d_loss_real = discriminator.train_on_batch(image_batch_hr, real_data_Y)
                d_loss_fake = discriminator.train_on_batch(generated_images_sr, fake_data_Y)
                discriminator_loss = 0.5 * np.add(d_loss_fake, d_loss_real)

                rand_nums = np.random.randint(0, x_train_hr.shape[0], size=batch_size)
                image_batch_hr = x_train_hr[rand_nums]
                image_batch_lr = x_train_lr[rand_nums]

                gan_Y = np.ones(batch_size) - np.random.random_sample(batch_size)*0.2
                discriminator.trainable = False
                gan_loss = gan.train_on_batch(image_batch_lr, [image_batch_hr,gan_Y])


            print("discriminator_loss : %f" % discriminator_loss)
            print("gan_loss :", gan_loss)
            gan_loss = str(gan_loss)

            loss_file = open(model_save_dir + 'losses.txt' , 'a')
            loss_file.write('epoch%d : gan_loss = %s ; discriminator_loss = %f\n' %(e, gan_loss, discriminator_loss)
)
            loss_file.close()

            if e == 1 or e % 5 == 0:
                Utils.plot_generated_images(output_dir, e, generator, x_test_hr, x_test_lr)
            if e % 500 == 0:
                generator.save(model_save_dir + 'gen_model%d.h5' % e)
                discriminator.save(model_save_dir + 'dis_model%d.h5' % e)


    if __name__== "__main__":

        parser = argparse.ArgumentParser()

        parser.add_argument('-i', '--input_dir', action='store', dest='input_dir', default='./data/' ,
                            help='Path for input images')

        parser.add_argument('-o', '--output_dir', action='store', dest='output_dir', default='./output/' ,
                            help='Path for Output images')

        parser.add_argument('-m', '--model_save_dir', action='store', dest='model_save_dir', default='./model/' ,
                            help='Path for model')

        parser.add_argument('-b', '--batch_size', action='store', dest='batch_size', default=64,
                            help='Batch Size', type=int)

        parser.add_argument('-e', '--epochs', action='store', dest='epochs', default=1000 ,
```

```
                help='number of iteratios for trainig', type=int)

        parser.add_argument('-n', '--number_of_images', action='store', dest='number_of_images', default=1000 ,
                help='Number of Images', type= int)

        parser.add_argument('-r', '--train_test_ratio', action='store', dest='train_test_ratio', default=0.8 ,
                help='Ratio of train and test Images', type=float)

        values = parser.parse_args()

        train(values.epochs, values.batch_size, values.input_dir, values.output_dir, values.model_save_dir,
values.number_of_images, values.train_test_ratio)
```