МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Програмний засіб перенесення нейронного стилю для зміни представлення зображень

Виконавець: Вовк В.А.

Керівник: Артамонов Є.Б.

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Освітнього ступеня магістр
Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

Спеціалізація 123.02 "Системне програмування"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.

«____»_____2020 р.

ЗАВДАННЯ
на виконання дипломної роботи (проекту)

Вовка Володимира Андрійовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема роботи: "Програмний засіб перенесення нейронного стилю для зміни представлення зображень "
затверджена наказом ректора від " 27 " серпня 2020 року № 1203 /ст.
2. Термін виконання роботи: з 05.10.2020 до 31.12.2020
3. Вихідні дані до роботи: програмне забезпечення стилізації зображень
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) дослідження принципів передачі нейронного стилю;
2) методи реалізації нейронних мереж
3) розробка програмного забезпечення для перенесення нейронного стилю

5. Перелік обов'язкового графічного матеріалу:
1) приклад згортки малюнку;
2) етапи формування нейронного стилю;
3) навчання карти об'єктів на різних шарах;
4) приклади перенесення нейронного стилю;
5) схема алгоритму перенесення нейронного стилю

6. Календарний план

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1			
2			
3			
4			
5			
6			

7. Дата видачі завдання 05.10.2020

Керівник Артамонов Є.Б.
(підпис)

Завдання прийняв до виконання Вовк В.А.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Програмний засіб перенесення нейронного стилю для зміни представлення зображень”: 91 с., 12 рис., 29 літературних джерел, 1 додаток.

ПЕРЕДАЧА СТИЛЮ, РЕДАГУВАННЯ МАЛЮНКІВ, НЕЙРОННА МЕРЕЖА, АНАЛІЗ РИСУНКІВ, ВИБІРКА, МАШИННЕ НАВЧАННЯ.

Мета дослідження – розробити програмний засіб для перенесення стилю зображення з одного малюнку на інший.

Об’єкт дослідження – стилізація зображень.

Предмет дослідження – програмний засіб перенесення нейронного стилю для зміни представлення зображень.

Практична значимість полягає у розробці програмного забезпечення, що планується використовувати у навчальному процесі, як практична складова у вивченні застосування нейронних мереж в графічних системах.

Прогнозні припущення щодо подальшого розвитку матеріалів роботи полягає у встановленні та реалізації ПЗ автоматичної генерації випадкових стилів зображень. Результати дипломної роботи було представлено на науково-практичній конференції та опубліковані у збірнику тез доповідей: Вовк В.А., Цирень М.В. Можливості використання нейронних мереж для представлення зображень та визначення стійкості паролів до зламу// Тези доповідей наук.-практ. конф. “Сучасні тенденції розвитку системного програмування” (25-26 листопада 2020 р.). – К.: НАУ, 2020. – С. 17.

ЗМІСТ

ВСТУП

Передача стилю полягає у створенні зображення з тим самим "вмістом", що і базове зображення, але зі "стилем" іншого зображення (як правило, художнього). Це досягається за рахунок оптимізації функції втрат, яка складається з 3 компонентів: "втрата стилю", "втрата вмісту" та "загальна втрата змін":

- Повна втрата варіації накладає місцеву просторову неперервність між пікселями комбінованого зображення, надаючи йому візуальну цілісність.
- Втрата стилю полягає там, де глибоке навчання зберігається - це визначається за допомогою глибокої згорткової нейронної мережі. Точніше, він складається із суми L2 відстаней між матрицями Грама подань базового зображення та еталонного зображення стилю, витягнутих з різних шарів коннету (тренованих на ImageNet). Загальна ідея полягає у захопленні інформації про колір / текстуру в різних просторових масштабах (досить великі масштаби - визначаються глибиною розглянутого шару).
- Втрата вмісту - це відстань L2 між ознаками базового зображення (витягнутого з глибокого шару) та особливостями комбінованого зображення, утримуючи сформоване зображення досить близько до вихідного.

Мета дослідження – розробити програмний засіб для перенесення стилю зображення з одного малюнку на інший.

Об’єкт дослідження – стилізація зображень.

Предмет дослідження – програмний засіб перенесення нейронного стилю для зміни представлення зображень.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРИНЦИПІВ ПЕРЕДАЧІ НЕЙРОННОГО СТИЛЮ

1.1. Основи передачі нейронного стилю

У серпні 2015 року вийшов документ під назвою `` Нейронний алгоритм художнього стилю ". Через кілька місяців вийшов додаток Prisma, і люди просто збожеволіли від цього. Призма – це в основному додаток, який дозволяє застосовувати стилі живопису відомих живописців до власних фотографій, а результати цілком візуально приємні. Це не як фільтри Instagram, де якесь перетворення застосовується до зображення лише в колірному просторі. Це набагато складніше, а результати ще цікавіші. Ось цікаве фото, яке я знайшов в Інтернеті.

Рис. 1.1. Зразок малюнку

В основному, у Neural Style Transfer ми маємо два стилі зображень та вміст. Нам потрібно скопіювати стиль із зображення стилю та застосувати його до зображення вмісту. Під стилем ми в основному маємо на увазі візерунки, мазки тощо.

Для цього ми використовуємо попередньо навчену мережу VGG-16. Оригінальний документ використовує мережу VGG-19, але я не зміг знайти ваги VGG-19 для TensorFlow, тому я пішов із VGG-16. Насправді це (із використанням VGG-16 замість VGG-19) не має великого впливу на кінцеві результати. Отримані зображення ідентичні.

Так інтуїтивно, як це працює?

Щоб зрозуміти це, я дам вам основне вступ до того, як працює ConvNets.

ConvNets працює за основним принципом згортки. Скажімо, наприклад, у нас є зображення та фільтр. Ми насуваємо фільтр по зображенню і приймаємо як вихідну інформацію зважену суму входів, охоплених фільтром, перетворену нелінійністю, такою як сигмоїдна або ReLU або танх. Кожен фільтр має власний набір вагових коефіцієнтів, які не змінюються під час операції згортки. Це чудово зображено у нижченаведеному GIF-

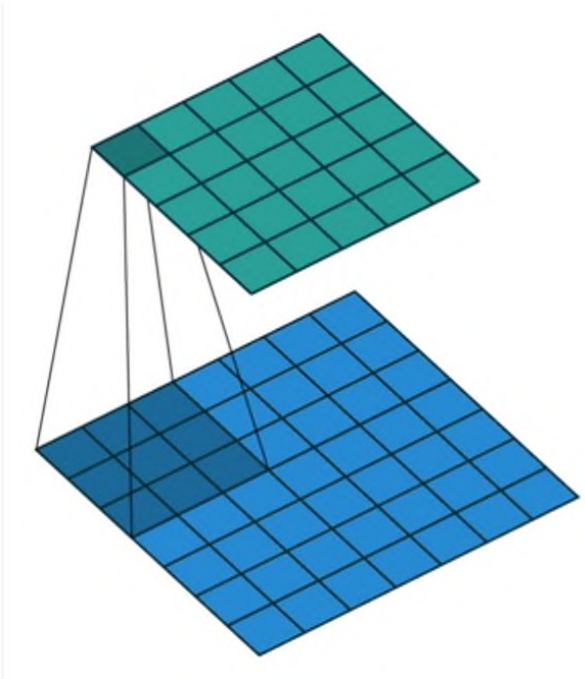


Рис. 1.2. Згортка малюнку

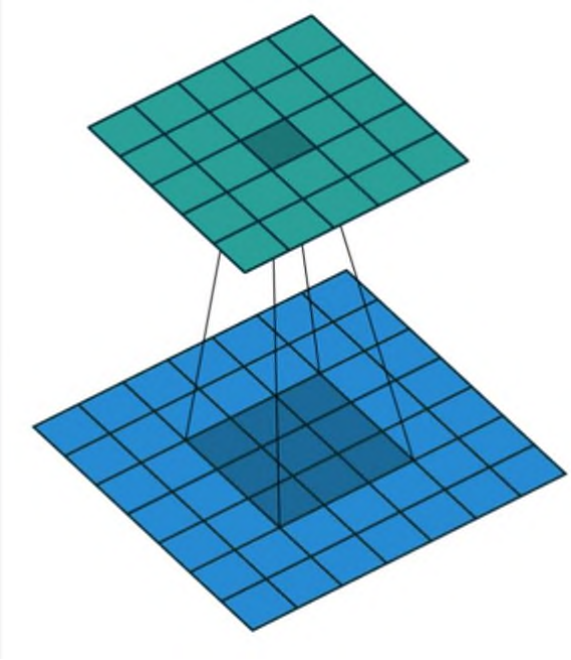


Рис. 1.3. Переміщення згортки малюнку

Тут синьою сіткою є вхідні дані. Ви можете побачити область 3x3, вкриту слайдом фільтра через вхід (темно-синя область). Результат цієї згортки називається картою об'єктів, яка зображена зеленою сіткою.

Ось графіки функцій активації ReLU і tanh (рис. 1.4).

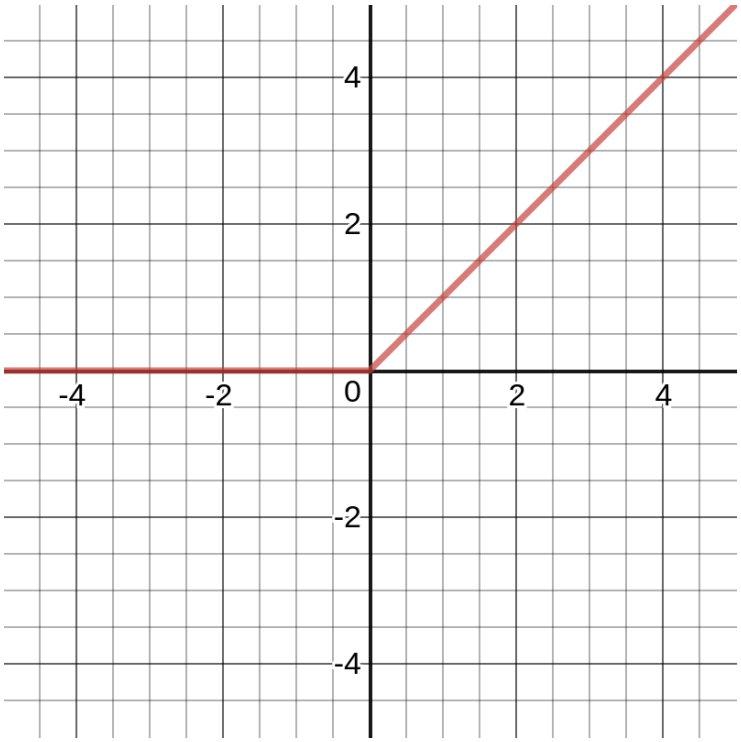


Рис. 1.4. Функція активації ReLU

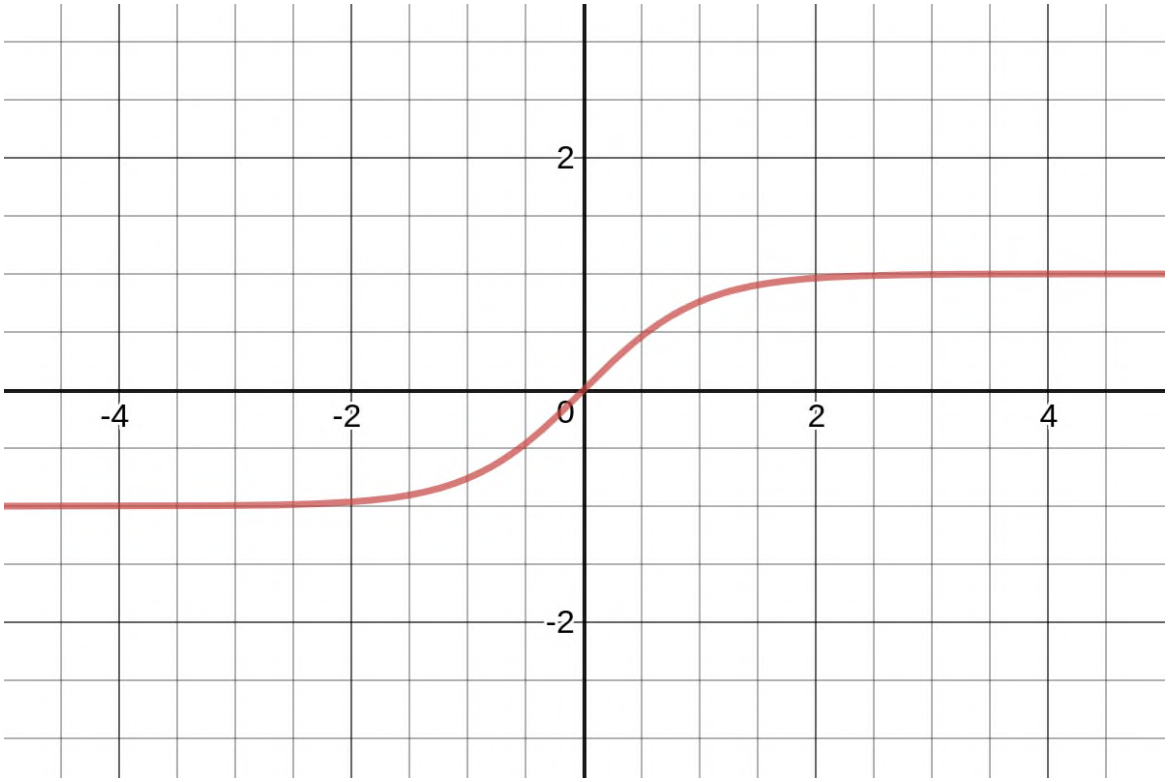


Рис. 1.5. Функція активації Tanh

Отже, у ConvNet вхідне зображення конвертується з декількома фільтрами та генеруються карти фільтрів. Потім ці карти фільтрів змішуються з деякими додатковими фільтрами і генеруються ще деякі карти функцій. Це чудово проілюстровано на схемі нижче.

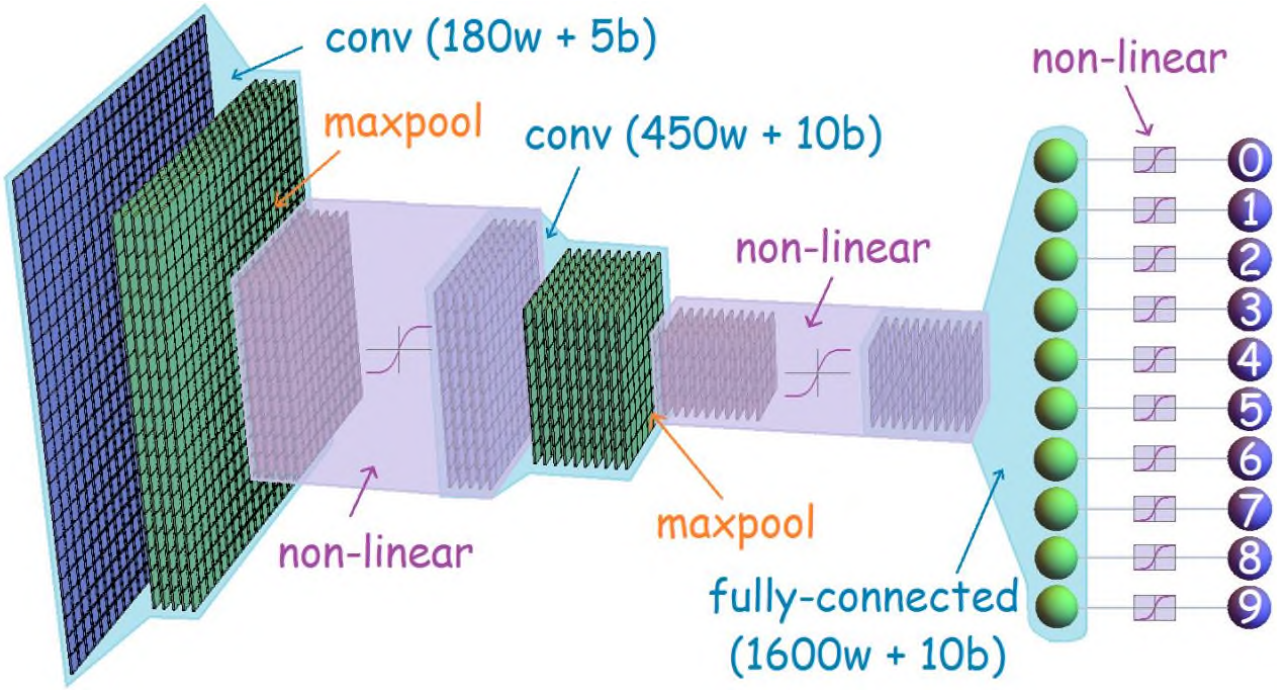


Рис. 1.6. Формування стилю

Можна побачити термін maxpool на наведеному вище зображенні. Макспулейний шар в основному використовується з метою зменшення розмірності. В операції maxpool ми просто пересуваємо вікно, скажімо, розміром 2x2, по зображенню і беремо як вихідний максимум значень, охоплених вікном. Ось приклад (рис. 1.7).

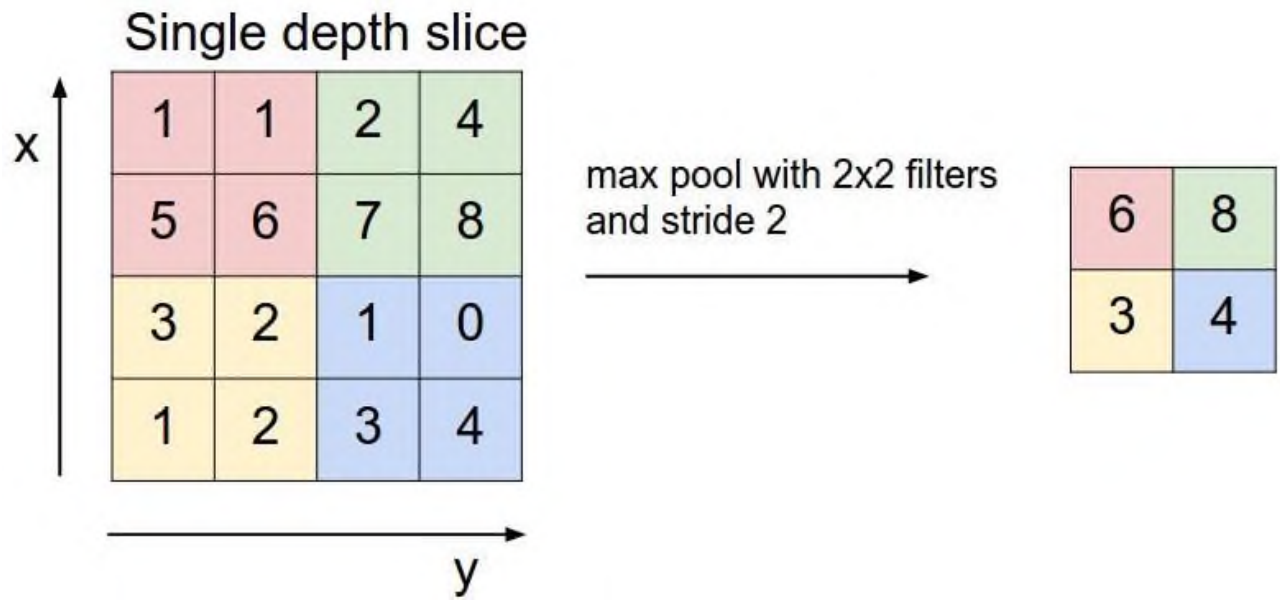


Рис. 1.8. вихідний максимум значень, охоплених вікном

На рисунку 1.9 проходить навчання карти об’єктів на різних шарах.

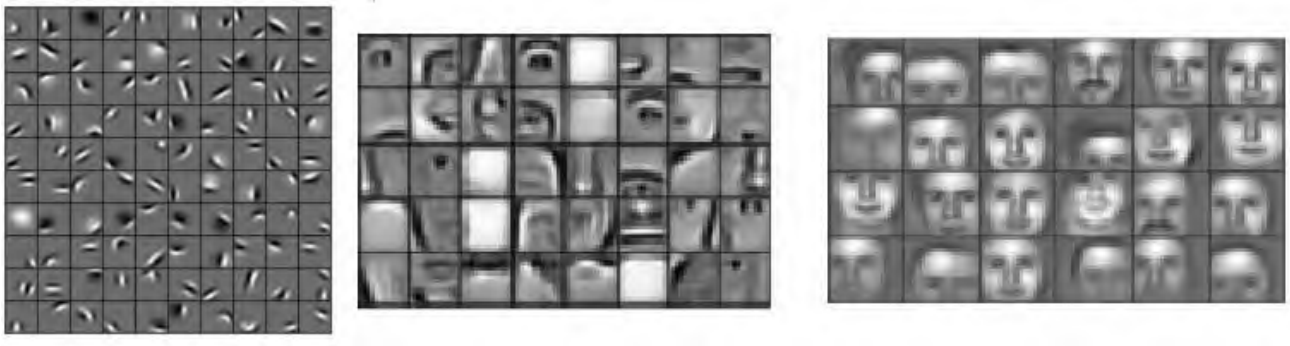


Рис. 1.9. Навчання карти об’єктів на різних шарах

Ви, напевно, помітили, що - карти в нижніх шарах шукають об’єкти низького рівня, такі як лінії або краплі (фільтри Габора). Коли ми переходимо до вищих шарів, зовнішні риси стають дедалі складнішими. Інтуїтивно ми можемо

думати про це таким чином - нижні шари фіксують елементи низького рівня, такі як лінії та крапки, шар вище, який базується на цих елементах низького рівня та обчислює дещо складніші особливості тощо ...

Таким чином, можна зробити висновок, що ConvNets розробляє ієрархічне представлення ознак.

Ця властивість є основою передачі стилю.

Тепер пам’ятайте - виконуючи передачу стилю, ми не тренуємо нейронну мережу. Швидше за все, ми робимо це - ми починаємо з порожнього зображення, що складається із випадкових значень пікселів, і оптимізуємо функцію витрат, змінюючи значення пікселів зображення. Простіше кажучи, ми починаємо з порожнього полотна та функції витрат. Потім ми ітеративно модифікуємо кожен піксель, щоб мінімізувати нашу функцію витрат. Іншими словами, під час тренування нейронних мереж ми оновлюємо свої ваги та упередження, але, передаючи стиль, ми підтримуємо ваги та упередження постійними, і замість цього оновлюємо наш образ.

Для цього важливо, щоб наша функція витрат правильно представляла проблему. Функція витрат має два терміни - термін втрати стилю та термін втрати вмісту, обидва з яких пояснюються нижче.

1.1.1. Втрата вмісту

Нейронний стиль базується на інтуїції, що зображення з подібним вмістом матимуть подібне представлення у вищих шарах мережі.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l} \right)^2$$

*P – л*являє собою подання вихідного зображення, а F - подання сформованого зображення на картах особливостей шару l.

1.1.2. Втрата стилю

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Тут A ^ l - зображення вихідного зображення, а G ^ l - зображення генерованого зображення в шарі l. Nl - кількість карт об'єктів, а Ml - розмір сплощеної карти об'єктів у шарі l. wl - вага, який надається втраті стилю шару l.

Під стилем ми в основному маємо на увазі мазки пензля та візерунки. Тому ми в основному використовуємо нижні шари, які фіксують особливості низького рівня. Також зверніть увагу на використання грам-матриці тут. Грамовою матрицею вектора X є X.X_transpose. Інтуїція використання грам-матриці полягає в тому, що ми намагаємося зафіксувати статистику нижніх шарів. Не обов’язково використовувати матрицю Грама. Деякі інші статистичні дані (наприклад, середнє значення) були випробувані і теж працювали досить добре.

1.1.3. Загальні втрати

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

де альфа та бета - ваги вмісту та стилю відповідно. Їх можна змінити, щоб змінити наш кінцевий результат.

Отже, наша функція загальної втрати в основному представляє нашу проблему - нам потрібно, щоб вміст кінцевого зображення був подібним до вмісту зображення вмісту, а стиль кінцевого зображення повинен бути подібним до стилю зображення стилю.

Тепер нам залишається лише мінімізувати ці втрати. Ми мінімізуємо ці втрати, змінюючи вхід до самої мережі. В основному ми починаємо з порожнього

сірого полотна і починаємо змінювати значення пікселів, щоб мінімізувати втрати. Для мінімізації цих втрат можна використовувати будь-який оптимізатор. Тут для простоти я використав простий градієнтний спуск. Але люди скористались Адамом та L-BFGS і отримали непогані результати щодо цього завдання.

1.2. Аналіз логіки передачі нейронного стилю

Передача нейронного стилю - це оптимізаційна техніка, яка використовується для зйомки трьох зображень, зображення вмісту, еталонного зображення стилю (наприклад, зображення відомого художника) та вхідного зображення, яке потрібно стилізувати, - і поєднати їх так, щоб вхідне зображення перетворюється так, щоб виглядати як вміст зображення, але «намальований» у стилі зображення стилю.

Наприклад, давайте візьмемо зображення цієї черепахи та Великої хвилі Кацусіки Хокусая біля Канагави:

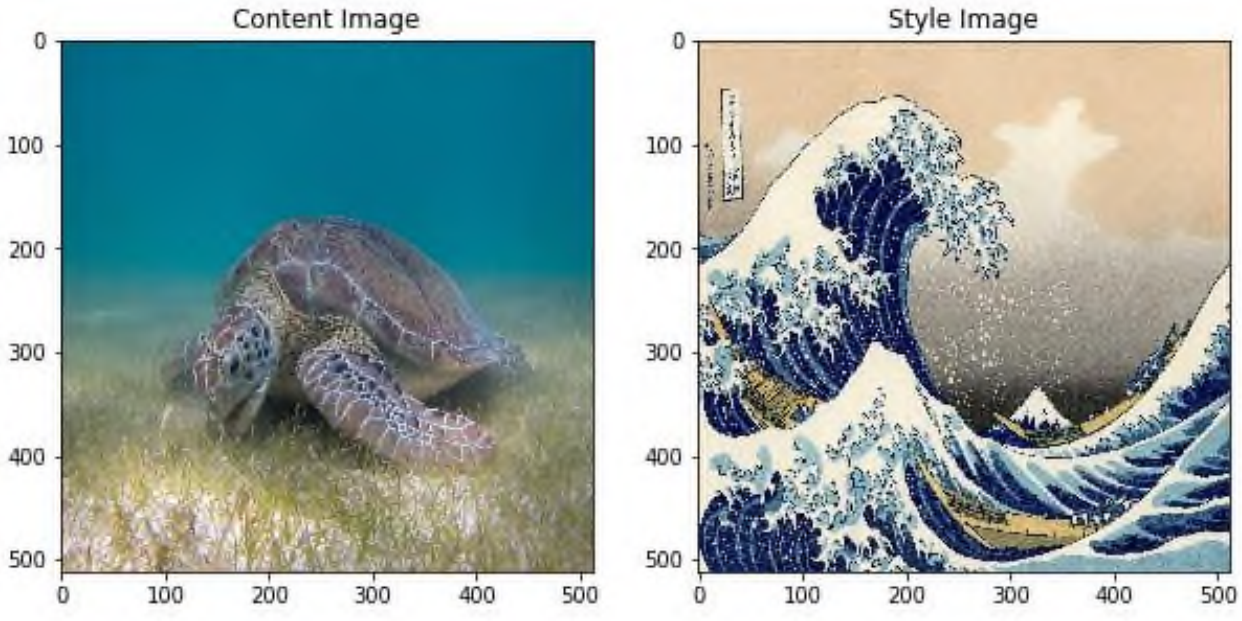


Рис. 1.10. Вихідні рисунки

А як би це виглядало, якби Хокусай вирішив додати фактуру або стиль своїх хвиль до образу черепахи? Щось на зразок цього?

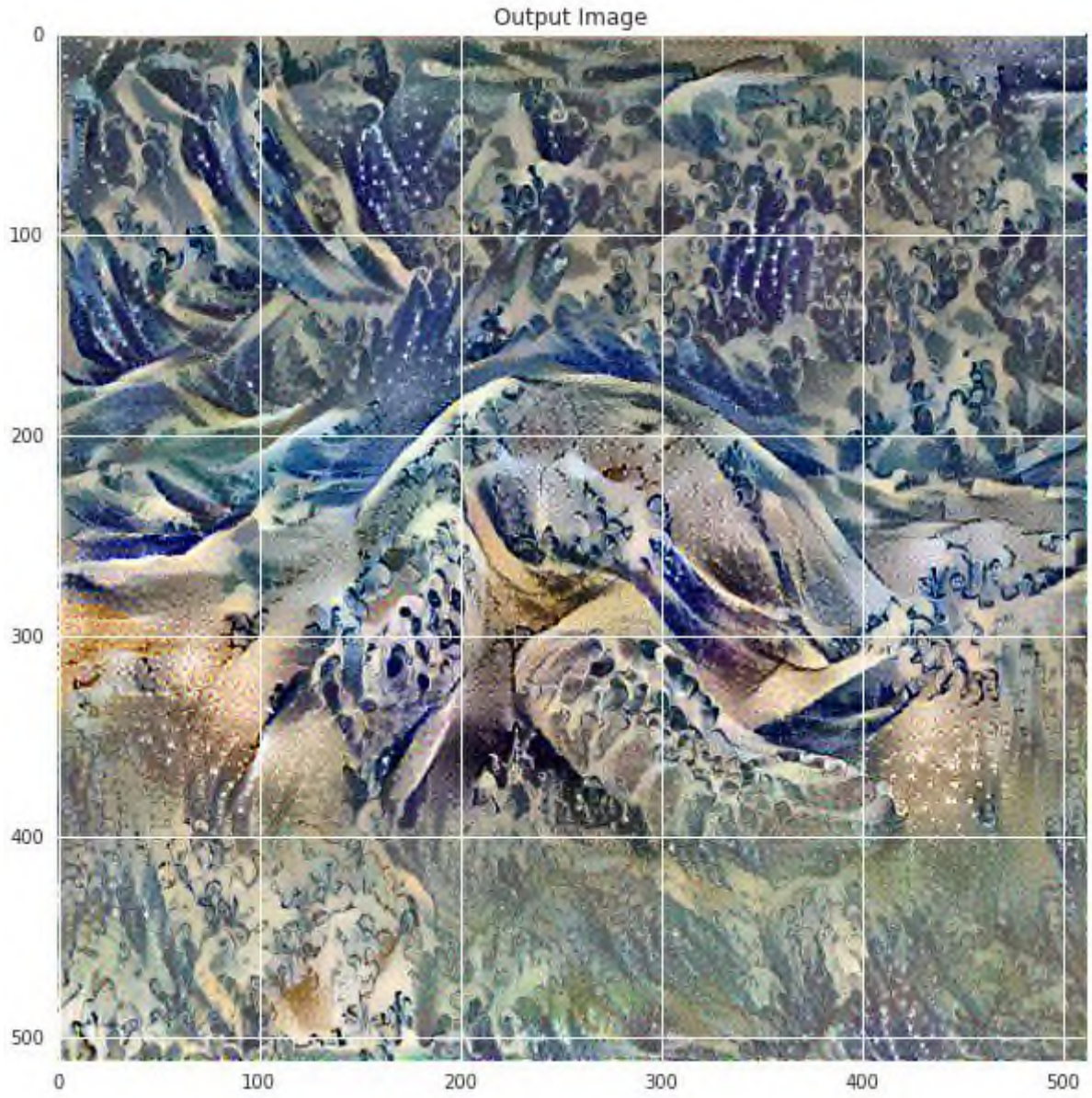


Рис. 1.11. Отримане зображення з переданим стилем

Передача стилю - це цікава техніка, яка демонструє можливості та внутрішні уявлення нейронних мереж.

Принцип передачі нейронного стилю полягає у визначенні двох функцій відстані, одна, яка описує, наскільки різним є вміст двох зображень, Lcontent, і одна, яка описує різницю між двома зображеннями з точки зору їх стилю, Lstyle. Потім, отримуючи три зображення, потрібне зображення стилю, потрібне зображення вмісту та вхідне зображення (ініціалізоване зображенням вмісту), ми

намагаємось трансформувати вхідне зображення, щоб мінімізувати відстань вмісту до зображення вмісту та відстань стилю до стиль зображення.

Підсумовуючи, ми візьмемо базове вхідне зображення, зображення вмісту, якому ми хочемо відповідати, і зображення стилю, якому ми хочемо відповідати. Ми перетворимо базове вхідне зображення, мінімізуючи відстань вмісту та стилю (втрати) із зворотним розповсюдженням, створивши зображення, яке відповідає вмісту зображення вмісту та стилю зображення стилю.

Використання карт функцій попередньо навченої моделі - Дізнайтеся, як користуватися попередньо навченими моделями та їх картами функцій

1.3. Впровадження передачі нейронного стилю

Ми почнемо з включенняпрагнення втрати. Стрімке виконання дозволяє нам пропрацювати цю техніку найбільш чітко і зрозуміло.

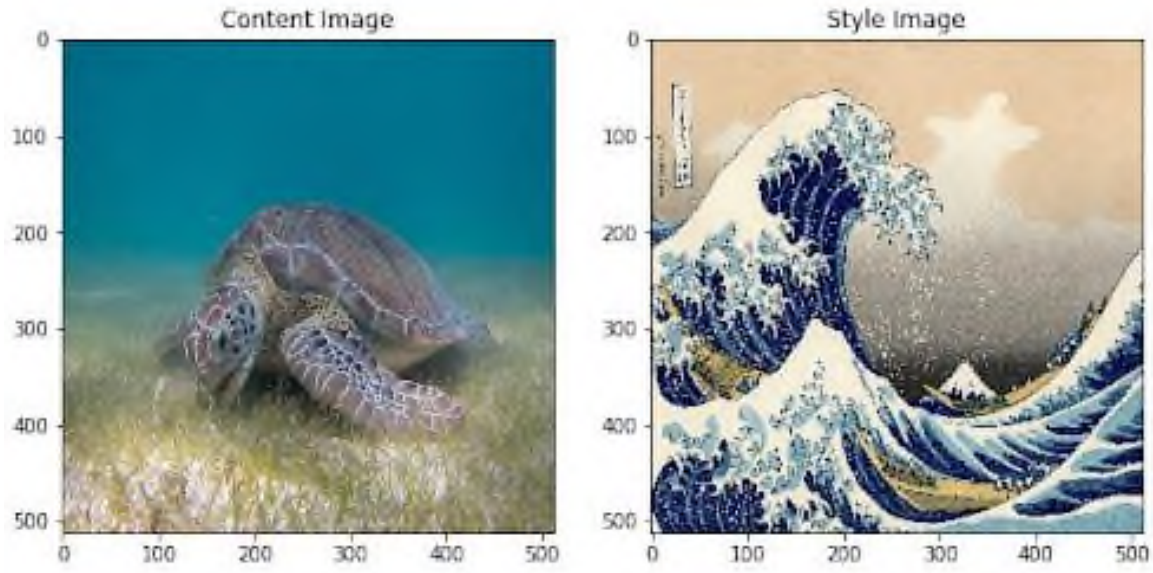


Рис. 1.12. Аналіз передачі нейронного стилю

Для того, щоб отримати як вміст, так і стилі зображення нашого зображення, ми розглянемо деякі проміжні шари в нашій моделі. Проміжні шари представляють карти функцій, які стають дедалі вищими впорядкованими, коли

ви заглиблюєтесь. У цьому випадку ми використовуємо архітектуру мережі VGG19, попередньо навчену мережу класифікації зображень. Ці проміжні шари необхідні для визначення відображення вмісту та стилю на наших зображеннях. Для вхідного зображення ми спробуємо узгодити відповідні подання стилю та цільового вмісту на цих проміжних шарах.

На високому рівні це явище можна пояснити тим, що для того, щоб мережа виконувала класифікацію зображень (до чого наша мережа навчена), вона повинна розуміти зображення. Це передбачає взяття вихідного зображення як вхідних пікселів та побудову внутрішнього уявлення за допомогою перетворень, які перетворюють вихідні пікселі зображення у складне розуміння особливостей, присутніх у зображенні. Це також частково, чому згорткові нейронні мережі можуть добре узагальнювати: вони здатні фіксувати незмінність та визначальні особливості в класах (наприклад, коти проти собак), які є агностичними для фонового шуму та інших неприємностей. Таким чином, десь між тим, де подається вихідне зображення та виводиться мітка класифікації, модель виконує функцію витяжки складних ознак; отже, отримуючи доступ до проміжних шарів, ми можемо описати вміст та стиль вхідних зображень.

1.4. Висновки до розділу

У цьому випадку ми завантажуємоVGG19, і подаємо наш вхідний тензор до моделі. Це дозволить нам витягнути карти функцій (а згодом і вміст та подання стилів) вмісту, стилю та створених зображень.

Ми використовуємо VGG19, як запропоновано в роботі. Крім того, оскільки VGG19 є відносно простою моделлю (порівняно з ResNet, Inception тощо), карти функцій насправді працюють краще для передачі стилів.

Для того, щоб отримати доступ до проміжних шарів, що відповідають нашим картам стилів та вмісту, ми отримуємо відповідні результати за допомогою KerasФункціональний APIщоб визначити нашу модель з бажаними вихідними активаціями.

За допомогою функціонального API визначення моделі просто передбачає визначення вхідних та вихідних даних:модель = модель (входи, виходи).

У наведеному вище фрагменті коду ми завантажимо нашу попередньо навчену мережу класифікації зображень. Потім ми беремо шари, що цікавлять, як ми визначили раніше. Потім ми визначаємо Модель, встановлюючи вхідні дані моделі для зображення, а виходи - вихідні дані стилю та вмісту. Іншими словами, ми створили модель, яка візьме вхідне зображення та виведе проміжні шари вмісту та стилю!

РОЗДІЛ 2
МЕТОДИ РЕАЛІЗАЦІЇ НЕЙРОННИХ МЕРЕЖ

2.1. Методи поглиблнггого навчання нейронних мереж

Поглиблене навчання сьогодні є гарячою темою. Але що саме робить його особливим і виділяє з-поміж інших аспектів машинного навчання? Це глибоке питання (вибачте за каламбур). Щоб навіть почати відповідати на нього, нам потрібно буде вивчити основи нейронних мереж.

Нейронні мережі - це робочі коні глибокого навчання. І хоча вони можуть виглядати як чорні ящики, в глибині душі (вибачте, я зупиню жахливі каламбури) вони намагаються виконати те саме, що будь-яка інша модель - робити хороші прогнози.

У цій публікації ми розглянемо тонкощі простої нейронної мережі. І до кінця, сподіваємось, ви (і я) отримали глибше та інтуїтивніше розуміння того, як нейронні мережі роблять те, що роблять.

Вид на 30000 футів

Почнемо з огляду дійсно високого рівня, щоб ми знали, з чим працюємо. Нейронні мережі - це багатошарові мережі нейронів (сині та пурпурові вузли на діаграмі нижче), які ми використовуємо для класифікації речей, прогнозування тощо. Нижче наведена схема простої нейронної мережі з п’ятьма входами, 5 виходами та двома приховані шари нейронів.

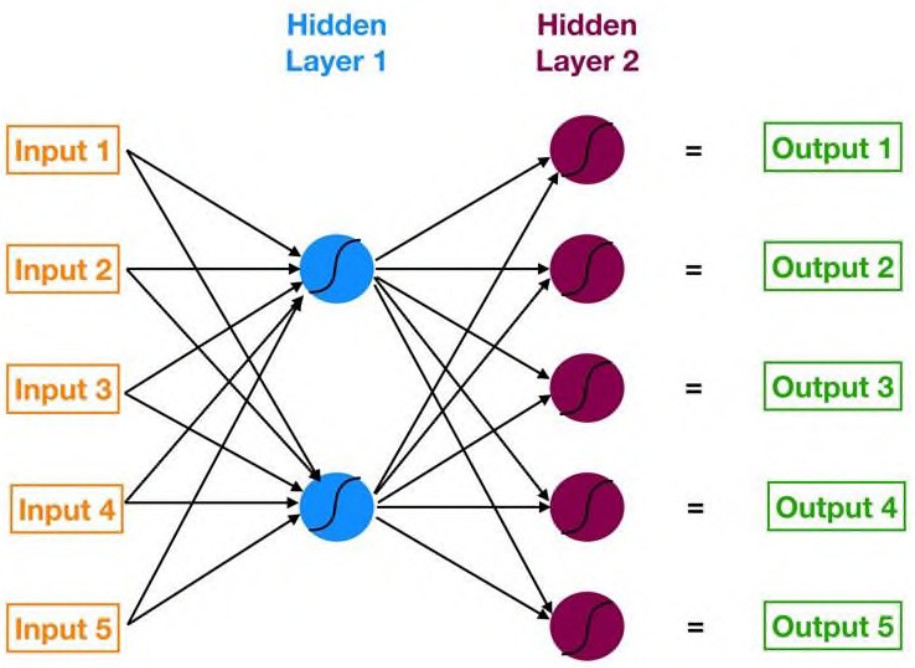


Рис. 2.1. Нейромережа з двома прихованими шарами

Починаючи зліва, маємо:

– Вхідний шар нашої моделі оранжевим кольором.

– перший прихований шар нейронів блакитного кольору.

– другий прихований шар нейронів у пурпуровому кольорі.

– Вихідний рівень (він же прогноз) нашої моделі зеленим кольором.

Стрілки, що з'єднують точки, показують, як всі нейрони взаємопов'язані і як дані переміщаються від вхідного шару аж до вихідного шару.

Пізніше ми будемо покроково обчислювати кожне вихідне значення. Ми також будемо спостерігати, як нейронна мережа вчиться на своїй помилці, використовуючи процес, відомий як зворотне розповсюдження.

Що саме намагається зробити нейронна мережа? Як і будь-яка інша модель, вона намагається зробити хороший прогноз. У нас є набір вхідних даних і набір цільових значень - і ми намагаємось отримати прогнози, які максимально точно відповідають цим цільовим значенням.

Забудьте на секунду про більш складну на вигляд нейронну мережу, яку я намалював вище, і зупиніться на цій простішій нижче.

Логістична регресія (лише з однією функцією), реалізована за допомогою нейронної мережі

Це одна особливість логістичної регресії (ми надаємо моделі лише одну змінну X), виражена через нейронну мережу (якщо вам потрібне оновлення щодо логістичної регресії,Я писав про це тут). Щоб побачити, як вони з'єднуються, ми можемо переписати рівняння логістичної регресії, використовуючи наші кольорові коди нейронної мережі.

2.1.1. Рівняння логістичної регресії

Давайте розглянемо кожен елемент:

$$Sigmoid(B1 * X + Bo) = Predicted\ Probability$$

X (оранжевим) - це наш вхід, одинока особливість, яку ми надаємо нашій моделі для обчислення прогнозу.

B1 (у бірюзовому, він же синьо-зелений) - це розрахунковий параметр нахилу нашої логістичної регресії - B1 повідомляє нам, наскільки змінюються Log_Odds при зміні X. Зверніть увагу, що B1 живе на бірюзовій лінії, яка з'єднує вхід X із синім нейроном у прихованому шарі 1.

B0 (синім кольором) - це упередження - дуже схоже на термін перехоплення від регресії. Ключова відмінність полягає в тому, що в нейронних мережах кожен нейрон має власний термін зміщення (тоді як при регресії модель має особливий термін перехоплення).

Синій нейрон також включає афункція сигмоїдної активації(позначається кривою лінією всередині синього кола). Пам'ятайте, що саме ми використовуємо сигмоподібну функціюперейти від log-odds до ймовірності (виконайте control-f пошук "sigmoid" у моєму попередньому дописі).

І нарешті, ми отримуємо нашу передбачувану ймовірність, застосовуючи сигмоїдну функцію до величини (B1 * X + B0).

Не надто погано, правда? Тож підсумуємо. Надзвичайно проста нейронна мережа складається лише з наступних компонентів:

Зв'язок (хоча на практиці, як правило, буде кілька з'єднань, кожен зі своєю вагою, що переходить у певний нейрон), з вагою, "що живе всередині нього", яка трансформує ваш вхід (за допомогою B1) і передає його нейрону .

Нейрон, що включає термін упередження (B0) та функцію активації (у нашому випадку сигмоїдна).

І ці два об’єкти є фундаментальними будівельними блоками нейронної мережі. Складніші нейронні мережі - це лише моделі з більш прихованими шарами, а це означає більше нейронів і більше зв’язків між нейронами. І ця більш складна павутина зв’язків (і ваг та упереджень) - це те, що дозволяє нейронній мережі «вивчати» складні взаємозв’язки, приховані в наших даних.

Давайте додамо трохи складності зараз

Тепер, коли у нас є основний фреймворк, повернімось до нашої дещо складнішої нейронної мережі і подивимося, як вона переходить від входу до виводу. Ось він знову для довідки:

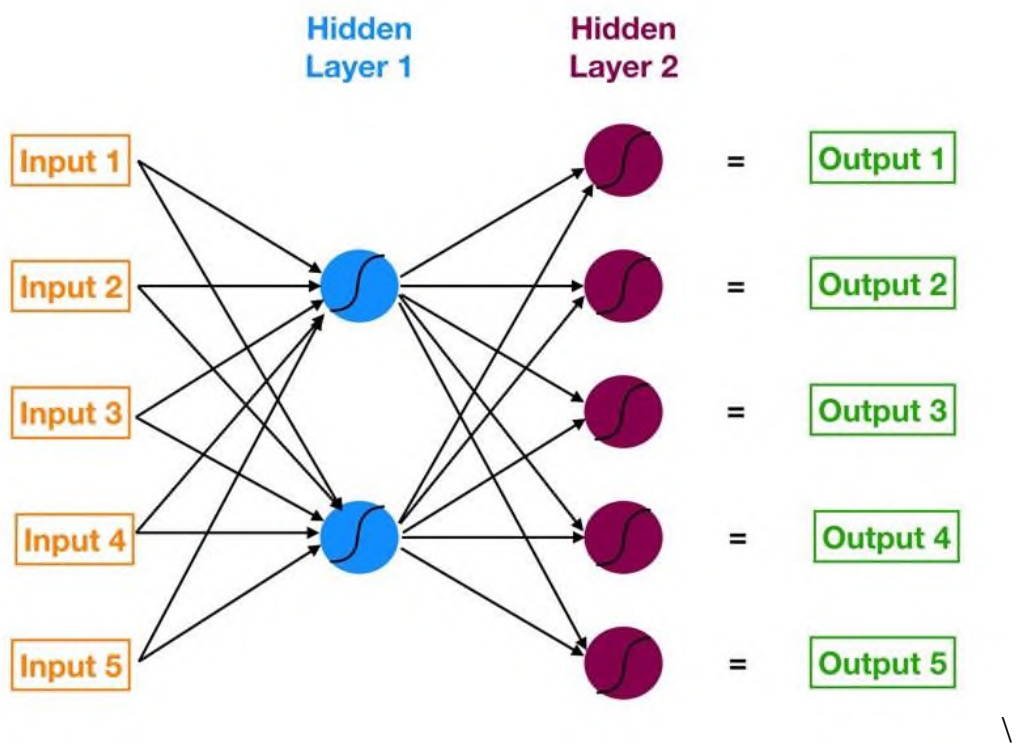


Рис. 2.2. Наша дещо складніша нейронна мережа

Перший прихований шар складається з двох нейронів. Отже, щоб підключити всі п’ять входів до нейронів прихованого шару 1, нам потрібно десять з’єднань. Наступне зображення (нижче) показує лише зв’язки між входом 1 і прихованим шаром 1.

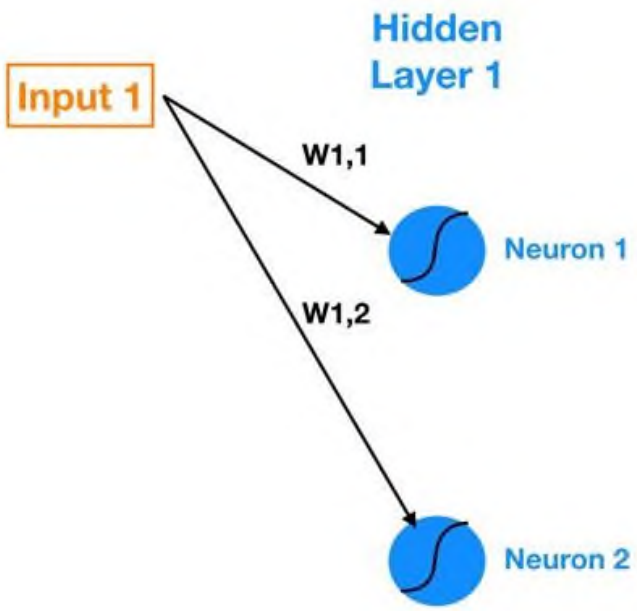


Рис. 2.4. Зв'язки між входом 1 і прихованим шаром 1

Зверніть увагу на наші позначення ваг, які живуть у зв’язках - W1,1 позначає вагу, яка живе у зв’язку між входом 1 та нейроном 1, а W1,2 позначає вагу у зв’язку між входом 1 та нейроном 2. Отже, загальне позначення що я буду слідувати, це Wa, b позначає вагу зв'язку між входом a (або нейроном a) та нейроном b.

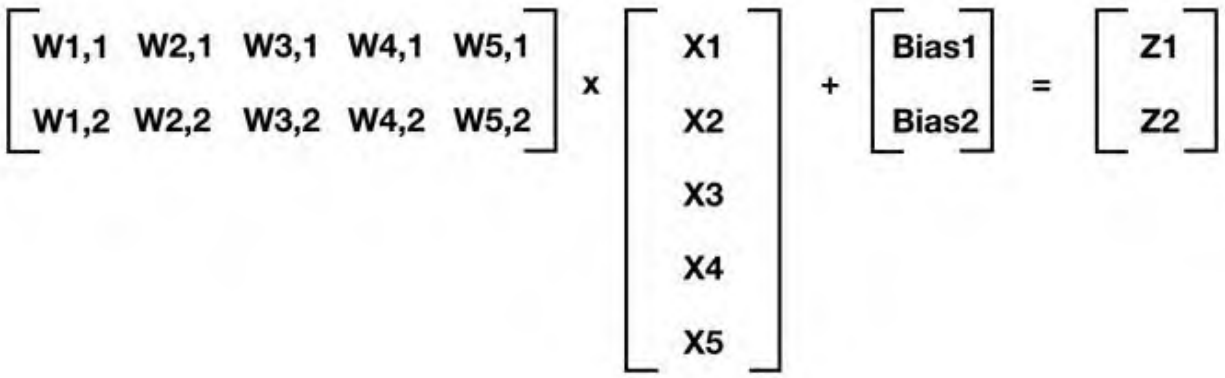
Тепер давайте обчислимо результати кожного нейрона в прихованому рівні 1 (відомий як активація). Ми робимо це, використовуючи наступні формули (W позначає вагу, In позначає введення).

Z1 = W1 * In1 + W2 * In2 + W3 * In3 + W4 * In4 + W5 * In5 + зміщення_Neuron1

Активація нейрону 1 =Сигмовидний(Z1)

Ми можемо використовувати матричну математику, щоб підсумувати це обчислення (пам’ятайте наші правила позначення - наприклад, W4,2 позначає вагу, яка живе у зв'язку між входом 4 та нейроном 2):

2.1.2. Матрична представлення нейронної мережі



Для будь-якого шару нейронної мережі, де попередній шар має m елементів глибиною, а поточний шар - n елементів глибиною, це узагальнює на:

[W] @ [X] + [Упередження] = [Z]

Де [W] - ваша матриця ваг n на m (зв’язки між попереднім шаром і поточним шаром), [X] - ваша матриця m на 1 або початкових входів, або активацій з попереднього шару, [Bias] - це ваша n на 1 матриця нейронних упереджень, а [Z] - ваша n на 1 матриця проміжних виходів. У попередньому рівнянні я слідую нотації Python і використовую @ для позначення множення матриць. Отримавши [Z], ми можемо застосувати функцію активації (сигмоїдна в нашому випадку) до кожного елемента [Z], і це дає нам наші нейронні виходи (активації) для поточного шару.

Нарешті, перш ніж рухатися далі, давайте візуально відобразимо кожен з цих елементів назад на нашій нейромережевій діаграмі, щоб зв’язати все це ([Упередження] вбудовано в сині нейрони).

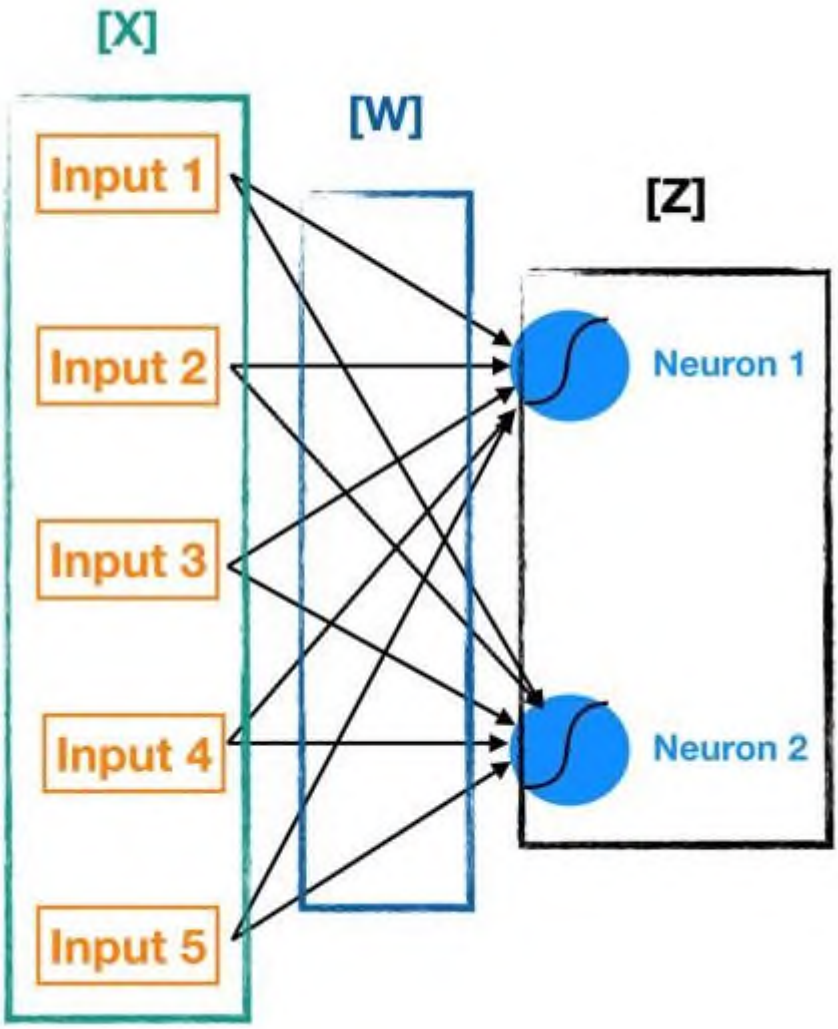


Рис. 2.5. Візуалізація [W], [X] та [Z]

Повторно обчислюючи [Z] і застосовуючи до нього функцію активації для кожного наступного шару, ми можемо переходити від вхідного до вихідного. Цей процес відомий як пряме розповсюдження. Тепер, коли ми знаємо, як обчислюються результати, настав час почати оцінювати якість результатів і навчати нашу нейронну мережу.

2.2. Навчення нейронної мережі

Навчальний процес нейронної мережі на високому рівні подібний до процесу багатьох інших моделей науки про дані - визначте функцію витрат та використовуйтеоптимізація градієнтного спускущоб її мінімізувати.

Спочатку давайте подумаємо, які важелі ми можемо потягнути, щоб мінімізувати функцію витрат. У традиційній лінійній або логістичній регресії ми шукаємо бета-коефіцієнти (B0, B1, B2 тощо), які мінімізують функцію витрат. Щодо нейронної мережі, ми робимо те саме, але набагато більший і складніший масштаб.

Під час традиційної регресії ми можемо змінювати будь-яку конкретну бета-версію ізольовано, не впливаючи на інші бета-коефіцієнти. Отже, застосовуючи невеликі поодинокі потрясіння до кожного бета-коефіцієнта та вимірюючи його вплив на функцію витрат, порівняно просто зрозуміти, у якому напрямку нам потрібно рухатись, щоб зменшити та врешті-решт мінімізувати функцію витрат.

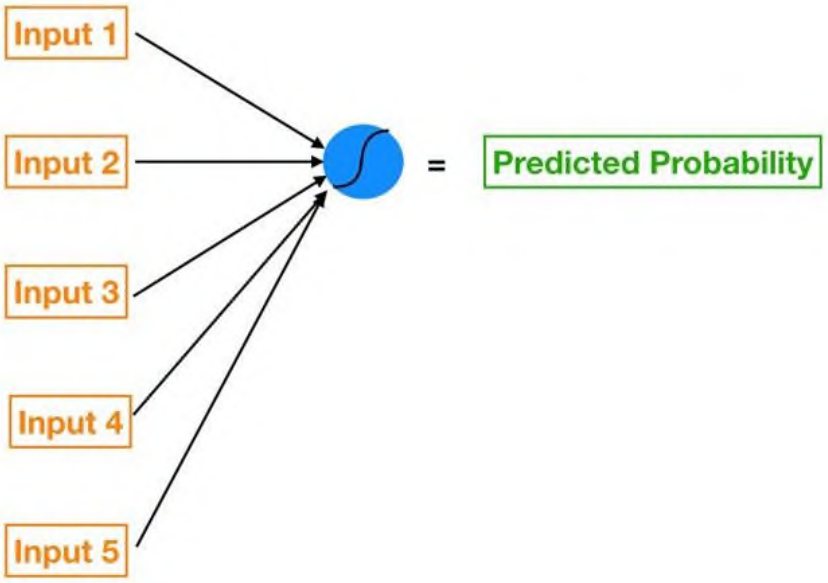


Рис.

П'ять особливостей логістичної регресії, реалізованої за допомогою нейронної мережі

У нейронній мережі зміна ваги будь-якого з’єднання (або зміщення нейрона) надає реверсивний ефект на всі інші нейрони та їх активацію в наступних шарах.

Це тому, що кожен нейрон у нейронній мережі схожий на власну маленьку модель. Наприклад, якби ми хотіли мати п’ять функціональних логістичних

регресій, ми могли б виразити це через нейронну мережу, як та зліва, використовуючи лише особливий нейрон!

Отже, кожен прихований шар нейронної мережі - це, в основному, набір моделей (кожен окремий нейрон у шарі діє як власна модель), вихід яких надходить у ще більше моделей далі за течією (кожен наступний прихований шар нейронної мережі вміщує ще більше нейронів) .

Функція витрат

Отже, з огляду на всю цю складність, що ми можемо зробити? Насправді це не так погано. Давайте зробимо це поетапно. По-перше, дозвольте мені чітко сформулювати нашу мету. Враховуючи набір навчальних матеріалів (наші особливості) та результати (ціль, яку ми намагаємося передбачити):

Ми хочемо знайти набір вагових коефіцієнтів (пам’ятайте, що кожна сполучна лінія між будь-якими двома елементами нейронної мережі містить вагу) та упередженості (кожен нейрон містить зміщення), які мінімізують нашу функцію витрат - де функція витрат є наближенням того, як помилкові наші прогнози відносно цільового результату.

Для навчання нашої нейронної мережі ми будемо використовуватиСередня квадратична помилка (MSE)як функція витрат:

MSE = Сума [(Прогноз - Фактичний) ²] * (1 / кількість_спостережень)

MSE моделі в середньому говорять нам про те, наскільки ми помиляємось, але із закруткою - шляхом квадратування помилок наших прогнозів перед їх усередненням, ми караємо передбачення, які є набагато суворішими, ніж ті, що трохи відхиляються. Функції витрат лінійної регресії та логістичної регресії діють дуже подібним чином.

Гаразд, круто, у нас є функція витрат для мінімізації. Час розпалюватиградієнтний спусктак?

Не так швидко - для використання градієнтного спуску нам потрібно знати градієнт нашої функції витрат, вектор, який вказує у напрямку найбільшої крутизни (ми хочемо неодноразово робити кроки у протилежному напрямку градієнта, щоб з часом досягти мінімуму).

За винятком нейронної мережі, у нас так багато змінних ваг та упереджень, які всі взаємопов'язані. Як ми обчислимо градієнт всього цього? У наступному розділі ми побачимо, як зворотне розповсюдження допомагає нам вирішити цю проблему.

Короткий огляд градієнтного спуску

Градієнт функції - це вектор, елементи якого є її частковими похідними щодо кожного параметра. Наприклад, якби ми намагалися мінімізувати функцію витрат, C (B0, B1), лише з двома змінними параметрами, B0 і B1, градієнт буде таким:

Градієнт C (B0, B1) = [[dC / dB0], [dC / dB1]]

Отже, кожен елемент градієнта повідомляє нам, як змінилася б функція витрат, якби ми застосували невелику зміну до цього конкретного параметра - тож ми знаємо, що налаштувати і на скільки. Підводячи підсумок, ми можемо пройти до мінімуму, виконавши такі дії:

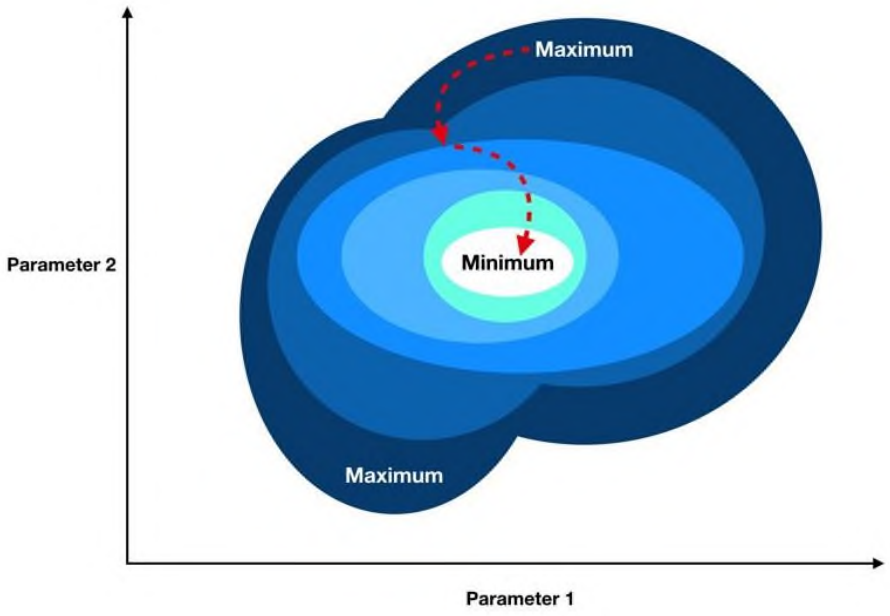


Рис. Ілюстрація градієнтного спуску

Обчисліть градієнт нашого “поточного місцезнаходження” (обчисліть градієнт, використовуючи наші поточні значення параметрів).

Змініть кожен параметр на величину, пропорційну його елементу градієнта та в протилежному напрямку від елемента градієнта. Наприклад, якщо часткова похідна нашої функції витрат відносно B0 є додатною, але крихітною, а часткова

похідна щодо B1 від’ємною і великою, тоді ми хочемо зменшити B0 на крихітну величину і збільшити B1 на велику величину до знизити нашу функцію витрат.

Повторно обчисліть градієнт, використовуючи наші нові налаштовані значення параметрів, і повторюйте попередні кроки, поки ми не досягнемо мінімуму.

2.3. Зворотне розмноження

Я відкладу на це чудовий підручник (в Інтернеті та безкоштовно!) для детальної математики (якщо ви хочете глибше зрозуміти нейронні мережі, неодмінно ознайомтесь). Натомість ми зробимо все можливе, щоб створити інтуїтивне розуміння того, як і чому працює зворотне розмноження.

Пам’ятайте, що пряме поширення - це процес руху вперед по нейронній мережі (від входів до кінцевого результату або передбачення). Зворотне розмноження - це зворотне. За винятком того, що замість сигналу, ми переміщуємо помилку назад через нашу модель.

Деякі прості візуалізації дуже допомогли, коли я намагався зрозуміти процес зворотного розповсюдження. Нижче наведено моє уявлення про просту нейронну мережу, яка поширюється вперед від входу до виходу. Процес може бути узагальнений наступними кроками:

Вхідні дані подаються в блакитний шар нейронів і модифікуються вагами, ухилом та сигмоподібною оболонкою кожного нейрона, щоб отримати активації. Наприклад: Activation_1 = Sigmoid (Bias_1 + W1 * Input_1)

Активація 1 і Активація 2, які виходять із синього шару, подаються в пурпуровий нейрон, який використовує їх для остаточної вихідної активації.

І метою прямого розповсюдження є обчислення активацій на кожному нейроні для кожного наступного прихованого шару, поки ми не дійдемо до результату.

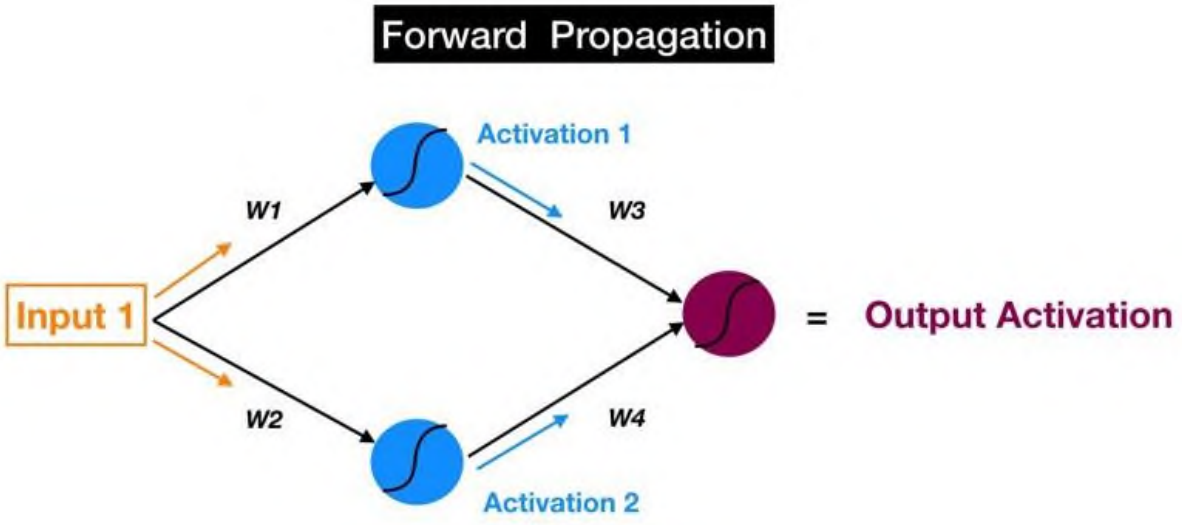


Рис. Поширення вперед у нейронній мережі

Тепер давайте просто повернемо його назад. Якщо слідувати червоним стрілкам (на малюнку нижче), ви помітите, що зараз ми починаємо з виходу пурпурового нейрона. Це наша активація на виході, яку ми використовуємо для прогнозування, і остаточне джерело помилок у нашій моделі. Потім ми переміщуємо цю помилку назад через нашу модель за допомогою тих самих ваг і з'єднань, які ми використовуємо для прямого розповсюдження нашого сигналу (тому замість Активації 1, тепер у нас є Error1 - помилка, що приписується верхньому синьому нейрону).

Пам'ятаєте, ми говорили, що мета прямого розповсюдження полягає в розрахунку активації нейронів шар за шаром, поки не дійдемо до результату? Тепер ми можемо сформулювати мету зворотного розмноження подібним чином:

Ми хочемо обчислити похибку, що приписується кожному нейрону (я просто буду називати цю величину помилки помилкою нейрона, оскільки повторювати «атрибутивність» знову і знову не цікаво), починаючи від шару, найближчого до виходу, аж до початкового шар нашої моделі.

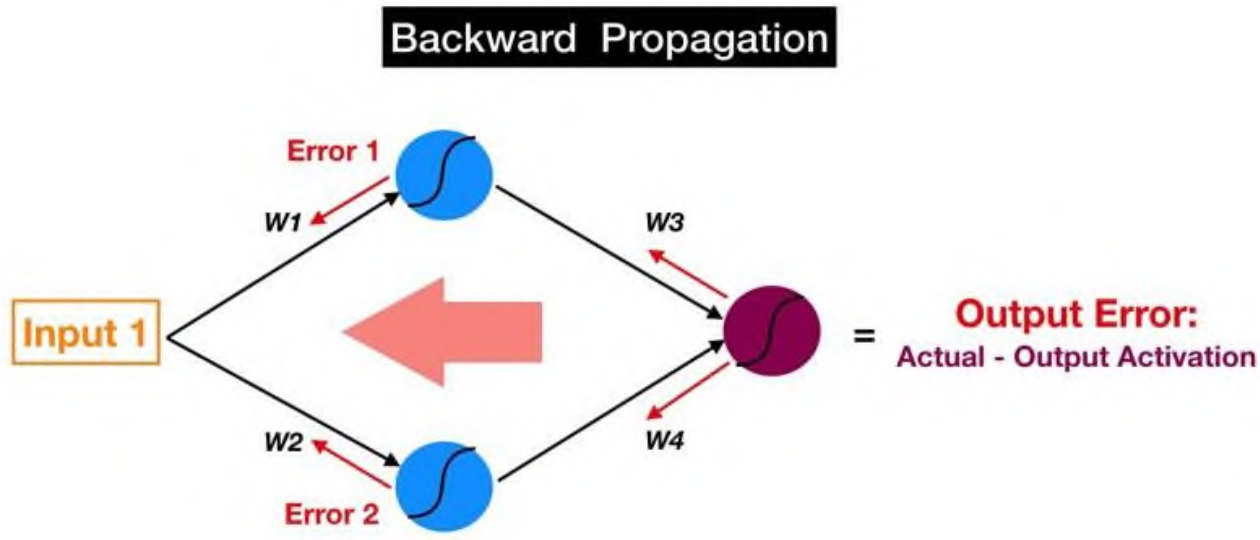


Рис. Зворотне розмноження в нейронній мережі

То чому ми дбаємо про помилку кожного нейрона? Пам’ятайте, що двома будівельними блоками нейронної мережі є зв’язки, які передають сигнали до певного нейрона (з вагою, що живе в кожному зв’язку), і до самого нейрона (з ухилом). Ці ваги та упередження по всій мережі також є циферблатами, які ми налаштовуємо, щоб змінити прогнози, зроблені моделлю.

Величина похибки конкретного нейрона(відносно помилок усіх інших нейронів) прямо пропорційний впливу виходу цього нейрона (він же активації) на нашу функцію витрат.

Отже, помилка кожного нейрона є проксі для часткової похідної функції вартості відносно вхідних даних цього нейрона. Це має інтуїтивний сенс - якщо певний нейрон має набагато більшу похибку, ніж усі інші, тоді налаштування ваги та упередженості нашого нейрона, що порушує, матиме більший вплив на загальну помилку нашої моделі, ніж возиння з будь-якими іншими нейронами.

А часткові похідні щодо кожної ваги та упередженості - це окремі елементи, які складають вектор градієнта нашої функції витрат. Отже, в основному зворотне розповсюдження дозволяє нам розрахувати похибку, що приписується кожному нейрону, а це, в свою чергу, дозволяє розрахувати часткові похідні і, врешті-решт, градієнт, щоб ми могли використовувати градієнтний спуск. Ура!

2.4. Метод навчанн «Гра за звинувачення»

Ну нейрони через зворотне розповсюдження є господарями гри у звинуваченні. Коли помилка знову поширюється на певний нейрон, цей нейрон швидко та ефективно спрямовуватиме пальцем на колегу (або колеги), що винен у винній причині помилки (тобто нейрони шару 4 спрямовуватимуть палець на нейрони шару 3, нейрони рівня 3 на рівні 2 нейрони тощо).

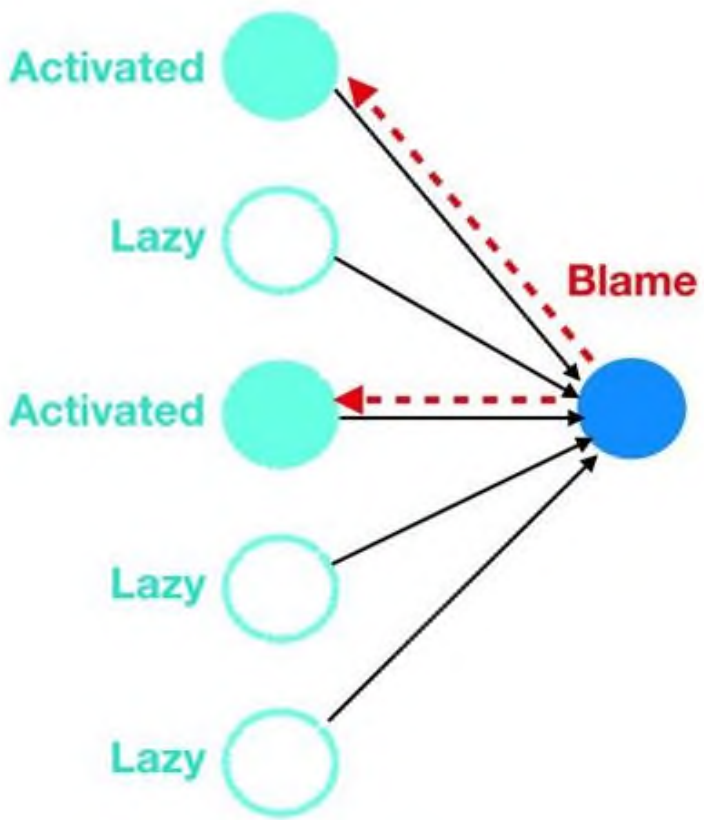


Рис. Нейрони звинувачують найбільш активні нейрони вище по течії

І як кожен нейрон знає, кого звинувачувати, оскільки нейрони не можуть безпосередньо спостерігати за помилками інших нейронів? Вони просто дивляться на те, хто надіслав їм найбільше сигналу з точки зору найвищих і найчастіших активацій. Як і в реальному житті, ледачі, які грають у безпеці (низькі та рідкісні активації), катаються без вини, в той час як нейрони, які виконують найбільшу роботу, звинувачують і змінюють їх вагу та упередження. Цинічно так, але також дуже ефективно для того, щоб отримати оптимальний

набір ваг та упереджень, які мінімізують нашу функцію витрат. Зліва - зображення того, як нейрони кидають один одного під автобус.

І це в двох словах - це інтуїція процесу зворотного розмноження. На мій погляд, це три основні способи зворотного розмноження:

Це процес зміщення помилки назад шар за шаром і приписування правильної кількості помилок кожному нейрону нейронної мережі.

Помилка, пов’язана з певним нейроном, є хорошим наближенням того, як зміна ваги нейрона (від зв’язків, що ведуть до нейрона) та упередженості вплинуть на функцію витрат.

Якщо дивитись назад, то більш активні нейрони (не ледачі) є тими, яких процес зворотного розмноження звинувачує та допрацьовує.

Ознакою зображення називається його проста відмітна характеристика або властивість. Деякі ознаки є природними в тому сенсі, що вони встановлюються візуальним аналізом зображення, тоді як інші, так звані штучні ознаки, виходять в результаті його спеціальної обробки або вимірювань. До природних ознак відносяться світлина (яскравість) і текстура різних областей зображення, форма контурів об'єктів. Гістограми розподілу яскравості і спектри просторових частот дають приклади штучних ознак.

2.5. Ознаки зображень для передачі нейронного стилю

2.5.1. Яркостні ознаки зображень

Найбільш важливою ознакою зображення є яскравість. Яскравість виражається через такі величини, як спектральна інтенсивність випромінювання, координати кольору і т. д., які називатимуться ознаками яскравості. Вимірювання ознак яскравості можна проводити або в окремих точках зображення, або в їх околицях. Наприклад, середня яскравість околиці крапки $\{j, k\}$ зображення розміром $(2W + 1)$ x $(2W + 1)$ елементів визначається як:

$$\bar{Y}(j,k) = (1/(2W+1)^2) \sum_{m=-W}^{W} \sum_{n=-W}^{W} Y(j+m,k+n) \qquad (2.1)$$

Існує безліч різних способів визначення ознак яскравості. Можна використовувати значення яскравості або координат кольору безпосередньо або перейти до нових ознак яскравості, виконавши деяке лінійне, нелінійне або, можливо, необоротне перетворення.

Вимірювання ознак яскравості набуває особливої важливості при виділенні зображених об'єктів (символічний опис) і при маркіровці таких об'єктів (інтерпретація).

2.5.2. Просторово спектральні ознаки

Спектральні коефіцієнти, знайдені в результаті двовимірного перетворення, визначають ваги базисних зображень (двовимірних базисних функцій), відповідних цьому перетворенню, при яких зважена сума базисних функцій ідентична зображенню. Можна вважати, що ці коефіцієнти показують ступінь кореляції відповідних базисних функцій із зображенням. Якщо базисне зображення має ту ж просторову форму, що і ознака, яку необхідно виявити на зображенні, то виявлення ознаки можна виконати просто шляхом спостереження значення відповідного спектрального коефіцієнта. Практична складність полягає в тому, що об'єкти, які необхідно виявити, часто мають складні форму і розподіл яркостей і, отже, не відповідають точно простішим образам яскравості, які представляються базисними функціями більшості перетворень.

Лендеріс і Стенлі [3, 2] досліджували застосування безперервного двовимірного перетворення Фурье, отриманого за допомогою когерентно-оптичного пристрою, для виділення ознак зображення. Оптична система створює електричне поле, пропорційне спектру:

$$F(\omega_x,\omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(x,y) \exp\{-i(\omega_x x + \omega_y y\} dx dy M(\omega_x,\omega_y) = \left|F(\omega_x,\omega_y)\right| \qquad (2.2)$$

де (ω_x, ω_y) - просторові частоти. Оптичний прочитуючий пристрій дає на виході функції:

$$M(\omega_x, \omega_y) = \left| F(\omega_x, \omega_y) \right|^2 \qquad (2.3)$$

значення яких пропорційні інтенсивності спектру. Слід зазначити, що функції і *F (x, y)* є парою, зв'язаною однозначним перетворенням, тоді як функція $\left| M(\omega_x, \omega_y) \right|$ неоднозначно пов'язана з *F (x, y)*. Наприклад, функція не змінюється, якщо початок координат на площині початкового зображення зрушується. Для деяких застосувань інваріантність функції може показатися гідністю.

Інтеграція функції по куту на площині просторових частот дає просторово-частотну ознаку, інваріантну щодо зрушення і обертання. Представивши функцію в полярних координатах, набудемо цієї ознаки в наступному вигляді:

$$N(p) = \int_0^{2\Pi} M(\rho, \theta) d\theta \qquad (2.4)$$

де $\theta = arctg(\omega_x / \omega_y)$ *и* $\rho^2 = (\omega_x^2 + \omega_y^2)$. Інваріантністю щодо зміни масштабу володіє ознака:

$$P(\theta) = \int_0^{2\Pi} M(\rho, \theta) \rho \cdot d\rho \qquad (2.5)$$

Якщо вхідне зображення має обмежені розміри, то поле отримане в результаті перетворення Фур'є, затухатиме в межах певного діапазону. Можна легко показати, що якщо функцію *F (x, y)* помножити на функцію вікна *W (x, y),* яка рівна одиниці усередині деякого прямокутника і нулю поза ним, то фур'є-спектр цього твору рівний згортку з Фур'є-спектром функції вікна $\left| W(\omega_x, \omega_y) \right|$.

Загасання, викликане обмеженими розмірами вікна, слід враховувати при визначенні відносних значень коефіцієнтів Фурье на різних просторових частотах[6].

Для виділення ознак зображення яскравістю образ зазвичай розглядається в областях специфічної форми. Принцип Фурье для різних областей визначається таким чином:

1) Горизонтальна щілина

$$S_1(m) = \int_{\omega_y(m)}^{\omega_y(m+1)} M(\omega_x, \omega_y) d\omega_y \qquad (2.6)$$

2) Вертикальна щілина

$$S_2(m) = \int_{\omega_x(m)}^{\omega_x(m+1)} M(\omega_x, \omega_y) d\omega_x \qquad (2.7)$$

3) Кільце

$$S_3(m) = \int_{\rho(m)}^{(m+1)} M(\rho, \theta) d\rho \qquad (2.8)$$

4) Сектор

$$S_3(m) = \int_{\theta(m)}^{\theta(m+1)} M(\rho, \theta) d\theta \qquad (2.9)$$

Для дискретного зображення, що описується масивом чисел *F (j, k),* як джерело ознак можна розглядати безпосередньо дискретний спектр Фур'є:

$$F(u,v) = (1/N)\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} F(j,k)\exp\left\{\frac{-2\Pi i}{N}(ux+vy)\right\} \quad (2.10)$$

при *i, v* = 0 .... *N – I.* В цьому випадку ознаки для горизонтальної щілини, вертикальної щілини, кільця і сектора можна визначити аналогічно виразам (2.6) —(2.9). Описану ідею можна розповсюдити також і на інші унітарні перетворення, такі, як перетворення Адамара і перетворення Хаару[8]. Виділення ознак,

представлених у вигляді спектральних коефіцієнтів, було досліджене в різних практичних завданнях, в яких ці ознаки використовувалися як вхідні дані для системи розпізнавання образів. Спектральні ознаки знаходять широке застосування — від класифікації земельних ресурсів [5] до діагностики хвороб по рентгенівських знімках [6—8].

2.5.3. Контурні ознаки

Різкі зміни (розриви) яскравості, координат кольору або параметрів, що характеризують текстуру, є важливими простими ознаками, оскільки вони часто визначають контури зображених. об'єктів. Локальні розриви значень яскравості називаються перепадами яскравості, або контурами яскравості (*luminance edge*). Протяжні розриви, звані відрізками межі об'єкту. Розглянемо перепади яскравості, що розділяють області з майже однаковою яскравістю.

У одновимірному випадку перепад характеризується заввишки, кутом нахилу і координатою центру схилу. Перепад існує, якщо його кут нахилу і висота більше деякого заданого порогу. Для двовимірного випадку важлива також орієнтація перепаду по відношенню до осі *x.* Ідеальний детектор перепаду при обробці областей зображення повинен указувати на наявність перепаду в єдиній крапці, розташованій в центрі схилу.

Загальний підхід до виявлення перепадів на одноколірному зображенні ілюструється у вигляді блок-схеми на рис. 2.1. Початкове зображення, представлене масивом чисел *F* (*j, K),* піддається лінійній або нелінійній обробці з тим, щоб підсилити перепади яскравості. В результаті утворюється масив чисел

G (*j*, до), що описує зображення з підкресленими змінами яркостей. Потім виконується операція порівняння з порогом і визначається положення елементів зображення з яскраво вираженими перепадами[9]. Якщо

$$G(j,k) < T_L(j,k) \qquad (2.11)$$

то має місце низхідний перепад, а при

$$G(j,k) = T_L(j,k) \qquad (2.12)$$

висхідний перепад. Величини *TL* (*j*, k_A) і *Ti* (*j*, *k*) є нижнє і верхнє порогові значення. Ці значення можна зробити змінними в площині зображення для компенсації впливу сильних змін яскравості на результати виявлення перепадів. Вибір порогу є одним з ключових питань виділення перепадів. При дуже високому рівні порогу не будуть виявлені структурні елементи з низьким контрастом. Навпаки, дуже низький рівень порогу з'явиться причиною того, що шум буде помилково прийнятий за перепад[11]. Для позначення положення перепадів на зображенні часто формують контурний препарат — масив елементів *E* (*j*,*k*). Наприклад, положення точок висхідних препаратів можна було б відзначати білими крапками на чорному фоні. Можна також точки висхідних препаратів відзначати білим кольором, низхідних – чорним, а решта елементів зображення – деяким середнім рівнем яскравості.

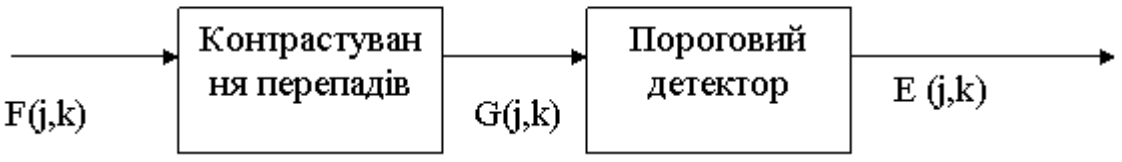


Рис. 2.11. Порогова система виявлення перепадів

Інший важливий підхід до виявлення перепаду полягає в апроксимації фрагмента реального зображення деяким ідеальним одно- або двовимірним перепадом. Якщо апроксимація виявляється достатньо точною, то вважається, що перепад існує і йому приписуються параметри ідеального перепаду.

2.6. Виявлення і поєднання об'єктів

Модель *MatLab* охоплює два завдання, що відносяться до аналізу зображень: виявлення об'єктів і поєднання зображень. Виявлення зображень пов'язане зі встановленням наявності об'єктів, щодо яких передбачається, що вони є на картинці. Поєднання зображень полягає в знаходженні поточкової відповідності (прив'язці) двох видів однієї сцени.

Один з основних способів виявлення об'єктів на зображенні полягає в зіставленні з еталоном. При цьому еталон об'єкту, що цікавить нас, порівнюється зі всіма невідомими об'єктами, що знаходяться на зображенні. Якщо схожість між невідомим об'єктом і еталоном достатньо велика, то цей об'єкт позначається як відповідний еталонному об'єкту.

Як простий приклад зіставлення з еталоном розглянемо набір двійкових фігур, утворених чорними лініями на білому фоні (рис.2.2, а). В даному прикладі завдання полягає в тому, щоб виявити на зображенні рівнобедрений прямокутний трикутник і визначити його місцеположення. На рис.2.2,б показаний еталонний трикутник. Товщина сторін вибрана так, щоб забезпечити інваріантність результатів по відношенню до невеликих змін форми шуканого трикутника і при цьому не дуже багато втратити в точності його локалізації. При роботі еталон послідовно переміщається по полю зображення і досліджується його схожість з різними ділянками зображення.

Повний збіг еталону з якою-небудь частиною зображення буває рідко через дію шумів і спотворень, викликаних просторовою дискретизацією і квантуванням яскравості, а також унаслідок відсутності апріорної інформації щодо точної форми і структури об'єкту, який потрібно виявити. Тому зазвичай за допомогою деякої конкретної міри відмінності *D (m, n)* між еталоном і зображенням в крапці (*m, n)* указують на наявність виділеного об'єкту там, де це відмінність менше деякого встановленого порогу *LD (m, n).* Як правило, поріг вибирається постійним для всіх точок зображення[4]. Зазвичай як міра відмінності береться среднеквадратическая помилка, визначувана як:

$$D(m,n) = \sum_j \sum_k \left[F(j,k) - T(j-m,k-n)\right]^2 \qquad (2.13)$$

де *F (j, k)* — елемент масиву зображення, на якому проводиться пошук, а *T (j,k)* — елемент еталонного масиву.

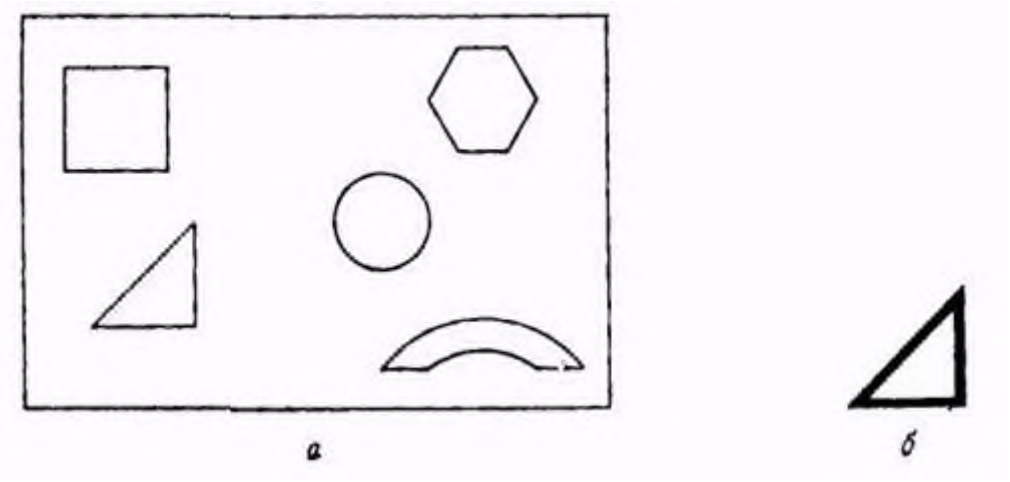


Рис. 2.12. Зіставлення з еталоном:
а – безліч об'єктів, *б* – еталон для трикутника

Пошук звичайно, обмежений областю перекриття зміщеного еталону і зображення. Вважається, що є схожість з еталоном в крапці з координатами *(m,n*), якщо:

$$D(m,n) < L_D(m,n) \qquad (2.14)$$

Тепер представимо рівність (3.13) в наступному вигляді:

$$D(m,n) = D_1(m,n) - 2D_2(m,n) + D_3(m,n) \qquad (2.15)$$

де

$$D_1(m,n) = \sum_j \sum_k \left[F(j,k)\right]^2 \qquad (2.16)$$

$$D_2(m,n) = \sum_j \sum_k \left[F(j,k)\right] \cdot T(j-m,k-n) \qquad (2.17)$$

$$D_3(m,n) = \sum_j \sum_k [T(j-m,k-n)]^2 \quad (2.18)$$

Доданок *D3 (m, n)* —это енергія еталону, яка постійна і не залежить від координат *(m, n).* Енергія зображення в межах вікна, представлена першим доданком *D1 (m, n),* при зміні координат зазвичай міняється досить поволі. Другий доданок — взаємна кореляція *RFT (m, n)* зображення і еталону. При збігу зображення і еталону взаємна кореляція повинна бути велика, що приводить до малих значень среднеквадратической помилки. Проте величина взаємної кореляції не завжди адекватно відображає відмінність зображення від еталону, оскільки енергія зображення *Dt (m, n)* залежить від значенні координат. Взаємна кореляція може збільшитися навіть за відсутності відповідності зображення еталону, якщо яскравість зображення в околиці крапки з координатами *(m, n)* велика. Цю трудність можна обійти, порівнюючи нормовану взаємну кореляцію:

$$R_{FT}(m,n) = \frac{D_2(m,n)}{D_1(m,n)} = \frac{D_2(m,n) = \sum_j \sum_k [F(j,k)] \cdot T(j-m,k-n)}{D_1(m,n) = \sum_j \sum_k [F(j,k)]^2} \quad (2.19)$$

з порогом *LR (m, n).* Вважається, що схожість з еталоном має місце, якщо:

$$R_{FT}(m,n) > L_R(m,n) \quad (2.20)$$

Нормована взаємна кореляція має максимальну величину, рівну одиниці, тоді і тільки тоді, коли зображення у вікні точно співпадає з еталоном[6]. Головний недолік методу зіставлення з еталоном полягає в необхідності використання величезної кількості еталонів для обліку змін об'єктів, що виникають при їх повороті і збільшенні (зменшенні) розмірів. З цієї причини при зіставленні з еталоном бажано обмежитися ознаками, які менше залежать від змін розміру і форми об'єкту. Такими ознаками можуть бути, наприклад, розгалуження контурних ліній, створюючі фігуру типа *Y* або *T.*

2.7. Прив'язка зображень

У багатьох прикладних програмах обробки зображень необхідно виконувати поелементне порівняння двох зображень одного і того ж об'єкту, зареєстрованих різними датчиками, або двох зображень деякого об'єкту, отриманих за допомогою одного датчика, але в різний час. Щоб здійснити таке порівняння, необхідно виконати взаємну прив'язку цих зображень і таким чином скоректувати відносні просторові зрушення, відмінності в посиленні, зсуви, викликані поворотом, а також геометричні спотворення і спотворення яскравості кожного зображення. Часто виявляється можливим виключити або мінімізувати дію більшості джерел помилок прив'язки шляхом відповідного статичного калібрування і корекції датчиків. Проте в деяких випадках виявлення подібних помилок і їх подальшу корекцію потрібно виконувати динамічно для кожної пари зображень[20]. У даній дипломній роботі вирішується завдання поєднання зображень за наявності просторового зрушення одного з них щодо іншого. Отримані результати можна використовувати для компенсації відмінностей, викликаних поворотом і зміною масштабу, якщо збільшити розмірність завдання або виконати відповідне перетворення координат. (Наприклад при переході до полярних координат поворот приводиться до зрушення.)

Класичний спосіб кореляційної прив'язки пари функцій полягає в тому, що формується величина, що вимірює кореляцію між цими функціями, і знаходиться положення максимуму функції кореляції [11,12].

Розглянемо застосування цього способу у разі двох вимірювань. Хай масиви $F_1(j,k)$ і $F_2(j,k)$ представляють два дискретні зображення, які потрібно прив'язати один до одного. У простій формі міра кореляції визначається таким чином (рис 2.13):

Площа
вікна

Зона
пошуку

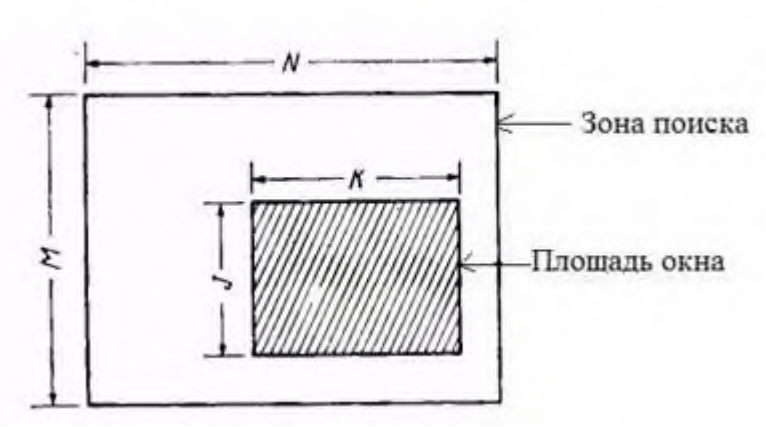


Рис.2.13. Зона пошуку при кореляційній прив'язці і площа вікна.

$$R(m,n)=\frac{\sum_{j=1}^{J}\sum_{k=1}^{K}F_1(j,k)F_2(j-m,k-n)}{\left[\sum_{j=1}^{J}\sum_{k=1}^{K}F_1^{\,2}(j,k)\right]^{1/2}\left[\sum_{j=1}^{J}\sum_{k=1}^{K}F_2^{\,2}(j-m,k-n)\right]^{1/2}} \qquad (2.21)$$

де *(j,k)*—индексы елементів у вікні *W* розміром *JXK* елементів, яке розташоване усередині зони пошуку *S* розміром *MXN* елементів. Рис. 2.6 ілюструє співвідношення між зоною пошуку і вікном. Взагалі, функцію кореляції потрібно обчислити для всіх *(M – J* + 1) *(N –k*+ 1) можливих зсувів вікна в зоні пошуку для того, щоб визначити її максимальне значення і отримати оцінку помилки поєднання.

Застосування цій простій заходи кореляції пов'язано з двома основними труднощами. По-перше, функція кореляції може мати досить розмитий максимум, що утрудняє його виявлення. Слід зазначити, що міра кореляції не враховує просторову структуру порівнюваних зображень. По-друге, шум на зобра-

женні може приховати максимум кореляції. Обидві труднощі можна подолати, поліпшивши міру кореляції так, щоб в ній враховувалися статистичні властивості зображень $F_1(j, k)$ і $F_2(j, k)$.

Покращувана міра кореляції визначається як:

$$R(m,n) = \frac{\sum_{j=1}^{J}\sum_{k=1}^{K} G_1(j,k)G_2(j-m,k-n)}{\left[\sum_{j=1}^{J}\sum_{k=1}^{K} G_1^{\,2}(j,k)\right]^{1/2}\left[\sum_{j=1}^{J}\sum_{k=1}^{K} G_2^{\,2}(j-m,k-n)\right]^{1/2}} \qquad (2.22)$$

де масиви $G_i(j, k)$ виходять сверткой масивів, що описують зображення, з функціями $D_i(j, k)$, що є імпульсними характеристиками просторових фільтрів:

$$G_i(j,k) = F_i(j,k) \cdot D_i(j,k) \qquad (2.23)$$

Імпульсні характеристики вибираються так, щоб максимізувати пікову кореляцію у тому випадку, коли порівнювані зображення суміщені найкращим чином. Хай f_1 — вектор, отриманий розгорткою по стовпцях фрагмента $F_1(j, k)$, відповідного окну, а вектор $f_2\ (m, n)$ складений з елементів фрагмента $F_2\ (j, k)$ при заданому зрушенні (m, n). Повне число різних векторів $f_2\ (m, n)$ складає $M*N$. Елементи векторів f_1 і f_2 (m, n) зазвичай сильно корельовані. Тому відповідно до методу узгодженої фільтрації випадкових полів перший крок обробки повинен полягати в «вибілюванні», т. е, в множенні цих векторів на матриці вибілюючих фільтрів H_1 і H_2:

$$g\mathbf{1} = [H_1]^{-1} f_1 \qquad (2.24)$$

$$g_2(m,n) = [H_2]^{-1} f_2(m,n) \qquad (2.25)$$

Матриці H1 і H2 визначаються через ковариационные матриці зображень:

$$k_1 = H_1 \cdot H_1^{\,T} \qquad (2.26)$$

$$k_2 = H_2 \cdot H_2^{\,T} \qquad (2.27)$$

Матриці H1 і H2 можна представити в наступному вигляді:

$$H_1 = E_1 \Lambda_1^{1/2} \tag{2.28}$$

$$H_2 = E_2 \Lambda_2^{1/2} \tag{2.29}$$

де матриці E_1 і E_2 утворені з власних векторів ковариационных матриць $K1$ і K_2, а Λ_1 і — діагональні матриці з їх власних значень.

Міра кореляції (2.20) записується у вигляді нормованого скалярного добутку:

$$R_s(m,n) = \frac{g_i^T g_2(m,n)}{(g_1^T g_1)^{1/2}(g_2^T(m,n)g_2(m,n))^{1/2}} \tag{2.30}$$

Можна показати, що можливе інше представлення цього заходу:

$$R_S(m,n) = \frac{[(k^T)^{-1} f_1]^T f_2}{\{[(k^T)^{-1} f_1]^T [(k^T)^{-1} f_1][f_2^T(m,n)][f_2(m,n)]\}} \tag{2.31}$$

де $K = H_2 * H_1$ Використовуючи формулу (2.28) треба проводити «вибілювання» вектора f_1 і всіх $M*N$ векторів $f2$**,** тоді як формула вимагає лише одного множення вектора $f1$ на матрицю $G = (k^T)^{-1}$. Ясно, що друга формула має перевагу над першою, якщо всі обчислення виконуються звичайними засобами.

Щоб знайти матриці *G*, необхідно обчислити два набори власних векторів і власних значень ковариационных матриць двох порівнюваних зображень в межах вікна[15]. Якщо вікно містить *JXK* елементів, то кожна з ковариационных матриць K_1 і K_2 матиме розмір $(J*K)x(J*K)$. Наприклад, якщо $J = K = 16$, то ковариационные матриці K_1 і K_2 матимуть розмір 256*x*256. Взагалі обчислення власних векторів і власних значень для таких великих матриць виявляється трудомістким завданням для всіх обчислювальних машин, за винятком наймогутніших. Проте в особливих випадках ці обчислення можна помітно спростити [12]. Наприклад, якщо зображення моделюються реалізаціями

роздільного марківського процесу і відсутній шум, то згортка (2.23) зводиться до згортку зображення з маскою виявлення перепадів (2.24), коли:

$$D_j = \begin{bmatrix} \rho^2 & -\rho(1+\rho^2) & \rho^2 \\ -\rho(1+\rho^2) & (1+\rho^2)^2 & -\rho(1+\rho^2) \\ \rho^2 & -\rho(1+\rho^2) & \rho^2 \end{bmatrix} \qquad (2.32)$$

де — коефіцієнт кореляції суміжних елементів зображення. Якщо обидва зображення просторово некорельовано, то $\rho = 0$ і операція згортки не потрібний. У іншому граничному випадку, коли $\rho = 1$

$$D_i = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad (2.33)$$

Цей оператор є одним з видів оператора Лапласа. Таким чином, коли зображення сильно корельовані, обчислення покращуваної міри кореляції (2.20) зводиться до звичайної кореляції контурних зображень двох сцен.

Основний недолік кореляційного способу прив'язки зображень полягає в тому, що якщо площа вікна і зона пошуку великі, то необхідно виконати великий об'єм обчислень. *С.* допомогою кореляційного способу не можна отримати

рішення до тих пір, поки не буде обчислений кореляційний масив *Rs (m, n)* для всіх *(m, n).* Крім того, об'єм обчислень для визначення *Rs(m,n)* однаковий при будь-якій відмінності двох зображень. Ці недоліки привели до пошуку послідовних алгоритмів, які змогли б по суті давати оцінку величини відмінності зображень при меншому числі обчислень.

Барнеа і Сильверман [18] запропонували метод послідовних випробувань. Основна форма цього алгоритму брехливо проста. Обчислюється міра відмінності зображень у вікні:

$$E_S = \sum_j \sum_k \left[F_1(j,k) - F_2(j-m, k-n) \right] \qquad (2.34)$$

Обчислення проводяться послідовно від крапки до крапки. Якщо поточне значення міри відмінності перевищить наперед певний поріг, перш ніж все *J x N* елементів в межах вікна будуть досліджені, то вважається, що випробування для даного вікна не вдалося, і перевіряється нове вікно. Якщо значення міри відмінності росте поволі, то число крапок, розглянутих до того моменту, коли, нарешті, поріг буде перевершений, записується і позначається як оцінка випробуваного вікна. Після дослідження всіх вікон вікно з найбільшою величиной оцінки вважається таким, що знаходиться в правильній позиції. Максимально можлива оцінка рівна числу крапок у вікні розміром *JXK*. Слід відмітити, що у разі вікон, яким відповідають великі розузгодження, потрібне відносно невелике число обчислень.

Для збільшення швидкості збіжності і підвищення надійності запропоновано декілька модифікацій алгоритму послідовних випробувань [18]. Природною модифікацією служить гібридна система, в якій для відкидання крапок з великим розузгодженням використовується алгоритм послідовних випробувань з подальшим обчисленням міри кореляції для крапок, що залишилися, підлягають випробуванню. Така система могла б поєднувати переваги робочих характеристик способу вимірювання покращуваної міри кореляції і переваги в швидкості методу послідовних випробувань.

2.8. Нейромережеві алгоритми

Нейромережеві методи - це методи, що базуються на застосуванні різних типів нейронних мереж (НМ). Основні напрями застосування різних НМ для розпізнавання образів і зображень:

- застосування для витягання ключових характеристик або ознак заданих образів;

- класифікація самих образів або вже витягнутих з них характеристик (у першому випадку витягання ключових характеристик відбувається неявно усередині мережі);

- рішення оптимізаційних задач.

Архітектура штучних НМ має деяку схожість з природними нейронними мережами. НМ, призначені для вирішення різних завдань, можуть істотно розрізнятися алгоритмами функціонування, але їх головні властивості наступні.

НМ складається з елементів, званих формальними нейронами, які самі по собі дуже прості і пов'язані з іншими нейронами. Кожен нейрон перетворить набір сигналів, що поступають до нього на вхід у вихідний сигнал. Саме зв'язки між нейронами, що кодуються вагами, грають ключову роль[12]. Одна з переваг НМ (а так само недолік при реалізації їх на послідовній архітектурі) це те, що всі елементи можуть функціонувати паралельно, тим самим істотно підвищуючи ефективність рішення задачі, особливо в обробці зображень. Крім того, що НМ дозволяють ефективно вирішувати багато задач, вони надають могутні гнучкі і універсальні механізми навчання, що є їх головною перевагою перед іншими методами (імовірнісні методи, лінійні роздільники, вирішальні дерева і т.п.). Навчання позбавляє від необхідності вибирати ключові ознаки, їх значущість і відносини між ознаками. Але проте вибір початкового представлення вхідних даних (вектор в *n*-мірному просторі, частотні характеристики, вейвлети і т.п.), істотно впливає на якість рішення і є окремою темою. НМ володіють хорошою узагальнювальною здатністю (краще чим у вирішальних дерев), тобто можуть успішно поширювати досвід, отриманий на кінцевому повчальному наборі, на всю безліч образів.

Для розпізнавання людини по зображенню обличчя можливо застосовувати багатошарові нейронні мережі (БНМ).

Архітектура багатошарової нейронної мережі (БНМ) складається з послідовно сполучених шарів, де нейрон кожного шару своїми входами пов'язаний зі всіма нейронами попереднього шару, а виходами - наступного. НМ з двома вирішальними шарами може з будь-якою точністю апроксимувати будь-яку багатовимірну функцію. НМ з одним вирішальним шаром здатна формувати лінійні розділяючі поверхні, що сильно звужує круг завдань ними вирішуваних, зокрема така мережа не зможе вирішити задачу типу що "виключає або". НМ з нелінійною функцією активації і двома вирішальними

шарами дозволяє формувати будь-які опуклі області в просторі рішень, а з трьома вирішальними шарами - області будь-якої складності, у тому числі і неопуклою. При цьому БНМ не втрачає своєї узагальнювальної здатності. Навчаються БНМ за допомогою алгоритму зворотного розповсюдження помилки, градієнтного спуску, що є методом, в просторі вагів з метою мінімізації сумарної помилки мережі. При цьому помилки (точніше за величину корекції вагів) розповсюджується у зворотному напрямі від входів до виходів, крізь ваги, що сполучають нейрони[5].

Просте застосування одношарової НМ (званою автоасоціативною пам'яттю) полягає в навчанні мережі відновлювати зображення, що подаються. Подаючи на вхід тестове зображення і обчислюючи якість реконструйованого зображення, можна оцінити, наскільки мережа розпізнала вхідне зображення. Позитивні властивості цього методу полягають в тому, що мережа може відновлювати спотворені і зашумленные зображення, але для серйозніших цілей він не підходить [20].

Якщо ця активність нижча за деякий поріг, то вважається, що поданий образ не відноситься ні до одного з відомих класів. Процес навчання встановлює відповідність образів, що подаються на вхід, з приналежністю до певного класу. Це називається навчанням з вчителем. У застосуванні до розпізнавання людини по зображенню особи, такий підхід хороший для завдань контролю доступу невеликої групи осіб. Такий підхід забезпечує безпосереднє порівняння мережею самих образів, але із збільшенням числа класів час навчання і роботи мережі зростає експоненціально. Тому для таких завдань, як пошук схожої людини у великій базі даних, вимагає витягання компактного набору ключових характеристик, на основі яких можна проводити пошук. Підхід до класифікації з використанням частотних характеристик всього зображення, описаний в [7]. Застосовувалася одношарова НМ, заснована на багатозначних нейронах. Відмічене 100% розпізнавання на базі даних *MIT*, але при цьому здійснювалося розпізнавання серед зображень, яким мережа була навчена. Застосовується БНМ для

класифікації зображень осіб на основі таких характеристик, як відстані між деякими специфічними частинами особи (ніс, рот, очі)[13]. В цьому випадку на вхід НС подавалися ці відстані. Використовувалися так само гібридні методи - в першому на вхід НМ подавалися результати обробки прихованою марківською моделлю, а в другому - результат роботи НМ подавався на вхід марківської моделі. У другому випадку переваг не спостерігалося, що говорить про те, що результат класифікації НМ достатній.

Бачимо застосування НМ для класифікації зображень, коли на вхід мережі поступають результати декомпозиції зображення по методу головних компонент. У класичній БНМ міжшарові нейронні з'єднання полносвязны, і зображення представлене у вигляді одновимірного вектора, хоча воно двовимірне. Архітектура згортальної НМ направлена на подолання цих недоліків. У ній використовувалися локальні рецепторні поля (забезпечують локальну двовимірну зв'язність нейронів), загальні ваги (забезпечують детектування деяких рис в будь-якому місці зображення) і ієрархічна організація з просторовими підвибірками (*spatial subsampling*).

Згортальна НМ (ЗНМ) забезпечує часткову стійкість до змін масштабу, зсувів, поворотів, спотворень. Архітектура ЗНМ складається з багатьох шарів, кожний з яких має декілька площин, причому нейрони наступного шару пов'язані тільки з невеликим числом нейронів попереднього шару з околиці локальної області (як в зоровій корі людини)[15]. Ваги в кожній точці однієї площини однакові (згортальні шари). За згортальним шаром слідує шар, що зменшує його розмірність шляхом локального усереднювання. Потім знову згортальний шар, і так далі. Таким чином, досягається ієрархічна організація. Пізніші шари витягують більш загальні характеристики, менше залежні від спотворень зображення. Навчається ЗНМ стандартним методом зворотного розповсюдження помилки.

Порівняння БНМ і ЗНМ показало істотні переваги останньої як за швидкістю, так і по надійності класифікації. Корисною властивістю ЗНМ є і те, що характеристики, що формуються на виходах верхніх шарів ієрархії,

можуть бути застосовні для класифікації по методу найближчого сусіда (наприклад, обчислюючи евклідову відстань), причому ЗНМ може успішно витягувати такі характеристики і для образів, які відсутні в навчальному наборі. Для ЗНМ характерні швидка швидкість навчання і роботи. Тестуванні ЗНМ на базі даних *ORL*, що містить зображення осіб з невеликими змінами освітлення, масштабу, просторових поворотів, положення і різними емоціями, показало приблизно 98% точність розпізнавання, причому для відомих осіб, пред'являлися варіанти їх зображень, отсутствующие в повчальному наборі. Такий результат робить цю архітектуру перспективною для подальших розробок в області розпізнавання зображень просторових об'єктів.

БНМ застосовуються і для виявлення об'єктів певного типу. Крім того, що будь-яка навчена БНМ в деякій мірі може визначати приналежність образів до "своїх" класів, її можна спеціально навчити надійному детектуванню певних класів. В цьому випадку вихідними класами будуть класи що належать і не належать до заданого типу образів[18]. Застосовується нейросетевой детектор для виявлення зображення особи у вхідному зображенні. Зображення сканувалося вікном 20x20 пікселів, яке подавалося на вхід мережі, вирішальної, чи належить дана ділянка до класу осіб. Навчання проводилося як з використанням позитивних прикладів (різних зображень осіб), так і негативних (зображень, що не є особами). Для підвищення надійності детектування використовувався колектив НМ, навчених з різними початковими вагами, унаслідок чого НМ помилялися по різному, а остаточне рішення ухвалювалося голосуванням всього колективу.

НМ застосовується так само для витягання ключових характеристик зображення, які потім використовуються для подальшої класифікації. Тут показаний спосіб нейромережевої реалізації методу аналізу головних компонент. Суть методу аналізу головних компонент полягає в отриманні максимально декорельованих коефіцієнтів, що характеризують вхідні образи. Ці коефіцієнти називаються головними компонентами і використовуються для статистичного стиснення зображень, в якому невелике число коефіцієнтів

використовується для представлення всього образу. НМ з одним прихованим шаром N нейронів (яке багато менше, ніж розмірність зображення), що містить, навчена по методу зворотного розповсюдження помилки відновлювати на виході зображення, подане на вхід, формує на виході прихованих нейронів коефіцієнти перших N головних компонент, які і використовуються для порівняння. Зазвичай використовується від 10 до 200 головних компонент. Із збільшенням номера компоненти її репрезентативність сильно знижується, і використовувати компоненти з великими номерами не має сенсу. При використанні нелінійних активаційних функцій нейронних елементів можлива нелінійна декомпозиція на головні компоненти. Нелінійність дозволяє точніше відобразити варіації вхідних даних. Застосовуючи аналіз головних компонент до декомпозиції зображень осіб, отримаємо головні компоненти, звані власними особами, яким так само властиво корисна властивість, - існують компоненти, які в основному відображають такі істотні характеристики особи як підлога, раса, емоції. При відновленні компоненти мають вигляд, схожий на обличчя, причому перші відображають найбільш загальну форму особи, останні - різні дрібні відмінності між особами. Такий метод добре застосовний для пошуку схожих зображень осіб у великих базах даних. Показана так само можливість подальшого зменшення розмірності головних компонент за допомогою НМ. Оцінюючи якість реконструкції вхідного зображення можна дуже точно визначати його приналежність до класу осіб.

2.6. Висновки до розділу

В данному розділ було розглянуто методи використання нейронних мереж для передачі стилю зображення.

Просте застосування одношарової НМ (званою автоасоціативною пам'яттю) полягає в навчанні мережі відновлювати зображення, що подаються. Подаючи на вхід тестове зображення і обчислюючи якість реконструйованого зображення, можна оцінити, наскільки мережа розпізнала

вхідне зображення. Позитивні властивості цього методу полягають в тому, що мережа може відновлювати спотворені і зашумленные зображення, але для серйозніших цілей він не підходить.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЕРЕНЕСЕННЯ НЕЙРОННОГО СТИЛЮ

3.1. Проектування програмної системи

3.1.1 Метод теорії розпізнавання образу

Теорія розпізнавання образу - розділ інформатики та суміжних дисциплін, що розвиває основи і методи класифікації та ідентифікації предметів, явищ, процесів, сигналів, ситуацій і т.п. Об'єктів, які характеризуються кінцевим набором деяких властивостей і ознак. Такі завдання вирішуються досить часто, наприклад, при переході або проїзді вулиці за сигналами світлофора. Розпізнавання кольору загорілася лампи світлофора і знання правил дорожнього руху дозволяє прийняти правильне рішення про те, чи можна чи не можна переходити вулицю.

Необхідність в такому розпізнаванні виникає в самих різних областях - від військової справи і систем безпеки до оцифровки аналогових сигналів.

Проблема розпізнавання образу придбала видатне значення в умовах інформаційних перевантажень, коли людина не справляється з лінійно-послідовним розумінням надходять до нього повідомлень, в результаті чого його мозок перемикається на режим одночасності сприйняття і мислення, якому властиво таке розпізнавання.

Не випадково, таким чином, проблема розпізнавання образу виявилася в поле міждисциплінарних досліджень - в тому числі в зв'язку з роботою зі створення штучного інтелекту, а створення технічних систем розпізнавання образу привертає до себе все більшу увагу.

3.1.2. Формальна постановка задачі

Можна виділити два основних напрямки :

– вивчення здібностей до розпізнавання, якими володіють живі істоти,

пояснення і моделювання їх;

– розвиток теорії та методів побудови пристроїв, призначених для вирішення окремих завдань в прикладних цілях.

В даній роботі розглядається задача обробки фотографій та переносу певного заданого стилю на початкове зображення. Тобто на підставі наявних двох зображень – початкової фотографії та графічно-стилістичного контенту, який задає стилістику, формується нова стилізована фотографія. Ця стилізована фотографія залишається впізнаваною, але на неї накладаються певні структури, кольори та форми.

При застосуванні різних графічних контентів можна сформувати фото з різною стилістикою – наприклад, у стилі імпресіонізму, абстракціонізму, тощо. Таким чином, робота представляє інтерес як з художньої, так і з дослідницької точки зору. Зокрема, важливою є розробка та реалізація алгоритму обробки зображень на базі математичного апарату штучних нейронних мереж.

Розпізнавання образів - це віднесення вихідних даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних.

При постановці завдань розпізнавання намагаються користуватися математичною мовою, прагнучи - на відміну від теорії штучних нейронних мереж , де основою є отримання результату шляхом експерименту, - замінити експеримент логічними міркуваннями і математичними доказами .

Класична постановка задачі розпізнавання образів: дано безліч об'єктів. Щодо них необхідно провести класифікацію. Безліч представлено підмножинами, які називаються класами. Задані: інформація про класи, опис усієї множини і опис інформації про об'єкт, належність якого до певного класу невідома. Потрібно за наявною інформацією про класи і описі об'єкта встановити - до якого класу належить цей об'єкт.

Найбільш часто в задачах розпізнавання образів розглядаються монохромні зображення, що дає можливість розглядати зображення як функцію на площині. Якщо розглянути точкове безліч на площині T, де функція $f(x, y)$ висловлює в кожній точці зображення його характеристику - яскравість, прозорість, оптичну

щільність, то така функція є формальна запис зображення.

І багато всіх можливих функцій *f (x, y)* на площині *T* є модель безлічі всіх зображень *X*. Вводячи поняття подібності між образами можна поставити задачу розпізнавання. Конкретний вид такої постановки сильно залежить від наступних етапів при розпізнаванні відповідно до тих чи інших підходом.

3.1.3. Деякі методи розпізнавання графічних образів

Для оптичного розпізнавання образів можна застосувати метод перебору виду об'єкта під різними кутами, масштабами, зміщеннями і т.д. Для букв потрібно перебирати шрифт, властивості шрифту і т. д.

Другий підхід - знайти контур об'єкта і досліджувати його властивості (зв'язність, наявність кутів і т. д.)

Ще один підхід - використовувати штучні нейронні мережі. Цей метод вимагає або великої кількості прикладів завдання розпізнавання (з правильними відповідями), або спеціальної структури нейронної мережі, яка враховує специфіку даного завдання.

.3.2 Згорткова нейронна мережа

Згорткова нейронна мережа – структурний вид багатошарової штучної нейронної мережі, яка використовується для ефективної роботи з зображеннями. Метод базується на використанні математичної операції «згортка». До шарів в цій мережі застосовується операція згортки на вході, а результат передається до наступного шару. Така організація роботи мережі імітує реакцію біологічного нейрона на візуальні подразники. Згортка виконується з допоміжною функцією, що задається кінцевими імпульсними характеристиками серії цифрових фільтрів, які послідовно накладаються на вхідний сигнал – зображення, що підлягає обробці. Ці фільтри є матрицями – зазвичай тривимірними наборами значень, що у просторі утворюють квадратну призму. При обробці зображень часто використовуються матриці з висотою та шириною, що дорівнюють 3 або 5.

Глибина матриці відповідає кількості каналів зображення. Три кольорові канали(RGB) означають глибину, що дорівнює трьом.

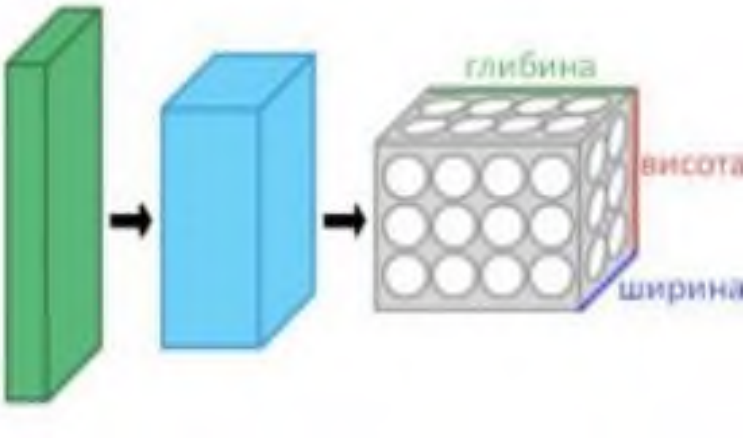


Рис.3.1. Модель згорткової нейронної мережі

3.3. Програмна реалізація нейронної передачі стилю

Нейронна передача стилю – метод формування зображення шляхом об’єднання двох інших зображень. При цьому задається два зображення: картинка-контент та картинка-стиль. Результуюче зображення містить в собі об’єкти контенту та дрібні деталі картинки-стилю. Зазвичай, контент – це фотографія, а стиль – абстрактний малюнок, в результаті одержується ніби намальована фотографія. Вхідні зображення пропускаються через згорткову нейронну мережу, після чого для вихідного зображення вираховуються значення функції втрати контенту та функції втрати стилю.

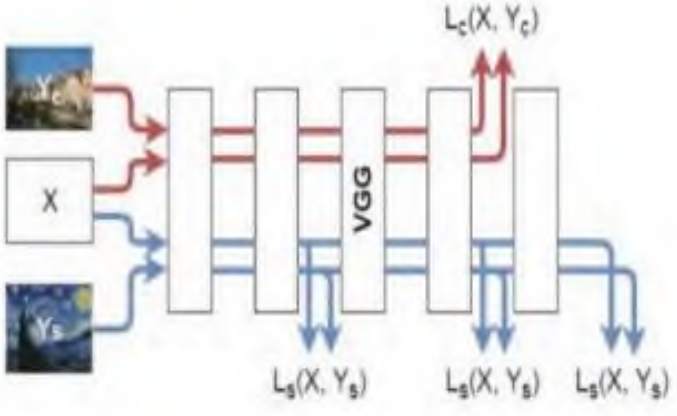


Рис. 3.2 Принцип роботи функцій втрат контенту та стилю

Функція втрати контенту визначається як:

$$L^l_{content}(X, Y_{content}) = \frac{1}{HWC}\sum_{c=1}^{C}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(X^l_{ijc} - Y^l_{ijc}\right)^2,$$

де *Xl, Yl* – ознаки, отримані після обробки зображень *X* та *Y* в l-му шарі нейромережі; *H,W,C* – розмір та кількість каналів відповідно.

Функція втрати контенту демонструє, наскільки дані зображення схожі між собою, причому порівняння здійснюється не безпосередньо (попіксельно),а шляхом визначення та подальшого порівняння високорівневих ознак зображень.

Функція втрати стилю:

$$L^l_{style}(X, Y_{style}) = \sum_{i}^{C}\sum_{j}^{C}\left(G^{X^l}_{ij} - G^{Y^l_{style}}_{ij}\right)^2,$$

де *GA* – матриця Грама для збірки ознак *A*:

$$G^A_{ij} = \frac{1}{HWC}\sum_{x=1}^{W}\sum_{y=1}^{H} a^i_{xy} a^j_{xy},$$

Матриця Грама ілюструє зв’язок між нейронами в різних каналах, а функція втрати стилю обмежує різницю в матрицях та демонструє, наскільки зображення схожі за стилем.

Далі вирішується задача оптимізації:

$$X = \arg\min_x \begin{bmatrix} \alpha \sum\limits_{l \in Q_{content}} L^l_{content}(X, Y_{content}) + \\ \beta \sum\limits_{l \in Q_{content}} L^l_{style}(X, Y_{style}) \end{bmatrix},$$

де *Qcontent* та *Qstyle* – набори номерів тих шарів, в яких потрібно визначити функції втрат, α та β – вага контенту та стилю відповідно (чим більшим є відношення α/β , тим менше стилізованим буде фото, і навпаки).

Були використані конволюційні нейронні мережі які також називаються *CNN*, є основними будівельними блоками, необхідними для передачі нейронного стилю. Такі види глибоких мереж використовуються для виявлення шаблонів /

обрисів зображення і зазвичай використовуються при класифікації та генерації зображень. Нижче наводиться структура базового *CNN*.

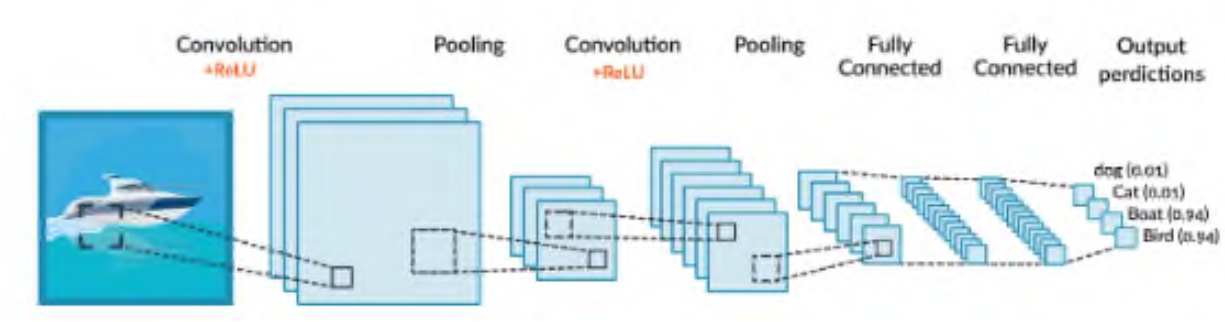


Рис 3.3. Типовий CNN

CNN означає, що відбувається над частинами активацій у шарі, визначеному розміром фільтра (ядра) протягом багатьох шарів, поки ми нарешті не досягнемо нашого вихідного шару, який класифікує наше зображення за наявними входами. Основна ідея - використовувати різні активації, що надаються проміжними шарами CNN, на використанні нашого вмісту та стильових зображень в якості вхідних даних. Ми визначимо дві окремі **функції втрати** , втрату вмісту та втрату стилю, які відстежуватимуть, наскільки «далі» є наш генерований образ, від нашого контентного зображення та образу стилю. Метою нашої Нейронної мережі буде мінімізувати ці функції втрат і, таким чином, покращити сформований імідж з точки зору як змісту, так і стилю.

Постановка задач машинного навчання математично дуже проста. Будь-яке завдання класифікації, регресії або кластеризації - це по суті звичайна оптимізаційна задача з обмеженнями. Незважаючи на це, існує різноманіття алгоритмів і методів їх вирішення робить професію аналітика даних однієї з найбільш творчих IT-професій. Щоб рішення задачі не перетворилося на нескінченний пошук «золотого» рішення, а було прогнозованим процесом, необхідно дотримуватися досить чіткій послідовності дій. Цю послідовність дій описують такі методології, як CRISP-DM.

Методологія аналізу даних CRISP-DM згадується в багатьох постах на Хабре, але я не зміг знайти її докладних російськомовних описів і вирішив своєю статтею заповнити цю прогалину. В основі мого матеріалу -оригінальне

описіадаптоване опис від IBM. Оглядову лекцію про переваги використання CRISP-DM можна подивитися, наприклад,тут.

CRoss Industry Standard Process for Data Mining (CRISP-DM) - стандарт, що описує загальні процеси і підходи до аналітики даних, що використовуються в промислових data-mining проектах незалежно від конкретного завдання і індустрії.

На відомому аналітичному порталі kdnuggets.org періодично публікується опитування (наприклад,тут), Згідно з яким серед методологій аналізу даних перше місце за популярністю регулярно займає саме CRISP-DM, далі з великим відривом йде SEMMA і найрідше використовується KDD Process.

В цілому, ці три методології дуже схожі один на одного (тут складно придумати щось принципово нове). Однак CRISP-DM заслужила популярність як найбільш повна і детальна. У порівнянні з нею KDD є більш загальної та теоретичної, а SEMMA - це просто організація функцій за цільовим призначенням в інструменті SAS Enterprise Miner і зачіпає виключно технічні аспекти моделювання, неможливо торкаючись бізнес-постановки задачі.

Методологія розроблена в 1996 році за ініціативою трьох компаній (нинішні DaimlerChrysler, SPSS і Teradata) і далі допрацьовувалася за участю 200 компаній різних індустрій, які мають досвід data-mining проектів. Всі ці компанії використовували різні аналітичні інструменти, але процес у всіх був побудований дуже схоже.

Методологія активно просувається компанією IBM. Наприклад, вона інтегрована в продукт IBM SPSS Modeler (колишній SPSS Clementine).

Важлива властивість методології - приділення уваги бізнес-цілям компанії. Це дозволяє керівництву сприймати проекти з аналізу даних не як «пісочницю» для експериментів, а як повноцінний елемент бізнес-процесів компанії.

Друга особливість - це досить детальне документування кожного кроку. На думку авторів, добре задокументований процес дозволяє менеджменту краще розуміти суть проекту, а аналітикам - більше впливати на прийняття рішень.

Згідно CRISP-DM, аналітичний проект складається з шести основних етапів, які виконуються послідовно:

- Бізнес-аналіз (Business understanding)
- Аналіз даних (Data understanding)
- Підготовка даних (Data preparation)
- Моделювання (Modeling)
- Оцінка результату (Evaluation)
- Впровадження (Deployment)

Методологія не жорстка. Вона допускає варіацію залежно від конкретного проекту - можна повертатися до попередніх кроків, можна якісь кроки пропускати, якщо для розв'язуваної задачі вони не важливі:

Кожен з цих етапів в свою чергу ділиться на завдання. На виході кожного завдання повинен виходити певний результат. Завдання такі:

В описі кроків я свідомо не буду заглиблюватися в математику і алгоритми, оскільки в статті фокус робиться саме на процес. Припускаю, що читач знайомий з основами машинного навчання, але про всяк випадок в наступному параграфі наводиться опис базових термінів.

Також зверну увагу, що методологія однаково застосовна як для внутрішніх проектів, так і для ситуацій, коли проект робиться консультантами.

Як правило, основним результатом аналітичного проекту є математична модель. Що таке модель?

Нехай у бізнесу є якась цікава йому величина - y (наприклад, ймовірність відтоку клієнта). А також є дані - x (наприклад, звернення клієнта в техпідтримку), від яких може залежати y. Бізнес хоче розуміти, як саме y залежить від x, щоб в подальшому через настройку x він міг впливати на y. Таким чином, завдання проекту - знайти функцію f, яка найкраще моделює досліджувану залежність $y = f(x)$.

Під моделлю ми будемо розуміти формулу f (x) або програму, що реалізовує цю формулу. Будь-яка модель описується, по-перше, своїм алгоритмом навчання (це може бути регресія, дерево рішень, градієнтний бустінг та інше), а по-друге, набором своїх параметрів (які у кожного алгоритму свої). Навчання моделі - процес пошуку таких параметрів, при яких модель найкраще апроксимує спостерігаються дані.

Навчальна вибірка- таблиця, яка містить пари x і y. Рядки в цій таблиці називаються кейсами, а стовпці - атрибутами. Атрибути, які мають достатню предсказательной здатністю, будемо називати предикторами. У випадку з навчанням «без вчителя» (наприклад, в задачах кластеризації), навчальна вибірка складається тільки з x. Скоринг - це застосування знайденої функції f (x) до нових даних, за якими y поки невідомий. Наприклад, в задачі кредитного скорингу спочатку моделюється ймовірність несвоєчасної оплати боргу клієнтом, а потім розроблена модель застосовується до нових заявникам для оцінки їх кредитоспроможності.

Покроковий опис методології

3.3.1. Бізнес-аналіз (Business Understanding)

На першому кроці нам потрібно визначитися з цілями і Скоуп проекту.

Проект має на меті (Business objectives)

Насамперед знайомимося з замовником і намагаємося зрозуміти, що ж він насправді хоче (або розповідаємо йому). На наступні питання добре б отримати відповідь.

- Організаційна структура: хто бере участь в проекті з боку замовника, хто виділяє гроші під проект, хто приймає ключові рішення, хто буде основним користувачем? Збираємо контакти.
- Яка бізнес-мета проекту?

Наприклад, зменшення відтоку клієнтів.

- Чи існують якісь вже розроблені рішення? Якщо існують, то які і чому саме поточне рішення не влаштовує?

Коли разом із замовником розібралися, що ми хочемо, потрібно оцінити, що ми можемо запропонувати з урахуванням поточних реалій.

Оцінюємо, чи вистачає ресурсів для проекту.

- Чи є доступне залізо або його необхідно закуповувати?
- Де і як зберігаються дані, чи буде надано доступ в ці системи, чи потрібно додатково докуповувати / збирати зовнішні дані?

• Чи зможе замовник виділити своїх експертів для консультацій на даний проект?

Потрібно описати ймовірні ризики проекту, а також визначити план дій по їх зменшенню.

Типові ризики наступні.

• Чи не вкластися в терміни.

• Фінансові ризики (наприклад, якщо спонсор втратить зацікавленість в проекті).

• Мала кількість або погана якість даних, які не дозволять отримати ефективну модель.

• Дані якісні, але закономірності в принципі відсутні і, як наслідок, отримані результати не цікаві замовнику.

Важливо, щоб замовник і виконавець говорили на одній мові, тому перед початком проекту краще скласти глосарій і домовитися про використовувану в рамках проекту термінології. Так, якщо ми робимо модель відтоку для телекому, необхідно відразу домовитися, що саме ми будемо вважати відтоком - наприклад, відсутність значних нарахувань за рахунком протягом 4 тижнів поспіль.

Далі варто (хоча б грубо) оцінити ROI. У machine-learning проектах обґрунтовану оцінку окупності часто можна отримати тільки по завершенню проекту (або пілотного моделювання), але розуміння потенційної вигоди може стати хорошим драйвером для всіх.

Після того, як завдання поставлене в бізнес-термінах, необхідно описати її в технічних термінах. Зокрема, відповідаємо на такі питання.

• Яку метрику ми будемо використовувати для оцінки результату моделювання (а вибрати є з чого: Accuracy, RMSE, AUC, Precision, Recall, F-міра, R2, Lift, Logloss і т.д.)?

• Який критерій успішності моделі (наприклад, вважаємо AUC рівний 0.65 - мінімальним порогом, 0.75 - оптимальним)?

• Якщо об'єктивний критерій якості використовувати не будемо, то як будуть оцінюватися результати?

Як тільки отримані відповіді на всі основні питання і ясна мета проекту, час скласти план проекту. План повинен містити оцінку всіх шести фаз впровадження.

3.3.2. Аналіз даних (Data Understanding)

Починаємо реалізацію проекту і для початку дивимося на дані. На цьому кроці ніякого моделювання немає, використовується тільки описова аналітика.

Мета кроку - зрозуміти слабкі і сильні сторони наданих даних, визначити їх достатність, запропонувати ідеї, як їх використовувати, і краще зрозуміти процеси замовника. Для цього ми будуємо графіки, робимо вибірки і розраховуємо статистики.

2.1 Збір даних (Data collection)

Для початку потрібно розуміти, якими дані має замовник. Дані можуть бути:

- власні (1stparty data),
- сторонні дані (3rdparty),
- «Потенційні» дані (для отримання яких необхідно організувати збір).

Необхідно проаналізувати всі джерела, доступ до яких надає замовник. Якщо власних даних недостатньо, можливо, варто закупити сторонні або організувати збір нових даних.

Далі дивимося на доступні нам дані.

- Необхідно описати дані у всіх джерелах (таблиця, ключ, кількість рядків, кількість стовпців, обсяг на диску).
- Якщо обсяг дуже великий для використовуваного ПО, створюємо семпл даних.
- Вважаємо ключові статистики по атрибутам (мінімум, максимум, розкид, кардинальність і т.д.).

За допомогою графіків і таблиць досліджуємо дані, щоб сформулювати гіпотези щодо того, як ці дані допоможуть вирішити задачу.

У міні-звіті фіксуємо, що цікавого знайшли в даних, а також список атрибутів, які потенційно корисні.

Важливо ще до моделювання оцінити якість даних, так як будь-які невідповідності можуть вплинути на хід проекту. Які можуть бути труднощі з даними?

- Пропущені значення.

Наприклад, ми робимо модель класифікації клієнтів банку по їх продуктовим перевагам, але, оскільки анкети заповнюють тільки клієнти-позичальники, атрибут «рівень з / п» у клієнтів-вкладників не заповнений.

- Помилки даних (помилки)
- Неконсістентная кодування значень (наприклад «М» і «male» в різних системах)

3.3.3. Підготовка даних (Data Preparation)

Підготовка даних - це традиційно найбільш витратний за часом етап machine learning проекту (в описі йдеться про 50-70% часу проекту, за нашим досвідом може бути ще більше). Мета етапу - підготувати навчальну вибірку для використання в моделюванні.

Для початку потрібно відібрати дані, які ми будемо використовувати для навчання моделі. Відбираються як атрибути, так і кейси.

Наприклад, якщо ми робимо продуктові рекомендації відвідувачам сайту, ми обмежуємося аналізом тільки від зареєстрованих користувачів.

При виборі даних аналітик відповідає на наступні питання.

Так, електронна пошта або номер телефону клієнта як предиктори для прогнозування явно не приносять користі. А ось домен пошти (mail.ru, gmail.com) або код оператора в теорії вже можуть мати самий корінь здатністю.

- Чи достатньо якісний атрибут для використання в моделі?

Якщо бачимо, що велика частина значень атрибута порожня, то атрибут, швидше за все, не потрібен.

- Чи варто включати корелюють один з одним атрибути?
- Чи є обмеження на використання атрибутів?

Наприклад, політика компанії може забороняти використання атрибутів з персональною інформацією в якості предикторів.

Коли відібрали потенційно цікаві дані, перевіряємо їх якість.

- Пропущені значення => потрібно або їх заповнити, або видалити з розгляду
- Помилки в даних => спробувати виправити вручну або видалити з розгляду
- Невідповідна кодування => привести до єдиної кодуванні

На виході виходить 3 списку атрибутів - якісні атрибути, виправлені атрибути і забраковані.

Часто генерація ознак (feature engineering) - це найбільш важливий етап в підготовці даних: грамотно складений ознака може істотно поліпшити якість моделі.

До генерації даних можна віднести:

- агрегацію атрибутів (розрахунок sum, avg, min, max, var і т.д.),
- генерацію кейсів (наприклад, oversampling або алгоритмSMOTE),
- конвертацію типів даних для використання в різних моделях (наприклад,SVMтрадиційно працює з інтервальними даними, аCHAIDз номінальними),
- нормалізацію атрибутів (feature scaling),
- заповнення пропущених даних (missing data imputation).

Добре, коли дані беруться з корпоративного сховища (КХД) або заздалегідь підготовленої вітрини. Однак часто дані необхідно завантажувати з декількох джерел і для підготовки навчальної вибірки потрібно їх інтеграція. Під інтеграцією розуміється як «горизонтальне» з'єднання (Merge), так і «вертикальне» об'єднання (Append), а також агрегація даних. На виході, як правило, маємо єдину аналітичну таблицю, придатну для поставки в аналітичне ПЗ в якості навчальної вибірки.

Нарешті, потрібно навести дані до формату, придатного для моделювання (тільки для тих алгоритмів, які працюють з певним форматом даних). Так, якщо

мова йде про аналіз тимчасового ряду - наприклад, прогнозуємо щомісячні продажі торгової мережі - можливо, його потрібно попередньо відсортувати.

3.3.4. Моделювання (Modeling)

На четвертому кроці нарешті починається найцікавіше - навчання моделей. Як правило, воно виконується ітераційно - ми пробуємо різні моделі, порівнюємо їх якість, робимо перебір гіперпараметров і вибираємо кращу комбінацію. Це найбільш приємний етап проекту.

Необхідно визначитися, які моделі будемо використовувати (благо, їх безліч). Вибір моделі залежить від розв'язуваної задачі, типів атрибутів і вимог по складності (наприклад, якщо модель буде далі впроваджуватися в Excel, то RandomForest і XGBoost явно не підійдуть). При виборі слід звернути увагу на наступне.

- Чи достатньо даних, оскільки складні моделі як правило вимагають більшої вибірки?
- Чи зможе модель обробити пропуски даних (якісь реалізації алгоритмів вміють працювати з пропусками, якісь немає)?
- Чи зможе модель працювати з наявними типами даних або необхідна конвертація?

Далі треба вирішити, на чому ми будемо навчати, а на чому тестувати нашу модель.

Традиційний підхід - це поділ вибірки на 3 частини (навчання, валідацію і тест) в приблизною пропорції 60/20/20. В цьому випадку навчальна вибірка використовується для підгонки параметрів моделі, а валідація і тест для отримання очищеної від ефектуперенавчанняоцінки її якості. Більш складні стратегії передбачають використання різних варіантівкрос-валідації.

Тут же прикидаємо, як будемо робити оптимізацію гіперпараметров моделей - скільки буде ітерацій по кожному алгоритму, чи будемо робитигrid-searchабоrandom-search.

Запускаємо цикл навчання і після кожної ітерації фіксуємо результат. На виході отримуємо кілька навчених моделей.

Крім того, для кожної навченої моделі фіксуємо наступне.

- Чи показує модель якісь цікаві закономірності?

Наприклад, що точність передбачення на 99% пояснюється лише одним атрибутом.

- Яка швидкість навчання / застосування моделі?

Якщо модель навчається 2 дня, можливо, варто пошукати більш ефективний алгоритм або зменшити навчальну вибірку.

- Чи були проблеми з якістю даних?

Наприклад, в тестову вибірку потрапили кейси з пропущеними значеннями, і через це не вся вибірка проскорілась.

Після того, як був сформований пул моделей, потрібно їх ще раз детально проаналізувати і вибрати моделі-переможці. На виході непогано мати список моделей, відсортоване за об'єктивним і / або суб'єктивним критерієм.

Завдання кроку:

- провести технічний аналіз якості моделі (ROC, Gain, Lift і т.д.),
- оцінити, чи готова модель до впровадження в КХД (або куди потрібно),
- досягаються задані критерії якості,
- оцінити результати з точки зору досягнення бізнес-цілей. Це можна обговорити з аналітиками замовника.

Якщо критерій успіху не досягнуто, то можна або покращувати поточну модель, або пробувати нову.

Перш ніж переходити до впровадження потрібно переконатися, що:

- результат моделювання зрозумілий (модель, атрибути, точність)
- результат моделювання логічний

Наприклад, ми прогнозуємо відтік клієнтів і отримали ROC AUC, рівний 95%. Занадто хороший результат - привід перевірити модель ще раз.

- ми спробували всі доступні моделі
- інфраструктура готова до впровадження моделі

Замовник: «Давайте впроваджувати! Тільки у нас місця немає у вітрині ... ».

3.3.5. Оцінка результату (Evaluation)

Результатом попереднього кроку є побудована математична модель (model), а також знайдені закономірності (findings). На п'ятому кроці ми оцінюємо результати проекту.

Якщо на попередньому етапі ми оцінювали результати моделювання з технічної точки зору, то тут ми оцінюємо результати з точки зору досягнення бізнес-цілей.

Адресуємо наступні питання:

- Формулювання результату в бізнес-термінах. Бізнесу набагато легше спілкуватися в термінах $ і ROI, ніж в абстрактних Lift або R2

Класичний приклад діалогу

Аналітик: Наша модель показує десятикратний lift! Бізнес: Я не вражений ... Аналітик: Ви заробите додаткових 100K $ в рік! Бізнес: З цього треба було починати! Детальніше, будь ласка ...

- В цілому наскільки добре отримані результати вирішують бізнес-завдання?
- Знайдена чи якась нова цінна інформація, яку варто виділити окремо?

Наприклад, компанія-рітейлер сфокусувала свої маркетингові зусилля на сегменті «активна молодь», але, зайнявшись прогнозуванням ймовірності відгуку, з подивом виявила, що їх цільової сегмент зовсім інший - «забезпечені жінки 40+».

Варто зібратися за кухлем пива за столом, проаналізувати хід проекту і сформулювати його сильні і слабкі сторони. Для цього потрібно пройтися по всіх кроків:

- Чи можна було якісь кроки зробити більш ефективними?

Наприклад, через неповороткість IT-відділу замовника цілий місяць пішов на узгодження доступів. Чи не гуд!

- Які були допущені помилки і як їх уникнути в майбутньому?

На етапі планування недооцінили складність вивантаження даних з джерел і в результаті не вклалися в терміни.

• Були ледве не спрацювали гіпотези? Якщо так, чи варто їх повторювати?

Аналітик: «А давайте тепер спробуємо сверточное нейронну мережу ... Все стає краще з нейросетями!»

• Чи були несподіванки при реалізації кроків? Як їх передбачити в майбутньому?

Замовник: «Ok. А ми думали, що навчальна вибірка для розробки моделі не потрібна ... »

Далі потрібно або впроваджувати модель, якщо вона влаштовує замовника, або, якщо видно потенціал для поліпшення, спробувати ще її поліпшити.

Якщо на даному етапі у нас кілька задовольняють моделей, то відбираємо ті, які будемо далі впроваджувати.

3.3.6. Впровадження (Deployment)

Перед початком проекту з замовником завжди обмовляється спосіб поставки моделі. В одному випадку це може бути просто проскоренная база клієнтів, в іншому - SQL-формула, в третьому - повністю відпрацьований аналітичне рішення, інтегроване в інформаційну систему.

На даному етапі здійснюється впровадження моделі (якщо проект передбачає етап впровадження). Причому під впровадженням може розумітися як фізичне додавання функціоналу, так і ініціювання змін в бізнес-процесах компанії.

Нарешті зібрали в купу всі отримані результати. Що тепер?

• Важливо зафіксувати, що саме і в якому вигляді ми будемо впроваджувати, а також підготувати технічний план впровадження (паролі, явки та інше)

• Продумати, як з впроваджуваної моделлю працюватимуть користувачі

Наприклад, на екрані співробітника колл-центру показуємо схильність клієнта до підключення додаткових послуг.

• Визначити принцип моніторингу рішення. Якщо потрібно, підготуватися до дослідно-промислової експлуатації.

Наприклад, домовляємося про використання моделі протягом року і тюнінгу моделі раз в 3 місяці.

Дуже часто в проект включаються роботи з підтримки рішення. Ось що обмовляється.

- Які показники якості моделі будуть відслідковуватися?

У своїх банківських проектах ми часто використовуємо популярний в банках показникpopulation stability indexPSI.

- Як розуміємо, що модель застаріла?

Наприклад, якщо PSI більше 0.15, або просто домовляємося про регулярне перерахунку раз в 3 місяці.

- Якщо модель застаріла, чи достатньо буде її перенавчити або потрібно організовувати новий проект?

При істотних змінах в бізнес-процесах тюнінгу моделі недостатньо, потрібен повний цикл перенавчання - з додаванням нових атрибутів, відбором предикторов і.т.д.

6.3 Звіт за результатами моделювання (Final Report)

Після закінчення проекту, як правило, пишеться звіт про результати моделювання, в який додаються результати по кожному кроці, починаючи від первинного аналізу даних і закінчуючи впровадженням моделі. У цей звіт також можна включити рекомендації щодо подальшого розвитку моделі.

Написаний звіт презентується замовнику і всім зацікавленим особам. За відсутності ТЗ цей звіт є головним документом проекту. Також важливо поговорити з задіяними в проекті співробітниками (як з боку замовника, так і з боку виконавця) і зібрати їх думку про проект.

Важливо розуміти, що методологія не є універсальним рецептом. Це просто спроба формально описати послідовність дій, яку в тій чи іншій мірі виконує будь-який аналітик, який займається аналізом даних.

У нас вCleverDATАслідування методології на дата-майнінгових проектах не є жорсткою вимогою, але, як правило, при складанні плану проекту наша деталізація досить точно укладається в цю послідовність кроків.

Методологія може бути застосована до абсолютно різних завдань. Ми слідували їй в ряді маркетингових проектів, в тому числі, коли передбачали можливість відгуку клієнта торгової мережі на рекламну пропозицію, робили модель оцінки кредитоспроможності позичальника для комерційного банку і розробляли сервіс рекомендацій товарів для інтернет-магазину.

Після кожного кроку повинен писатися якийсь звіт. Однак на практиці це не дуже реалістично. Як і у всіх, у нас бувають проекти, коли замовник ставить дуже стислі терміни і необхідно швидко отримати результат. Зрозуміло, що в таких умовах немає сенсу витрачати час на детальне документування кожного кроку. Всю проміжну інформацію, якщо вона потрібна, ми в таких випадках фіксуємо олівцем «на серветці». Це дозволяє максимально швидко зайнятися реалізацією моделі і вкластися в терміни.

На практиці багато речей робляться куди менш формально, ніж вимагає методологія. Ми, наприклад, зазвичай не витрачаємо час на вибір і узгодження використовуваних моделей, а тестуємо відразу всі доступні алгоритми (звичайно, якщо ресурси дозволяють). Аналогічно робимо з атрибутами - готуємо відразу кілька варіантів кожного атрибута, щоб можна було випробувати максимальну кількість варіантів. Нерелевантні атрибути при такому підході відсіваються автоматично за допомогою алгоритмівfeature selection- автоматичному визначенні предсказательной здатності атрибутів.

Вважаю, формалізм методології пояснюється тим, що вона писалася ще в 90-е, коли не було такої кількості обчислювальних потужностей і важливо було грамотно спланувати кожну дію. Зараз доступність і дешевизна «заліза» спрощує багато речей.

Використовується *TensorFlow* в якості нашої основи глибокого навчання. Кілька інших бібліотек імпортуються для включення таких утиліт, як збереження, імпорт, зміна розміру та показ зображення.

Нейронний стиль передачі - це технологія оптимізації, яка використовується для зйомки двох зображень - контентного зображення та стильового зображення (наприклад, художнього твору відомого художника) - і

поєднує їх разом, щоб вихідне зображення виглядало як вміст зображення, але "намальовано" у стилі еталонного зображення стилю.

Це реалізується шляхом оптимізації вихідного зображення для відповідності статистиці вмісту зображення вмісту та статистиці стилю еталонного зображення стилю. Ці статистичні дані витягуються із зображень за допомогою згорткової мережі.

Наприклад, береться зображення цієї собаки (рис. 3.4).

Рис. 3.4. Контенте зображення

Тепер як би виглядало, як би картина цього собаки була намальована виключно еталонним зображенням стилю.

Рис 3.5. Стильове зображення

3.4. Реалізація імпорту та налаштування модулів

```
import tensorflow as image_ttf
import IPython.display as display
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12,12)
mpl.rcParams['axes.grid'] = False
import nn_my as np
import PIL.Image
import time
import functools
def tensor_to_image(tensor):
  tensor = tensor*255
  tensor = np.array(tensor, dtype=np.uint8)
  if np.ndim(tensor)>3:
    assert tensor.shape[0] == 1
```

```
    tensor = tensor[0]
  return PIL.Image.fromarray(tensor)
```

3.5. Візуалізація введення:

Визначення функції для завантаження зображення та обмежте його максимальний розмір до 512 пікселів.

```
def load_img(path_to_img):
  max_dim = 512
  img = image_ttf.io.read_file(path_to_img)
  img = image_ttf.image.decode_image(img, channels=3)
  img = image_ttf.image.convert_image_dtype(img, image_ttf.float32)
  shape = image_ttf.cast(image_ttf.shape(img)[:-1], image_ttf.float32)
  long_dim = max(shape)
  scale = max_dim / long_dim
  new_shape = image_ttf.cast(shape * scale, image_ttf.int32)
  img = image_ttf.image.resize(img, new_shape)
  img = img[image_ttf.newaxis, :]
  return img
```

Створення простої функції для відображення зображення:

```
def imshow(image, title=None):
  if len(image.shape) > 3:
    image = image_ttf.squeeze(image, axis=0)
  plt.imshow(image)
  if title:
    plt.title(title)
content_image = load_img(content_path)
style_image = load_img(style_path)
plt.subplot(1, 2, 1)
imshow(content_image, 'Content Image')
plt.subplot(1, 2, 2)
```

imshow(style_image, 'Style Image')

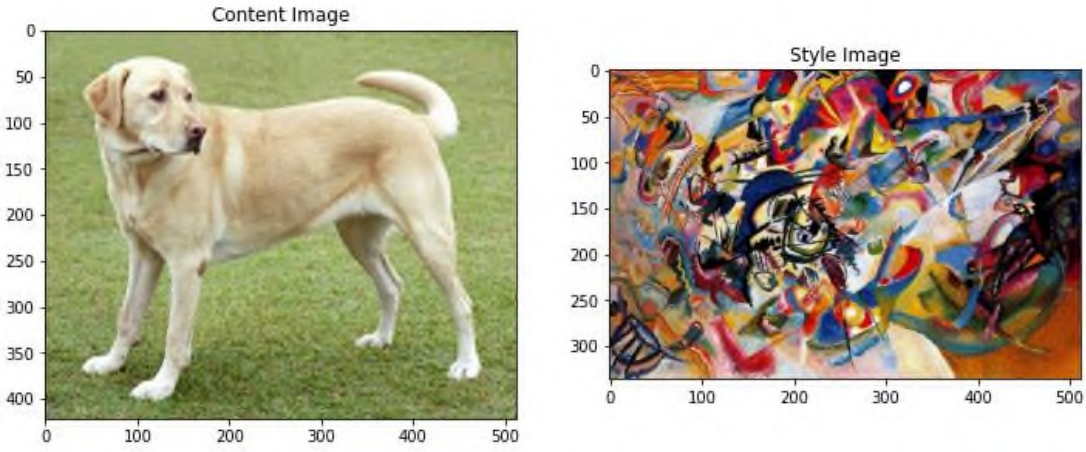


Рис.3.6. Контенте зображення/ Стильове зображення

Швидка передача стилю за допомогою *IMAGE_TTF-Hub*

import tensorflow_hub as hub

hub_module=hub.load('https://image_ttfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1')

stylized_image=hub_module(image_ttf.constant(content_image), image_ttf.constant(style_image))[0]

tensor_to_image(stylized_image)

Рис. 3.7. Стильозоване зображення

3.6. Визначення вмісту та стильового уявлення

Використовується проміжні шари моделі, щоб отримати зміст і стильові зображення. Починаючи з вхідного шару мережі, перші кілька активацій шару представляють функції низького рівня, такі як краї та текстури. Останні кілька шарів представляють особливості вищого рівня - деталі предмета, такі як колеса або очі . У цьому випадку використовується мережева архітектуа *VGG19* - мережу класифікації зображень, що перевіряється. Ці проміжні шари необхідні для визначення подання змісту та стилю із зображень. Для вхідного зображення відповідає відповідним поданням стилю та вмісту на цих проміжних шарах.

Завантажимо *VGG19* і протестуємо його на нашому зображенні, щоб переконатися, що воно правильно використовується:

*x = image_ttf.keras.applications.vgg19.preprocess_input(content_image*255)*

x = image_ttf.image.resize(x, (224, 224))

vgg = image_ttf.keras.applications.VGG19(include_top=True, weights='imagenet')

prediction_probabilities = vgg(x)

prediction_probabilities.shape

predicted_top_5 = image_ttf.keras.applications.vgg19.decode_predictions(prediction_probabilities.nn_my())[0]

[(class_name, prob) for (number, class_name, prob) in predicted_top_5]

Тепер завантажуються *VGG19* без класифікаційної голови та перелічіть назви шарів

vgg = image_ttf.keras.applications.VGG19(include_top=False, weights='imagenet')

print()

for layer in vgg.layers:

print(layer.name)

content_layers = ['block5_conv2']

style_layers = ['block1_conv1',

'block2_conv1',

'block3_conv1',

'block4_conv1',

'block5_conv1']

num_content_layers = len(content_layers)

num_style_layers = len(style_layers)

3.7. Реалізація проміжних шарів для стилю та змісту

На високому рівні, щоб мережа здійснювала класифікацію зображень (що ця мережа навчена робити), вона повинна розуміти зображення. Для цього потрібно взяти необроблене зображення як вхідні пікселі та побудувати внутрішнє подання, яке перетворює пікселі необробленого зображення в складне розуміння особливостей, наявних у зображенні.

Це також є причиною того, що конволюційні нейронні мережі здатні добре узагальнити: вони здатні фіксувати інваріації та визначати особливості в класах (наприклад, кішки проти собак), що спричиняють фоновий шум та інші неприємності. Таким чином, десь між тим, де сире зображення подається в модель та етикеткою класифікації виходу, модель виступає як екстрактор складних функцій. Доступ до проміжних шарів моделі, можна описати зміст та стиль вхідних зображень.

Побудува моделі

Мережі *image_ttf.keras.applications* розроблені таким чином, що ви можете легко отримати значення проміжних рівнів за допомогою функціонального *API Keras.*

Щоб визначити модель за допомогою функціонального *API*, вкажіть входи та виходи:

model = Model(enter_my, outputs)

Ця функція будує модель *VGG19*, яка повертає список виходів проміжного рівня:

def vgg_layers(layer_names):

""" Creates a vgg model that returns a list of intermediate output values."""

 # Load our model. Load pretrained VGG, trained on imagenet data

vgg = image_ttf.keras.applications.VGG19(include_top=False, weights='imagenet')

vgg.trainable = False

outputs = [vgg.get_layer(name).output for name in layer_names]

model = image_ttf.keras.Model([vgg.input], outputs)

return model

І щоб створити модель:

style_extractor = vgg_layers(style_layers)

*style_outputs = style_extractor(style_image*255)*

#Look at the statistics of each layer's output

for name, output in zip(style_layers, style_outputs):

print(name)

print("shape: ", output.nn_my().shape)

print("min: ", output.nn_my().min())

print("max: ", output.nn_my().max())

print(" mean: ", output.nn_my().mean())

print()

Обчислення стилю

Зміст зображення представлений значеннями карт проміжних ознак.

Виявляється, стиль зображення можна описати засобами та співвідношеннями різних карт .Обчислюється матриця Грама, що включає цю інформацію, взявши зовнішній продукт векторного ознаки з собою в кожному місці та усереднюючи цей зовнішній продукт по всіх місцях. Ця матриця Грама для певного шару може бути обчислена як:

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

Це може бути реалізовано стисло за допомогою *image_ttf.linalg.einsum* функції:

```
def gram_matrix(input_tensor):
  result = image_ttf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
  input_shape = image_ttf.shape(input_tensor)
  num_locations = image_ttf.cast(input_shape[1]*input_shape[2], image_ttf.float32)
  return result/(num_locations)
```

Витяг стилю та змісту

Будується модель, яка повертає тензори стилів та вмісту.

```
class StyleContentModel(image_ttf.keras.models.Model):
  def __init__(self, style_layers, content_layers):
    super(StyleContentModel, self).__init__()
    self.vgg =  vgg_layers(style_layers + content_layers)
    self.style_layers = style_layers
    self.content_layers = content_layers
    self.num_style_layers = len(style_layers)
    self.vgg.trainable = False

  def call(self, enter_my):
    "Expects float input in [0,1]"
    enter_my = enter_my*255.0
    preprocessed_input = image_ttf.keras.applications.vgg19.preprocess_input(enter_my)
    outputs = self.vgg(preprocessed_input)
    style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                      outputs[self.num_style_layers:])

    style_outputs = [gram_matrix(style_output)
                     for style_output in style_outputs]
```

```
    content_dict = {content_name:value
                    for content_name, value
                    in zip(self.content_layers, content_outputs)}

    style_dict = {style_name:value
                  for style_name, value
                  in zip(self.style_layers, style_outputs)}

    return {'content':content_dict, 'style':style_dict}
```

Закликаючи зображення, ця модель повертає грам матрицю (стиль) *style_layers* та вмісту *content_layers*:

```
extractor = StyleContentModel(style_layers, content_layers)
results = extractor(image_ttf.constant(content_image))
print('Styles:')
for name, output in sorted(results['style'].items()):
  print("  ", name)
  print("    shape: ", output.nn_my().shape)
  print("    min: ", output.nn_my().min())
  print("    max: ", output.nn_my().max())
  print("    mean: ", output.nn_my().mean())
  print()
print("Contents:")
for name, output in sorted(results['content'].items()):
  print("  ", name)
  print("    shape: ", output.nn_my().shape)
  print("    min: ", output.nn_my().min())
  print("    max: ", output.nn_my().max())
  print("    mean: ", output.nn_my().mean())
```

Виконується градієнтний спуск

За допомогою цього інструменту для вилучення стилів та вмісту тепер можна реалізувати алгоритм передачі стилів. Обчисливши середню квадратичну помилку для виведення зображення щодо кожної цілі, а потім зважену суму цих втрат.

Встановлюються цільові значення для стилю та вмісту:

style_targets = extractor(style_image)['style']

content_targets = extractor(content_image)['content']

Визначимо, *image_ttf.Variable* щоб містити зображення для оптимізації. Щоб зробити це швидко, ініціалізуємо його із зображенням вмісту (воно *image_ttf.Variable* має бути такої ж форми, як зображення вмісту):

image = image_ttf.Variable(content_image)

Визначається функція для збереження значень пікселів між 0 і 1:

def clip_0_1(image):

return image_ttf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)

Створюється оптимізатор.

opt = image_ttf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)

Щоб оптимізувати це, використовується зважена комбінація двох втрат, щоб отримати загальний збиток:

style_weight=1e-2

content_weight=1e4

def style_content_loss(outputs):

style_outputs = outputs['style']

content_outputs = outputs['content']

*style_loss = image_ttf.add_n([image_ttf.reduce_mean((style_outputs[name]-style_targets[name])**2)*

for name in style_outputs.keys()])

*style_loss *= style_weight / num_style_layers*

```
        content_loss =
image_ttf.add_n([image_ttf.reduce_mean((content_outputs[name]-
content_targets[name])**2)
                         for name in content_outputs.keys()])
        content_loss *= content_weight / num_content_layers
        loss = style_loss + content_loss
        return loss
```

Використовується *image_ttf.GradientTape* для оновлення зображення.

```
@image_ttf.function()
def train_step(image):
  with image_ttf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)
  grad = tape.gradient(loss, image)
  opt.apply_gradients([(grad, image)])
  image.assign(clip_0_1(image))
```

Виконується кілька кроків для перевірки:

```
train_step(image)
train_step(image)
train_step(image)
tensor_to_image(image)
```

Рис 3.8. Оптимізоване зображення

Виконається більш довга оптимізація

```
import time
start = time.time()
epochs = 10
steps_per_epoch = 100
step = 0
for n in range(epochs):
  for m in range(steps_per_epoch):
    step += 1
    train_step(image)
    print(".", end='')
  display.clear_output(wait=True)
  display.display(tensor_to_image(image))
  print("Train step: {}".format(step))
end = time.time()
print("Total time: {:.1f}".format(end-start))
```

Рис. 3.9. Початкова оптимізація зображення

3.8. Реалізація загальної втрати варіації

Недоліком цієї основної реалізації є те, що вона створює багато високочастотних артефактів. Їх можна зменшити за допомогою явного терміна регуляризації на високочастотних компонентах зображення. У передачі стилів це часто називають загальною втратою варіації :

```
def high_pass_x_y(image):
  x_var = image[:,:,1:,:] - image[:,:,:-1,:]
  y_var = image[:,1:,:,:] - image[:,:-1,:,:]
  return x_var, y_var
x_deltas, y_deltas = high_pass_x_y(content_image)
plt.figure(figsize=(14,10))
plt.subplot(2,2,1)
imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Original")
plt.subplot(2,2,2)
imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Original")
x_deltas, y_deltas = high_pass_x_y(image)
plt.subplot(2,2,3)
imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Styled")
```

plt.subplot(2,2,4)

*imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Styled")*

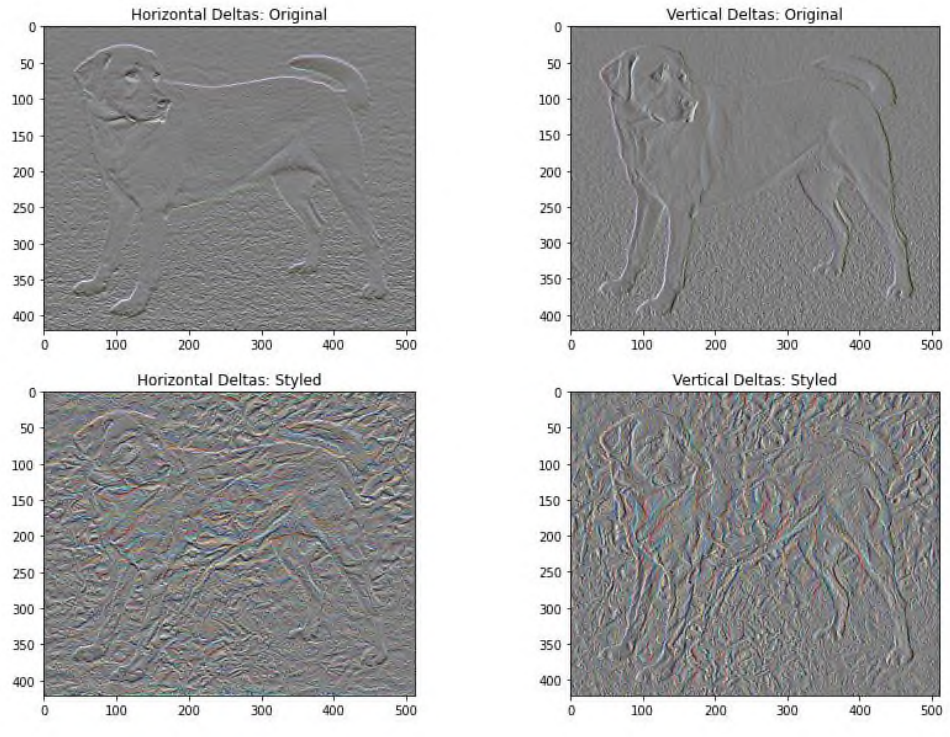


Рис. 3.10. Високочастотні компоненти

Приклади обробки зображень на рис. 3.11.

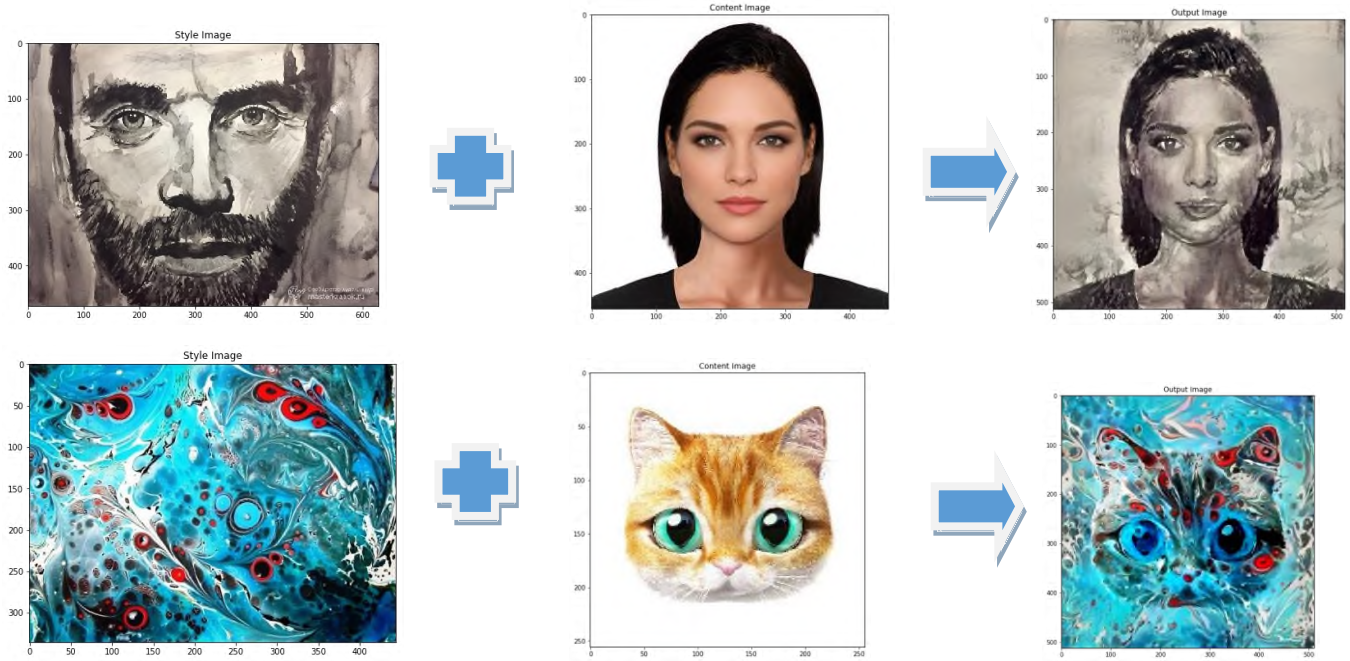


Рис. 3.11. Приклади перенесення стилю

3.9. Висновки до розділу

У образотворчому мистецтві, особливо живописі, люди оволоділи майстерністю створювати унікальні візуальні переживання шляхом складання складної взаємодії між змістом та стилем зображення. Поки що алгоритмічна основа цього процесу невідома, і не існує штучної системи з подібними можливостями. Однак в інших ключових областях візуального сприйняття, таких як розпізнавання предметів і облич, що знаходяться поблизу людини, нещодавно було продемонстровано класом біологічно натхненних моделей зору під назвою Deep Neural Networks. Тут ми впроваджуємо штучну систему на основі глибокої нейронної мережі, яка створює художні образи високої перцептивної якості. Система використовує нейронні уявлення для розділення та рекомбінації змісту та стилю довільних зображень, забезпечуючи нейронний алгоритм створення художніх образів.

ВИСНОВКИ

Останнім часом спостерігається розвиток нейронних мереж, а разом з ними і всебічне їх використання. Процес розпізнавання зображень є складною багатоетапною процедурою. Багатоетапність обумовлена тим, що різні завдання обробки насправді тісно зв'язані і якість вирішення однієї з них впливає на вибір методу вирішення інших. Так вибір методу розпізнавання залежить від конкретних умов представлення вхідних зображень, зокрема характеру фону, шумової обстановки і пов'язаний з вибором методів попередньої обробки, сегментації, фільтрації.

У даній дипломній роботі була розглянута тематика накладання стилю на зображення. В результаті виконання дипломної роботи був створений комплекс формування нових стилів зображення. Крім того набув навичок роботи з професійним програмним забезпеченням даного виду, закріпив і поглибив теоретичні і практичні знання в галузі програмування

Було розглянуто поняття нейронної мережі, всі її види та характеристики.

Також було розглянуто програми, які забезпечують роботу з зображеннями. Ознайомлено з методом скелетизації, котрий при розпізнаванні зображень реалізується впровадженням процесору до зняття шару за шаром з літери для покращення точності розпізнавання і не тільки, це дає змогу розпізнавати складні рисунки.

В третьому розділі було описано процес розробки програмного модулю накладання стилю. Була розроблена програма для створення вибірки для нейронної мережі, та модулю по групуванню, навчанню та розпізнаванню.

Було детально розглянуто інтерфейс розробленого програмного модулю для створення вибірки стилів. Він дуже простий та практичний, це надає програмі великі переваги: легкість керування, збору вибірки та інтуїтивний інтерфейс. Велику роль грає швидкість роботи нейронної мережі. Це дозволяє користувачу швидше проаналізувати рисунок з якого потрібно зробити розпізнавання.

Було розкрито питання, щодо застосування нейронних мереж у сферах діяльності зв’язаних з творчістю. Нейронні мережі стрімко розвиваються в безлічі інших областей, які прагнуть скоротити час опрацювання того чи іншого процесу, підвищити ефективність.

Також були наведені основні фрагменти коду програмної частини які відповідають за організацію роботи обчислень в програмі, словесно описані алгоритми роботи створеного засобу для розпізнавання тексту.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ