Міністерство освіти і науки України
ДВНЗ «Ужгородський національний університет»
Факультет інформаційних технологій
Кафедра інформаційних управляючих систем і технологій

В.О. ЛАВЕР, О.М. ЛЕВЧУК

ОРОБКА ЗОБРАЖЕНЬ ТА МУЛЬТИМЕДІЯ

Навчально-методичний посібник
(для студентів IV курсу денної та заочної форм навчання за спеціальністю 122 «Комп'ютерні науки»)

Ужгород - 2021

УДК 338.48-4(075.8)
ББК 65.433я73
Ч-64

Лавер В.О., Левчук О.М. Обробка зображень: навч.-метод. посіб. / В.О. Лавер, О.М. Левчук. – Ужгород : вид-во ПП «АУТДОР - ШАРК», 2021. – 51 с.

Рецензенти:

Повхан І.Ф., кандидат технічних наук, доцент, декан факультету інформаційних технологій ДВНЗ «Ужгородський національний університет»

Міца О.В., кандидат технічних наук, доцент, завідуючий кафедри інформаційних управляючих систем і технологій ДВНЗ «Ужгородський національний університет»

Рекомендовано до друку методичною комісією
факультету інформаційних технологій ДВНЗ
«Ужгородський національний університет»
(протокол № _ від __.01.2021 р.)

ЗМІСТ

ПЕРЕДМОВА

Розвиток інформаційних технологій зумовлює активний розвиток цифрових методів обробки сигналів. Підсилює цей процес інтеграція сучасних комп'ютерних та телекомунікаційних технологій. Особливого розвитку в умовах сьогодення набувають методи цифрової обробки зображень, оскільки вони становлять значну частину загального трафіку мультисервісних мереж.

Даний навчально-методичний посібник призначений для студентів ДВНЗ «Ужгородський національний університет» за спеціальністю 122 «Комп'ютерні науки».

Навчальна дисципліна «Обробка зображень» вивчається на четвертому курсі денної та заочної форм навчання протягом другого семестру.

Навчально-методичний посібник призначенний допомогти студентам при підготовці до практичних, лекційних занять, а також написання рефератів.

Метою викладання навчальної дисципліни «Обробка зображень» є ознайомлення студентів з сучасними методами обробки зображень, основами стиснення та злиття зображень на основі перетворень, практичні навички з використання методів просторової фільтрації растрів і перетворення Фур'є з метою поліпшення та відновлення зображень, виділення і розпізнавання різноманітних об'єктів.

Завданням вивчення дисципліни «Обробка зображень» є забезпечення студентів вміннями використовувати методи цифрової обробки зображень в практичній діяльності.

У результаті вивчення навчальної дисципліни студент повинен мати компетентності пов'язані із здатністю:

– демонструвати поглиблені знання з комп'ютерних наук;

– самостійно здобувати за допомогою інформаційних технологій і практичної діяльності нові знання та вміння;

- розширювати і поглиблювати свій науковий світогляд; –

- вільно володіти професійними знаннями для аналізу і обробки зображень.
- вільно володіти основами роботи із сучасними бібліотеками для обробки зображень (OpenCV), вміти використовувати нейронні мережі для розпізнавання та класифікації зображень (Tensorflow).

ОСНОВНІ ПОНЯТТЯ ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

1.1. Поняття зображення

Що таке зображення? На це питання можна відповісти по-різному. Найпростішим і найширшим визначенням цього поняття є таке: **Зображення** — це те, що ми бачимо.

Інше визначення: **Зображення** — це інформація, придатна для візуального сприйняття.

В залежності від походження, умовно можна виділити наступні типи зображень:

1. *Рисоване або друковане* (джерелом є художник, поліграфія, принтер);

2. *Оптичне* (розподіл інтенсивності електромагнітного поля, що створюється оптичним прибором у деякій області простору (області локалізації), наприклад, на сітківці ока, на екрані при проектуванні, у площині приймача об'єктиву фотоапарата);

3. *Фотографічне* (оптичне зображення, зареєстроване на фотоматеріалі у результаті хімічного процесу);

4. *Електронне або цифрове* (оптичне зображення, зареєстроване з допомогою електронного приймача, наприклад ПЗЗ-матриці (прилад із зарядовим зв'язком), сканера).

Електронним також називають зображення, що відображається на екрані монітора.

Легко бачити, що цей поділ є умовним. Зображення із одного типу відразу переходить у інший. Ланцюжок цих перетворень у більшості випадків закінчується зображенням на сітківці ока і образом у мозку людини.

Поняття зображення можна формалізувати, описати математично і маніпулювати ним для досягнення певних цілей. Ці маніпуляції назвемо обробкою зображень.

Метою оброки зображень може бути:

1. *Зміна (спотворення) зображення* з метою досягнення тих чи інших ефектів (художнє покращення);

2. *Image Processing* — візуальне (помітне оку) покращення якості зображення (корекція яскравості і контрасту, корекція кольорів та ін.); об'єктивне покращення якості зображення (усунення спотворень зображення різних типів);

3. *Image Analysis* — проведення вимірів на зображенні (аналіз інтерферограм, гартманограм та ін.);

4. *Image Understanding* — розпізнавання образів (розпізнавання символів, відбитків пальців, обличь та ін.).

Для досягнення цих цілей слід розглянути розв'язання таких задач:

1. Дискретизація, квантування і кодування зображень;
2. Геометричні перетворення зображень;
3. Логічні і арифметичні операції над зображеннями;
4. Фільтрація зображень.
5. Препарування зображень.

1.2. Області застосування обробки зображень

У даному підрозділі мова піде про найпопулярніші аспекти застосування технологій та методів цифрової обробки зображень, про їх актуальність та

практичне значення. Аналіз областей використання методів цифрової обробки зображень показав, що вона проникла майже у всі види інформаційної діяльності людини.

Компресія цифрових зображень (стиснення, компактне подання) – одна з найактуальніших проблем цифрової обробки зображень, вона пов'язана з необхідністю економії місця на фізичних носіях інформації. Прикладами практичного використання компресії зображень є:

– стиснення зображень на штучному супутнику Землі з метою збільшення об'єму інформації, що передається за сеанс зв'язку;

– зменшення об'ємів зображень для швидкого завантаження веб-сторінок;

– компактне подання зображень оптоелектронними пристроями (фотоапаратами, камерами) для економного використання дискового простору;

– застосування графічних архіваторів для збереження архівних, історичних, художніх та інших документів;

– стиснення зображень як окремих кадрів відеопотоку для зменшення об'ємів носіїв та скорочення вимог до якості каналів зв'язку.

Фільтрація шумів – одне із завдань ідентифікації об'єктів у військовій справі, важливе значення має при підвищенні чіткості зображень у цифровому телебаченні, фільтруванні сигналів у відповідних блоках сучасної апаратури.

1.3. Поняття сигналу

Розгляду поняття цифрового сигналу та того, у чому полягає його обробка.

Сигнал — це зміна фізичної величини (температури, тиску повітря, світлового потоку, сили струму тощо), що використовується для пересилання даних.

Загалом можна сказати, що сигнал є функцією. Наприклад, сигнал може бути заданим одновимірною функцією $x(t)$,

де t — незалежна змінна, або двовимірною функцією $x(t_1,t_2)$, де t_1 і t_2 — незалежні змінні, що містять певні дані. Будь-яка величина, що вимірюється у часі та/або просторі — це, швидше за все, сигнал. Так, швидкість автомобіля, що рухається, є сигналом. Сигналом є і електрокардіограма - тут час відображається по горизонтальній осі, а рівень сигналу — по вертикальній. Такого роду сигнал показує здоровий стан чийогось серця. Сигнали у системах грають важливу роль у багатьох областях науки і технологій, від медицини до космонавтики. Розрізняють *аналогові* та *цифрові* сигнали.

Аналоговим сигналом називається сигнал, який є неперервним у часі.

Аналоговий сигнал або є вираженим синусоїдальним коливанням, або, у загальному випадку, розкладеним у ряд накладанням синусоїдальних коливань певної амплітуди і частоти. Визначення цифрового сигналу дамо трохи згодом.

Розглянемо приклад: систему запису, передачі, обробки та відтворення голосу (рис. 1.1).

Рис. 1.1. Обробка звуку

Під впливом тиску всередині легень, повітря, за допомогою мовленнєвого апарату, перетворюється у звук. Мовлення — це аналоговий сигнал, який передається звуковими хвилями, що використовують вібрацію молекул повітря. Цей акустичний сигнал, коли доходить до мікрофона, що виступає у ролі перетворювача, перетворюється на електричний сигнал. Обидва ці сигнали є аналоговими

(неперервними). Їхніми графіками є графіки неперервних функцій.

Наступним кроком є перетворення аналогового сигналу у цифровий. На схемі за цей крок відповідає блок A/D (Analog/Digital).Для цього спершу потрібно розбити вісь, що відображає час, на скінченні проміжки. Ця операція називається *дискретизацією*. Далі кожному значенню присвоюється певна величина сигналу. Цей етап називається *квантуванням* (очевидно, що під час квантування ми привносимо певні похибки). Сигнал, до якого застосовано дискретизацію і квантування, називається *цифровим*.

Отже, **цифровий сигнал** — це сигнал, який можна представити у вигляді дискретних (цифрових) значень. На даний момент найпопулярнішими є двійкові цифрові сигнали.

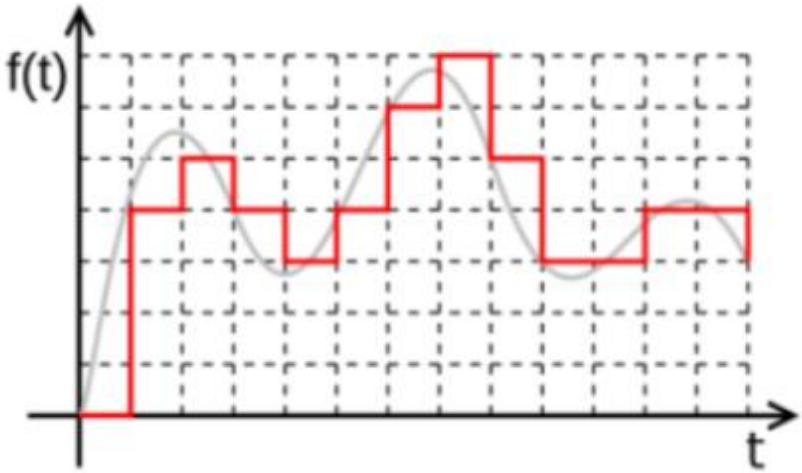


Рис. 1.2. Цифровий сигнал.

На наступному етапі цифровий сигнал поступає у комп’ютер. Потім, після певної обробки, він знову перетворюється на електричний, а згодом і на аналоговий сигнал.

Метою даного курсу є зосередитись на тому, що відбувається у центральній частині рис. 1.1, тобто на обробці цифрових сигналів при роботі із відео та зображеннями.

Рис. 1.3. Обробка зображень

На рис. 1.3 зображені генерація, запис, обробка та сприйняття зображення. Так, електромагнітна енергія видимої частини спектра, що йде від Сонця, відображається від деякого об'єкта, проходить через повітря і фіксується фотоапаратом. У звичайному (нецифровому) фотоапараті сенсор шляхом фотохімічних процесів переводив енергію світла у хімічні зміни на плівці. Ця плівка після обробки — негатив, являє собою аналогове зображення. Для того, щоб перетворити його у цифрове у блоці A/D, можна використати денситометр (що вимірює концентрацію срібних часток на плівці) або сканер. Звісно, зараз ці частини об'єднуються в одну, тож ми отримуємо цифрові фотоапарати.

Після цього зображення обробляється на комп'ютері, для того, щоб зробити його придатним для візуального сприйняття людиною. Тому на зворотному шляху цифрове зображення знову трансформується в аналогове, потім цей сигнал подається на вхід монітору, який перетворює зображення у електромагнітну хвилю, яка вже досягає ока людини. Знов-таки, зараз більш поширені цифрові монітори, до яких прямо поступає цифровий сигнал, а відеоадаптер грає роль D/A-конвертера.

Розглянемо зараз як зображення представляються у пам'яті комп'ютера.

1.4. Представлення зображень у пам'яті комп'ютера

Для перетворення аналогового сигналу у цифровий, у сучасних фотоапаратах використовуються спеціальні сенсори, за допомогою яких ми отримуємо дискретну матрицю $n \times m$. Чим більша кількість світлочутливих елементів, тим кращу якість фотографій можна отримати. Це визначається параметрами матриці.

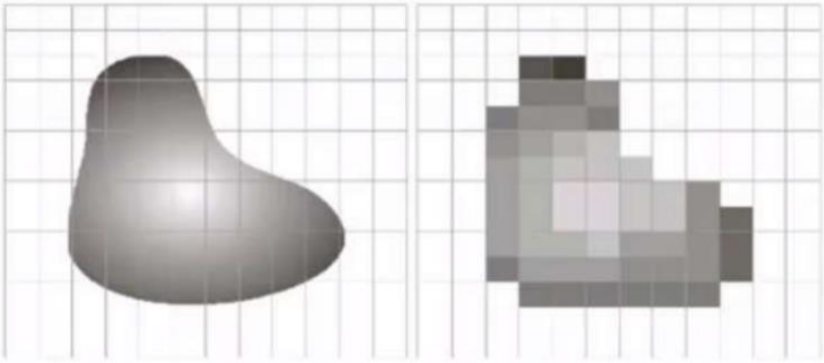


Рис. 1.4. Приклад переводу аналогового зображення у цифрове

При переводі аналогового зображення у цифрове (рис. 1.4), ми розбиваємо на частини не тільки область нашого малюнку, а і квантуємо кольори. Якщо у нас хороша камера, то зображення розбивається на стільки маленьких частинок-пікселів (від англ. picture element), що наше око не помічає різниці.

На якість зображення (чорно-білого) впливає як кількість пікселів, виділених на це зображення, так і кількість біт, виділених на один піксель. Так, якщо на один піксель виділено 8 біт, то у ньому можна представити 256 відтінків. При цьому 0 буде позначати чорний колір, а 255 — білий.

Якщо для представлення чорно-білого малюнку потрібна одна матриця, то для кольорових малюнків ми використовуємо три матриці — червону, зелену і синю (RGB).

За типом даних зображення поділяють на:

– бітові (булевські, логічні);

– байтові (зі знаком і без знаку);

– цілочисельні (зі знаком і без знака);

– дійсні (з фіксованою і плаваючою крапкою);

– кольорові (спеціальний тип даних);

– векторні (піксель є масивом або списком чисельних значень).

ОСНОВИ РОБОТИ ІЗ ЗОБРАЖЕННЯМИ В PYTHON

2.1. Бібліотека OpenCV

Однією із основних бібліотек для комп’ютерного зору є бібліотека OpenCV (англ. Open Source Computer Vision Library, бібліотека комп'ютерного зору з відкритим кодом) — бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях.

Бібліотека розроблена Intel і нині підтримується Willow Garage та Itseez. Сирцевий код бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Біндинги підготовлені для різних мов програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Може вільно використовуватися в академічних та комерційних цілях.

Офіційно проект OpenCV був запущений у 1999 році за ініціативою Intel Research з ціллю розвивати CPU-ресурсомісткі додатки. Основними вкладниками у проект була Intel's Performance Library Team.. На перших етапах розвитку OpenCV основними задачами бібліотеки були:

- Розвивати дослідження у напрямку комп'ютерного зору, забезпечуючи добре оптимізований та відкритий код бібліотеки.
- Поширювати знання у сфері комп'ютерного зору, забезпечуючи загальну інфраструктуру, яку б могли розвивати

розробники, таким чином код ставатиме більш легким для сприйняття та обміну.

- Розвивати засновані на роботі з комп'ютерним зором комерційні додатки, створюючи не залежну від платформи, оптимізовану та безкоштовну бібліотеку. Для цього використовувалася ліцензія, яка не вимагала від таких комерційних додатків бути відкритими.

Перша альфа-версія OpenCV була оприлюднена на IEEE конференції з комп'ютерного зору й розпізнавання образів у 2000 році, і п'ять бета-версій було випущено у період між 2001 і 2005 роками. Перша версія 1.0 була випущена у 2006 році. У середині 2008 року, OpenCV отримала корпоративну підтримку від Willow Garage і знову перейшла у стадію активної розробки. «Пре-релізна» версія 1.1 була випущена у жовтні 2008 року.

Другий великий випуск OpenCV відбувся у жовтні 2009 року. OpenCV 2 включала у себе серйозні зміни у інтерфейсі C++. Ці зміни спрямовані на більш простi, тип-безпечні моделі, додавання нових функцій, і кращу реалізацію існуючих моделей в плані швидкодії (особливо на багатоядерних системам). Офіційні релізи надалі відбуваються кожні 6 місяців.

У серпні 2012 року, підтримку OpenCV було передано некомерційній організації, OpenCV.org.

Бібліотека містить понад 2500 оптимізованих алгоритмів, серед яких повний набір як класичних так і практичних алгоритмів машинного навчання і комп'ютерного зору. Алгоритми OpenCV застосовують у таких сферах:

1. Аналіз та обробка зображень
2. Системи з розпізнавання обличчя
3. Ідентифікації об'єктів
4. Розпізнавання жестів[en] на відео

5. Відстежування переміщення камери
6. Побудова 3D моделей об'єктів
7. Створення 3D хмар точок зі стерео камер
8. Склеювання зображень між собою, для створення зображень всієї сцени з високою роздільною здатністю
9. Система взаємодії людини з комп'ютером
10. Пошуку схожих зображень із бази даних
11. Усування ефекту червоних очей при фотозйомці зі спалахом
12. Стеження за рухом очей
13. Аналіз руху
14. Ідентифікація об'єктів
15. Сегментація зображення
16. Трекінґ відео
17. Розпізнавання елементів сцени і додавання маркерів для створення доповненої реальності

та інших.

2.2. Зчитування, вивід та збереження зображень

Розглянемо основні операції для роботи із зображеннями.

Перш за все, потрібно підключити необхідні бібліотки (рис. 2.1).

```
[1]: import cv2
     import numpy
     from matplotlib import pyplot as plt

     img = cv2.imread('doggo.jpg')
     print(img.shape)
     img

     (374, 374, 3)

[1]: array([[[ 50,  54,  43],
             [ 61,  65,  54],
             [ 74,  75,  65],
             ...,
```

Рис. 2.1. Зчитування зображення

Для зчитування зображень використовується функція **cv2.imread**. Дана функція має два параметри:

- **path** - шлях до файлу, який необхідно відно відкрити;
- **flag** - опція, що вказує, яким чином слід відкрити файл.

Функція **cv2.imread** повертає масив numpy. Якщо файл не буде знайдено, то буде повернено порожній масив.

Параметр **flag** може приймати наступні значення:

- **cv2.IMREAD_COLOR**: завантажуване зображення розглядається як кольорове. Будь-яку прозорість зображення буде проігноровано. Це є значенням параметру за замовчуванням.Замість цього значення можна також передати значення 1.
- **cv2.IMREAD_GRAYSCALE**: завантажуване зображення розглядається як кольорове. Альтернативним варіантом задання цього параметру є число 0.
- **cv2.IMREAD_UNCHANGED**: показує, що завантажуване зображення містить альфа канал (тобто містить інформацію не тільки про значення кольорових каналів, наприклад інформацію про прозорі частини зображення). Альтернативним варіантом задання цього параметру є число -1.

Для виведення зображення на екран існує два можливі варіанти:

1. Вивід в окремому вікні;
2. Вивід в комірці Jupyter Notebook.

Код для першого прикладу наведено на Рис. 2.2.

```
[2]: window_name = 'image'
     cv2.imshow(window_name, img)
     cv2.waitKey(0)
     cv2.destroyAllWindows()
```

Рис. 2.2. Вивід зображення в окремому вікні.

У цьому випадку для виводу використано функцію **cv2.imshow**. Після появи віконця із зображенням, слід викликати функцію **cv2.waitKey(0)** для паузи, та **cv2.destroyAllWindows()** для закриття усіх вікон.

Розглянемо вивід зображення за допомогою бібліотеки matplotlib.

```
img = cv2.imread('doggo.jpg',0)
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])  # сховати осі
plt.show()
```

Рис. 2.2. Вивід зображення за допомогою matplotlib.

Для збереження зображення слід використовувати функцію **cv2.imwrite**.

```
[11]: cv2.imwrite('doggo_grey.png',img_grey)
      img_new = cv2.imread('doggo_grey.png')
```

Рис. 2.3. Збереження зображення.

ЗАУВАЖЕННЯ

Кольорові зображення в OpenCV завантажуються в режимі BGR. Matplotlib виводить зображення в RGB. Тож кольорові зображення, завантажені за допомогою OpenCV можуть некоректно виводитися за допомогою Matplotlib.

Розглянемо деякі основні властивості зображень.

2.3. Базові операції над зображеннями

Оскільки зображення представляються у вигляді матриць (або масивів матриць, як у випадку із кольоровими зображеннями), то над операції над матрицями можуть бути застосовані і до зображень. Так, можна змінювати пікселі зображень, виділяти окремі області зображень і т.д. Аналогічно ми можемо змінювати значення пікселя та виділяти окремі області зображення. Також ми можемо розділяти канали зображення, склеювати їх і т.д.

```
img = cv2.imread('doggo.jpg')

px = img[100,100]
print('Значення пікселя (в форматі BGR):',px)
print('Значення блакитного каналу:',px[0])
```

```
Значення пікселя (в форматі BGR): [96 97 88]
Значення блакитного каналу: 96
```

```
img[100,100] = [255,255,255]
print(img[100,100])
```

```
[255 255 255]
```

Рис. 2.4. Ілюстрація базових операцій

Над зображеннями також можна використовувати арифметичні операції. Якщо додати число до усіх елементів зображення, то збільшиться інтенсивність (зображення стане світліше), якщо відняти – зменшиться.

Самі зображення також можна додавати. При цьому використовується формула:

$g(x)=(1-\alpha)f_0(x)+\alpha f_1(x)$.

Змінюючи α від нуля до одиниці, отримуємо плавний перехід між зображеннями.

Для додавання зображень використовується функція **cv2.addWeighted()**, яка додає зображення за формулою

$$img=\alpha\cdot img1+\beta\cdot img2+\gamma.$$

Приклад відповідної операції наведено на рис. 2.5.

```
[23]: cameraman = cv2.imread('cameraman.png', 0)
      rice = cv2.imread('rice.png', 0)

      result = cv2.addWeighted(cameraman,0.3,rice,0.7,0)

      show_image(result, cmap = 'gray')
```

Рис. 2.5. Зважене додавання зображень.

Над зображеннями можна також проводити побітові логічні операції та змінювати колірні моделі.

Ко́лірна модель — абстрактна модель опису представлення кольорів у вигляді кортежів (наборів) чисел, зазвичай з трьох або чотирьох значень, званих колірними компонентами або колірними координатами. Разом з методом інтерпретації цих даних (наприклад, визначення умов відтворення та / або перегляду — тобто завдання способу реалізації), множина кольорів колірної моделі визначає колірний простір.

Також під колірною моделлю необхідно розуміти спосіб відображення колірної гами в дискретному вигляді, для представлення її в обчислювальних, цифрових системах.

В OpenCV ми здебільшого матимемо справу із трьома колірними моделями: grayscale (чорно-білі зображення),BGR та HSV (або HSB). Розглянемо ці моделі детальніше:

- **Greyscale** - це модель, що зводить усю колірну інформацію до відтінків сірого, тобто до яскравості окремих частин зображення. Ця модель дуже корисна, коли інформація про яскравість є достантньою (наприклад при розпізнаванні облич). Зазвичай кожен піксель представляється 8-бітним числом, від 0 (чорне), до 255 (біле).
- **BGR** - у цій моделі кожен піксель представляється трійкою значень, кожне з яких відповідає за окремий канал: синій, зелений та червоний. В даній моделі трійка (0,0,0) позначає чорний колір, (255,0,0) - синій, (0,255,0) - зелений, (0,0,255) - червоний.
- **HSV** (також HSB) - дана колірна модель, заснована на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation) і значенні кольору (Value), який також називають яскравістю (Brightness). Тон змінюється у проміжку [0,179], насиченість у [0,255], а значення кольору у проміжку [0,255]. Різні програми використовують різні шкали, тож при обробці зображень в OpenCV може виникнути необхідність нормалізації значень.

Для конвертації між різними колірними моделями в OpenCV використовується функція **cv2.cvtColor(input_image, flag)**, де **flag** визначає тип конвертації. Для переведення BGR у Grayscale використовується значення прапорця **cv2.COLOR_BGR2GRAY**. Аналогічно, для переведення BGR у HSV, ми використовуємо прапорець **cv2.COLOR_BGR2HSV**.

АФІННІ ПЕРЕТВОРЕННЯ ЗОБРАЖЕНЬ

Афінне перетворення (лат. affinis, «пов'язаний з») — відображення $f:\mathbb{R}^n \to \mathbb{R}^n$, яке можна записати у вигляді

$f(x)=M \cdot x+v$,

де M — невироджена матриця і $v \in \mathbb{R}^n$.

Інакше кажучи, відображення називається афінним, якщо його можна отримати наступним способом:

- Обрати «новий» базис простору з "новим" початком координат v;
- Координатам x кожної точки простору поставити у відповідність нові координати $f(x)$, які мають те саме положення в просторі відносно "нової" системи координат, яке координати x мали в "старій".

Афінне відображення задовольняє наступним властивостям:

- **Колінеарність**: точки, які лежать на одній лінії до перетворення, лежатимуть на одній прямій і після перетворення.
- **Паралельність**: паралельні лінії залишатимуться паралельними і після трансформації;
- **Опуклість**: опукла множина залишиться опуклою після перетворення;
- **Відношення відразків, що лежать на паралельних прямих**: відношення довжин відрізків, які лежать на двох паралельних прямих, не зміниться після перетворення.

Афінне перетворення застосовуєтсья наступним чином:

- Розглядаємо кожен піксель зображення;
- Знаходимо скалярний добуток координат пікселя та матриці перетворення (матриця залежить від характеру перетворення) для того, щоб знайти нові координати пікселя;

- Визначаємо значення пікселя у перетвореному зображенні. Оскільки скалярний добуток може давати не цілі значення, ми застосовуватимемо інтерполяцію.

В OpenCV є дві функції для афінних перетворень: **cv2.warpAffine** і **cv2.warpPerspective** за допомогою яких можна задати довільні афінні перетворення. При цьому **cv2.warpAffine** приймає матрицю розмірності 2×3, а **cv2.warpPerspective** - матрицю розмірності 3×3.

Розглянемо три основні афінні перетворення:

- **Перенесення**;
- **Поворот**;
- **Маштабування**.

Перенесення є процесом зміщення зображення по різним осям (x, y та z). Для двовимірномго зображення ми можемо застосовувати перенесення по одній або по двом осям незалежно. Матриця перетворення для перенесення має вигляд:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Якщо координати пікселя (x,y,1)(x,y,1) і виконаємо множення на матрицю перетворення, отримаємо координати пікселя у перетвореному зображенні:

$$C_{\text{перетворене}} = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix} = \begin{pmatrix} x + t_x & y + t_y & 1 \end{pmatrix}$$

Отже, кожен піксель у перетвореному зображенні буде зміщений на t_x і t_y по осям x та y, відповідно. Значення t_x та t_y можуть бути як додатними, так і від'ємними.

Для задання перенесення в OpenCV матрицю перетворення слід задати у вигляді

$$M = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{pmatrix}.$$

Дану матрицю можна задати як масив **np.float32** і передати як параметр у **cv2.warpAffine()**.

Третім параметром функції **cv2.warpAffine()** є розмір перетвореного зображення, який задається у вигляді пари (width, height), де width це кількість стовпців, height - кількість рядків у матриці зображення.

Поворот зображення проти годинникової стрілки задається наступною матрицею:

$$T = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Якщо розглядається піксель із координатами $(x \;\; y \;\; 1)$ і виконаємо множення на матрицю перевторення, отримаємо наступні координати для перетвореного зображення:

$$C_{\text{перетворене}} = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta & x\cos\theta + y\sin\theta & 1 \end{pmatrix}.$$

В OpenCV поворот реалізований із можливістю задання центру повороту. Модифікована матриця перетворення має вигляд:

$$\begin{pmatrix} \alpha & \beta & (1-\alpha)\cdot x_{center} - \beta \cdot y_{center} \\ -\beta & \alpha & \beta \cdot x_{center} + (1-\alpha)\cdot y_{center} \end{pmatrix},$$

$$\alpha = scale \cdot \cos\theta;$$
$$\beta = scale \cdot \sin\theta.$$

Для знаходження матриці перетворення можна скористатися функцією **cv2.getRotationMatrix2D**.

Маштабування є зміною відстаней між точками (стиснення або розтягення) вздовж однієї або більше осей. В результаті отримаємо зображення більше або менше, ніж початкове. Маштабуючий множник може бути різним вздовж різних осей. Матриця перетворення визначається як

$$T = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Якщо k_x або k_y менше, ніж 1, то зображення стискується, а значення незаповнених пікселів заповнюються нулями, або іншими значення, залежно від параметру стиснення. Якщо k_x або k_y більше, ніж 1, то зображення розтягуться.

Якщо розглянемо піксель із координатами (x y 1), перетворені координати матимуть вигляд

$$C_{\text{перетворене}} = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} k_x & 0 & 1 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x \cdot k_x & y \cdot k_y & 1 \end{pmatrix}.$$

В OpenCV для маштабування використовується функція **cv2.resize()**. Розмір зображення може бути заданим вручну, або за допомогою маштабуючого множника. При цьому використовуються різні методи інтерполяції, такі як **cv2.INTER_AREA** для зменшення, **cv2.INTER_CUBIC** (повільний метод) та **cv2.INTER_LINEAR** для збільшення. За замовчуванням використовується **cv2.INTER_LINEAR**.

ФІЛЬТРИ У ПРОСТОРОВІЙ ОБЛАСТІ

Зазвичай зображення, сформовані різними інформаційними системами, спотворюються дією шумів. Це ускладнює як їхній візуальний аналіз, так і автоматичну обробку. При вирішенні деяких завдань обробки зображень у ролі шуму можуть виступати ті або інші компоненти самого зображення. Наприклад, при аналізі космічного знімка земної поверхні може стояти завдання визначення границь між її окремими ділянками - лісом і полем, водою й сушею тощо. З погляду цього завдання окремі деталі зображення всередині розділених областей є шумом.

Ослаблення дії шуму досягається **фільтрацією**. При фільтрації яскравість (сигнал) кожної точки вихідного зображення, спотвореного шумом, замінюється деяким іншим значенням яскравості, яке в меншій мірі було спотворене.

Фільтрація зображень здійснюється в **просторовій** і **частотній** областях. **Просторова область зображення** являє собою сукупність пікселів зображення. Відстані на зображенні (у пікселях) відповідають справжнім відстаням (у метрах, дюймах і т.д.). **Частотною областю зображення** називається область, отримана внаслідок дії перетворення Фур'є на зображення.

При **просторовій фільтрації зображень** перетворення виконується безпосередньо над значеннями пікселів зображення. Результатом фільтрації є оцінка корисного сигналу зображення. Це досягається завдяки тому, зображення часто являє собою двовимірну функцію просторових координат, що змінюється по цих координатах повільніше, ніж шум, що також є двовимірною функцією. Це дозволяє при оцінці корисного сигналу в кожній точці зображення взяти до уваги сусідні точки, скориставшись певною подібністю сигналу. В інших випадках, навпаки, ознакою корисного

сигналу є різкі перепади яскравості. Однак, як правило, частота цих перепадів відносно невелика, так що на значних проміжках сигнал або постійний, або змінюється повільно. І в цьому випадку властивості сигналу проявляються при спостереженні не тільки його окремої точки, але й при аналізі її околиці. Поняття околиці є досить умовним.

Фільтри можна поділити на такі два класи:

- лінійні (усереднюючий фільтр, Лапласіан, Ліапласіан Гаусіана і т.д.);
- нелінійні (медіанний фільтр, фільтри "максимум" та "мінімум", фільтри Собеля, Прюітта, Кенні).

4.1. Основи фільтрації

Для фільтрації зображення використовується маска (яку ще називають фільтром), яка зазвичай являє собою двовимірне квадратне вікно, яке рухається по зображенню, діючи тільки на один піксель за різ. Кожне число у фільтрі розглядається як коєфіціент. Коєфіціенти фільтра визначають ефект від фільтрації та вигляд результуючого зображення.

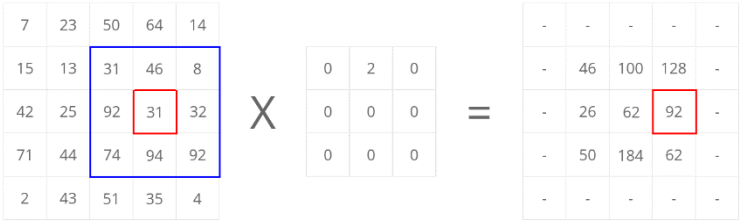


Рис. 4.1. Приклад фільтрації

Даний процес повторюється для кожного пікселя зображення. Цей процес також називають **згорткою**, я фільтр **- ядром згортки**.

Якщо ми застосовуватимемо фільтр до пікселів, що лежать на границі зображення, частина фільтра виходитиме за межі області зображення. Тоді ми можемо додати "рамку"

навколо зображення. Цей процес називається padding. При цьому нові граничні пікселі заповнюються за певними правилами. Є, зокрема, такі варіанти:

- Заповнення нулями;
- Заповнення сталим значенням (наприклад числом 5 чи іншим);
- Заповнення значенням найближчого сусіднього пікселя;
- Заповнення повторенням останнього рядка (стовпця);
- Заповнення останнього рядка першим, і навпаки.

В математиці функції можна поділити на два великі класи: лінійні та нелінійні. Функція називається **лінійною**, якщо для довільних змінних $x, y,$ та довільних коефіцієнтів α та β має місце

$$f(\alpha x+\beta y)=\alpha f(x)+\beta f(y).$$

Функція, яка не є лінійною, називається **нелінійною**.

Лінійні фільтри являють собою певне узагальнення лінійних функцій. Одним із найпоширеніших лінійних фільтрів є **усереднюючий фільтр**.

Ядро усереднюючого фільтру розмірності 5×5, наприклад, матиме наступний вигляд:

$$K = \frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Для фільтрації у OpenCV існує функція **cv2.filter2D**. За педдінг у цій функції відповідає необов'язковий параметр **borderType**.

Рис. 4.2. Усереднюючий фільтр

Цю саму операцію (усереднюючу фільтрацію) можна виконати і за допомогою функцій **cv2.blur()** або **cv2.boxFilter()**. При цьому слід задати розмір фільтра.

Переваги усереднюючого фільтра:

- Прибирає шум;
- Висвітлю зображення, покращує якість.

Недоліки:

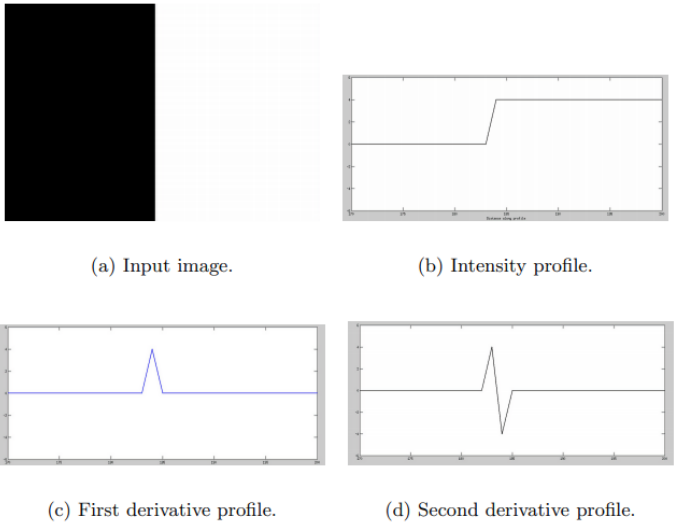
- Границі в зображенні розмиваються;
- Зменшує розкид значень пікселів.

На відміну від усереднюючого фільтру, медіанний фільтр ставить у відповідність кожному пікселю медіанне значення з його околиці. Відповідна функція - **cv2.medianBlur**. Фільтр «максимум» замінює кожен піксель максимальним значенням із його околиці. Функція в OpenCV - **cv2.dilate**. Фільтр «мінімум» фільтр замінює кожен піксель мінімальним значенням із його околиці. Функція в OpenCV - **cv2.erode**.

4.2. Виявлення контурів

Контури являють собою множину точок зображення, у яких є зміна інтенсивності між однією та іншою сторонами зображення.

Зміни інтенсивності сигналу можуть бути виміряні за допомогою перших та других похідних.



(a) Input image. (b) Intensity profile.

(c) First derivative profile. (d) Second derivative profile.

Рис. 4.3. Приклад похідних

Перщі похідні обчислюються за формулами:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x);$$
$$\frac{\partial f}{\partial y} = f(y+1) - f(y).$$

Тобто ми обчислюємо зміну інтенсивності пікселів по відповідній змінній. Другі похідні, відповідно, матимуть вигляд

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x);$$
$$\frac{\partial^2 f}{\partial y^2} = f(y+1) + f(y-1) - 2f(y).$$

4.2.1. Фільтр Собеля

Даний фільтр використовується для знаходження вертикальних та горизотальних контурів. Фільтр Собеля використовує наступні маски:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Рис. 4.4. Маски фільтра Собеля

Важливими властивостями даного фільтра є:

- Сума коєфіцієнтів у масці рівна нулю.
- Побічним ефектом застосування даного фільтру є шум.

Для згалжування шуму використовуються коєфіцієнти 2 та -2.

Рис. 4.5. Фільтр Собеля

4.2.2. Фільтр Прюітта

Даний фільтр використовує наступні маски:

-1	0	+1
-1	0	+1
-1	0	+1

G_x

+1	+1	+1
0	0	0
-1	-1	-1

G_y

Рис. 4.6. Маски фільтра Прюітта

Крім горизонтальних та вертикальних контурів, є маски для діагональних контурів.

Рис. 4.7. Фільтр Прюітта

4.2.3. Фільтр Кенні

Кенні (John F. Canny; 1953 р.) вивчив математичну проблему пошуку фільтра, оптимального за критеріями виділення, локалізації та мінімізації кількох відгуків одного краю. Дектектор повинен реагувати на контури, але при цьому ігнорувати хибні межі. Кенні ввів поняття Non-Maximum Suppression, яке означає, що пікселями границь оголошуються тотчки, в яких досягається локальний максимум градієнта у напрямку вектора градієнта. І хоч роботу Кенні було опубліковано в 1986, фільтр, запропонований ним, актуальний і досі.

Алгоритм складається із п'яти кроків:

1. **Зглажування.** Розмиття зображення для видалення шуму.

2. **Пошук ргадієнтів.** Границі відмічаються там, де градієнт зображення набуває максимального значення.

3. **Приглушення не-максимумів.** Тільки локальні максимум відмічаються як межі.

4. **Подвійна порогова фільтрація.** Потенціальні межі відмічаються порогами.

5. **Трассування області неоднозначності.** Границі визначаються шляхом подавлення усіх країв, не зв'язаних із сильними границями.

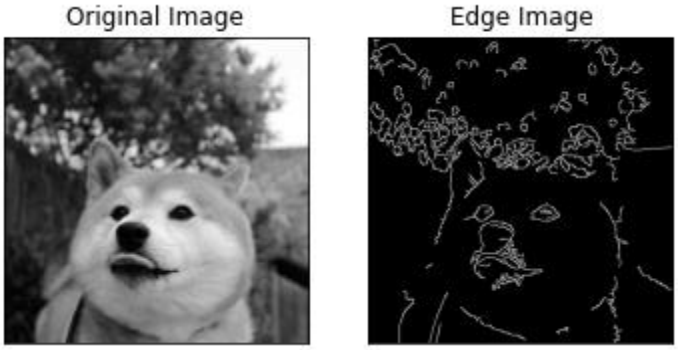


Рис. 4.8. Фільтр Кенні

РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ НЕЙРОМЕРЕЖ

Згорткова нейронна мережа (Convolutional Neural Network - CNN) – основний інструмент для класифікації та розпізнавання об'єктів, облич на фотографіях, розпізнавання мови. Є безліч варіантів застосування CNN, такі як Deep Convolutional Neural Network (DCNN), Region-CNN (R-CNN), Fully Convolutional Neural Networks (FCNN), Mask R-CNN та інші.

Для реалізації згорткової нейронної мережі ми використаємо пакет **TensorFlow** – бібліотеку програмного забезпечення з відкритим кодом для чисельних розрахунків з використанням графів потоку даних. Вузли графу представлені у вигляді математичних операцій, в той час як ребра графу представляються багатовимірними масивами даних (тензорами), що передаються між вузлами.

TensorFlow був створений і підтримується командою Google Brain в рамках дослідницької організації Google Machine Intelligence для ML та DL. Зараз він випускається під ліцензією відкритого коду Apache 2.0. Інтерфейси програмування TensorFlow включають Python та C ++ з планами для API Java, GO, R та Haskell, також підтримуються хмарні середовища Google та Amazon.

На відміну від інших бібліотек DL, які в основному орієнтовані на дослідження (наприклад, Theano), TensorFlow був розроблений для використання як у системах досліджень, так і у розробці та виробництві ПЗ. TensorFlow підтримується процесорами, графічними процесорами, мобільними пристроями і широкомасштабними розподіленими системами, які складаються з сотень вузлів. Крім того, існує TensorFlow Lite – легке рішення TensorFlow для мобільних та вбудованих

пристроїв. Це дає змогу випускати ML-орієнтовані рішення на пристрої користувачів дуже швидко та з невеликим двійковим розміром, але має підтримку обмеженого набору систем. Також підтримується апаратне прискорення за допомогою Android Neural Networks API.

5.1. Архітектура CNN

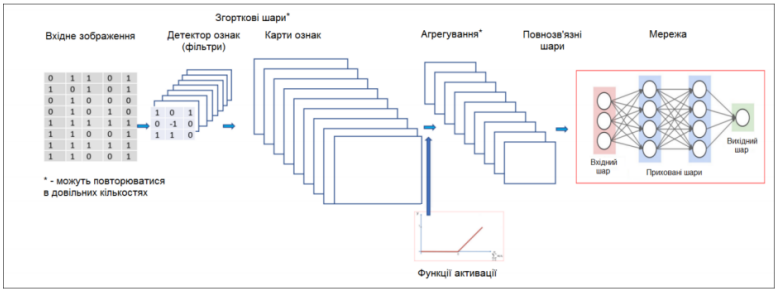


Рис. 5.1. Архітектура згорткової нейромережі

В основі згорткових шарів нейроннох мережі лежить операція згортки.

Згортка - це процес додавання кожного елемента зображення до його сусідів, зважених ядром. Важливо зауважити, що виконувана матрична операція - згортка - це не звичайне множення, хоча й позначається *.

Наприклад, якщо ми маємо дві 3x3 матриці, перша - ядро, друга - шматок зображення, згортка - це процес транспонування рядків і стовпчиків ядра з наступним множенням і додаванням. Елемент з координатами [2, 2] (тобто, центральний елемент) отриманого зображення буде зваженою комбінацією всіх елементів матриці зображення, з вагами взятими з ядра:

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Значення кожного пікселя у вихідному зображенні дорівнює сумі добутків значень матриці згортки і відповідних пікселів вхідного зображення.

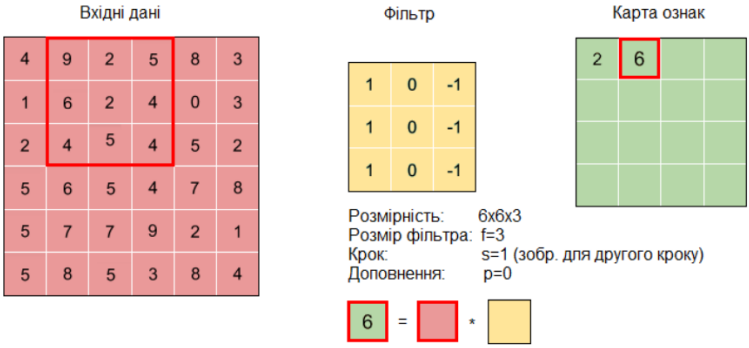


Рис. 5.2. Згортковий шар

Агрегувальні шари або шари субдискретизації (англ. pooling layers) є одним з основних структурних елементів згорткових нейронних мереж, як і згорткові шари. Такі шари можуть бути як глобальними, так і локальними, тобто розповсюджуватись тільки на окремі групи вхідних даних, які будуть рецептивними полями для нейронів поточного шару.

Головною задачею агрегувальних шарів є зменшення розмірності даних з одночасним збереженням найважливіших характеристик шляхом формування залежності між кількома елементами (нейронами) з попереднього шару з єдиним елементом даного шару. Тому при побудові мережі вони зазвичай використовуються з певною періодичністю між згортковими шарами.

Слід зазначити, що агрегувальні шари зберігають глибину вхідних даних, при цьому значно запобігаючи перенавчанню.

Агрегувальні шари бувають двох підтипів: усереднювальні (англ. average) та максимізаційні (англ. maximal). Інколи використовуються також мінімізаційні шари (англ. minimal) або агрегувальні шари за L2-нормою.

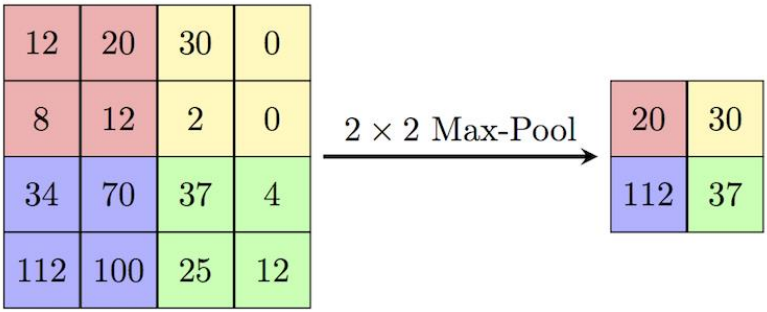


Рис. 5.3. Агрегувальний шар

Повнозв'язними рівнями (англ. fully-connecccted layers) в згорткових нейронних мережах називають такі рівні, де всі нейрони з наступного шару поєднані зв'язками з нейронами попереднього шару, як і у більшості шарів у звичайних нейронних мережах. Використання таких рівнів на початкових та прихованих рівнях мережі невиправдане, адже воно ускладнює модель і навіть може ігнорувати знайдені раніше за допомогою згорток та агрегації ознаки та закономірності. Проте, повнозв'язні рівні зазвичай використовуються на передостанньому кроці роботи мережі для підготовки знаходження результатів на виході мережі.

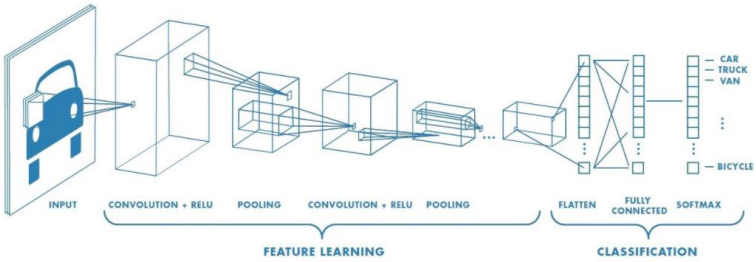


Рис. 5.4. Архітектура CNN для мультикласової класифікації

5.2. Бінарна класифікація зображень за допомогою Tensorflow

Процес побудови моделі можна розділити на такі кроки:

1. Дослідження даних (візуалізація);
2. Попередня обробка даних (підготовка до нейромережі);
3. Створення моделі (починаємо з найпростішої);
4. Тренування моделі.
5. Оцінка моделі.
6. Налаштування параметрів, удосконалення моделі;
7. Повторюємо, поки не будемо задоволені якістю.

Пройдемося по кожному із кроків.

Для класифікації розглянемо задачу розпізнавання стейків та піци. Візуалізуємо дані.

```
plt.figure()
plt.subplot(1, 2, 1)
steak_img = view_random_image("pizza_steak/train/", "steak")
plt.subplot(1, 2, 2)
pizza_img = view_random_image("pizza_steak/train/", "pizza")
```

```
Image shape: (512, 512, 3)
Image shape: (384, 512, 3)
```

Рис. 5.5. Візуалізація даних

На цьому етапі ми розбиваємо дані на тренувальну і тестову підможини. Для розпізнавання зображень прийнято зберігати ці дві підмножини у різних теках.

На наступному кроці нам слід розбити дані на маленькі групки (batch). Замість того, щоб дивитися відразу на усі тренувальні дані, модель буде навчатися, використовуючи дані "порційно".

Навіщо це? Є кілька причин:

- Велика кількість зображень може не вміститися в пам'яті процесора;
- Модель може неефективно навчатися.

Загальноприйнятий розмір батчу - 32.

Для розбиття на батчі створимо ImageDataGenerator для кодного із датасетів.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)
```

```
# Завантажимо зображення
train_data = train_datagen.flow_from_directory(directory=train_dir,
                                               target_size=(224, 224),
                                               class_mode='binary',
                                               batch_size=32)

test_data = test_datagen.flow_from_directory(directory=test_dir,
                                             target_size=(224, 224),
                                             class_mode='binary',
                                             batch_size=32)
```

```
Found 1500 images belonging to 2 classes.
Found 500 images belonging to 2 classes.
```

Рис. 5.6. Попередня обробка зображень

На наступному кроці можемо створити першу модель. Починати слід із простих архітектур.

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Activation
from tensorflow.keras import Sequential
```

```
model_3 = Sequential([
  Conv2D(filters=10,
         kernel_size=3,
         strides=1,
         padding='valid',
         activation='relu',
         input_shape=(224, 224, 3)), # input layer (specify input shape)
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  Flatten(),
  Dense(1, activation='sigmoid') # output layer (specify output shape)
])
```

```
model_3.compile(loss='binary_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy'])
```

Рис. 5.7. Створення моделі

Ми створили модель із трьох згорткових шарів.

На наступному етапі слід натренувати модель. Використаємо 5 епох.

```
history_3 = model_3.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

```
Epoch 1/5
47/47 [==============================] - 11s 214ms/step - loss: 1.8038 - accuracy: 0.6326 - val_loss: 0.4536 - val_accuracy: 0.7900
Epoch 2/5
47/47 [==============================] - 10s 209ms/step - loss: 0.4707 - accuracy: 0.8002 - val_loss: 0.4408 - val_accuracy: 0.8020
Epoch 3/5
47/47 [==============================] - 10s 209ms/step - loss: 0.3964 - accuracy: 0.8280 - val_loss: 0.3859 - val_accuracy: 0.8240
Epoch 4/5
47/47 [==============================] - 10s 209ms/step - loss: 0.2355 - accuracy: 0.9183 - val_loss: 0.3885 - val_accuracy: 0.8180
Epoch 5/5
47/47 [==============================] - 10s 207ms/step - loss: 0.0953 - accuracy: 0.9794 - val_loss: 0.5503 - val_accuracy: 0.7960
```

Рис. 5.8. Тренування моделі

Коли модель натреновано, можемо оцінити криві навчання моделі.

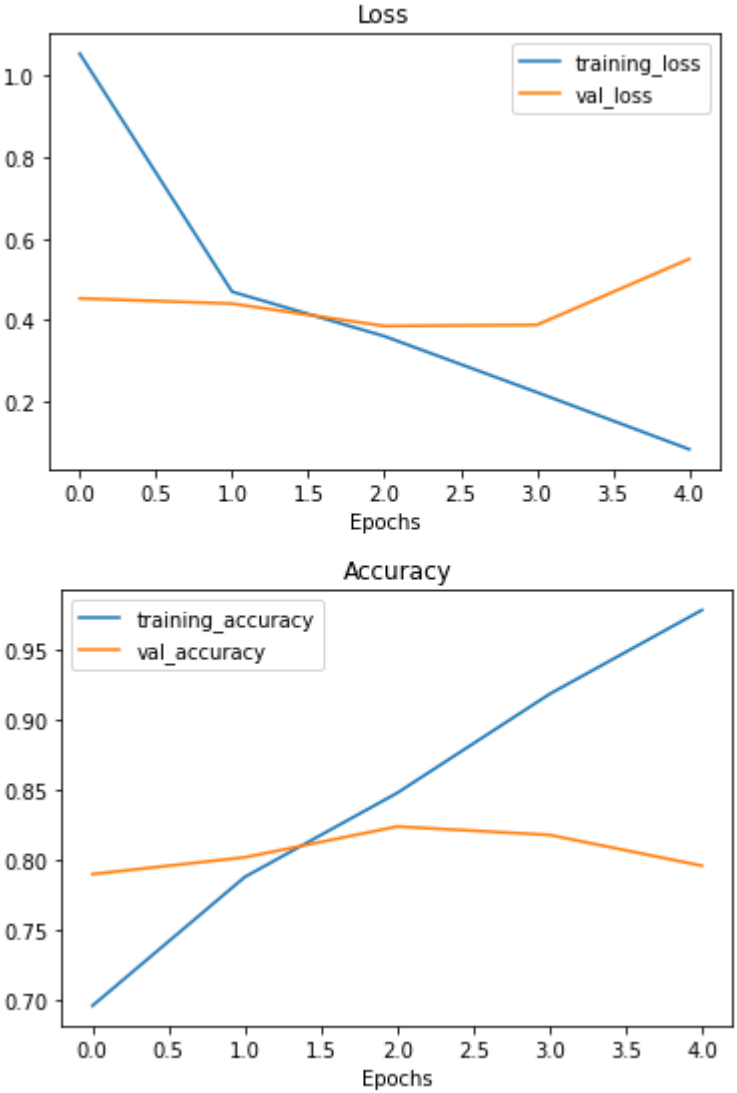


Рис. 5.9. Криві втрат

З оцінки кривих видно, що наша модель перенавчається на тренувальних даних, тож точність прогнозу на тестових даних падає.

Слід використати регуляризацію, для покращення точності моделі на тестових даних.

Для зменшення перетренованості моделі можна використати такі техніки:

- використання агрегувальних шарів;
- аугментація даних.

Стандартна структура CNN з агрегувальними шарами:

Вхідний шар -> Згорктовий шар + ReLU (для нелінійності) + Max Pooling агрегування -> Повнозв'язний шар (dense layer) як Output

Додамо кілька агрегувальних шарів.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 222, 222, 10)      280
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 10)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 109, 109, 10)      910
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 10)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 52, 52, 10)        910
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 26, 26, 10)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6760)              0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 6761
=================================================================
Total params: 8,861
Trainable params: 8,861
Non-trainable params: 0
_________________________________________________________________
```

Рис. 5.10. Архітектура моделі з агрегувальними шарами

Нова модель вже не перенавчається, і дає більшу точність при більшій кількості епох.

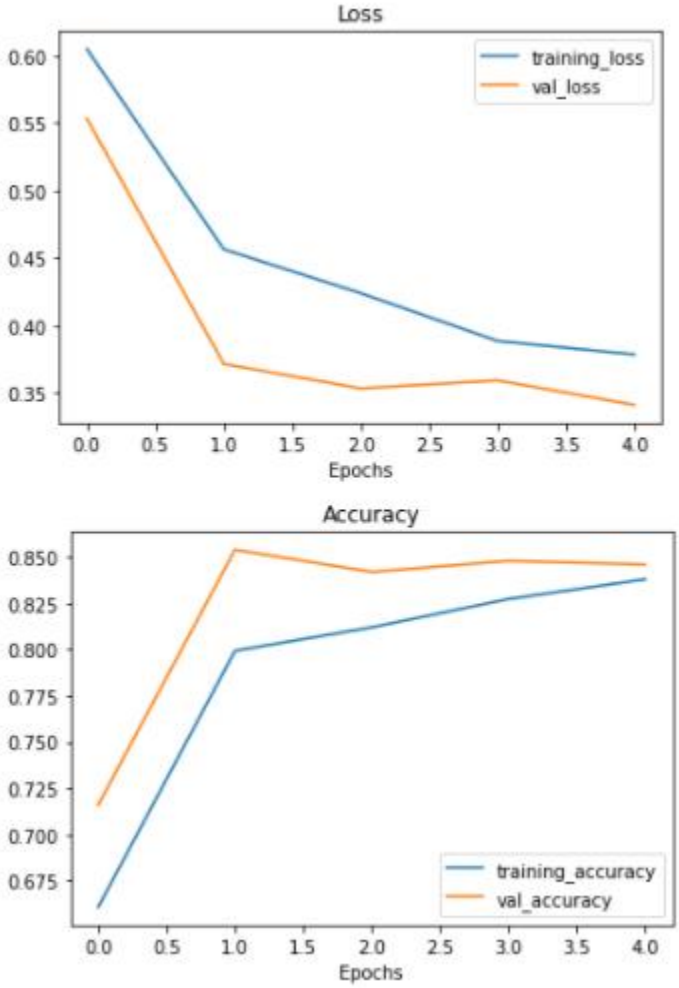


Рис. 5.11. Криві втрат нової моделі

Розглянемо іншу техніку регуляризації – аугментацію (спотворення) даних. Аугментація даних (data augmentation) –

методика створення нових тренувальних даних із наявних. Для досягнення хороших результатів нейромережі мають навчатися на великих об'ємах даних. Якщо даних мало - слід використати аугментацію.

Ця техніка полягає у зміщенні, повороті, розтягненні та інших спотвореннях зображення, з метою навчити модель розпізнавати навіть такі дані. Аугментацію можна задати в генераторі тренувальних даних.

```
train_datagen_augmented = ImageDataGenerator(rescale=1/255.,
                                              rotation_range=0.2, # поворот
                                              shear_range=0.2, # зсув
                                              zoom_range=0.2, # наближення
                                              width_shift_range=0.2, # здвиг по ширині
                                              height_shift_range=0.2, # здвиг по висоті
                                              horizontal_flip=True) # віддзеркалення по горизонталі
```

Рис. 5.12. Аугментація даних

Проілюструємо дані перетворення на рис. 5.13.

Рис. 5.13. Аугментовані зображення

Та сама модель на аугментованих даних дає наступні криві втрат:

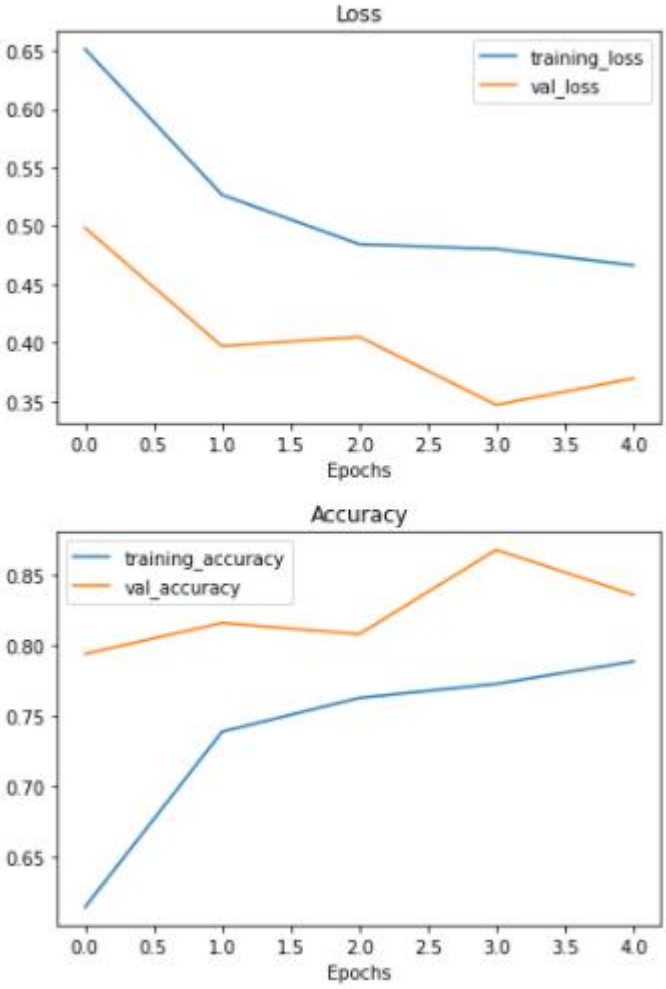


Рис. 5.14. Криві втрат для аугментованих даних

В даному випадку, модель можна припинити навчати на третій епосі.

Можливі шляхи для вдосконалення нашої моделі:

- збільшення числа шарів;
- збільшення числа фільтрів у згорткових шарах (з 10 до 32,64,128);
- навчати модель довше;
- знайти оптимальний learning rate;
- отримати більше данних;
- використовувати transfer learning (вже натреновані моделі).

Використаємо для даної задачі нейромережу з архітектурою TinyVGG і аугментованими даними.

```
model_7 = Sequential([
  Conv2D(10, 3, activation='relu', input_shape=(224, 224, 3)), # same input shape as our images
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Conv2D(10, 3, activation='relu'),
  Conv2D(10, 3, activation='relu'),
  MaxPool2D(),
  Flatten(),
  Dense(1, activation='sigmoid')
])

# Compile the model
model_7.compile(loss="binary_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

# Fit the model
history_7 = model_7.fit(train_data_augmented_shuffled,
                        epochs=5,
                        steps_per_epoch=len(train_data_augmented_shuffled),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

```
Epoch 1/5
47/47 [==============================] - 26s 538ms/step - loss: 0.6908 - accuracy: 0.5241 - val_loss: 0.4422 - val_accuracy: 0.8260
Epoch 2/5
47/47 [==============================] - 25s 528ms/step - loss: 0.5169 - accuracy: 0.7636 - val_loss: 0.4795 - val_accuracy: 0.8300
Epoch 3/5
47/47 [==============================] - 25s 530ms/step - loss: 0.5373 - accuracy: 0.7525 - val_loss: 0.4002 - val_accuracy: 0.8540
Epoch 4/5
47/47 [==============================] - 25s 527ms/step - loss: 0.5155 - accuracy: 0.7612 - val_loss: 0.4277 - val_accuracy: 0.8120
Epoch 5/5
47/47 [==============================] - 25s 528ms/step - loss: 0.4892 - accuracy: 0.7819 - val_loss: 0.3772 - val_accuracy: 0.8460
```

Рис. 5.15. Навчання нової моделі

Отримаємо наступні криві втрат:

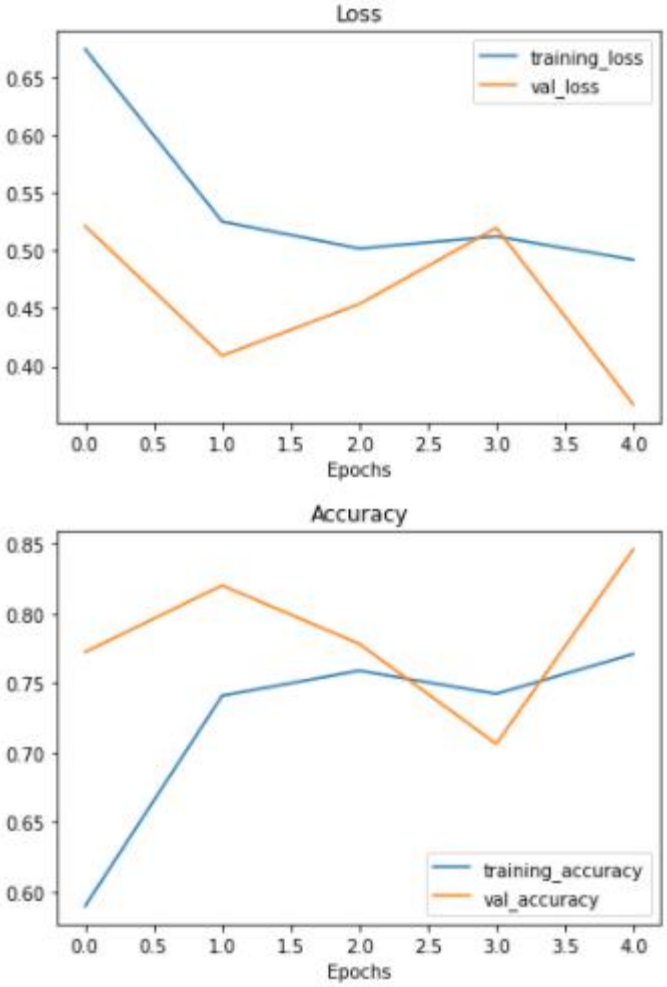


Рис. 5.16. Криві втрат для TinyVGG

Список рекомендованих джерел

1. Шапиро Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман. – М. : Бином. Лаборатория знаний, 2006. – 716 с.
2. Форсайт Д. Компьютерное зрение. Современный подход / Д. Форсайт, Ж. Понс. – М. : Вильямс, 2004. – 928 с.
3. Савиных В. П. Аэрокосмическая фотосьемка / В. П. Савиных, А. С. Кучко, А. Ф. Стеценко. – М. : КартоГеоЦентр Геоиздат, 1997. – 378 с.
4. Янтуш Д. А. Дешифрирование азрокосмических снимков / . А. Янтуш. – М. : Недра, 1991. – 240 с.
5. Лисицин В. З. Практикум по фотограмметрии и дистанционному зондированию / В. З. Лисицин. – Харьков : ХНАГХ, 2006. – 200 с.
6. Кашкин В. Б. Дистанционное зондирование Земли из космоса. Цифровая обработка изображений / В. Б. Кашкин, А. И. Сухинин. – М. : Логос, 2001. − 264 с.
7. Цифровая обработка изображений в информационных системах /И. С. Грузман, В. С. Киричук и др. – Новосибирск : НГТУ, 2002. − 352 с.

8. Ватолин Д. Методы сжатия данных / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М. : Диалог-Мифи, 2002. – 384 с.
9. Прэтт У. Цифровая обработка изображений / У. Прэтт. – М. : Мир, 1982. – 480 с
10. Solem, Jan Erick. Programming computer vision with Python. Beijing; Cambridge; Sebastopol [etc.]: O'Reilly, 2012.
11. Geron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems (2nd ed.). O'Reilly.