КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

«Інформаційна технологія обробки та трансформації зображень картин з використанням нейронних мереж»

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:
студент 2 курсу магістратури, групи ТШІ-21

Березовський Владислав Юрійович
(ПІБ)

Науковий керівник: Гайна Георгій Анатолійович
(ПІБ)
кандидат технічних наук, професор
(науковий ступінь, вчене звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____
підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № ____ від «____» _____ 20__ р.

Зав. кафедри _____ ПІБ
підпис

Київ 2022

РЕФЕРАТ

Робота містить 76 сторінок тексту, 26 рисунків, 2 таблиці, 36 посилань на літературні джерела та 1 додаток.

Актуальність. З середини 1990-х років теорії мистецтва, які стоять за привабливими творами мистецтва, привертають увагу не лише художників, а й багатьох дослідників інформатики. Існує багато досліджень і методів, спрямованих на автоматичне перетворення зображення на синтетичні твори мистецтва. Серед цих досліджень надихають досягнення в області нефотореалістичної візуалізації. Однак, більшість алгоритмів НФВ розроблена для певних художніх стилів і не може бути легко поширена на інші стилі.

У 2015 році вперше було використано ЗНМ для відтворення відомих стилів живопису на природних зображеннях. Хоча з того часу з'явилось багато нових досягнення в області нейронного перенесення стилю, і деякі з них навіть знайшли промислове застосуваннях, проте, все ще існують невирішені проблеми в роботі алгоритмів НПС, а також в їх ефективності та доступності.

До можливих застосувань НПС можна віднести імітацію або створення робіт мистецтва, дизайн та споріднені напрями. Застосування НПС в соціальній комунікації зміцнює зв'язки між людьми, а також має позитивний вплив як на наукові кола, так і на промисловість. Інше використання НПС полягає в тому, щоб змусити його діяти як інструмент для створення мистецтва користувачем.

Об'єкт дослідження – процес перенесення стилю одного зображення на інший без втрати змісту.

Предмет дослідження – метод перенесення стилю з використанням згорткових нейронних мереж.

Мета дослідження – розробити інформаційну технологію перенесення стилю з одного зображення на інше з використанням нейронних мереж.

Новизна отриманих результатів. Запропоновано та реалізовано інформаційну технологію перенесення стилю зображень з використанням нейронних мереж мовою Python, а також надані рекомендації стосовно використання нашої реалізації на основі проведених експериментів.

Структура та обсяг роботи. Кваліфікаційна робота на здобуття освітнього ступеня «магістр» складається з вступу, трьох розділів та висновків.

У *вступі* до магістерської роботи надано загальну характеристику роботи, визначено актуальність теми досліджень, сформульовано мету, завдання та методи досліджень.

У *першому* розділі проаналізовано сучасні відомості про стан питання, що досліджується, обґрунтовано актуальність досліджень, розглянуто існуючі рішення поставленої задачі, в тому числі на основі нейронних мереж, та проведено їх порівняльний аналіз.

У *другому* розділі розглянуто та проаналізовано методи та складові алгоритму перенесення стилю з використанням нейронних мереж. На основі аналізу обрано основу для реалізації інформаційної технології.

У *третьому* розділі обґрунтовано вибір методів розробки, а також алгоритм програми. Описано його структуру та принцип роботи. Проведено експерименти з використанням програмного застосунку та зроблено аналіз отриманих результатів.

У *висновках* представлені підсумки проведених досліджень.

Ключові слова: перенесення стилю, перенесення текстури, нейронне перенесення стилю, перенесення стилю на основі нейронних мереж, згорткові нейронні мережі, VGG19, матриця Грама, обробка зображень, трансформація зображень.

ABSTRACT

The qualification work contains 76 pages of text, 26 figures, 2 tables, 36 references and 1 application.

Theme relevance. Since the mid-1990s, the theories of art behind attractive works of art have attracted the attention not only of artists but also of many computer scientists. There are many studies and methods aimed at automatically converting images into synthetic works of art. Among these studies are inspiring advances in non-photorealistic rendering. However, most NPR algorithms are designed for certain art styles and cannot be easily applied to other styles.

In 2015, for the first time, the CNNs were used to reproduce well-known styles of painting in natural images. Even though since then many advances in neural style transfer have appeared, and some of them have already found industrial applications, however, there are still unresolved issues in the operation of NST algorithms, as well as in their efficiency and availability.

Possible applications of NST include imitation or creation of art works, design and related areas. The use of NST in social communication strengthens the bonds between people and has a positive impact on both academia and industry. Another use of NST is to make it act as an art tool for a user.

The object of research is the process of transferring the style of one image to another without losing content.

The subject of research is a method of style transfer using convolutional neural networks.

The purpose of the qualification work is to develop information technology for transferring style from one image to another using neural networks.

Scientific novelty of the obtained results. The information technology of image style transfer using neural networks written with Python is offered and implemented, as well as recommendations on the use of this implementation based on the conducted experiments are given.

Structure and scope of work. The qualifying work for the master's degree consists of an introduction, three sections and conclusions.

In *the introduction* the general characteristic of the work is given, the relevance of the research theme is determined, the purpose, tasks and methods of research are formulated.

The first section analyzes current information about the state of the research issue, substantiates the relevance of research, considers existing solutions to this problem, including on the basis of neural networks, and gives a comparative analysis to these solutions.

The second section considers and analyzes the methods and components of the style transfer algorithm using neural networks. Based on the analysis, the basis for the implementation of information technology has been chosen.

The third section substantiates the choice of development methods, as well as the architecture of the algorithm. Its structure and principle of operation are described. Experiments with the use of software application have been performed and the results have been analyzed.

The conclusions present the results of the research.

Keywords: style transfer, texture transfer, neural style transfer, neural network-based style transfer, convolutional neural networks, VGG19, Gram matrix, image processing, image transformation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

англ. – англійська

ВПМ – випадкові поля Маркова

ЗНМ – згорткова нейромережа

НПС – нейронне перенесення стилю

НФВ – нефотореалістична візуалізація

ХВОЗ – художня візуалізація на основі зображень

NFT – non-fungible token

ВСТУП

Однією з особливостей, притаманних людському розуму, є здатність до творчості. Живопис – популярний вид мистецтва. Людей приваблювало мистецтво живопису своєю здатністю до створення неповторних візуальних переживань через взаємодію між стилем та змістом. Особливо це стає помітно з появою багатьох привабливих творів мистецтва, наприклад, «Зоряна ніч» Ван Гога. Проте, раніше для отримання копії зображення в певному стилі потрібен був добре підготовлений художник і багато часу.

З середини 1990-х років теорії мистецтва, які стоять за привабливими творами мистецтва, привертають увагу не лише художників, а й багатьох дослідників інформатики. Існує багато досліджень і методів, спрямованих на автоматичне перетворення зображення на синтетичні твори мистецтва. Серед цих досліджень надихають досягнення в області нефотореалістичної візуалізації.

Однак, більшість алгоритмів НФВ розроблена для певних художніх стилів і не може бути легко поширена на інші стилі.

У спільноті комп'ютерного зору передача стилю зазвичай вивчається як узагальнена проблема синтезу текстури, яка полягає у вилученні та передачі текстури від джерела до цілі. Однак загальним обмеженням цих методів є те, що вони використовують лише низькорівневі ознаки зображення і часто не можуть ефективно зафіксувати зміст зображення.

У 2015 році натхнений потужністю згортковий нейронних мереж Гатіс та ін. вперше вивчив, як використовувати ЗНМ для відтворення відомих стилів живопису на природних зображеннях. Хоча досягнення в області нейронного перенесення стилю надихають, і вони вже знайшли промислове застосуваннях, проте, все ще існують невирішені проблеми в роботі алгоритмів НПС, а також в їх ефективності та доступності.

До можливих застосувань НПС можна віднести імітацію або створення робіт мистецтва, дизайн та споріднені напрями. Застосування

НПС в соціальній комунікації зміцнює зв’язки між людьми, а також має позитивний вплив як на наукові кола, так і на промисловість.

Інше використання НПС полягає в тому, щоб змусити його діяти як інструмент для створення мистецтва користувачем.

Все це і формує **актуальність** цієї роботи.

Об’єкт дослідження – процес перенесення стилю одного зображення на інший без втрати змісту.

Предмет дослідження – метод перенесення стилю з використанням згорткових нейронних мереж.

Мета дослідження – розробити інформаційну технологію перенесення стилю з одного зображення на інше з використанням нейронних мереж.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

1. Висвітлення сутності задачі перенесення стилю та існуючих методів її вирішення, їх порівняння.
2. Дослідження теоретичних засобів вирішення задачі. Вибір засобів, що забезпечать ефективне досягнення поставленої мети дослідження.
3. Розробка інформаційної технології нейронного перенесення стилю на основі обраних теоретичних засобів.
4. Оцінка результатів роботи інформаційної технології.

Для вирішення поставлених завдань були використані такі **методи** дослідження:

- метод зіставного аналізу сприяв встановленню переваг та недоліків існуючих алгоритмів перенесення стилю;
- описовий метод був застосований для комплексного дослідження задачі перенесення стилю.

РОЗДІЛ 1. АНАЛІЗ ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ МЕТОДІВ ПЕРЕНЕСЕННЯ СТИЛЮ

1.1. Задача перенесення стилю

З середини 1990-х років теорії мистецтва, які стоять за привабливими творами мистецтва, привертають увагу не лише художників, а й багатьох дослідників інформатики. Існує багато досліджень і методів, які намагаються досягти автоматичного перетворювати зображення на синтетичні твори мистецтва. Серед цих досліджень надихають досягнення в області нефотореалістичної візуалізації. Однак більшість цих алгоритмів розроблені для певних стилів і не є гнучкими.

Живопис – це популярний вид мистецтва, який дозволяє людям створювати візуальні переживання через складну взаємодію змісту і стилю зображення. Алгоритмічна основа цього процесу невідома, і поки не існує штучної системи, що могла б відтворити цей процес.

Перенесення стилю з одного зображення на інше можна вважати проблемою перенесення текстури. Постановка задачі полягає у синтезі зображення з текстурою, що відповідає стилю іншого зображення, накладаючи обмеження на збереження змісту.

Натхнені потужністю згорткових нейронних мереж, вчені почали використовувати ЗНМ для відтворення відомих стилів живопису на природних зображеннях[1].

Найперше застосування, що спадає на думку у контексті перенесення стилю – це імітація або створення робіт мистецтва, дизайн та споріднені напрями.

Застосування НПС в соціальній комунікації зміцнює зв’язки між людьми, а також має позитивний вплив як на наукові кола, так і на промисловість. Для науковців, коментарі користувачів можуть допомогти удосконалити алгоритм.

Інше використання НПС полягає в тому, щоб змусити його діяти як інструмент для створення мистецтва користувачем. НПС може спростити і

покращити досвід художників при створенні творів мистецтва певного стилю, особливо під час створення комп’ютерних творів мистецтва. Це особливо стає ще більш доречним зараз, в епоху розквіту цифрового мистецтва, завдяки появі технології NFT [2].

Крім того, за допомогою алгоритмів НПС можливо створювати стилізовані елементи одягу для модельєрів.

Проте варто зауважити, що цілком можливими і доречними можуть бути й інші застосування НПС, особливо в тих областях, де необхідно отримати цілісне для сприйняття людиною зображення, наприклад, візуалізація і покращення супутникових знімків.

1.2. Класичні алгоритми перенесення стилю

Художня стилізація – давня тема дослідження. Завдяки широкому спектру застосувань, вона була і залишається важливою областю досліджень протягом більше двох десятиліть. До появи НПС пов'язані дослідження поширилися на область, яка називається нефотореалістичною візуалізацією (НФВ). У цьому підрозділі ми коротко розглянемо деякі з цих алгоритмів художнього візуалізації без ЗНМ. Зокрема, ми зосередимося на художній стилізації 2D-зображень, яка ще також називається художньою візуалізацією на основі зображень (ХВОЗ), таксономію методів ХВОЗ представив у своїй праці Кіпріанідіс [3].

1.2.1. Візуалізація на основі штрихів або мазків.

Візуалізація на основі штрихів відноситься до процесу розміщення віртуальних мазків (наприклад, мазків пензля, плиток, крапок) на цифровому полотні для відтворення зображення в певному стилі. Цей процес зазвичай починається з вихідної фотографії, за ним слідує поступове компонування штрихів у відповідності до фотографії, і, нарешті, створення

нефотореалістичного зображення, яке виглядає як фото, але має художній стиль (див. рис. 1.).

Рис. 1. Процес роботи алгоритму на основі штрихів

Під час цього процесу цільова функція призначена для керування жадібним або ітеративним розміщенням штрихів. Метою цих алгоритмів є достовірне зображення заданого стилю, тому вони, як правило, ефективні для моделювання певних типів стилів (наприклад, олійних картин, акварелі, ескізів). Однак кожен алгоритм на основі штрихів ретельно розроблений лише для одного конкретного стилю і не може імітувати довільний стиль, а отже не є гнучким [3].

1.2.2. Методи на основі регіонів

Візуалізація на основі регіонів передбачає сегментацію зображення на регіони, щоб уможливити адаптацію візуалізації на основі вмісту в регіонах. Ранні алгоритми ХВОЗ, засновані на регіонах, використовують форму областей для керування розташуванням штрихів. Таким чином, можна створити різні шаблони штрихів у різних семантичних областях зображення. Сонг та ін. запропонували алгоритм ХВОЗ на основі регіонів,

що маніпулює геометрією для художніх стилів. Їхній алгоритм створює спрощені ефекти відтворення форми, замінюючи області кількома канонічними фігурами (див. рис. 2 і 3).

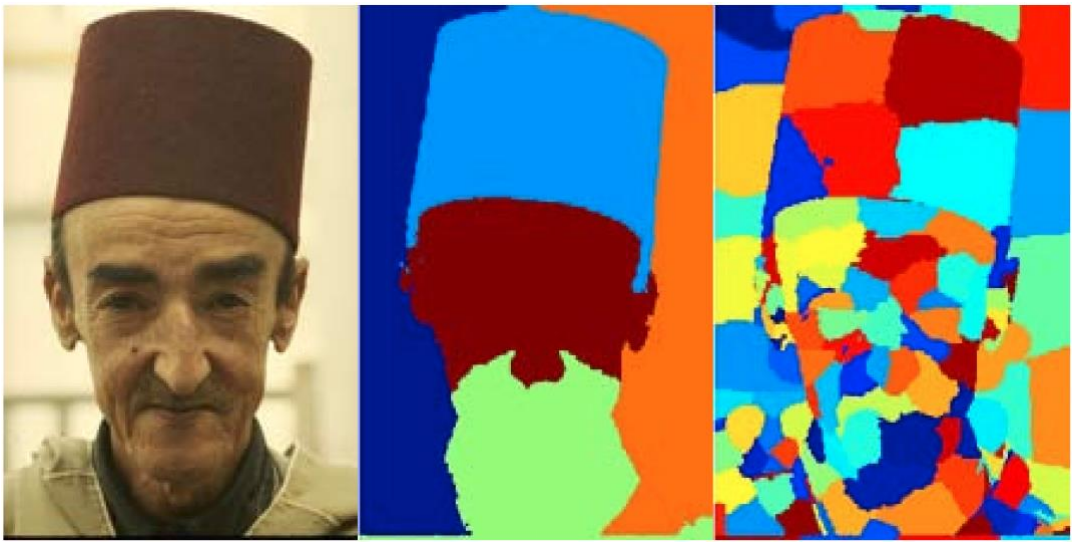


Рис. 2. Процес розбиття зображення на регіони

Рис. 3. Результати роботи методу візуалізації на основі регіонів

Врахування регіонів при візуалізації дозволяє локально контролювати рівень деталізації. Однак проблема в цих методах залишається така як і в

візуалізації на основі штрихів: один алгоритм візуалізації на основі регіону не здатний імітувати довільний стиль [5].

1.2.3. Візуалізація на основі прикладів.

Метою візуалізації на основі прикладів є вивчення відношення між зразковою парою. Ця категорія методів ХВОЗ була першою впровадженою Hertzmann та ін., які запропонували структуру під назвою аналогії зображень. Аналогії зображень спрямовані на вивчення відношення між парою вихідних зображень і цільових стилізованих зображень у контрольований спосіб. Навчальний набір зображень містить пари нестилізованих вихідних зображень і відповідних стилізованих зображень з певним стилем. Алгоритм аналогії зображень потім вивчає перетворення у прикладах навчальних пар і створює аналогічні стилізовані результати, при умові отримання тестової вхідної фотографії (див. рис. 4).

Рис. 4. Навчальні зображення для візуалізації на основі прикладів

Загалом, аналогії зображень ефективні для різних художніх стилів. Однак на практиці пари навчальних даних зазвичай недоступні. Іншим обмеженням є те, що аналогії зображень використовують лише низькорівневі функції зображення. Таким чином вони зазвичай не можуть ефективно охопити зміст і стиль, що і обмежує продуктивність [6].

1.2.4. Обробка та фільтрація зображень.

Створення художнього зображення – це процес, який спрямований на його спрощення та абстрагування. Тому цілком природно розглянути можливість використання та комбінування деяких пов’язаних між собою фільтрів обробки зображень для візуалізації вхідної фотографії. Winnemoller та ін. вперше використали цей метод для автоматичного створення ефектів, схожих на мультфільм (див. рис. 5).

Рис. 5. Результат використання методу фільтрації зображень

У порівнянні з іншими категоріями методів ХВОЗ, алгоритми візуалізації на основі фільтрації зображень, як правило, прості в реалізації

та ефективні на практиці. За рахунок цього вони дуже обмежені в стильовому розмаїтті [7].

Отже, хоча деякі алгоритми ХВОЗ без ЗНМ здатні достовірно відображати певні прописані стилі, вони зазвичай мають обмеження в гнучкості, різноманітності стилів та ефективній диференціації структури зображень. Бажання усунути ці обмеження і породжують попит на нові більш досконалі методи, а саме НПС.

1.3. Похідні нейронного перенесення стилю

Для кращого розуміння розвитку НПС розглянемо його похідні. Для автоматичного передавання художнього стилю першим і найважливішим завданням є моделювання та екстракція стилю із зображення. Оскільки стиль дуже пов’язаний з текстурою, то простим способом може бути моделювання візуального стилю з раніше добре вивченими методами візуального моделювання текстур.

Після отримання відображення стилю наступне завдання полягає у відновленні зображення з потрібною інформацією про стиль, зберігаючи його зміст, що вирішується методами реконструкції зображення.

1.3.1. Візуальне моделювання текстур.

Протягом усієї історії існувало два різних підходи до моделювання візуальних текстур, а саме:

- параметричне моделювання текстури зі зведеною статистикою;
- непараметричне моделювання текстури з випадковими полями Маркова.

Параметричне моделювання текстури зі зведеною статистикою. Одним із шляхів до моделювання текстури є отримання статистики зображення із зразка текстури та використання підсумкової статистичної

властивості для моделювання текстури. Ідея була вперше запропонована Юлешем, який моделював текстури як статистику *N*-го порядку на основі пікселів. Пізніше він також використовує відповіді фільтра для аналізу текстур замість прямих вимірювань на основі пікселів[8].

Згодом Портілла та Сімончеллі впроваджують модель текстури, що заснована на багатомасштабних орієнтованих відгуках фільтрів і використовують градієнтний спуск для покращення синтезованих результатів[9].

Більш сучасний підхід до параметричного моделювання текстури, запропонований Гатіс та ін., був першим, що вимірював підсумкову статистику в області ЗНМ.

Вони спроектували представлення на основі Грама для моделювання текстур, що є кореляціями між відповідями фільтрів у різних шарах попередньо навченої мережі класифікації (мережі VGG) [10].

Припустимо, що карта ознак зразка зображення з текстурою I_s на шарі l попередньо навченої мережі глибокої класифікації дорівнює $\mathcal{F}^l(I_s)' \in \mathbb{R}^{C \times H \times W}$, де C – кількість каналів, а H і W представляють висоту і ширину карти об'єктів $\mathcal{F}(I_s)$. Тоді представлення на основі Грама можна отримати шляхом обчислення матриці Грама $\mathcal{G}(\mathcal{F}^l(I_s)') \in \mathbb{R}^{C \times C}$ і з карти ознак $\mathcal{F}^l(I_s)' \in \mathbb{R}^{C \times (HW)}$ (перероблена версія $\mathcal{F}^l(I_s)$) (1.1):

$$\mathcal{G}(\mathcal{F}^l(I_s)') = [\mathcal{F}^l(I_s)'][\mathcal{F}^l(I_s)']^T \tag{1.1}$$

Це представлення текстур на основі Грама з ЗНМ є ефективним для моделювання різноманітних як природних, так і неприродних текстур. Проте представлення на основі Грама розроблено для збору глобальної статистики та ігнорує просторового розташування, що призводить до незадовільних результатів моделювання регулярних текстур із симетричними структурами на великій відстані. Щоб вирішити цю

проблему, Бергер і Мемісевич запропонували горизонтально і вертикально перекладати карти об’єктів на δ пікселів, щоб співвіднести об’єкт в положенні (i, j) з тими, що знаходяться в позиціях $(i + \delta, j)$ і $(i, j). + \delta)$.

Таким чином, представлення включає інформацію про просторове розташування і тому є більш ефективним для моделювання текстур із симетричними властивостями (див. рис. 6) [11].

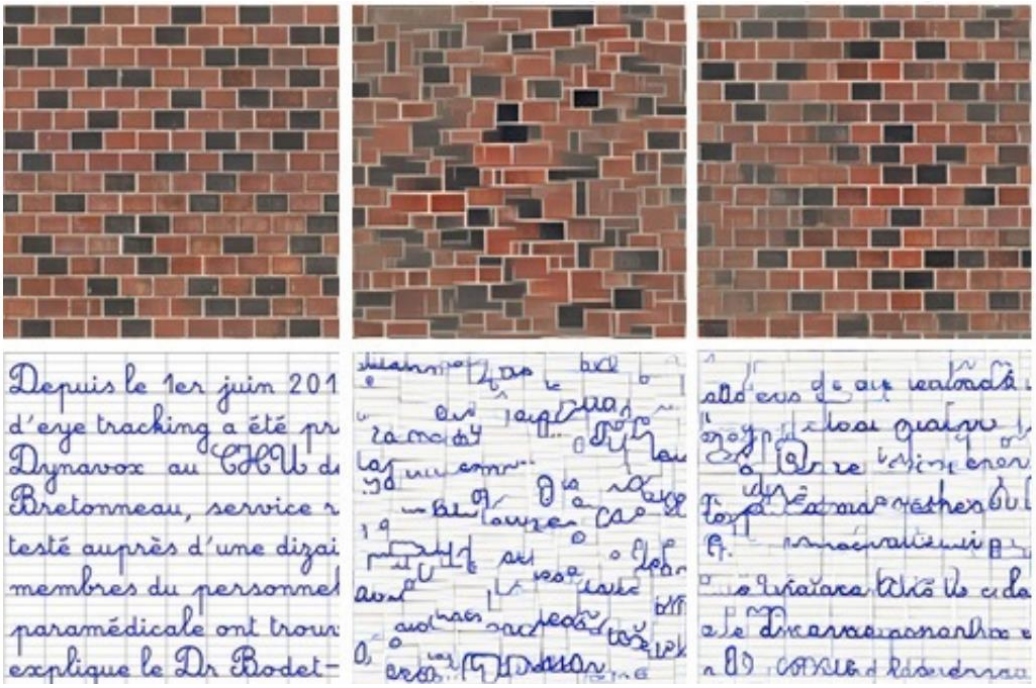


Рис. 6. Порівняння підходу Гатіс (центр) та Бергера і Мемісевича (праворуч) з початковим зображенням (ліворуч).

Непараметричне моделювання текстури за допомогою випадкових полів Маркова. Іншою помітною методологією моделювання текстури є використання непараметричної передискретизації. Існує багато непараметричних методів заснованих на моделі випадкових полів Маркова, яка передбачає, що в зображенні текстури кожен піксель повністю характеризується його просторовим сусідством. Відповідно до цього припущення, Ефрос і Леунг пропонують синтезувати кожен піксель один за одним шляхом пошуку подібних околиць у вихідному зображенні текстури та призначення відповідного пікселя. Їхня робота є одним із найперших

непараметричних алгоритмів з ВПМ[11]. Процес зіставлення сусідства можна прискорити шляхом постійного використання фіксованого сусідства.

1.3.2. Реконструкція зображення

Загалом, важливим кроком для багатьох завдань пов'язаних з комп'ютерним зором є вилучення абстрактного представлення з вхідного зображення. Реконструкція зображення є зворотним процесом, який полягає у відновленні всього вхідного зображення з вилученого зображення. Наша основна увага зосереджена на алгоритмах реконструкції зображень на основі представлення з використанням ЗНМ, які можна розділити на:

- онлайн реконструкцію зображення на основі оптимізації зображень (англ. Image-based online Image reconstruction, IOB-IR)
- автономну реконструкцію зображення на основі оптимізації моделі (англ. Model-based offline Image reconstruction, MOB-IR).

Онлайн-реконструкція зображення на основі оптимізації зображень. Перший алгоритм для зворотного уявлення ЗНМ запропоновано Махендраном і Ведальді. Враховуючи, що відображення ЗНМ зворотнє, їхній алгоритм ітераційно оптимізує зображення (зазвичай починаючи з випадкового шуму), поки воно не отримає аналогічне бажане представлення ЗНМ. Ітераційний процес оптимізації заснований на градієнтному спуску в просторі зображення. Тому процес займає багато часу, особливо коли бажане реконструйоване зображення є великим [12].

Автономна реконструкція зображення на основі оптимізації моделі. Для вирішення питання ефективності алгоритму наведеного вище Досовицький і Брокс запропонували заздалегідь навчити мережу з прямим поширенням і покласти обчислювальний тягар на етап навчання. На етапі тестування зворотний процес можна просто виконати за допомогою прямого проходу мережі. Їх алгоритм значно прискорює процес відновлення зображення. У своїй пізнішій роботі вони додатково

об’єднують генеративну змагальну мережу (GAN) для покращення результатів[13].

1.4. Алгоритми нейронного перенесення стилю

НПС є підмножиною методів ХВОЗ на основі прикладів згаданих у 1.2.3. У цьому підрозділі ми спочатку надамо категоризацію алгоритмів НПС, а потім детально пояснюємо основні нефотореалістичні алгоритми НПС на основі 2D зображень, адже вони становлять найбільший інтерес в рамках цієї роботи і для вирішення нашого завдання.

Визначення стилю є складним, а тому критерії важливі при створені «успішного» алгоритму його передачі є доволі суб’єктивними, так, наприклад, вже існує проблема естетичного критерію.

Поточні методи НПС можна віднести до однієї з двох категорій:

- Онлайн нейронні методи на основі оптимізації зображень (англ. Image-based online neural style transfer, IOB-NST)
- Автономні нейронні методи на основі оптимізації моделі (англ. Model-based offline neural style transfer, MOB-NST).

Перша категорія передає стиль шляхом ітеративної оптимізації зображення, тобто алгоритми, що належать до цієї категорії, побудовані на методах IOB-IR. Друга категорія оптимізує генеративну модель в автономному режимі та створює стилізоване зображення за один прохід вперед, що використовує ідею техніки MOB-IR.

1.4.1. Онлайн нейронні методи на основі оптимізації зображень.

DeepDream – це перша спроба створити художні зображення шляхом реверсування представлень ЗНМ за допомогою методів IOB-IR[14]. Завдяки подальшому поєднанню методів візуального моделювання текстури зі стилем моделювання і з’являються алгоритми IOB-NST, які створюють ранні основи для галузі НПС. Їх основна ідея полягає в тому, щоб спочатку змоделювати та витягти інформацію про стиль та зміст із відповідних

зображень, а потім об’єднати їх у цільове представлення щоб згодом ітеративно реконструювати стилізований результат, який відповідає цільовому представленню. Загалом, різні алгоритми НПС на основі оптимізації зображення мають однакову техніку IOB-IR, але відрізняються за способом моделювання візуального стилю, який побудований на двох категоріях методів візуального моделювання текстур згаданих у 1.3.1. Загальне обмеження алгоритмів НПС на основі оптимізації зображенні полягає в тому, що вони є дорогими в обчислювальному відношенні через ітераційну процедуру оптимізації зображення.

1.4.1.1. Параметричний НПС на основі оптимізації зображення

Перша підмножина методів НПС на основі оптимізації зображень заснована на параметричному моделюванні текстури зі зведеною статистикою. Стиль характеризується як набір просторової зведеної статистики. До цих методів відноситься і алгоритм запропонований Гатіс та ін. Реконструюючи представлення з проміжних шарів мережі VGG-19, вчені зауважили, що глибока згорткова нейронна мережа здатна витягувати зміст зображення з довільної фотографії та деяку інформацію про зовнішній вигляд із добре відомих картин [15].

1.4.1.2. Непараметричний НПС на основі оптимізації зображення

Непараметричний НПС на основі оптимізації зображення побудований на основі непараметричного моделювання текстури з ВПМ (випадкові поля Маркова). Ця категорія розглядає НПС на локальному рівні, тобто роботу з клаптиками, які відповідають стилю.

Лі і Венд першими запропонували алгоритм НПС на основі ВПМ. Вони виявили, що параметричний метод НПС із підсумковою статистикою фіксує лише кореляції на рівні окремих піксельних характеристик і не обмежує просторове розташування, що призводить до гірших результатів у використанні для фотореалістичних стилів. Їхнє рішення полягає в тому,

щоб змоделювати стиль непараметричним способом і ввести нову функцію втрати стилю, яка включає попередній ВПМ на основі клаптиків (1.2):

$$\mathcal{L}_s = \sum_{l\in\{l_s\}} \sum_{i=1}^{m} \left\| \Psi_i\left(\mathcal{F}^l(I)\right) - \Psi_{NN(i)}\left(\mathcal{F}^l(I_s)\right) \right\|^2 \quad (1.2)$$

де $\Psi\left(\mathcal{F}^l(I)\right)$ – множина всіх локальних клаптиків із карти ознак $\mathcal{F}^l(I)$. Ψi позначає *i*-ий локальний клаптик, а $\Psi_{NN(i)}$ є найбільш подібним стилем клаптика з *i-им* локальним клаптиком у стилізованому зображенні *I*. m – загальна кількість локальних клаптиків.

Оскільки їхній алгоритм відповідає стилю на рівні клаптика, точна структура та упорядкування можуть бути збережені набагато краще. Проте це зазвичай не вдається, коли зображення вмісту та стилю мають сильні відмінності в перспективі та структурі, оскільки латки зображення не можуть бути правильно підібрані. Він також обмежений у збереженні чітких деталей та інформації про глибину [16].

1.4.2. Автономні нейронні методи на основі оптимізації моделі

Хоча НПС на основі оптимізації зображення можуть давати вражаючі стилізовані зображення, все ж існують деякі обмеження. Найбільшим обмеженням є питання ефективності. Друга категорія НПС на основі оптимізації моделі вирішує проблему швидкості та обчислювальної вартості, використовуючи автономну реконструкцію зображення на основі оптимізації моделі для реконструкції стилізованого результату, тобто мережа прямого поширення g оптимізована для великого набору зображень I_c для одного або кількох зображень стилю I_s (1.3):

$$\theta^* = \arg\min_{\theta} \mathcal{L}_{total}\left(I_c, I_s, g_{\theta^*}(I_c)\right), I^* = g_{\theta^*}(I_c) \quad (1.3)$$

Залежно від кількості художніх стилів, які може створити одна g, алгоритми НПС на основі оптимізації моделі далі поділяються на додаткові методи: Методи MOB-NST з одним стилем на модель (англ. Per-Style-Per-Model, PSPM), Методи MOB-NST з кількома стилями на модель (англ. Multiple-Style-Per-Model, MSPM) і Методи MOB-NST з довільним стилем на модель (англ. Arbitrary-Style-Per-Model, ASPM).

1.4.2.1. Параметричні PSPM.

Перші два алгоритми MOB-NST запропоновані Джонсоном та ін. [17] та Ульяновим та ін. [18] відповідно. Ці два методи мають подібну ідею, яка полягає в тому, щоб попередньо навчити мережу прямого поширення налаштовану під специфічний стиль, і отримати стилізований результат за один прямий прохід на етапі тестування. І хоча вони відрізняються архітектурою мережі, цільова функція схожа на алгоритм Гатіс та ін. [15], що вказує, що вони також є параметричними методами зі зведеною статистикою.

Алгоритми Johnson та ін. та Ульянов та ін. досягти передачі стилю в режимі реального часу. Проте, дизайн їхнього алгоритму в основному відповідає алгоритму Gatys та ін., через що вони страждають від тих же вищезгаданих проблем, що й алгоритм Gatys та ін. (наприклад, відсутність врахування узгодженості деталей та глибинної інформації).

1.4.2.2. Непараметричний PSPM з ВПМ.

Інша робота Лі та Ванда [19] натхнена алгоритмом НПС на основі ВПМ у розділі 1.4.1. вони вирішують питання ефективності, навчаючи марковську мережу прямого поширення з використанням змагального навчання. Подібно до своєї попередньої роботи, їхній алгоритм є непараметричним методом з ВПМ на основі клаптиків. Їхній метод перевершує алгоритми Джонсона та Ульянова у збереженні пов’язаних текстур у складних зображеннях завдяки їхньому дизайну на основі

клаптиків. Однак їхній алгоритм має менш задовільну продуктивність зі стилями без текстур (наприклад, зображеннями обличчя), оскільки їх алгоритм не враховує семантику. Інші слабкі сторони їх алгоритму включають недостатнє врахування глибокої інформації та варіації мазків пензля, які є важливими візуальними факторами.

1.4.2.3. Багато стилів на модель. Прив'язування лише невеликої кількості параметрів до кожного стилю.

Хоча вищезгадані підходи PSPM можуть створювати стилізовані зображення на два порядки швидше, ніж попередні методи IOB-NST, окремі генеративні мережі повинні бути навчені для кожного конкретного зображення стилю, що є досить трудомістким та негнучким завданням. Проте багато картин (наприклад, картини імпресіоністів) мають схожі мазки фарби і відрізняються лише кольоровою палітрою. Інтуїтивно, тренувати окрему мережу для кожної з них є зайвим. У відповідь на це виникає метод Багатьох стилів на модель (англ. Multiple-Style-Per-Model, MSPM), який покращує гнучкість шляхом подальшого включення кількох стилів в одну модель.

Рання робота Дюмулена та ін. [20] побудовано на основі запропонованого шару індивідуальної нормалізації (еквівалент пакетної нормалізації коли розмір пакету становить 1) в алгоритмі Ульянова та ін.[21]. Вони несподівано виявили, що для моделювання різних стилів є достатнім використання тих самих параметрів згортки, проте до яких було застосовано масштабування та зсув в шарах індивідуальної нормалізації. Вчені пропонують алгоритм навчання умовної мережі передачі багатьох стилів на основі умовної індивідуальної нормалізації (CIN), який визначається як (1.4):

$$\mathrm{CIN}(\mathcal{F}(I_c), s) = \gamma^s \left(\frac{\mathcal{F}(I_c) - \mu(\mathcal{F}(I_c))}{\sigma(\mathcal{F}(I_c))} \right) + \beta^s , \qquad (1.4)$$

де $\mathcal{F}$ – активація вхідної функції, а s – індекс бажаного стилю з набору зображень стилю. Як показано в рівнянні, кондиціонування для кожного стилю I_s здійснюється шляхом масштабування та зсуву параметрів γ^s і β^s після нормалізації функції активації $\mathcal{F}(I_c)$, тобто кожного стилю I_s можна досягти шляхом налаштування параметрів афінного перетворення .

Інтерпретація подібна до роботи Ульянова та ін., тобто нормалізація статистики ознак з різними афінними параметрами може нормалізувати зображення вхідного змісту до різних стилів. Крім того, алгоритм Дюмулена та ін. також можна розширити, щоб об’єднати кілька стилів в одному стилізованому результаті шляхом комбінування афінних параметрів різних стилів. Результат роботи алгоритму наведено на рис 7.

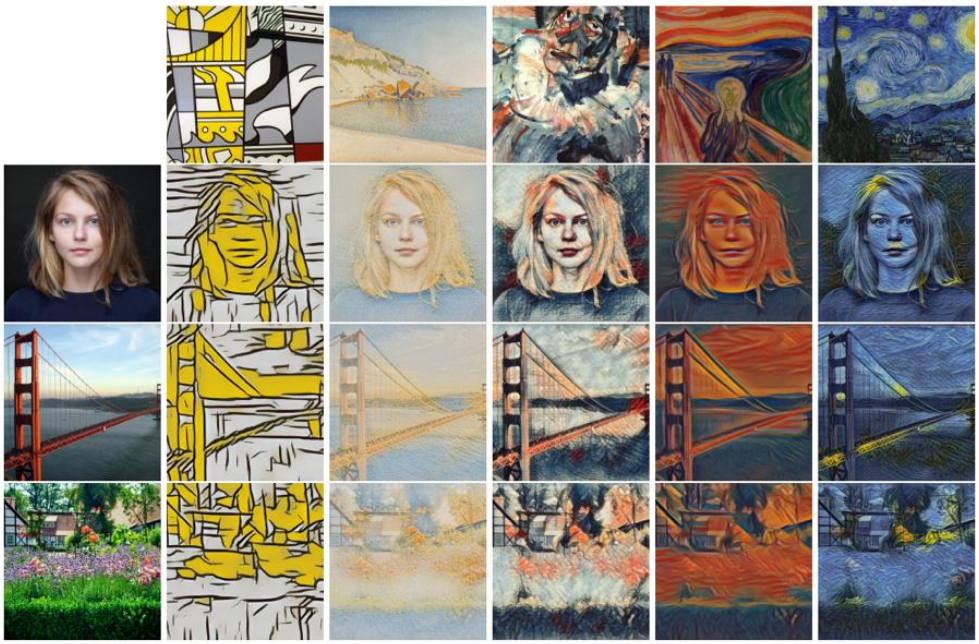


Рис. 7. З умовною індивідуальною нормалізацією нейромережа може працювати з 32 стилями

1.4.2.4. Багато стилів на модель. Поєднання стилю та змісту як вхідних даних.

Одним з недоліків прив'язування лише невеликої кількості параметрів до кожного стилю є те, що розмір моделі зазвичай стає більшим зі збільшенням кількості вивчених стилів.

Другий шлях MSPM усуває це обмеження шляхом повного використання можливостей однієї мережі та поєднання змісту та стилю, як входів в мережі, для визначення стилю. Різні алгоритми MSPM відрізняються за способом включення стилю в мережу.

Лі та ін. враховуючи N цільових стилів спроектувати блок вибору для вибору стилю, який є N-вимірним одноразовим вектором. Кожен біт в одиниці вибору представляє певний стиль I_s в наборі цільових стилів. Для кожного біта в одиниці вибору вони спочатку вибирають відповідну карту шуму $f(I_s)$ з рівномірного розподілу, а потім передають $f(I_s)$ у підмережу стилів, щоб отримати відповідні ознаки, закодовані у стилі $F(f(I_s))$. За допомогою подачі ланцюга закодованих стилем ознак $F(f(I_s))$ і закодованих змістом ознак $Enc(I_c)$ у частину Dec декодера мережі передачі стилів можна отримати бажаний стилізований результат (1.5):

$$I = Dec(\, F(\, f\,(I_s)) \oplus Enc(I_c)). \qquad (1.5)$$

Інша робота Чжана і Дани [22] спочатку поширює кожне зображення стилю в наборі стилів через попередньо навчену мережу VGG і отримує багатомасштабні активації функцій $F(I_s)$ у різних шарах VGG. Потім багатомасштабні $F(I_s)$ поєднуються з багатомасштабними закодованими ознаками $Enc(I_c)$ з різних шарів у енкодері через запропоновані вченими шари «натхнення». Шари «натхнення» призначені для зміни форми $F(I_s)$, щоб відповідати бажаному розміру, а також матрицю ваги, яка здатна до навчання, щоб налаштувати карти ознак для допомоги у мінімізації цільової функції.

Цей тип MSPM вирішує обмеження великого розміру модел, що було присутнє у першому типі MSPM. Проте це стає можливим жертвуючи масштабованістю стильового різноманіття, що стає набагато меншим, оскільки лише одна мережа використовується для кількох стилів.

1.4.2.5. Довільний стиль на модель.

Нейронні методи довільного стилю на модель (англ. Arbitrary-Style-Per-Model, ASPM) мають на меті створення однієї моделі для всіх стилів, тобто модель, яку можна навчати передачі довільних художніх стилів.

Існує два типи ДСНМ: один побудований на основі непараметричного моделювання текстури з ВПМ, а інший – на основі параметричного моделювання текстури зі зведеною статистикою.

Непараметричний ДСНМ з ВПМ був запропоновано Ченом і Шмідтом. Спочатку вони витягують набір клаптиків активації з активацій ознак змісту та стилю, обчислених у попередньо навченій мережі VGG. Потім вони поєднують кожен клаптику змісту з найбільш подібним клаптиком стилю і міняють їх місцями (так званий «стильовий обмін»). Стилізований результат можна отримати шляхом реконструкції результуючої карти активації після «стильового обміну» за допомогою методів IOB-IR або MOB-IR.

Алгоритм Чена і Шмідта є більш гнучким, ніж попередні підходи, завдяки підходу «одна модель для всіх», проте стилізовані результати менш привабливі, оскільки клаптики змісту, як правило, замінюються клаптиками стилю, які не відповідають бажаному стилю. В результаті зміст добре збережено, тоді як стиль, як правило, погано відображений [23].

Параметричний ДСНМ зі зведеною статистикою. Враховуючи інформацію наведену вище стосовно БСНМ у 1.4.4., а саме відомості з роботи Думуліна та ін., найпростіший підхід для передачі довільного стилю полягає у навчанні окремої мережі прогнозування параметрів P для прогнозування γ^s та β^s у рівнянні (1.6):

$$\mathrm{CIN}(\mathcal{F}(I_c), s) = \gamma^s \left(\frac{\mathcal{F}(I_c) - \mu(\mathcal{F}(I_c))}{\sigma(\mathcal{F}(I_c))} \right) + \beta^s, \qquad (1.6)$$

з низкою стилів навчання. Враховуючи тестове зображення стилю I_s, шари CIN в мережі передачі стилю беруть афінні параметри γ^s і β^s з $P(I_s)$ та нормалізують вхідне зображення з бажаним змістом до потрібного стилю за допомогою прямого проходу.

Інший подібний підхід, заснований на роботі Дюмуліна, пропонують Хуанг і Белонджі [24]. Замість навчання мережі передбачення параметрів, Хуанг і Белонджі пропонують змінити умовну нормалізацію екземплярів (CIN) на адаптивну нормалізацію екземплярів (AdaIN) (1.7):

$$\mathrm{AdaIN}\big(\mathcal{F}(I_c), \mathcal{F}(I_s)\big) = \sigma\big(\mathcal{F}(I_s)\big) \left(\frac{\mathcal{F}(I_c) - \mu(\mathcal{F}(I_c))}{\sigma(\mathcal{F}(I_c))} \right) + \\ + \mu\big(\mathcal{F}(I_s)\big). \qquad (1.7)$$

На відміну від роботи Дюмуліна та ін., кодер у мережі передачі стилю Хуанга і Белонджі є фіксованим і містить кілька перших шарів у попередньо навченій мережі VGG. Отже, F є активацією функції з попередньо навченої мережі VGG. Частину декодера потрібно навчити з великим набором зображень стилю та змісту, щоб декодувати результуючі активації функцій після AdaIN до стилізованого результату: $I = Dec(\mathrm{AdaIN}(F(I_c), F(I_s)))$.

Алгоритм Хуанга і Белонгі є першим алгоритмом ASPM, який досягає стилізації в реальному часі. Однак він залежний від даних та обмежений у узагальненні небачених стилів.

Робота Li та ін. [25] намагається використати серію трансформацій ознак для передачі довільного художнього стилю у вільний від стильового навчання спосіб. Подібно до роботи Хуанга і Белонгі, Лі та ін. використовують перші кілька шарів попередньо навченого VGG як кодер і

тренують відповідний декодер. Але вони замінюють шар AdaIN [51] між кодером і декодером парою перетворень відбілювання та фарбування (англ. whitening and colouring transformations, WCT): $I = Dec(\text{WCT}(F(I_c), F(I_s)))$.

1.5. Порівняння алгоритмів НПС

Таблиця 1.

Порівняння середньої швидкості роботи алгоритмів

Methods	**Time(s)**			**Styles/Model**
	256 × 256	**512 × 512**	**1024 × 1024**	
Gatys et al.	14.32	51.19	200.3	∞
Johnson et al.	0.014	0.045	0.166	1
Ulyanov et al.	0.022	0.047	0.145	1
Li and Wand	0.015	0.055	0.229	1
Zhang and Dana	0.019 (**0.039**)	0.059 (**0.133**)	0.230 (**0.533**)	$k(k \in Z^+)$
Li et al.	0.017	0.064	0.254	$k(k \in Z^+)$
Chen and Schmidt	0.123 (**0.130**)	1.495 (**1.520**)	–	∞
Huang and Belongie	0.026 (**0.037**)	0.095 (**0.137**)	0.382 (**0.552**)	∞
Li et al.	0.620	1.139	2.947	∞

Таблиця 2.

Узагальнене порівняння методів НПС

Тип	**Метод**	**Плюси і мінуси**			
		Е	**ДС**	**БН**	**ВЯ**
IOB-NST	Гатіс та ін.	-	+	+	Хороша, вважається золотим стандартом
PSPM-MOB-NST	Ульянов та ін.	+	-	-	Результати перших двох методів схожі на результати метода Гатіса та ін. Якість результатів третього алгоритму гірша.
	Джонсон та ін.	+	-	-	
	Лі та Ванд	+	-	-	
MSPM-MOB-NST	Дюмулен та ін.	+	-	-	Результати перших двох методів схожі на Гатіса та ін., проте розмір моделі зростає з кількістю вивчених стилів. Третій та четвертий вирішують проблему розміру, проте візуальна якість страждає.
	Чен та ін.	+	-	-	
	Лі та ін.	+	-	-	
	Чжан і Дана	+	-	-	
ASPM-MOB-NST	Чен і Шмідт.	+	+	-	Загалом результати ASPM менш вражаючі ніж інших типів НПС алгоритмів. Перший не комбінує достатньо стильових елементів, другий не ефективний в створенні складних
	Хуанг і Белонджі	+	+	-	
	Лі та ін.	+	+	+	

					стильових патернів, а третій в мазках і контурах.
Е – ефективність, ДС – довільний стиль, БН – без навчання, ВЯ – візуальна якість					

Висновки до першого розділу

У цьому розділі нами було:

1. Висвітлено сутність задачі перенесення стилю та проведено ґрунтовний аналіз методів її вирішення. Отже, перенесення стилю – задача, яку, незважаючи на інтуїтивно просте розуміння людиною, складно задати алгоритмічно. Хоча існує ряд методів, які в тій чи іншій мірі вирішують цю проблему, вони в абсолютній більшості є обмеженими або ресурсозатратними з необхідним виконання процесу навчання.
2. На основі порівняння методів НПС зроблено висновок, що останні часом найбільш перспективні методи застосовують згорткові глибинні нейронні мережі для розв’язку цієї проблеми.
3. Сформовано мету нашої роботи, а саме: розробити інформаційну технологію нейронного перенесення стилю з одного зображення на інше.

Протягом останніх кількох років перенесення стилю на основі нейронних мереж продовжувало ставати надихаючим дослідницьким напрямком, мотивованим як науковими проблемами, так і промисловими вимогами. Проте, незважаючи на великий прогрес останніх років, ця область далека від зрілого стану.

РОЗДІЛ 2. МЕТОДИ НПС. ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ.

2.1. Згорткова нейронна мережа

Згорткові нейронні мережі – це клас глибинних штучних неронних мереж прямого поширення, що відрізняються від інших нейронних мереж їх чудовою продуктивністю з зображенням, мовленням або звуковим сигналом.

ЗНМ складається з шарів входу та виходу, а також з прихованих шарів, що в свою чергу, як правило, поділяються на три основних типи шарів, а саме:

- Згортковий шар
- Агрегувальний шар
- Повноз’єднаний шар

Згортковий шар є першим шаром згорткової мережі. У той час як за згортковими шарами можуть слідувати додаткові згорткові шари або агрегувальні шари, повноз’єднаний шар є останнім шаром. З кожним шаром ЗНМ збільшується у своїй складності, ідентифікуючи більші частини зображення. Більш ранні шари зосереджені на простих елементах, таких як кольори та краї. Коли дані зображення просуваються через шари ЗНМ, вона починає розпізнавати більші елементи або форми об’єкта, поки нарешті не ідентифікує передбачуваний об’єкт (див. рис. 8) [26].

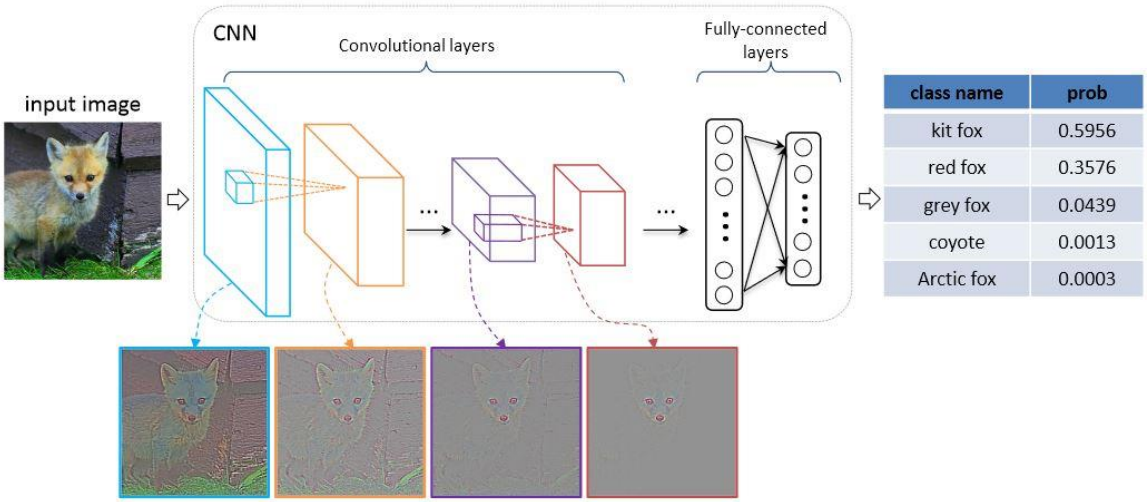


Рис. 8. Архітектура згорткової нейронної мережі. На вході – зображення, на виході – передбачення категорії.

2.1.1. Згортковий шар

Згортковий шар є основним будівельним блоком ЗНМ, і саме там відбувається більшість обчислень. Для цього потрібно кілька компонентів, які є вхідними даними, фільтром і картою ознак. Припустимо, що вхідним буде кольорове зображення, яке складається з матриці пікселів у 3D. Це означає, що вхідні дані будуть мати три виміри – висоту, ширину та глибину – які відповідають RGB в зображенні.

Також присутній детектор ознак, також відомий як ядро або фільтр, який буде переміщатися по сприйнятливих полях зображення, перевіряючи, чи присутня ознака. Цей процес відомий як згортка.

Детектор ознак – це двовимірний масив ваг, який представляє частину зображення. Хоча вони можуть відрізнятися за розміром, розмір фільтра зазвичай є матрицею 3x3; це також визначає розмір рецептивного поля. Фільтр застосовується до області зображення, і між вхідними пікселями та фільтром обчислюється точковий добуток. Цей точковий добуток потім подається у вихідний масив. Після цього фільтр зміщується на один крок, повторюючи процес, поки ядро не охопить усе зображення. Остаточний

вихід із ряду точкових добутків із входу та фільтра відомий як карта ознак, карта активації або згорнута функція.

Кожне вихідне значення на карті ознак не обов'язково має з'єднуватися з кожним значенням пікселя у вхідному зображенні (див. рис.9).

Варто зауважити, що ваги в детекторі ознак залишаються фіксованими, коли він рухається по зображенню, що також відомо як спільне використання параметрів.

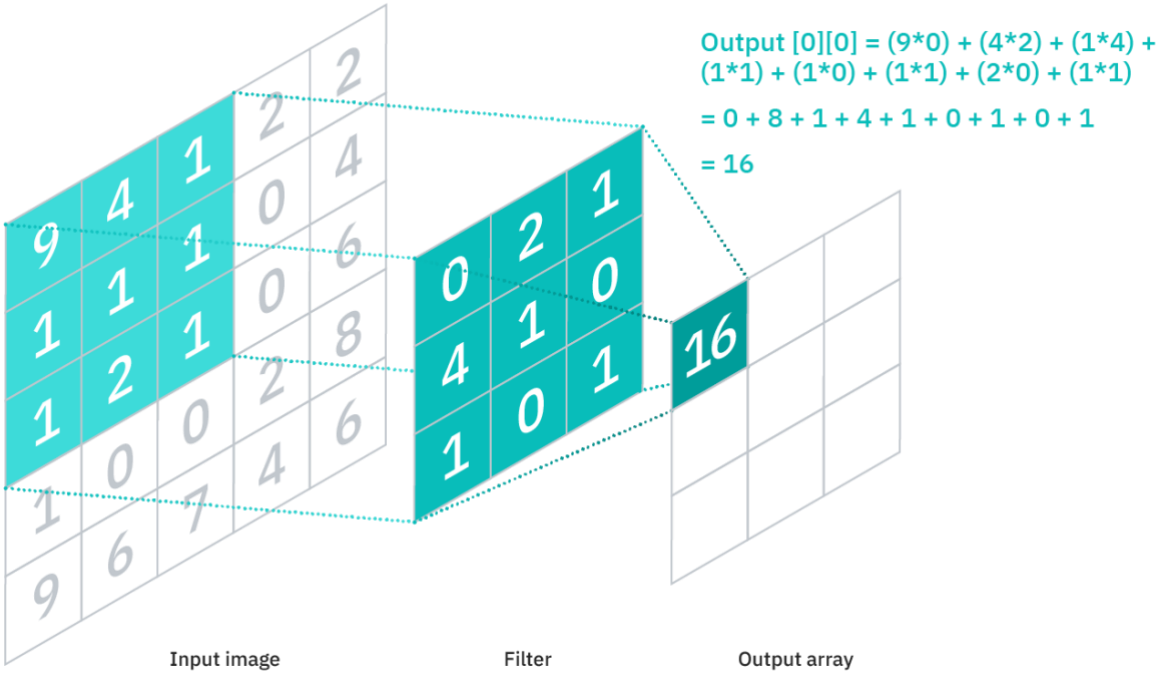


Рис. 9. Формування карти ознак

Деякі параметри, як-от значення ваги, коригуються під час навчання за допомогою процесу зворотного поширення та градієнтного спуску. Проте є три гіперпараметри, які впливають на розмір виводу, та які необхідно встановити перед початком навчання нейронної мережі. До них належать:

1. Кількість фільтрів. Вона впливає на глибину виводу. Наприклад, три різні фільтри дадуть три різні карти об'єктів, створюючи глибину у розміри три.

2. Крок – це відстань або кількість пікселів, на які ядро переміщається по вхідній матриці. Хоча значення кроку два або більше зустрічається рідко, більший крок дає менший результат.

3. Заповнення нулями зазвичай використовується, коли фільтри не підходять до вхідного зображення. Це обнуляє всі елементи, які виходять за межі вхідної матриці, створюючи більший або однаковий за розміром вихід. Існує три типи заповнення:

- Дійсний заповнення, що також відоме як відсутність заповнення. У цьому випадку остання згортка скидається, якщо розміри не співпадають.
- Таке ж заповнення – це заповнення гарантує, що вихідний шар має той самий розмір, що і вхідний шар
- Повне заповнення – цей тип заповнення збільшує розмір виводу шляхом додавання нулів до межі вхідних даних.

Після кожної операції згортки ЗНМ застосовує Rectified Linear Unit (ReLU) перетворення до карти ознак, вносячи нелінійність у модель.

Як вказано вище, інший шар згортки може слідувати за початковим шаром згортки. Коли це стається, структура ЗНМ може стати ієрархічною, оскільки пізні рівні можуть бачити пікселі в рецептивних полях попередніх шарів.

Як приклад, припустимо, що ми намагаємося визначити, чи містить зображення велосипед. Ми можемо думати про велосипед як про суму частин. Він складається з рами, керма, коліс, педалей тощо. Кожна окрема частина велосипеда складає шаблон нижнього рівня в нейронній мережі, а комбінація його частин представляє шаблон вищого рівня, створюючи ієрархію ознак у ЗНМ.

Зрештою, згортковий шар перетворює зображення в числові значення, дозволяючи нейронній мережі інтерпретувати та витягувати відповідні шаблони [27].

2.1.2. Агрегувальний шар

Агрегувальні шари, які ще називають зниженням дискретизації, понижують розмірність за допомогою зменшення кількості параметрів у вхідних даних. Подібно до згорткового шару, операція агрегування об’єднує фільтр по всьому входу, але різниця в тому, що цей фільтр не має жодних ваг. Замість цього ядро застосовує функцію агрегації до значень у рецептивному полі, заповнюючи вихідний масив. Існує два основних типи агрегування:

- Максимальне: під час переміщення фільтра по входу він вибирає піксель з максимальним значенням для відправки на вихідний масив. Крім того, цей підхід, як правило, використовується частіше в порівнянні зі середнім агрегуванням;
- Середнє: під час переміщення фільтра по входу, він обчислює середнє значення в поле сприйняття для відправлення на вихідний масив.

Хоча багато інформації втрачається в агрегувальному шарі, він також має ряд переваг для ЗНМ. Вони допомагають зменшити складність, підвищити ефективність та обмежити ризик перенавчання [27].

2.1.3. Повноз’єднаний шар.

Назва повноз’єднаного шару влучно описує його. Як згадувалося вище, значення пікселів вхідного зображення не пов’язані безпосередньо з вихідним шаром у частково з’єднаних шарах. Однак у повноз’єднаному шарі кожен вузол вихідного шару підключається безпосередньо до вузла попереднього шару.

Цей шар виконує завдання класифікації на основі ознак, витягнутих через попередні шари та їх різні фільтри. У той час як згорткові та агрегувальні шари, як правило, використовують функції ReLu, рівні повноз’єднані шари зазвичай використовують функцію активації softmax для належної класифікації вхідних даних, створюючи ймовірність від 0 до 1 [27].

2.2. Типи згорткових нейронних мереж

Куніхіко Фукусіма та Ян Лекун у своїй роботі 1980 року заклали основу дослідження згорткових нейронних мереж. Янн Лекун успішно застосував зворотне поширення для навчання нейронних мереж для визначення та розпізнавання шаблонів у серії рукописних поштових індексів. Він продовжив дослідження разом зі своєю командою протягом 1990-х років, кульмінацією якого був «LeNet-5», який застосовував ті самі принципи попередніх досліджень для розпізнавання документів. Відтоді з'явилася низка варіантів архітектури ЗНМ із введенням нових наборів даних, таких як MNIST і CIFAR-10, а також конкурсів, таких як ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [27]. Деякі з запропонованих архітектур:

- AlexNet;
- VGGNet;
- GoogLeNet;
- ResNet;
- ZFNet.

2.2.1. Згорткова нейронна мережа AlexNet

Архітектура складається з восьми шарів: п'яти згорткових шарів і трьох повноз'єднаних шарів (див. рис. 10).

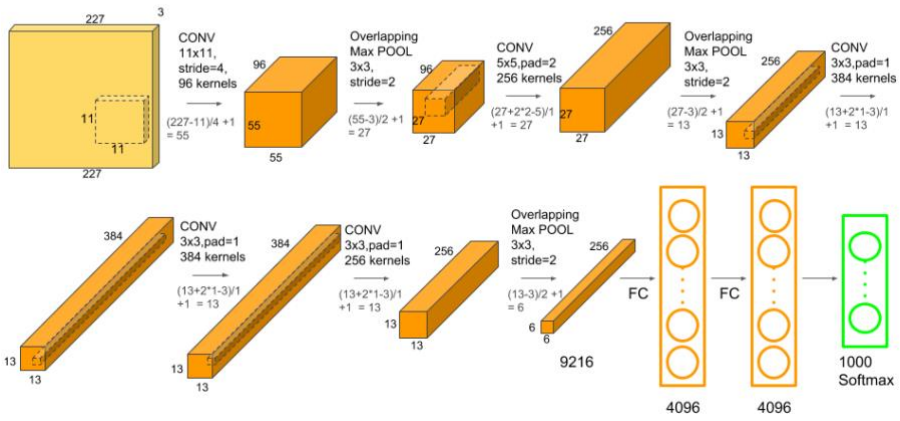


Рис. 10. Архітектура мережі AlexNet

Проте не тільки його архітектура робить AlexNet унікальним; ось деякі з використаних нових підходів до згорткових нейронних мереж:

- Нелінійність ReLU. AlexNet використовує Rectified Linear Units (ReLU) замість функції tanh, яка була стандартною на той час. Перевага ReLU полягає в часі навчання; ЗНМ, що використовує ReLU, може досягти 25% помилки в наборі даних CIFAR-10 в шість разів швидше, ніж ЗНМ, що використовує tanh.
- Кілька графічних процесорів. У той час графічні процесори все ще мали 3 гігабайти пам’яті. Це було особливо погано, оскільки навчальний набір містив 1,2 мільйона зображень. AlexNet дозволяє тренуватися з кількома GPU, поміщаючи половину нейронів моделі на один GPU, а іншу половину на інший GPU. Це не тільки дозволяє навчати більшу модель, але й скорочує час навчання.
- Агрегування з перекриттям. ЗНМ традиційно «агрегують» виходи сусідніх груп нейронів без перекриття. Однак, коли автори ввели перекриття, вони помітили зменшення похибки приблизно на 0,5% і виявили, що моделі з перекриттям агрегування зазвичай важче перенавчити.

Проблема перенавчання. У AlexNet було 60 мільйонів параметрів, що стало серйозною проблемою з точки зору перенавчання. Для його зменшення використовували два методи:

- Збільшення даних. Автори використовували трансформацію, що зберігає мітки, щоб зробити свої дані більш різноманітними.
- Опускання. Ця техніка полягає у «вимкненні» нейронів із заздалегідь визначеною ймовірністю (наприклад, 50%). Це означає, що кожна ітерація використовує різну вибірку параметрів моделі, що змушує кожен нейрон мати більш надійні функції, які можна використовувати з іншими випадковими нейронами [28].

2.2.2. Згорткова нейронна мережа GoogLeNet

Inception Network стала одним із головних проривів у галузі нейронних мереж, особливо для ЗНМ. Поки що існує три версії Inception Network, які називаються Inception Version 1, 2 і 3.

Перша версія з’явилася в 2014 році, і, як випливає з назви «GoogleNet», вона була розроблена командою Google. Ця мережа відповідала за встановлення нового сучасного рівня класифікації та виявлення в ILSVRC. Ця перша версія мережі Inception називається GoogleNet.

Якщо мережа побудована з великою кількістю глибоких шарів, вона може зіткнутися з проблемою перенавчання. Щоб вирішити цю проблему, автори запропонували архітектуру GoogleNet з ідеєю мати фільтри кількох розмірів, які можуть працювати на одному рівні. Завдяки цій ідеї мережа насправді стає ширшою, а не глибшою. Нижче наведено зображення, що показує Naive Inception модуль (див. рис. 11).

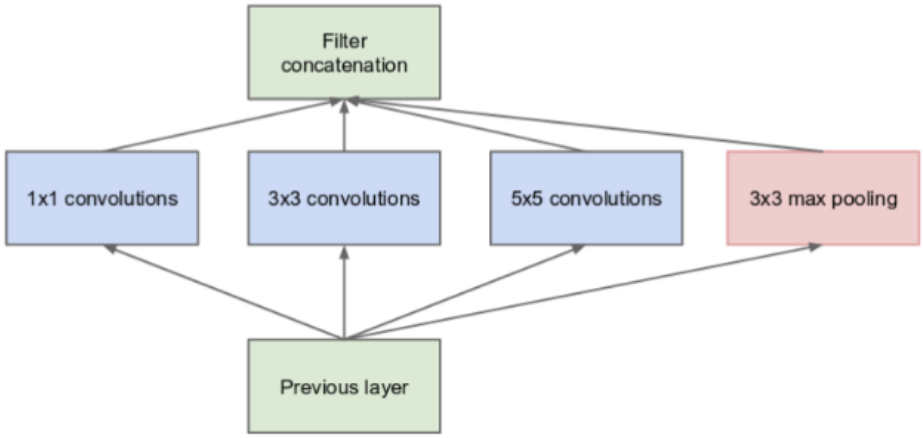


Рис. 11. Naive Inception модуль ЗНМ GoogLeNet

Як видно на діаграмі вище, операція згортки виконується на входах з трьома розмірами фільтрів: (1 × 1), (3 × 3) і (5 × 5). Операція максимального агрегування також виконується зі згортками, а потім надсилається в наступний початковий модуль.

Оскільки навчання нейронних мереж займає багато часу та дороге, автори обмежили кількість вхідних каналів, додавши додаткову (1 × 1) згортку перед згортками (3 × 3) і (5 × 5), щоб зменшити розміри мережі та виконувати більш швидкі обчислення (див. рис. 12).

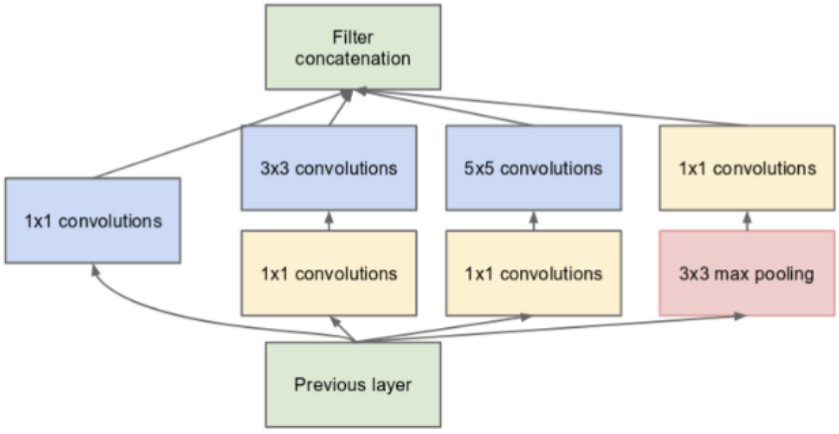


Рис. 12. Naive Inception модуль ЗНМ GoogLeNet з модифікацією

Це будівельні блоки GoogleNet. Архітектура GoogleNet складається з 22 шарів із 27 шарами агрегування. Всього є 9 Inception модулів, складених лінійно. Кінці цих модулів з’єднані з глобальним середнім шаром агрегування [29].

2.2.3. Згорткова нейронна мережа VGG

VGG означає Visual Geometry Group; це стандартна архітектура глибокої згорткової нейронної мережі з кількома шарами. Слово «глибока» відноситься до кількості шарів у VGG-16 або VGG-19, що складаються з 16 і 19 згорткових шарів.

Архітектура VGG є основою інноваційних моделей розпізнавання об'єктів. Розроблена як глибока нейронна мережа, VGGNet також перевершує середні результати для багатьох завдань і наборів даних за межами ImageNet. Більше того, зараз це одна з найпопулярніших архітектур розпізнавання зображень.

Модель VGG, або VGGNet, яка підтримує 16 шарів, також називається VGG16. Ця модель глибокої згорткової нейронної мережі була запропонована А. Зіссерманом і К. Сімоньяном з Оксфордського університету. Модель VGG16 досягає майже 92,7% точності тестів у топ-5 у ImageNet. Вона замінює фільтри великого розміру ядра кількома фільтрами розміру ядра 3×3 один за одним, тим самим роблячи значні покращення порівняно з AlexNet. Модель VGG16 навчалася за допомогою графічних процесорів Nvidia Titan Black протягом кількох тижнів.

Концепція моделі VGG19 (також VGGNet-19) така ж, як і VGG16, за винятком того, що вона підтримує 19 шарів. «16» і «19» означають кількість вагових шарів у моделі. Це означає, що VGG19 має на три згорткові шари більше, ніж VGG16 [30].

Архітектура VGG. Існує кілька шарів згортки, за якими слідує шар об’єднання, який зменшує висоту та ширину. Якщо ми подивимося на кількість фільтрів, які ми можемо використовувати, доступно близько 64 фільтрів, які ми можемо подвоїти приблизно до 128, а потім до 256 фільтрів. В останніх шарах ми можемо використовувати 512 фільтрів (див. рис. 13).

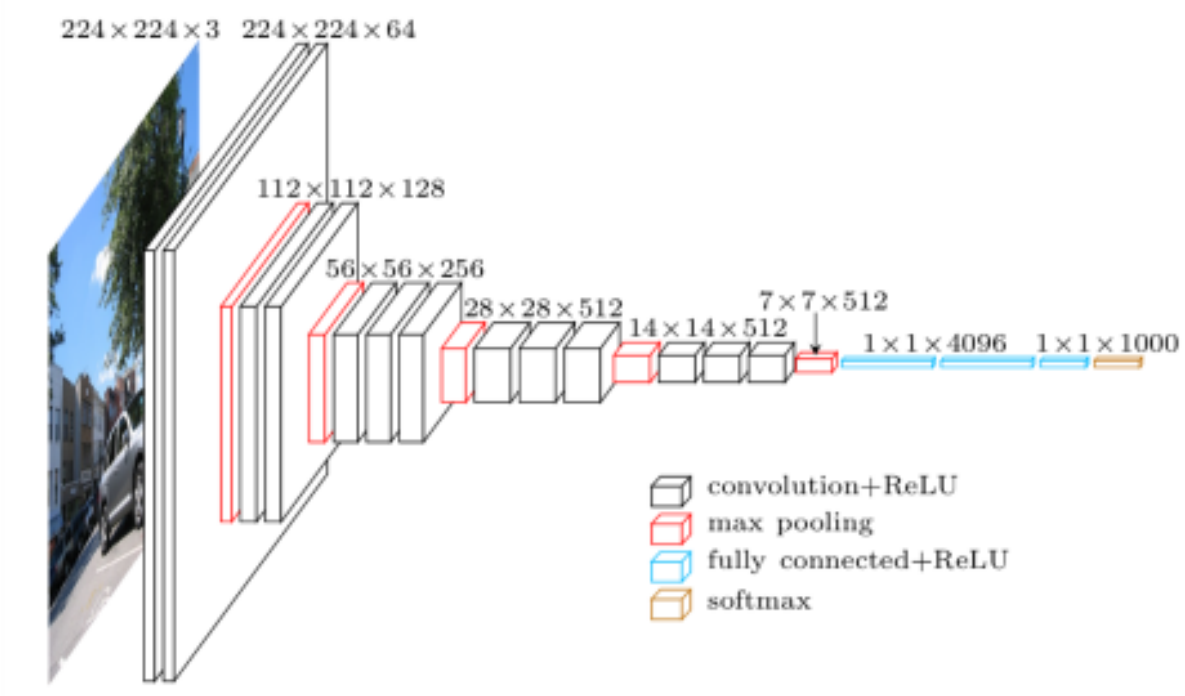


Рис. 13. Архітектура мережі VGG16

Згорткові шари VGG використовують мінімальне сприйнятливе поле, тобто 3×3, найменший можливий розмір, який все ще захоплює вгору/вниз і вліво/вправо. Крім того, існують також фільтри згортки 1×1, які діють як лінійне перетворення входу. Далі йде модуль ReLU, який є величезною інновацією від AlexNet, що скорочує час навчання. Крок згортки фіксується на рівні 1 піксель, щоб зберегти просторову роздільну здатність після згортки.

Усі приховані шари в мережі VGG використовують ReLU. VGG зазвичай не використовує локальну нормалізацію відповіді, оскільки це збільшує споживання пам’яті та час навчання, а також не покращує загальну точність.

VGGNet має три повноз’єднані підключені шари. З трьох шарів перші два мають по 4096 каналів кожен, а третій має 1000 каналів, по 1 для кожного класу.

Для вирішення задачі нейронного перенесення стилю нами буде використано мережу VGG19, що має 16 згорткових і 5 об’єднаних шарів, адже вона показує одні з найкращих результатів в задачі розпізнавання зображень, що є дотичним до визначення змісту зображення та подальшого відділення його від стилю.

На високому рівні, щоб мережа могла виконувати класифікацію зображень – вона повинна розуміти зображення. Це означає взяти вихідне зображення як вхідні пікселі та створити внутрішнє представлення, що перетворює необроблені пікселі зображення в комплексне розуміння ознак, присутніх у зображенні [26].

2.3. Модель нейронного перенесення стилю

Для навчання моделі передачі стилю потрібні дві мережі: попередньо навчений екстрактор функцій і мережа передачі.

НПС використовує попередньо навчену модель, навчену на ImageNet-VGG. Самі зображення не мають сенсу для моделі. Їх потрібно перетворити в необроблені пікселі та передати моделі, щоб перетворити їх в набір ознак, за що і відповідають згорткові нейронні мережі.

Таким чином, десь посередині між шарами, де зображення подається в модель, і шаром, який дає вихід, модель виконує роль комплексного витягувача ознак. Все, що нам потрібно отримати від моделі, це її проміжні шари, а потім використовувати їх для опису змісту та стилю вхідних зображень.

Вхідне зображення перетворюється на представлення, які мають більше інформації про зміст зображення, а не детальне значення пікселя.

Характеристики, які ми отримуємо від вищих рівнів моделі, можна вважати більш пов’язаними із змістом зображення. Щоб отримати уявлення стилю еталонного зображення, ми використовуємо кореляцію між різними відповідями фільтра (див. рис 14) [31].

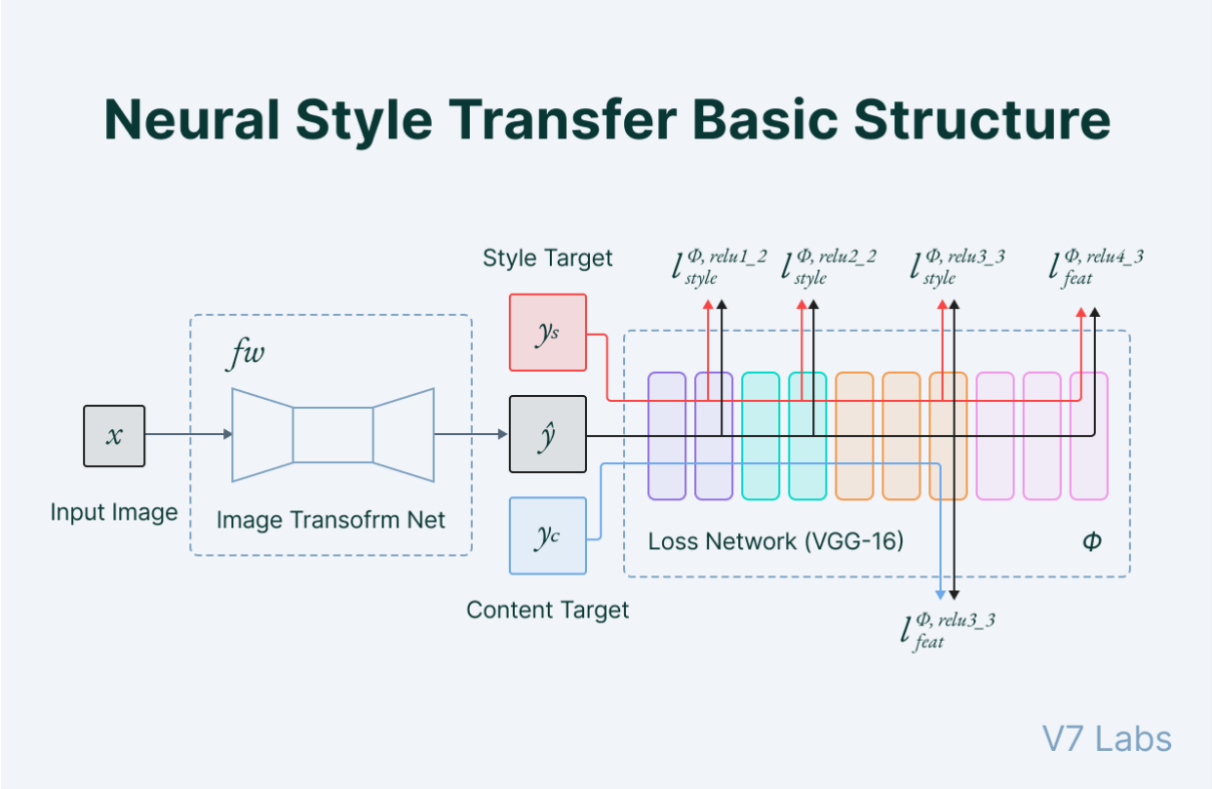


Рис. 14. Базова структура моделі НПС

2.3.1. Втрата змісту

Втрата змісту допомагає встановити схожість між зображенням змісту та створеним зображенням. Інтуїтивно зрозуміло, що вищі шари моделі більше зосереджуються на ознаках, присутніх у зображенні, тобто на загальному змісті зображення.

Втрата змісту розраховується за евклідовою відстанню між відповідним проміжним представленням об’єкта вищого рівня вхідного зображення (x) і зображення вмісту (p) на шарі l (2.1):

$$L_{content}^{l}(p,x) = \sum_{i,j}(F_{ij}^{l}(x) - P_{ij}^{l}(p))^2 \qquad (2.1)$$

Для моделі природно створювати різні карти ознак у вищих шарах, які активуються в присутності різних об’єктів. Це допомагає зробити висновок, що зображення з однаковим змістом також повинні мати подібні активації у вищих шарах.

2.3.2. Втрата стилю. Матриці Грама.

Втрата стилю концептуально відрізняється від втрати вмісту. Не можливо просто порівняти проміжні характеристики двох зображень і отримати втрату стилю. Тому і необхідне використання матриць Грама.

Матрицею Грама системи векторів $e1, e2, \dots, en$ називається матриця, що визначена таким чином (2.2):

$$G(e_1, e_2, \dots, e_n) = \begin{pmatrix} (e_1, e_1) & \dots & (e_1, e_n) \\ \vdots & \ddots & \vdots \\ (e_n, e_1) & \dots & (e_n, e_n) \end{pmatrix} \qquad (2.2)$$

Основою цього методу є обчислення квадратичної норми різниці матриць Грама цільового зображення та зображення стилю.

Величину скалярного добутку векторів можна вважати мірою їх подібності, масштабованою щодо норм векторів.

Матриця Грама – це спосіб інтерпретації інформації про стиль зображення, оскільки вона показує загальний розподіл ознак у даному шарі. Він вимірюється як величина кореляції, наявної між картами об’єктів у даному шарі (див. рис. 15) [31].

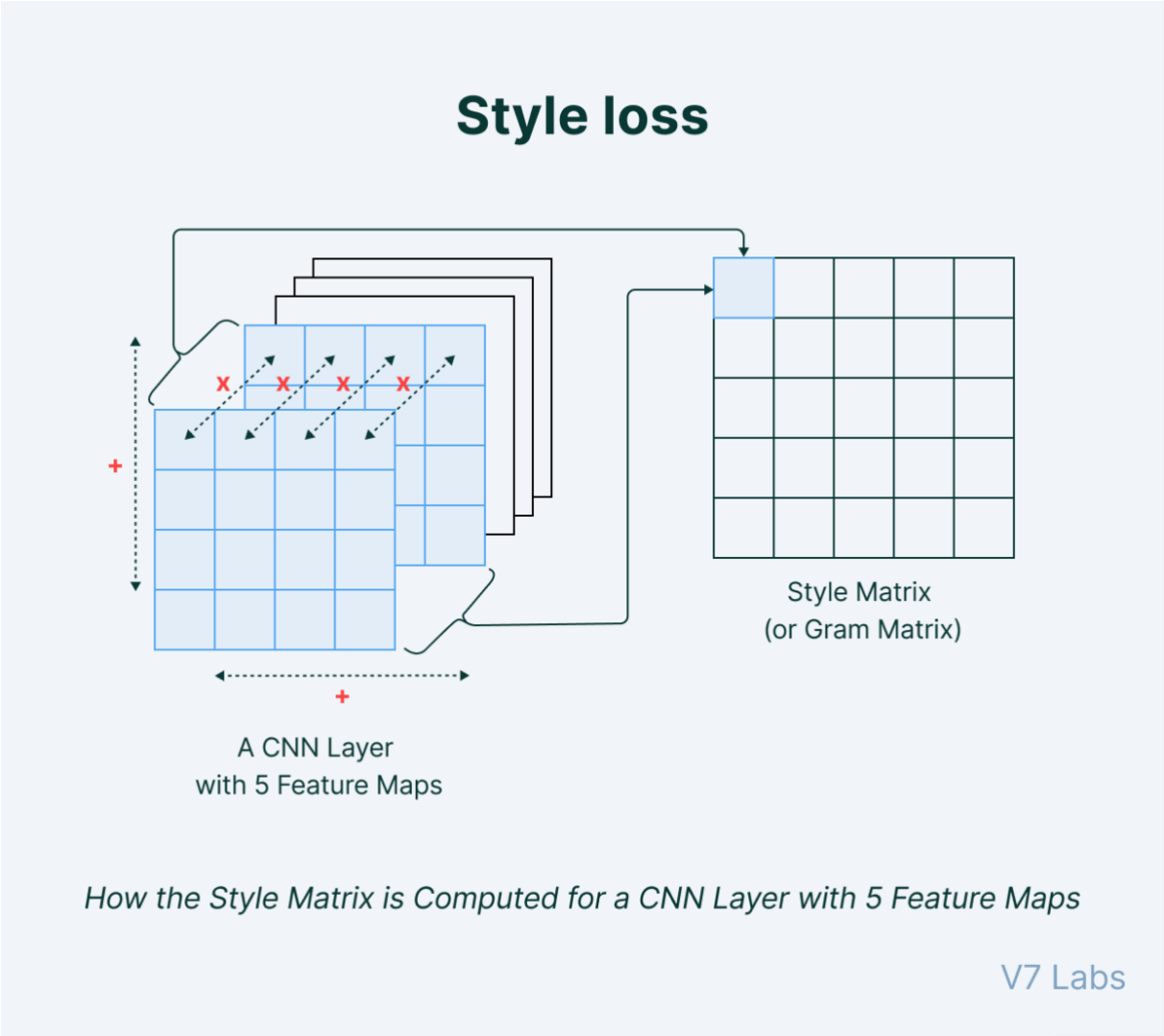


Рис. 15. Розрахунок стильової матриці у НПС

Втрата стилю розраховується за відстанню між грам-матрицями (або, іншими словами, представленням стилю) створеного зображення та еталонного зображення стилю. Внесок кожного шару в інформацію про стиль розраховується за наступною формулою (2.3):

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l \right)^2 \quad (2.3)$$

Таким чином, загальна втрата стилю на кожному шарі виражається як (2.4):

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l \quad (2.4)$$

де внесок кожного шару у втрату стилю зображується деяким фактором w_l.

2.4. Оптимізаційні алгоритми

Алгоритми машинного навчання часто покладаються на оптимізацію певної цільової функції, тому вибір алгоритму оптимізації є важливою частиною процесу навчання.

Методи, які використовують лише градієнт або першу похідну для пошуку оптимуму, називають методами першого порядку. Найпростішим таким алгоритмом є стохастичний градієнтний спуск, який робить кроки в напрямку негативного градієнта цільової функції. Цей алгоритм продемонстрував, що хоча він і простий, проте він досить добре працює в задачах машинного навчання. Більш складним методом першого порядку є алгоритм ADAM, який розроблений так, щоб бути менш чутливим до шумних і малих градієнтів [32].

2.4.1. ADAM

Назва ADAM походить від методу, використаного в алгоритмі, а саме адаптивних моментів. Алгоритм ADAM є методом оптимізації першого порядку для стохастичних цільових функцій, який базується на адаптивних оцінках моментів першого і другого порядку. Правило оновлення в алгоритмі засноване на експоненціально спадних ковзних середніх градієнта g і квадрата градієнта $g \odot g$. Ковзні середні, позначені s і r, обчислюються за допомогою (2.5):

$$\begin{aligned} \boldsymbol{s} &= p_1 \boldsymbol{s} + (1 - p_1)\boldsymbol{g} \\ \boldsymbol{r} &= p_2 \boldsymbol{r} + (1 - p_2)\boldsymbol{g} \odot \boldsymbol{g} \end{aligned} \tag{2.5}$$

і фактично є оцінками моментів першого та другого порядку. Алгоритм вводить два нові гіперпараметри *ρ1* і *ρ2*, які визначають, наскільки швидко вноситься внесок попередніх ітерацій. Оцінки s і r ініціалізуються до нуля і тому зміщені до нуля [32].

2.4.2. L-BFGS

BFGS з обмеженою пам’яттю (L-BFGS) – це алгоритм оптимізації в сімействі квазіньютоновських методів, який наближається до алгоритму Бройдена-Флетчера-Голдфарба-Шанно (BFGS), використовуючи обмежений обсяг пам’яті комп’ютера.

Це популярний алгоритм для оцінки параметрів у машинному навчанні. Цільова задача алгоритму полягає в тому, щоб мінімізувати $f(x)$ над необмеженими значеннями реального вектора x, де f – диференційована скалярна функція.

Як і оригінальний BFGS, L-BFGS використовує оцінку оберненої матриці Гессе, щоб спрямовувати свій пошук у змінному просторі, але L-BFGS зберігає лише кілька векторів, які неявно представляють

апроксимацію. У зв’язку з тим, що він потребує лінійної пам’яті, метод L-BFGS особливо добре підходить для задач оптимізації з багатьма змінними [32].

Висновки до другого розділу

Отже, дослідивши теоретичні засоби вирішення задачі, ми можемо обрати засоби, що забезпечать ефективне досягнення поставленої мети нашого дослідження, а саме створення інформаційної технології нейронного перенесення стилю.

Основою нашої моделі слугуватиме ЗНМ, адже згорткові нейронні мережі здатні вловлювати інваріантності та визначальні ознаки в межах класів. Таким чином, десь між тим, де вихідне зображення подається в модель, і вихідною міткою класифікації, модель служить як складний екстрактор ознак. Отримавши доступ до проміжних шарів моделі, ми зможемо отримати зміст і стиль вхідних зображень.

Нами було обрано модель VGG19, що є однією з найкращих мереж для класифікації зображень, а отже і гарною моделлю для екстракції ознак змісту та стилю.

Загальна функція втрати та функції втрати стилю та змісту були взяті з роботи Гатіса та ін.

У якості оптимізаційного алгоритму було обрано алгоритм ADAM.

РОЗДІЛ 3. ПОБУДОВА МОДЕЛІ НПС. РЕЗУЛЬТАТИ ЇЇ РОБОТИ.

3.1. Вибір мови програмування

В якості мови програмування для реалізації інформаційної технології було обрано Python 3 в середовищі Jupyter від Google Colab.

Основними характеристиками Python є:

- високорівневість – Python має синтаксис, схожий на англійську. Це полегшує читання та розуміння коду;
- інтерпретовність – Python безпосередньо виконує код рядок за рядком. У разі будь-якої помилки він зупиняє подальше виконання та повідомляє про помилку, яка сталася;
- багатофункціональність – стандартна бібліотека Python величезна, ви можете знайти майже всі функції, необхідні для вашого завдання. Отже, вам не потрібно залежати від зовнішніх бібліотек.

Python – це інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічним типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості мови сценаріїв або склеювання для з’єднання існуючих компонентів разом. Простий синтаксис Python, який легко вивчати, підкреслює читабельність і, отже, знижує витрати на обслуговування програми. Python підтримує модулі та пакунки, що сприяє модульності програм і повторному використанню коду. Інтерпретатор Python і велика стандартна бібліотека доступні у вихідній або двійковій формі безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Оскільки етапу компіляції немає, цикл редагування-тестування-дебагування неймовірно швидкий. Дебагувати програми написані на Python легко: помилка або неправильне введення ніколи не призведе до помилки

сегментації. Натомість, коли інтерпретатор виявляє помилку, він створює виняток. Якщо програма не вловлює виняток, інтерпретатор друкує трасування стека. Дебагер на рівні вихідного коду дозволяє перевіряти локальні та глобальні змінні, оцінювати довільні вирази, встановлювати точки зупину, переходити через код по рядку і так далі. Дебагер написаний на самому Python, що свідчить про інтроспективну силу Python [33].

Google Colab – це онлайн-середовище Jupyter Notebooks від Google. Він виконує все, що робив би локальний ноутбук (і більше), але він знаходиться в хмарі, тому не потрібно встановлювати програмне забезпечення, і воно доступне з будь-якого комп'ютера, підключеного до Інтернету.

Наразі воно працює лише на Python 3 і має велику кількість найпопулярніших бібліотек Python.

Google Colab дозволяє використовувати потужності GPU для вирішення задач пов'язаних з машинним навчанням, що в нашому випадку є особливо корисним, адже ми працюємо з зображеннями і глибокими згортковими мережами. Використання GPU дозволяє отримувати результати значно швидше [34].

3.2. Використання бібліотек та модулів Python

В рамках реалізації інформаційної технології було використано бібліотеки та модулі для мови Python. Їх опис наведено нижче.

- os – модуль, що надає розробнику можливість використовувати функції взаємодії з операційною системою. належить до стандартних модулів-утиліт Python. Цей модуль пропонує портативний спосіб використання функціоналу, що є специфічним для операційної системи.

- functools – модуль призначений для функцій вищого порядку: функцій, які діють або повертають інші функції. Загалом, будь-який викликаний об’єкт можна розглядати як функцію для цілей цього модуля.
- time –модуль надає різні функції, пов’язані з часом.

3.2.1. TensorFlow

TensorFlow – це бібліотека з відкритим вихідним кодом, розроблена Google в основному для програм глибокого навчання. Вона також підтримує традиційне машинне навчання. TensorFlow спочатку був розроблений для великих чисельних обчислень без урахування глибокого навчання. Однак він виявився дуже корисним і для розробки глибокого навчання, і тому Google відкрив доступ до нього всім охочим.

TensorFlow приймає дані у вигляді багатовимірних масивів вищих вимірів, які називаються тензорами. Багатовимірні масиви дуже зручні при обробці великих обсягів даних.

TensorFlow працює на основі графів потоків даних, це дозволяє набагато легше виконувати код розподілено на кластері комп’ютерів з використанням графічних процесорів.

Програми глибокого навчання дуже складні, а процес навчання вимагає багато обчислень. Це все займає багато часу через великий розмір даних, велику кількість ітераційних процесів, математичні обчислення, множення матриці тощо. Особливо, якщо виконувати ці дії на звичайному центральному процесорі.

Графічні процесори (GPU) популярні в контексті ігор, де потрібно, щоб екран і зображення мали високу роздільну здатність. Однак вони також використовуються для розробки програм глибокого навчання.

Однією з основних переваг TensorFlow є те, що він підтримує графічні процесори, а також центральні процесори. Він також має швидший час

компіляції, ніж інші бібліотеки глибокого навчання, такі як Keras і Torch [35].

3.2.2. Matplotlib

Matplotlib – одна з найуспішніших і часто використовуваних бібліотек, яка надає різноманітні інструменти для візуалізації даних на Python.

Це одна з найпотужніших бібліотек графіків у Python. Вона надає різні інструменти для створення двовимірних графіків із даних у списках або масивах.

Matplotlib був створений Джоном Д. Хантером на мові програмування Python у 2003 році. Поточна стабільна версія matplotlib – 3.4.2, випущена 8 травня 2021 року. Вона використовує NumPy, бібліотеку, яка надає числове математичне розширення для Python.

Ця бібліотека дає користувачеві можливість візуалізувати дані, використовуючи різноманітні типи графіків, щоб зробити дані зрозумілими. Можна використовувати ці різні типи діаграм (діаграми розсіювання, гістограми, стовпчасті діаграми, діаграми помилок, діаграми з рамкою тощо), написавши кілька рядків коду на python.

Пакет matplotlib можна використовувати в будь-якій оболонці Python, оболонці IPython, ноутбуку Jupyter, лабораторії jupyter, хмарі і тд.

matplotlib.pyplot – це інтерфейс для matplotlib на основі стану. Це набір функцій командного стилю, які змушують matplotlib працювати як MATLAB. Кожна функція pyplot вносить деякі зміни до графіка [36].

3.2.3. NumPy

NumPy – це фундаментальний пакет для наукових обчислень на Python. Це бібліотека Python, яка надає багатовимірний об’єкт масиву, різноманітні похідні об’єкти (наприклад, замасковані масиви та матриці), а також набір підпрограм для швидких операцій над масивами, включаючи

математичні, логічні, маніпуляції з формою, сортування, вибір, введення/виводу, дискретні перетворення Фур'є, базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого.

В основі пакету NumPy лежить об'єкт ndarray. Він інкапсулює n-вимірні масиви однорідних типів даних, при цьому багато операцій виконуються в скомпільованому коді для підвищення продуктивності. Існує кілька важливих відмінностей між масивами NumPy і стандартними послідовностями Python:

- Масиви NumPy мають фіксований розмір під час створення, на відміну від списків Python (які можуть динамічно зростати). Зміна розміру ndarray створить новий масив і видалить оригінал.
- Усі елементи в масиві NumPy повинні мати один тип даних і, таким чином, мати однаковий розмір пам’яті. Виняток: можна мати масиви об’єктів (Python, включаючи NumPy), що дозволяє використовувати масиви елементів різного розміру.
- Масиви NumPy полегшують розширені математичні та інші типи операцій над великою кількістю даних. Зазвичай такі операції виконуються ефективніше і з меншою кількістю коду, ніж це можливо за допомогою вбудованих послідовностей Python.

Все більша кількість науково-математичних пакетів на основі Python використовує масиви NumPy.

3.3. Архітектура програми

В основі програми лежить застосування мережі VGG19 для отримання активаційних карт із шарів. Ця мережа для класифікації зображень була попередньо навчена на датасеті «imagenet» (див. рис. 16).

```
#завантажуємо, попердньо навчену на imagenet, мережу VGG19 без класифікаційної верхівки
x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
x = tf.image.resize(x, (224, 224))
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

print()
for layer in vgg.layers:
  print(layer.name)
```

Рис. 16. Завантаження мережі VGG19

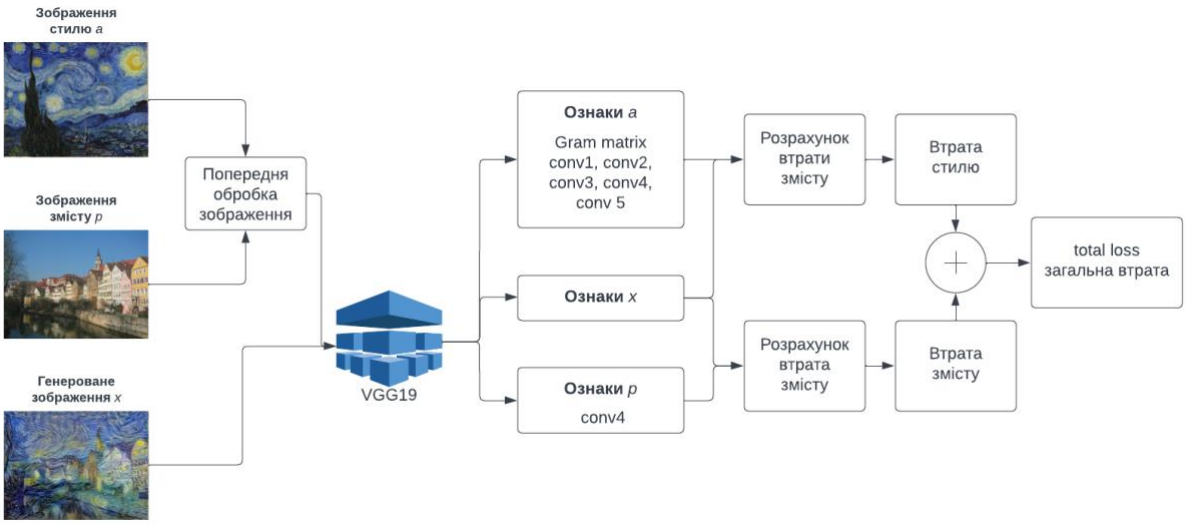


Рис. 17. Архітектура програми

Ми використовуємо проміжні шари моделі, щоб отримати представлення змісту та стилю зображення (див. рис. 17).

Так як стиль зображення можна описати за допомогою засобів і співвідношень на різних картах ознак, ми обчислюємо матрицю Грама, яка містить цю інформацію, взявши зовнішній добуток векторизованих карт ознак в кожному місці та усереднивши цей зовнішній добуток для всіх

місць. Цю матрицю Грама можна розрахувати для конкретного шару як (3.1):

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

(3.1)

Ми спільно мінімізуємо відстань представлення ознак зображення білого шуму від представлення змісту фотографії в одному шарі та представлення стилю картини, визначеного на кількох шарах згорткової нейронної мережі.

Функцію втрат ми мінімізуємо за формулою (3.2):

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{змісту}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{стилю}}(\vec{a}, \vec{x}) \quad (3.2)$$

де α і β – вагові коефіцієнти для реконструкції змісту та стилю,

p – оригінальне зображення змісту,

a – оригінальне зображення стилю,

x – генероване зображення.

При цьому втрату змісту для шару *l* ми обчислюємо по формулі vквадрата помилки між представленнями ознак оригінального і генерованого зображення (3.3):

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

(3.3)

А втрату стиля для шару *l* завдяки використанню градієнтного спуску від зображення білого шуму для мінімізації середньоквадратичної відстані

між записами матриць Грама вихідного та генерованого зображення за формулою (3.4):

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l \right)^2 \tag{3.4}$$

З якої загальна втрата стилю обчислюється за формулою (3.5):

$$\mathcal{L}_{\text{стилю}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \tag{3.5}$$

де w_l – вагові коефіцієнти внеску кожного шару в загальні втрати.

Код реалізації алгоритму наведено у Додатку 1.

3.4. Виконані експерименти

Для перевірки та підтвердження ефективності описаного методу було проведено декілька експериментів НПС, при таких змінних:

- α і β – вагові коефіцієнти для реконструкції змісту та стилю;
- steps_per_epoch – кроки на ітерацію;
- epochs – кількість ітерацій.

План наших експериментів передбачає перевірку залежності результатів від змінних.

Оригінальне зображення змісту було однакове у всіх експериментах на наведено в додатках (див. рис. 18.1).

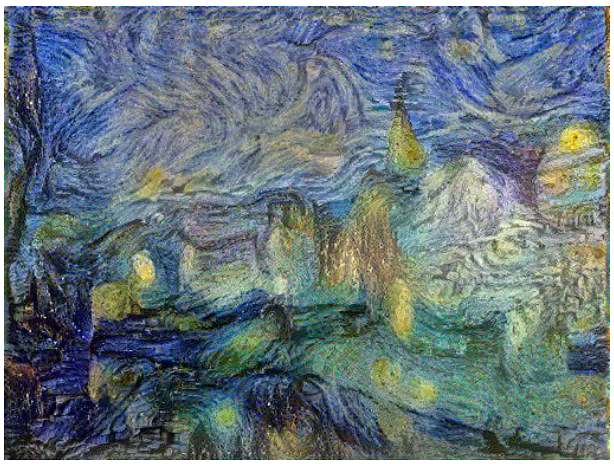
3.4.1. Зображення стилю – «Зоряна ніч»

Для першого експерименту за оригінальне зображення стилю було взято картину Ван Гога «Зоряна ніч» (див. рис. 18.2)

Рис. 18. (1) оригінальне зображення змісту, (2) зображення стилю

1) Порівняння впливу кроків епохи:
   - α і β – 1 та 0.1;
   - steps_per_epoch – 50 (а), 100 (б), 200 (в).
   - epochs – 10.

Результат (див. рис. 19).

(а) 

(б) 

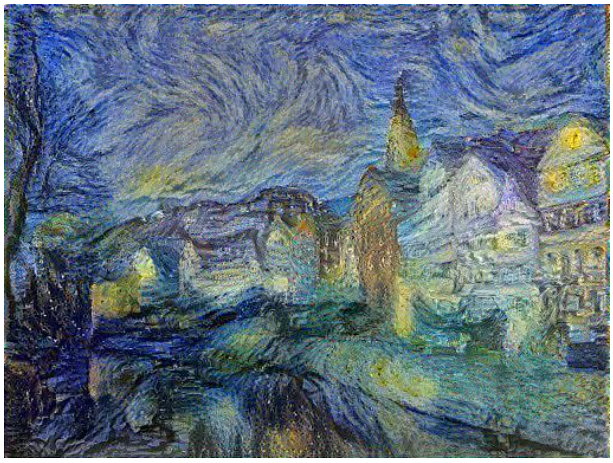
(в) 

Рис. 19. Результат роботи методу при зміні кроків на ітерацію: (а) 50, (б) 100, (в) 200.

2) Порівняння впливу вагових коефіцієнтів:

- α і β – 1 та 0.01 (а); 1 та 0.001 (б); 1 та 0.00001 (в).
- steps_per_epoch – 100;
- epochs – 10.

Результат (див. рис. 20).

(а) 

(б) 

(в) 

Рис. 20. Результат роботи методу при зміні вагових коефіцієнтів: (а) 1 та 0.01; (б) 1 та 0.001; (в) 1 та 0.00001.

3.4.2. Зображення стилю – «Соняшники»

Для другого експерименту за оригінальне зображення стилю було взято картину Ван Гога «Соняшники» (див. рис. 21.2)

Рис. 21. (1) оригінальне зображення змісту, (2) зображення стилю

Параметри експерименту:

- α і β – 1 та 0.01 (а); 1 та 0.001 (б).
- steps_per_epoch – 100;
- epochs – 10.

Результат (див. рис. 22).

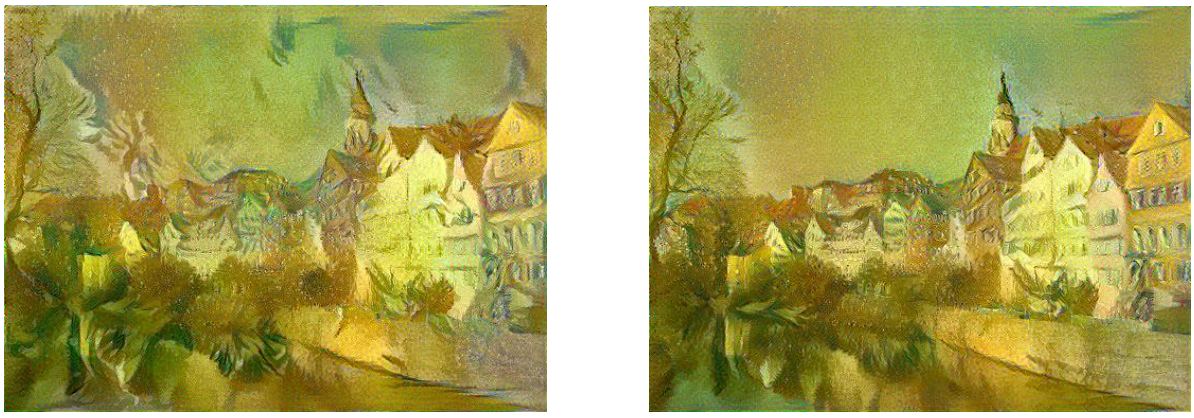


Рис. 22. результат роботи методу при зміні вагових коефіцієнтів змісту та стилю (а) 1 та 0.01; (б) 1 та 0.001.

3.4.3. Зображення стилю – «Авіньйонські дівиці»

Для третього експерименту за оригінальне зображення стилю було взято картину Пікассо «Авіньйонські дівиці» (див. рис. 23.2)

Рис. 23. (1) оригінальне зображення змісту, (2) зображення стилю

Параметри експерименту:

- α і β – 1 та 0.01 (а); 1 та 0.001 (б).
- steps_per_epoch – 100;
- epochs – 10.

Результат (див. рис. 24).

Рис. 24. результат роботи методу при зміні вагових коефіцієнтів змісту та стилю (а) 1 та 0.01; (б) 1 та 0.001.

3.4.4. Зображення стилю – «Сад земних насолод»

Для четвертого експерименту за оригінальне зображення стилю було взято картину Босха «Сад земних насолод» (див. рис. 25.2)

Рис. 25. (1) оригінальне зображення змісту, (2) зображення стилю

Параметри експерименту:

- α і β – 1 та 0.01 (а); 1 та 0.001 (б).
- steps_per_epoch – 100;
- epochs – 10.

Результат (див. рис. 26).

Рис. 26. результат роботи методу при зміні вагових коефіцієнтів змісту та стилю (а) 1 та 0.01; (б) 1 та 0.001

Висновок до третього розділу

Отже, на основі засобів і методів окреслених у другому розділі ми розробили інформаційну технологію нейронного перенесення стилю на Python.

Ми використали такі популярні у сфері машинного навчання бібліотеки як: TensorFlow, Matplotlib, NumPy. Це значно спростило реалізацію нашого алгоритму та покращило його ефективність.

Основою алгоритму стала мережа VGG19 без верхнього шару класифікації, що і слугувала засобом екстракції ознак стилю та змісту з вхідних зображень.

Ітераційна мінімізація загальної функції втрат дозволила нам отримати якісні результати роботи алгоритму НПС в результаті проведення декількох експериментів з різними вхідними зображеннями стилю.

ВИСНОВКИ

Отже, нами було **проаналізовано** задачу перенесення стилю та методи її вирішення, хоча ця задача є інтуїтивно простою для розуміння людиною – її складно задати алгоритмічно. Хоча існує ряд методів, які в тій чи іншій мірі вирішують цю проблему, вони в абсолютній більшості є обмеженими або ресурсозатратними з необхідним виконання процесу навчання.

Дослідивши теоретичні засоби вирішення задачі, ми обрали засоби, що забезпечили ефективне досягнення поставленої мети нашого дослідження, а саме створення інформаційної технології перенесення стилю.

Нам було **розроблено** алгоритм перенесення стилю на основі нейронної мережі, що здатен працювати з довільним стилем зображення. Сам алгоритм належить до IOB-NPS, описаних у першому розділі роботи, тобто алгоритмів на основі оптимізації зображення.

Основою інформаційної технології стала мережа VGG19 без верхнього шару класифікації, що і слугувала засобом екстракції ознак стилю та змісту з вхідних зображень.

Ітераційна мінімізація загальної функції втрат дозволила нам отримати якісні **результати роботи алгоритму** перенесення стилю на основі нейронних мереж.

У ході експериментів було підтверджено, що алгоритм показує добрі результати не залежно від стилю оригінального зображення, тобто він може працювати з довільним стилем. Алгоритм не потребує навчання, а його ефективність при роботі на GPU в середовищі Google Colab доволі висока, середній час перенесення стилю складає 120 секунд.

На основі результатів експериментів можна також зробити висновки, щодо параметрів які грають ключову роль у візуальній якості генерованого зображення. Так, при 10 епохах, кількість кроків на епоху в розкиді 50 – 200 не мала видимого впливу на результат. Зовсім протилежні результати були

отримані в ході зміни вагових коефіцієнтів змісту та стилю, що беруть участь у функції мінімізації загальних втрат.

Так, значне зменшення ваги стилю порівняно зі змістом призводило до гірших результатів НПС зі сторони візуальної якості, проте це не можна вважати проблемою алгоритму, а швидше його особливістю, адже при зменшенні ваги стилю – зміст зображення стає зрозумілішим.

Найкращий результат зі сторони якості показали наступні параметри вагових коефіцієнтів: стиль – 0.001, зміст – 1.

Також ми дійшли до висновку, що за умови сталості задачі НПС (перенесення стилю зі збереженням змісту), до більш абстрактних стилів краще застосовувати менші вагові коефіцієнти стилю (напр. 0.0001, або 0.00001). Це у результаті дає кращий результат алгоритму за умови сталості задачі НПС.

Одним з недоліків нашої реалізації є те, що вона створює багато високочастотних артефактів. Це може бути вирішено в майбутньому для покращення результатів роботи алгоритму.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gatys L. A. A neural algorithm of artistic style [Електронний ресурс] / L. A. Gatys, A. S. Ecker, M. Bethge // ArXiv e-prints. – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1508.06576.pdf (дата звернення: 05.03.2022).

2. Smee S. Will NFTs transform the art world? Are they even art? [Електронний ресурс] / Sebastian Smee // The Washington Post. – 2021. – Режим доступу до ресурсу: https://www.washingtonpost.com/arts-entertainment/2021/12/18/nft-art-faq/ (дата звернення: 21.02.2022)

3. J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg, «State of the ‘art’: A taxonomy of artistic stylization techniques for images and video», IEEE transactions on visualization and computer graphics, vol. 19, no. 5, pp. 866–885, 2013.

4. A. Hertzmann, «Painterly rendering with curved brush strokes of multiple sizes», in Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM, 1998, pp. 453– 460.

5. Y.-Z. Song, P. L. Rosin, P. M. Hall, and J. Collomosse, «Arty shapes», in Proceedings of the Fourth Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging. Eurographics Association, 2008, pp. 65–72.

6. A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, «Image analogies», in Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, 2001, pp. 327–340.

7. H. Winnemoller, S. C. Olsen, and B. Gooch, «Real-time video ¨ abstraction», in ACM Transactions On Graphics (TOG), vol. 25, no. 3. ACM, 2006, pp. 1221–1226.

8. D. J. Heeger and J. R. Bergen, «Pyramid-based texture analysis/synthesis», in Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. ACM, 1995, pp. 229– 238.

9. J. Portilla and E. P. Simoncelli, «A parametric texture model based on joint statistics of complex wavelet coefficients», International journal of computer vision, vol. 40, no. 1, pp. 49–70, 2000.

10. L. A. Gatys, A. S. Ecker, and M. Bethge, «Texture synthesis using convolutional neural networks», in Advances in Neural Information Processing Systems, 2015, pp. 262–270.

11. A. A. Efros and T. K. Leung, «Texture synthesis by nonparametric sampling», in Proceedings of the IEEE International Conference on Computer Vision, vol. 2. IEEE, 1999, pp. 1033–1038.

12. A. Mahendran and A. Vedaldi, «Understanding deep image representations by inverting them», in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5188–5196.

13. A. Dosovitskiy and T. Brox, «Inverting visual representations with convolutional networks», in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4829– 4837.

14. A. Mordvintsev, C. Olah, and M. Tyka, «Inceptionism: Going deeper into neural networks», 2015. [Електронний ресурс]. – Режим доступу до ресурсу: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html (дата звернення 15.03.2022).

15. L. A. Gatys, A. S. Ecker, and M. Bethge, «Image style transfer using convolutional neural networks», in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2414–2423.

16. C. Li and M. Wand, «Combining markov random fields and convolutional neural networks for image synthesis», in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2479–2486.

17. J. Johnson, A. Alahi, and L. Fei-Fei, «Perceptual losses for realtime style transfer and super-resolution», in European Conference on Computer Vision, 2016, pp. 694–711.

18. D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, «Texture networks: Feed-forward synthesis of textures and stylized images», in International Conference on Machine Learning, 2016, pp. 1349–1357.

19. C. Li and M. Wand, «Precomputed real-time texture synthesis with markovian generative adversarial networks», in European Conference on Computer Vision, 2016, pp. 702–716.

20. V. Dumoulin, J. Shlens, and M. Kudlur, «A learned representation for artistic style», in International Conference on Learning Representations, 2017.

21. D. Ulyanov, A. Vedaldi, and V. Lempitsky, «Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis», in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6924–6932.

22. Zhang H. Multi-style generative network for realtime transfer [Електронний ресурс] / H. Zhang, K. Dana // ArXiv e-prints. – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1703.06953.pdf (дата звернення: 05.04.2022).

23. T. Q. Chen and M. Schmidt, «Fast patch-based style transfer of arbitrary style», in Proceedings of the NIPS Workshop on Constructive Machine Learning, 2016.

24. X. Huang and S. Belongie, «Arbitrary style transfer in real-time with adaptive instance normalization», in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1501–1510.

25. Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.H. Yang, «Universal style transfer via feature transforms», in Advances in Neural Information Processing Systems, 2017, pp. 385–395.

26. Boesch G. VGG Very Deep Convolutional Networks (VGGNet) – What you need to know [Електронний ресурс] / Gaudenz Boesch // – Режим доступу до ресурсу: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/#:~:text=The%20concept%20of%20the%20VGG19,more%20convolutional%20layers%20than%20VGG16 (дата звернення: 15.04.2022).

27. Convolutional Neural Networks [Електронний ресурс] // IBM Cloud Education. – 2020. – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/convolutional-neural-networks (дата звернення 15.04.2022).

28. Wei J. AlexNet: The Architecture that Challenged CNNs [Електронний ресурс] / Jerry Wei // Towards Data Science. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951 (дата звернення 15.04.2022).

29. Kurama V. A Review of Popular Deep Learning Architectures: AlexNet, VGG16, and GoogleNet [Електронний ресурс] / Vihar Kurama // PaperspaceBlog. – 2020. – Режим доступу до ресурсу: https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/#:~:text=The%20GoogleNet%20Architecture%20is%2022,of%20the%20full%20GoogleNet%20architecture (дата звернення 15.04.2022).

30. Kaushik A. Understanding the VGG19 Architecture [Електронний ресурс] / Aakash Kaushik // OpenGenus IQ. – Режим доступу до ресурсу: https://iq.opengenus.org/vgg19-architecture/#:~:text=VGG19%20is%20a%20variant%20of,VGG19%20has%2019.6%20billion%20FLOPs (дата звернення 16.04.2022).

31. Baheti P. Neural Style Transfer: Everything You Need to Know [Guide] [Електронний ресурс] / Pragati Baheti // v7labs. – 2022. – Режим доступу до ресурсу: https://www.v7labs.com/blog/neural-style-transfer (дата звернення 17.04.2022).

32. Bonde O. A Comparison of Selected Optimization Methods for Neural Networks [Електронний ресурс] / O. Bonde, L. Karlsson // Degree project in Technology. – 2020. – Режим доступу до ресурсу: http://kth.diva-portal.org/smash/get/diva2:1438308/FULLTEXT01.pdf (дата звернення 18.04.2022).

33. What is Python? Executive Summary [Електронний ресурс]. – Режим доступу до ресурсу: https://www.python.org/doc/essays/blurb/ (дата звернення 01.05.2022).

34. Chng Z. M. Google Colab for Machine Learning Projects [Електронний ресурс] / Zhe Ming Chng // Machine Learning Mastery. – 2022. – Режим доступу до ресурсу: https://machinelearningmastery.com/google-colab-for-machine-learning-projects/ (дата звернення 01.05.2022).

35. What is Tensorflow: Deep Learning Libraries and Program Elements Explained [Електронний ресурс] // Simplilearn. – 2021. – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow (дата звернення 01.05.2022).

36. What is Matplotlib In Python? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/ (дата звернення 01.05.2022).

ДОДАТКИ

Додаток 1.

Код програми для перенесення стилю на основі нейронної мережі.

Імпорт бібліотек та налаштування модулів

```
[1]  import os
     import functools
     import time

     import tensorflow as tf
     import numpy as np
     import matplotlib.pyplot as pyplot
     import matplotlib as matplot
     matplot.rcParams['figure.figsize'] = (12, 12)
     matplot.rcParams['axes.grid'] = False

     import IPython.display as display
     import PIL.Image
```

```
[2]  def tensor_image_transfer(tensor):
       tensor = tensor*255
       tensor = np.array(tensor, dtype=np.uint8)
       if np.ndim(tensor)>3:
         assert tensor.shape[0] == 1
         tensor = tensor[0]
```

Завантаження і підготовка зображень

```
donwload_content = tf.keras.utils.get_file('Tuebingen_Beckarfront.jpg'
#image1 donwload_style = tf.keras.utils.get_file('vangogh_starry_night
#image2 donwload_style = tf.keras.utils.get_file('vangogh_sunflowers.j
#image3 donwload_style = tf.keras.utils.get_file('picasso_lesdemoisell
#image4 donwload_style = tf.keras.utils.get_file('bosch_garden.jpg','h
#image5 donwload_style = tf.keras.utils.get_file('pissarro_boulevard_m
donwload_style = tf.keras.utils.get_file('mondrian_broadway.jpg','http
```

```
Downloading data from https://upload.wikimedia.org/wikipedia/commons/0
409600/406531 [==============================] - 0s 0us/step
417792/406531 [==============================] - 0s 0us/step
Downloading data from https://upload.wikimedia.org/wikipedia/commons/t
327680/320751 [==============================] - 0s 0us/step
335872/320751 [==============================] - 0s 0us/step
```

Функція для завантаження зображення та обмеження його максимальний розміру (512 пікселів)

```
[4]  def image_upload(path_to_img):
       max_dim = 512
       img = tf.io.read_file(path_to_img)
       img = tf.image.decode_image(img, channels=3)
       img = tf.image.convert_image_dtype(img, tf.float32)

       shape = tf.cast(tf.shape(img)[:-1], tf.float32)
       long_dim = max(shape)
       scale = max_dim / long_dim

       new_shape = tf.cast(shape * scale, tf.int32)

       img = tf.image.resize(img, new_shape)
       img = img[tf.newaxis, :]
       return img
```

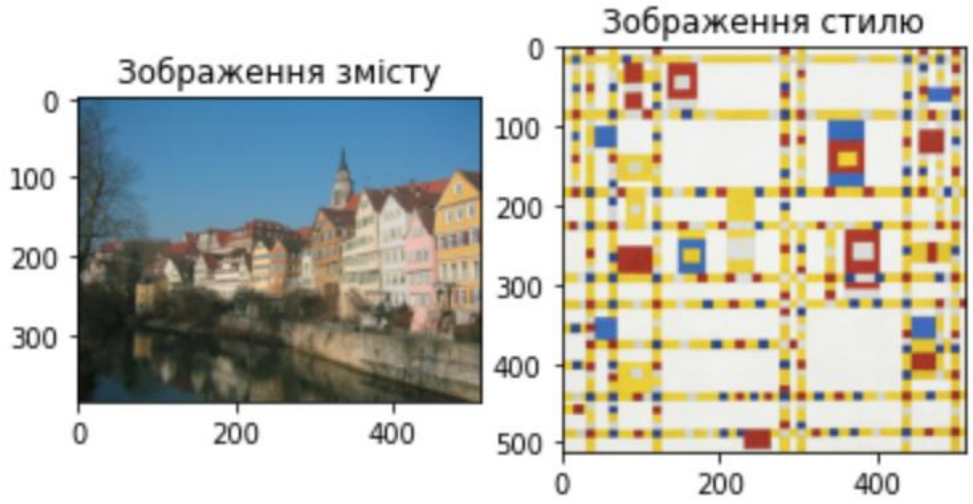
```
[5]  def imshow(image, title=None):
       if len(image.shape) > 3:
         image = tf.squeeze(image, axis=0)

       pyplot.imshow(image)
       if title:
         pyplot.title(title)
```

```
[6]  content_image = image_upload(donwload_content)
     style_image = image_upload(donwload_style)

     pyplot.subplot(1, 2, 1)
     imshow(content_image, 'Зображення змісту')

     pyplot.subplot(1, 2, 2)
     imshow(style_image, 'Зображення стилю')
```



Завантаження моделі та визначення представлень для стилю та змісту

```
[7] #завантажуємо, попердньо навчену на imagenet, мережу VGG19 без класифікаційної верхівки
    x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
    x = tf.image.resize(x, (224, 224))
    vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

    print()
    for layer in vgg.layers:
      print(layer.name)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weight
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step

input_1
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
block3_pool
block4_conv1
block4_conv2
```

```
#для визначення представлення змісту ми використовуємо 2(другий) згортковий шар 4 блоку мережі VGG19
content_layers = ['block4_conv2']

#для визначення представлення стилю ми використовуємо 1(перші) згорткові шари 5(п'яти) блоків мережі VGG19
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

```
[9] #Функція для створення модель VGG19, яка повертає список вихідних даних з проміжних шарів
    def all_layers(layer_names):
      vgg.trainable = False

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

      model = tf.keras.Model([vgg.input], outputs)
      return model
```

```
[10] style_extractor = all_layers(style_layers)
     style_outputs = style_extractor(style_image*255)
```

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

```
[12] class StyleContentModel(tf.keras.models.Model):
       def __init__(self, style_layers, content_layers):
         super(StyleContentModel, self).__init__()
         self.vgg = all_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, inputs):
         "Expects float input in [0,1]"
         inputs = inputs*255.0
         preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
         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}
```

```
[13] extractor = StyleContentModel(style_layers, content_layers)

     results = extractor(tf.constant(content_image))
```

```
[14] goals_style = extractor(style_image)['style']
     goals_content = extractor(content_image)['content']
```

```
[15] image = tf.Variable(content_image)
```

```
[16] def clip_0_1(image):
       return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)
```

```
[17] opt = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
```

```
[18] style_weight=1e1
     content_weight=1e4
```

```
[19] def total_loss(outputs):
       style_outputs = outputs['style']
       content_outputs = outputs['content']
       style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-goals_style[name])**2)
                              for name in style_outputs.keys()])
       style_loss *= style_weight / num_style_layers

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

```
[20] @tf.function()
     def iteration(image):
       with tf.GradientTape() as tape:
         outputs = extractor(image)
         loss = total_loss(outputs)

       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))
```

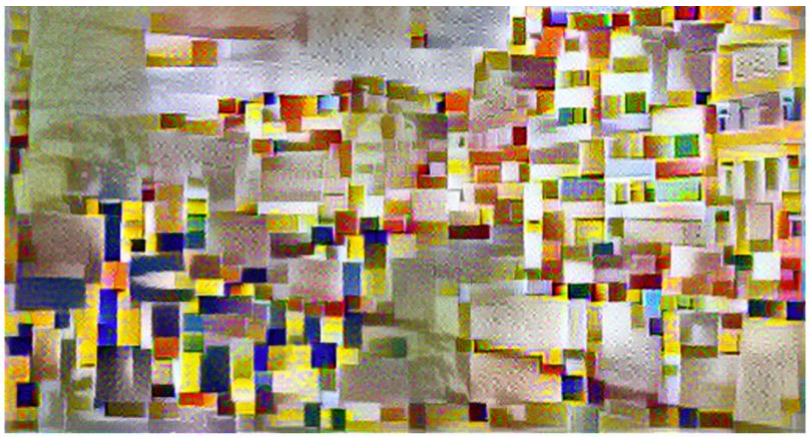
```
[21] import time
     start = time.time()

     epochs = 10
     iteration_per_epoch = 100

     step = 0
     for n in range(epochs):
       for m in range(iteration_per_epoch):
         step += 1
         iteration(image)
         print(".", end='', flush=True)
       display.clear_output(wait=True)
       display.display(tensor_image_transfer(image))
       print("Train step: {}".format(step))

     end = time.time()
     print("Total time: {:.1f}".format(end-start))
```

[21]



```
Train step: 1000
Total time: 70.9
```

```
file_name = "style_weight" + "_" + str(style_weight) + "_" + "content_weight" + "_" +

tensor_image_transfer(image).save(file_name)

try:
  from google.colab import files
except ImportError:
   pass
else:
  files.download(file_name)
```