МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ к.ф.м.н., доцент
Коплик І. В.

«___» _____ 20__р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

спеціальність 113 «Прикладна математика»

освітньо-професійна програма 113.2 «Наука про дані та моделювання складних систем»

тема роботи «**ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ СТИСНЕННЯ ТА ШИФРУВАННЯ ДАНИХ**»

Виконавець

студент факультету ЕлІТ

Татаренко М. Д. _____

Науковий керівник

к. ф.-м. н., доцент

Князь І.О. _____

Суми – 2021

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**
Кафедра **прикладної математики та моделювання складних систем**

Рівень вищої освіти **другий**

Галузь знань **11 Математика та статистика**
Спеціальність **113 Прикладна математика**

Освітня програма **освітньо-професійна 113.2 «Наука про дані та моделювання складних систем»**

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПМтаМСС
к.ф.м.н., доцент Коплик І. В. _____
«___» _____2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ ТАТАРЕНКО МИХАЙЛО ДЕНИСОВИЧ

Тема роботи: «Використання нейронних мереж для стиснення та шифрування даних»

Керівник роботи: Князь Ігор Олександрович, к. ф.-м. н., доцент

затверджено наказом по факультету ЕлІТ від «08» жовтня 2021 р. № 0687-VI

Термін подання роботи студентом «19» грудня 2021 р.

Вихідні данні до роботи : рекурентна нейронна мережа для стиснення шляхом кодування та декодування зображень.

Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити): Аналіз наявних методів стиснення інформації завдяки мережам, побудова базової нейронної мережі, модифікування моделі, отримання результатів та порівняння із наявними методами.

Перелік графічного матеріалу: 10 рисунків

Дата видачі завдання «22» листопада 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1	Збір інформації щодо теми кваліфікаційної роботи	22.11.2021 - 28.11.2021	Звіт
2	Оформлення літературного огляду в звіті	29.11.2021 - 30.11.2021	Звіт
3	Розробка алгоритму і написання моделі	01.12.2021 - 04.12.2021	Мат. модель
4	Розробка програми мовою програмування Python	05.12.2021 - 14.12.2021	Програма
5	Оформлення звітної документації з кваліфікаційної роботи	15.12.2021 - 19.12.2021	Звіт

Здобувач вищої освіти _____ Татаренко М. Д.

Керівник роботи _____ Князь І. О.

РЕФЕРАТ

Кваліфікаційна робота: 53 с., 10 рисунків, 33 джерела.

Мета роботи: Дослідження моделей нейронних мереж, підбір типу моделі, оптимальних значень та параметрів стиснення зображення за допомогою нейронної мережі. Зробити певні висновки, щодо виконаної роботи та порівняти отриману модель із існуючими методами.

Об’єкт дослідження: процес стиснення зображення без втрат із використанням методів нейронних мереж.

Предмет дослідження: модель рекурентної нейронної мережі, яка спочатку навчається на певному наборі даних, і завдяки цьому ми маємо можливість кодувати та декодувати зображення.

Робота містить аналіз галузі дослідження, огляд наукових робіт на тему стиснення, вивчення статистичних та математичних методів моделювання, висновки щодо проведеної роботи. Для зручності інтерпретування отриманих результатів, в роботі наведені вихідні зображення за певних модифікацій мережі.

Ключові слова: НЕЙРОННА МЕРЕЖА, РЕКУРЕНТНІСТЬ, СТИСНЕННЯ, МОДЕЛЬ, ВІЗУАЛІЗАЦІЯ, МОДЕЛЮВАННЯ.

ЗМІСТ

ВСТУП

В нашому житті зображення відіграють найважливішу роль, адже людина сприймає найбільше інформації із візуальних матеріалів таких як зображення та відео. Тому стиснення розмірів таких матеріалів стає на перший план зважаючи на те скільки необхідно місце для зберігання інформації. Та не менш важливим є швидкість передачі даних.

Для того щоб зменшити розмірністю зображені на початкових етапах використовувалися такі методи як: ентропійне кодування, кодування хоффмана також був використаний код Голомба та інші методи. Починаючи із 60-х років минулого століття було запропоноване кодування із використанням просторових частот. Також з’явився метод із перетворенням Фурьє. З плином часу та розвитком технологій був запропонований метод дискретно косинусного перетворення (DCT). Цей метод використовував енергію в низькочастотні області. Яка із використанням даного методу була набагато ефективніше за попередні методи.

Один із найпопулярніших методів стиснення зображення це JPEG. Оскільки він використовує попередні методи які були вище. Цей метод ділить первинне зображення на блоки та за допомогою метода DCT виконує певну модуляцією. Для зменшення витрат при пониженій розміру в цьому методі була розроблена відповідно таблиця яка зберігає більше низькочастотної інформації а ніж високочастотної. Тому що люди є менш чутливі до втрати інформації саме у високочастотних блоках з інформацією.

Також є інший достатньо популярний метод - JPEG 2000. На відміну від попереднього він застосовує двовимірне вайлет перетворення та також використовує метод арифметичного кодування.

Звісно ж прогрес не стоїть на місці, і тому останнім часом набули популярності нейронні мережі. Можна виділити згорткові нейронні мережі вони

досягли успіху в багатьох сферах. Дані мережі складаються з однієї або n-кількості внутрішніх шарів (залежить від задачі). Для того щоб навчити мережу необхідно задати певні параметри та правила для внутрішніх шарів. Це робиться за допомогою обробки великої кількості зображень або певної інформації що дає змогу навчити нашу нейронну мережу і після цього її можна використовувати для вирішення різноманітних задач. Таких як класифікація, розпізнавання тексту, зображень та відео матеріалів. Якщо модифікувати правила для перетворення зображення, то також дана модель може бути використана для стиснення інформації. А зважаючи на середні оцінки нейронних мереж, то таке рішення може бути достатньо вдалим.

РОЗДІЛ 1 АНАЛІТИЧНІЙ ОГЛЯД

1.1 Введення в нейронну мережу

В цьому розділі ми познайомимося взагалі із концепцією нейронної мережі, які етапи вона пройшла із часом та розглянемо принцип роботи.

Взагалі нейронна мережа з'явилася після певних дослідження математиків та представників різних нейронаук. Принцип роботи схожий з принципом роботи мозку. Мережа складається з кількох шарів простих нейронів, або як вони називаються у нейронній мережі – персептрони. Вони поєднані між собою за допомогою зважених зв’язків. Власне і самі нейрони активуються через ці зважені зв'язки, від нейронів які були активовані на попередньому кроці. Сам процес активації завжди використовується на всіх проміжних шарах. Схему роботи нейронної мережі можна побачити на Рис. 1.

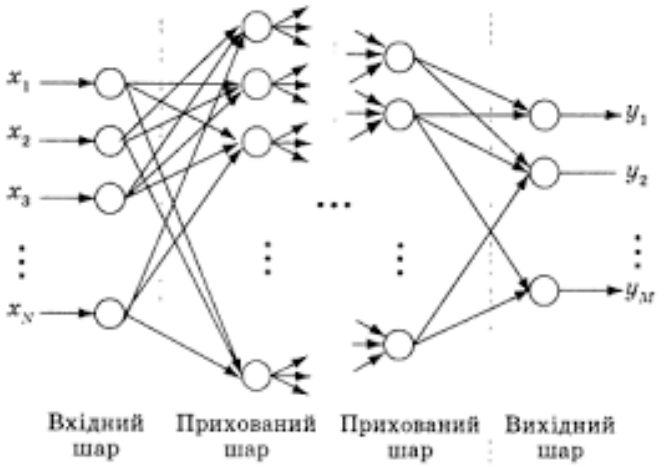

Рис.1 – Схематичне зображення нейронної мережі

Дана мережа складається із одного вхідного шару декількох прихованих шарів, кожен з яких містить різну кількість нейронів, та одного вихідного шару. Власне сама модель була запропонована ще в 60-х роках минулого століття проте за наявних технологій реалізувати їй було надзвичайно складно. Для вирішення проблеми навчання багатошарового перцептрона була запропонована процедура

зворотного розповсюдження. Вона була заснована на ланцюговому правилі. Після цього такі багатошарові архітектури навчають за допомогою стохастичного градієнтного спуску із зворотнім поширення похибки. Проте для цього необхідно великі обчислювальні витрати. Також не менш важливим є те що дана модель страждає від поганих локальних мінімумів. І не слід забувати, що зв'язки між сусідніми шарами таких мережах збільшують кількість параметрів моделі квадратично і цим зменшується ефективна швидкість обчислення.

Після багатьох років дослідження також була розроблена згорткова нейронна мережа, яка здебільшого застосовують для розпізнавання об’єктів.

Основними типами стиснення даних є стиснення із втратами та без втрат. Стиснення, при якому двійкові дані стискаються шляхом розпізнавання та видалення статистичної надмірності, називається стиском даних без втрат. Загалом, у разі стиснення даних без втрат відсутньої інформації не буде. І навпаки, процес, який зменшує двійкові дані шляхом видалення менш важливої або зашумленої інформації, називається стиском даних із втратами. Крім того, стиснення застосовується до різних типів цифрових носіїв, таких як текст, зображення, аудіо та відео, для зниження витрат на зберігання та передачу даних. Стиснення зображень - це процес, у якому відбувається стиснення даних, у якому кілька бітів кодуються в реальне зображення. Основним завданням є зменшення несуттєвої надмірності даних, присутніх на зображеннях. Швидкість передачі зображення нижче під час передачі фактичного зображення, але методи стиснення допомагають збільшити швидкість передачі. Стиснення зображень зосереджено переважно на зменшенні розміру даних зображення та збереженні більшості необхідних деталей для розуміння. Основною метою стиснення зображень є відображення цих зображень як невеликої кількості бітів без втрати необхідного інформаційного змісту в самому зображенні. Це пов'язано з тим, що кожне зображення містить важливу інформацію, яка не повинна бути втрачена при зменшенні обсягу

зображення. У зв'язку зі швидким розвитком технологій виникла потреба в управлінні великим обсягом даних зображень, а також правильному зберіганні цих зображень за допомогою ефективних методів. Зазвичай це призводить до стиснення зображень. Знову виникає проблема, пов'язана із різними підходами до оптимального стиснення медичних зображень. Як і у випадку зі стиском даних, існує два основних методи стиснення зображень. Тому стиснення зображень може бути із втратами або без втрат. Існує безліч алгоритмів та методик для стиснення зображень, що займаються видаленням різних надлишкових даних, таких як міжпіксельні, кодові, візуальні, та інші елементи зображення оптимальним чином. В даній роботі ми проаналізуємо наявні нейронні мережу і за допомогою різних компонентів та методів спробуємо стиснути зображення без втрат, а також проаналізуємо, за яких налаштувань мережі, будуть найкращі результати, і також порівняємо із іншими підходами стиснення зображень.

Спираючись на отриману інформацію можна виділити такі найпопулярніші та найефективніші нейронні мережі:

- Багатошаровий перцептрон (MLP)
- Згорткова нейронна мережа (CNN)
- Рекурентна нейронна мережа (RNN)
- Генеративно-змагальні мережі (GAN)

1.2 Багатошаровий перцептрон

Дана модель складається із одного вихідного шару, одного вхідного шару та декількох прихованих шарів, які знаходяться між ними.

Вихід кожного нейрона визначається за такою формулою

$$h_i = \sigma(\Sigma_{j=1}^{N} w_{ij} x_j + c_i), \tag{1}$$

де *σ()* функція активації, c_i байас лінійного перетворення, w_{ij} регульований параметр, який представляє зв'язок між шарами (вага).

Теоретично можна сказати що багатошаровий персептрон, в якому внутрішніх шарів більше ніж один може апроксимувати будь-яку функцію із заданою точністю. завдяки чому і можливо говорити про зменшення розмірів і стиснення даних за певних умов. Для коректної роботи і стиснення зображення необхідна розробка певних перетворень для всіх просторових даних.

Наприкінці 80-х років вчені запропонували наскрізну модель для стиснення зображення. Вони пройшли певні етапи для того щоб стиснути зображення, а саме розробили та перетворили дані і зображення в просторовій області, зробили перетворення коефіцієнтів та двійкове кодування як завдання оптимізації для реалізації функцій вартості. Яка буде виглядати наступним чином

$$e_{k,l} = \|\mathbf{X} - \hat{y}_{kl} \mathbf{u}_k^T \mathbf{u}_l\|^2 \tag{2}$$

$$\hat{y}_{kl} = (s_1 2^{-1} + s_2 2^{-2} + \cdots + s_{kl} 2^{-b_{kl}}) - m_{k,l}, \tag{3}$$

де $y\hat{}_{kl} \in [0, 1]$ - коефіцієнти відновленого перетворення, u_k - ядро ортогонального перетворення, $\{s1, s2, - - - , s_{bkl}\}$ - двійкові коди, які мають рівні квантування.

Після цього було використано нейронну мережу для того щоб знайти оптимальну комбінацію двійкових кодів а саме вирішити задачу оптимізації для бітового потоку. Після багатьох спроб та помилок нейронна мережа була здатна стискати зображення розміром 8 х 8 за допомогою зворотного поширення. Але була одна неточність адже таку стратегію було важко адаптувати для різного ступеня стиснення, адже вона фіксувала параметри для певної кількості двійкових кодів. Був запропонований інший метод для прискорення процесу навчання. Вхідне зображені ділиться на певні блоки які паралельно подаються на такі самі нейронні мережі. Але така конструкція з використанням декількох мереж вимагає щоб вхідні зображення були достатньо схожі, зважаючи на те, що різні мережі працюють по-різному в залежності від структури.

Проте адаптивність моделей які були згадані вище всі залежать від ручного встановлення кількості прихованих нейтронів, а не від створення мереж із великою кількістю внутрішніх шарів та зв'язків між ними, що може обмежити можливості стиснення зображення. Цю проблему було вирішено за допомогою предикативного кодування, яке використовувало просторовий контекст.

Модель навчалася за допомогою алгоритму зворотного розповсюдження помилки та для мінімізації похибки була використана середня квадратична похибка між вихідними та передбачуваними сигналами, які мали бути після обрахунку. Порівнюючи із результатами лінійної моделі, модель багатошарового перцептрона покращує ентропію помилки. Також існують передбачення, які можуть ще покращити дану модель із використанням авторегресіїної моделі, яка може добре обробляти різко виражені структури даних.

1.3 Випадкові нейронні мережі

Також наприкінці 90-х років була представлена нова нейронна мережа.

Випадкова нейронна мережа працюють дещо за іншим сценарієм ніж багатошаровий перцептрон. В ньому сигнали передаються у вигляді спайків одиничної амплітуди на відміну від сигналів багатошарового перцепторона, які обраховується за допомогою градієнтного методом зворотного розповсюдження похибки. Зв'язок у випадковій мережі моделюється як процес Пуассона, де позитивні сигнали є збуджуючими, а негативні сигнали навпаки.

Для перезапису параметрів використовується метод зворотного розповсюдження, який кожен раз обраховується для нової пари вхід-вихід і вимагає розв'язання n лінійних і нелінійних рівнянь.

Деякі дослідники спромоглися отримати значні результати у поєднанні випадкової нейронної мережі та процесу стиснення зображення. Найчастіше використовується мережа кодер-декодер із прямою передачею та одним або двома проміжними шарами. Перший шар сприймає зображення, потім проміжний шар обраховує ваги та вихідний шарф видає нове зображення.

Процес покращення не завершився на досягнутому і в подальшому було запропоновано адаптивну блокову випадкову нейронну мережу стиснення/декомпресії. В якій мережа стискає певні блоки паралельно, а вибір саме цих блоків залежить від оціненої якості декомпресованих результатів. Також після цього був ще метод який використовував інтегровану випадкову мережу у вайлет-область зображення.

1.4 Згортка нейронна мережа

Останнім часом CNN з великим відривом перевершує традиційні алгоритми у високорівневих завданнях комп'ютерного зору, таких як: виявлення об'єктів та класифікація зображень. Навіть для багатьох низькорівневих завдань комп'ютерного зору він також досягає значних результатів продуктивності, наприклад, зменшення артефактів перенавчання та стиснення. CNN використовує операцію згортки, щоб охарактеризувати кореляцію між сусідніми пікселями та каскадні операції згортки добре поєднуються з ієрархічною схемою. Крім того, локальні рецептивні поля і загальні ваги, що вводяться операціями згортки, також зменшують параметри CNN, що навчаються, що значно знижує ризик виникнення проблеми перенавчання. Натхненна потужним уявленням CNN для зображень, була зроблена велика робота з вивчення можливості стиснення зображень із втратами на основі CNN. Однак безпосередньо включити модель CNN у наскрізне стиснення зображень досить складно. Взагалі кажучи, навчання CNN ґрунтується на алгоритмі зворотного розповсюдження та стохастичному градієнтному спуску, який вимагає майже скрізь диференційності функції втрат щодо параметрів, що навчаються, таких як вага згортки та зміщення. Через модулі квантування при стисненні зображення майже всюди виходять нульові градієнти, що зупиняє оновлення параметрів CNN. Крім того, класичну оптимізацію швидкості важко застосувати до системи стиснення на основі CNN. Це пов'язано з тим, що наскрізне навчання для CNN вимагає функції втрат, але швидкість повинна бути розрахована на основі розподілу популяції всіх квантованих елемнтів, які зазвичай не диференціюються щодо аргументів CNN. Також було представилено оптимізовану наскрізну структуру CNN для стиснення зображень у припущенні скалярного квантування. Вона складається з двох модулів: аналізу та синтезу перетворень для кодера та декодера. У перетворенні аналізу є три етапи: згортка, субдискретизація та розподільча нормалізація.

Оскільки перетворення синтезу є зворотною операцією перетворення аналізу, всі параметри на трьох етапах будуть оптимізовані відповідно до об'єктивної функції "швидкість - перетворення" у наскрізному стилі. Щоб впоратися з нульовими похідними через квантування використовували адитивний однорідний шум для імітації квантувача у процедурі навчання CNN, що дозволяє використовувати стохастичний підхід градієнтного спуску для вирішення задачі оптимізації. Цей метод перевищує JPEG2000 за показниками PSNR та MSSSIM. Крім того був запропонований метод використовуючи гіпер-пріорне масштабування для оцінки ентропії, яка досягла схожої об'єктивної продуктивності кодування з HEVC. Для майбутнього практичного застосування необхідно продовжити вивчення апаратної підтримки та аналізу енергоефективності, оскільки авторегресійний компонент не може бути легко розпаралелений. Також часто використовують пірамідальну структуру злиття ознак у кодері та фільтр після обробки на основі CNN у декодері.

Недоліки нейронної мережі:

- Граничні пікселі (рамка)
- Класифікація за різними полями
- Низька продуктивність

1.5 Рекурентні нейронні мережі

На відміну від архітектури CNN, яку ми розглядали на попередньому кроці, RNN є різновидом нейронної мережі з пам'яттю для збереження недавньої поведінки. Зокрема, одиниці пам'яті в RNN мають зв'язки із собою, які передають перетворену інформацію з минулого виконання. Використовуючи цю збережену інформацію, RNN змінює поведінку поточного процесу живлення, щоб адаптуватися до контексту поточного входу. Більш сучасні вдосконалення, такі як Gated Recurrent Unit (GRU), представлені для спрощення процесів еволюції рекурентної мережі та збереження продуктивності рекурентної мережі на відповідних завданнях. За аналогією з CNN, для завдання стиснення зображень RNN, як і раніше, страждає від труднощів з поширенням градієнтів оцінки швидкості.

Вперше була запропонована схема стиснення зображень на основі PHC. Вона використовує масштабне адитивне кодування для обмеження кількості бітів кодування замість підходу оцінки швидкості у згортковій мережі. Більш конкретно, метод, є архітектурою багатоітераційного стиснення, яка підтримує варіативне стиснення бітової швидкості в прогресивному стилі. У ітерації беруть участь три модуля: мережу кодування E, бінаризатор B та мережу декодування D, де D та E містять рекурентні мережеві компоненти. Залишкові сигнали між вхідним зображенням та відновленим мережею декодування D можуть бути стиснуті на наступній ітерації. Для подальшого покращення стиснення зображень на основі PHC була представлена просторово-адаптивну систему стиснення зображень. У цій системі вхідні зображення поділяються на мозаїки, які схожі на існуючі кодеки зображень, такі як JPEG та JPEG2000. Для кожної мозаїки повністю конволюційна нейронна мережа генерує початкове наближення на основі сусідніх просторових мозаїк, які були декодовані в лівій та верхній областях.

RNN можна представити як кілька копій існуючої мережі, зациклених у собі, які передають інформацію та оцінюють таку кількість разів, скільки необхідно. Найбільш важливою особливістю RNN є здатність пов'язувати інформацію, що мала місце у попередній події, з поточною подією. У разі стиснення зображень із втратами, RNN використовуються для надання знань про раніше оброблений блок зображення для розуміння поточного блоку зображення. Це також допомагає на різних етапах стиснення із втратами зберегти якість зображення. Розмір зображення значно зменшується, але якість зображення зберігається навіть після ітерацій стиснення. Таким чином, втрата даних на зображеннях значно знижується. Ще однією перевагою RNN є можливість зменшення кількості параметрів за допомогою методу розподілу ваг у міру збільшення глибини мережі після різних тимчасових кроків. Крім того, коли зображення мають якісні послідовні дані, RNN дуже доцільно використовувати при стисканні.

Недоліками можуть слугувати:

- Проблеми із зникненням градієнту
- Важке навчання моделі
- Достатньо важко обробляє довгі послідовності.

1.6 Генеративні змагальні нейронні мережі

Генеративна змагальна мережа є одним із найбільш привабливих удосконалень у застосуванні глибоких нейронних мереж. GAN оптимізує одночасно дві мережеві моделі, тобто генератор та дискримінатор. Дискримінатор використовує глибоку нейронну мережу для розрізнення того, чи отримані зразки від генератора. У той же час генератор здатний подолати дискримінатор та зробити зразки, які проходять перевірку. Перевага змагальних втрат полягає в тому, що вони допомагають генератору покращити суб'єктивну якість зображень, а також можуть бути розроблені для різних завдань. У задачі стиснення зображень деякі дослідження фокусувалися на якості сприйняття декодованих зображень і використовували GAN для поліпшення ефективності кодування. Деякі приклади робіт інтегровані і мають добре оптимізовану компресію зображень на основі GAN, яка не тільки досягає приголомшливого покращення коефіцієнта стиснення, але і може бути виконана в реальному часі, використовуючи переваги величезних паралельних обчислювальних ядер GPU. Мережі стискають вхідне зображення в дуже компактний простір ознак в стислому вигляді, а генераторна мережа використовується для відновлення декодованого зображення з ознак. Найбільш очевидною відмінністю стиснення зображень на основі GAN від схем на основі CNN або RNN є запровадження змагальних втрат, які значно покращують суб'єктивну якість відновленого зображення. Генеративна мережа та змагальна мережа навчаються разом, що значно підвищує продуктивність генеративної моделі. Метод на основі GAN досягає значного поліпшення ступеня стиснення, наприклад, він виробляє стислі файли менше, ніж JPEG і JPEG2000, і також менше, ніж WebP. Натхнене досягненнями в синтезі уявлень на основі GAN, стиснення зображень у світловому полі може досягти значного виграшу в кодуванні шляхом генерації уявлень, що бракують, з використанням контекстних уявлень. Зокрема, згенерований GAN контент більшою мірою відповідає семантиці оригінального

контенту, ніж конкретні текстури. Особливо збільшуючи масштаб реконструйованих зображень, ми можемо побачити різницю у змісті конкретних текстур. Також була представлена однорідна глибока генеративна конволюційна модель для завдання стиснення зображень. На відміну від попередніх робіт в ній переважав концептуальний стиск, генеруючи якнайбільше семантичної інформації зображення. Детально розглядається заснована на GAN система екстремального стиснення зображень, націлена на швидкість передачі даних менше 0,1 bpp, що дозволяє різною мірою генерувати контент. В даний час стиснення на основі GAN успішно застосовується для зображень вузької області, таких як особи, і все ще потребує подальших досліджень для створення моделей для загальних природних зображень.

Для даного типу нейронних мереж властиві:

- Важке навчання моделі
- Незбіжність моделі
- Коливання коефіцієнтів, згортання, що обмежує кількість зразків
- Чутливість до вибору параметрів

РОЗДІЛ 2 МЕТОДИКА ДОСЛІДЖЕННЯ

2.1 Конфігурація компонентів

Існує декілька технік для конфігурування мережі програмним способом, але це не є "стандартним і прийнятим методом" для конфігурування мережі. Наслідуючи невеликий набір чітких правил, можна програмно встановити грамотну архітектуру мережі (тобто кількість і тип нейронних шарів і кількість нейронів, що входять до кожного шару), або методом підбору встановити найкращу кількість компонент. Якщо ви дотримуватиметеся цієї схеми, ви отримаєте компетентну, але, ймовірно, неоптимальну архітектуру. Але як тільки мережа ініціалізована, ви можете ітеративно коригувати конфігурацію під час навчання, використовуючи низку допоміжних алгоритмів. Одне з таких сімейств працює шляхом обрізання вузлів на основі (малих) значень вагового вектора після певної кількості епох, іншими словами, видалення непотрібних/надлишкових вузлів. Кожна нейронна мережа має три типи шарів: вхідний, прихований і вихідний. Тому створення архітектури означає створення значень кількості шарів кожного типу і кількості вузлів у кожному з цих шарів.

Простий вхідний шар: кожна мережа має рівно один такий шар, без винятків. Що стосується кількості нейронів, що становлять цей шар, то цей параметр визначається повністю і виключно після того, як ви дізнаєтесь форму ваших навчальних даних. Зокрема, кількість нейронів, що становлять цей шар, дорівнює кількості ознак (стовпців) у ваших даних. У деяких конфігураціях додається додатковий вузол члена зсуву.

Вихідний шар, які вхідний, кожна мережа має рівно один. Визначити його розмір (кількість нейронів) легко, він повністю визначається обраною конфігурацією моделі. Мережа може працювати у машинному режимі або в режимі регресії. Машинний режим: повертає позначку класу. На відміну режим регресії

повертає значення. Якщо нейрона мережа є регресором, вихідний шар має тільки один вузол. Якщо обрана мережа є класифікатором, він також має один вузол, якщо тільки не використовується softmax, в цьому випадку вихідний шар має один вузол на мітку класу в своїй моделі.

Приховані шари. Отже, ці декілька правил визначають кількість шарів та розмір (нейронів/шару) для вхідного та вихідного шарів. Ну, якщо ваші дані лінійно розподілені, то вам не потрібні приховані шари. Звичайно, що мережа зможе працювати і без проміжних шарів, але працюватиме вона погано, і не буде достатньо репрезентативна. Одне з питань в темі «нейронні мережі», на яке не існує правильної відповіді, - це різниця у продуктивності при додаванні додаткових прихованих шарів Ситуацій, коли продуктивність покращується при додаванні другого (або третього тощо) прихованого шару, дуже мало. Для переважної більшості завдань достатньо одного прихованого шару. Але це напряму залежить від різниці кількості нейронів у вхідному шарі на відміну від вихідного.

Тепер розглянемо питання щодо кількості нейронів у прихованих шарах. Існує кілька емпірично виведених правил, з яких найчастіше використовується таке: "оптимальний розмір прихованого шару зазвичай знаходиться між розміром вхідного та розміром вихідного шарів". Також часто використовується конфігурація із іншими варіантами кількості. В результаті, для більшості завдань ви, ймовірно, зможете отримати непогану продуктивність, навіть без другого кроку оптимізації, задавши конфігурацію прихованого шару за допомогою всього двох правил: 1) кількість прихованих шарів дорівнює одному; 2) кількість нейронів у цьому шарі дорівнює середньому значенню нейронів у вхідному та вихідному шарах. Оптимізація конфігурації мережі Обрізка визначає набір методів для зменшення розміру мережі для покращення обчислювальної продуктивності та іноді продуктивності роздільної здатності. Суть цих методів полягає у видаленні вузлів з мережі під час навчання шляхом визначення тих вузлів. Якщо їх видалити

з мережі, то вони не вплинуть на продуктивність мережі. Навіть не використовуючи формальну техніку обрізки, ви можете отримати приблизне уявлення про те, які вузли не важливі, подивившись на матрицю ваги після навчання. Також необхідно звернути увагу на ваги, дуже близькі до нуля; саме вузли, розташовані по обидва боки від цих вагів, часто видаляються під час обрізки. Якщо використовувати алгоритм обрізки під час навчання, то починайте з конфігурації мережі, яка з більшою ймовірністю матиме надлишкові вузли. Іншими словами, при виборі архітектури мережі, робіть вибір на користь більшої кількості нейронів. Інакше кажучи, застосовуючи алгоритм обрізки до мережі під час навчання, ви можете наблизитися до оптимальної конфігурації мережі; чи зможете ви це зробити за один похід невідомо, але в даний момент така двоетапна оптимізація більш поширена у створенні мережі для коректної і більш точної роботи.

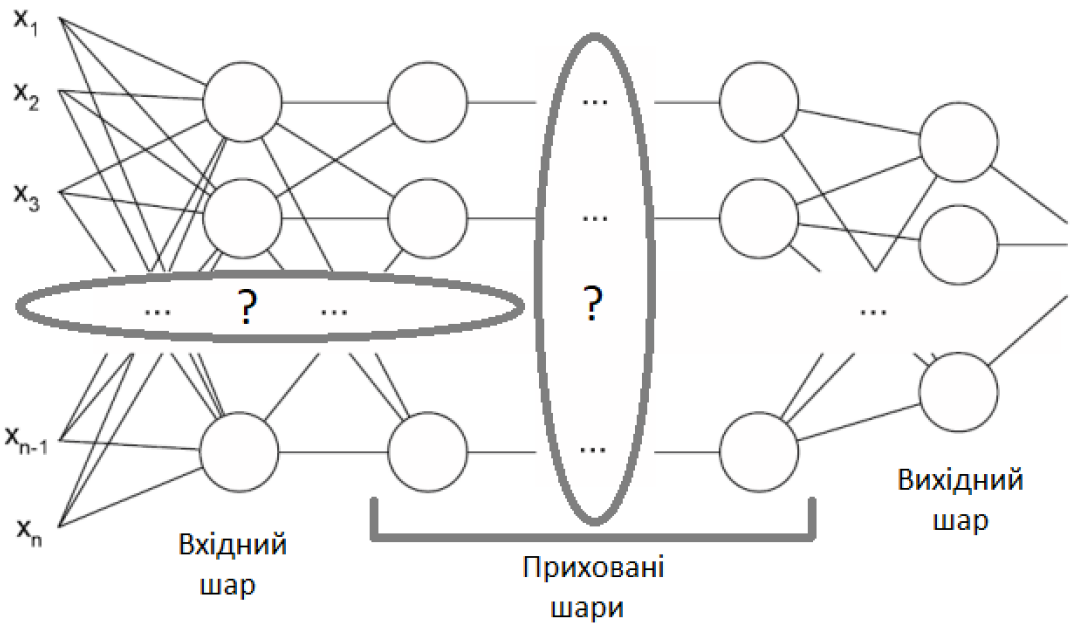

Рис.2 – Схематичне зображення нейронної мережі

2.2 LSTM

Під час обробки первинних даним модель передає попередній прихований стан наступного кроку в послідовності. Прихований стан діє як пам'ять нейронних мереж. Він містить інформацію про попередні дані, які мережа бачила раніше.

Спочатку вхідні дані та попередній прихований стан об'єднуються у вектор. Тепер цей вектор містить інформацію про поточний вхід та попередні входи. Вектор проходить через активацію tanh, і на виході ми отримуємо новий прихований стан або пам'ять мережі.

Активація tanh використовується для регулювання значень, що проходять через мережу. Функція tanh стискає значення так, щоб вони завжди знаходилися у проміжку [-1; 1]. Коли вектори проходять через нейронну мережу, вони піддаються безлічі перетворень у результаті різних математичних операцій. Можете побачити, як деякі значення можуть вибухнути та стати астрономічними, через що інші значення здаватимуться незначними. Функція tanh гарантує, що значення залишаються в проміжку [-1; 1], таким чином регулюючи вихід нейронної мережі. RNN використовує набагато менше обчислювальних ресурсів, ніж ішні варіанти, та підтримує покращення моделі за допомогою LSTM.

2.2.1 Головна концепція методу

LSTM має потік управління, схожий на потік управління нейронної рекурентної мережі. Він обробляє дані, які передають інформацію з її поширенням. Відмінності полягають в операціях всередині осередків LSTM.

Ці операції використовуються для того, щоб дозволити LSTM зберігати чи забувати інформацію. Розгляд цих операцій може бути трохи приголомшливим, тому ми більш детально розглянемо цей метод.

Центральною концепцією LSTM є стан комірки та її різних воріт. Стан клітини діє як транспортна магістраль, що передає відносну інформацію вздовж ланцюга послідовностей. Можна вважати, що це "пам'ять" мережі. Стан клітини, теоретично, може нести відповідну інформацію протягом усієї обробки послідовності. Тому навіть інформація з більш ранніх часових етапів може бути передана на пізніші часові етапи, що зменшує вплив короткочасної пам'яті. У міру того, як стан клітини продовжує свій шлях, інформація додається або видаляється у стані клітини через ворота. Ворота – це функції, які вирішують, яка інформація допускається у стан клітини. Ворота можуть дізнатися, яку інформацію потрібно зберегти або забути під час тренування.

2.2.2 Сигмоїдна активація

Ворота містять сигмоїдні активації. Сигмоїдна активація аналогічна активації tanh. Замість того, щоб зменшувати значення між -1 і 1, він усуває значення між 0 і 1. Це корисно для оновлення або забування даних, тому що будь-яке число, помножене на 0, дорівнює 0, що робить значення зникаючими або "забутими". Будь-яке число, помножене на 1, має те значення, тому це значення залишається незмінним або "зберігається". Мережа може дізнатися, які дані не важливі, щоб їх можна було забути, або які важливо зберегти.

2.2.3 Забуваючі ворота

По-перше, у нас є ворота забуття. Ці ворота вирішують, яку інформацію слід відкинути чи зберегти. Інформація попереднього прихованого стану та інформація з поточного входу пропускається через сигмоїдну функцію. Значення знаходяться в діапазоні від 0 до 1. Ближче до 0 означає забути, а ближче до 1 зберегти.

2.2.4 Вхідні ворота

Для оновлення стану осередку ми маємо вхідні ворота. Спочатку ми передаємо попередній прихований стан та поточний вхід до сигмоїдної функції. Це

вирішує, які значення буде оновлено шляхом перетворення значень між 0 і 1. 0 означає не важливо, а 1 означає важливо. Вона також передає прихований стан та поточний вхід у функцію tanh для зменшення значень між -1 та 1, щоб допомогти регулювати мережу. Потім він множить вихід tanh на вихід сигмоїду. Сигмоїдний вихід вирішуватиме, яку інформацію важливо виключити з tanh-виходу.

2.2.5 Стан клітин

Тепер ми маємо достатньо інформації для розрахунку стану клітини. Спочатку стан клітини точково множиться на вектор забування. Це може усунути значення у стані осередку, якщо вона множиться на значення, близькі до 0. Потім ми беремо вихід із вхідних воріт і виконуємо точкову суму, яка оновлює стан осередку до нових значень, які нейронна мережа вважає релевантними. Це дає нам новий стан клітки.

2.2.6 Вихідні ворота

Зрештою, у нас є вихідні ворота. Вихідні ворота вирішують, яким має бути наступний прихований стан. Слід пам'ятати, що прихований стан містить інформацію про попередні входи. Цей стан також використовується для прогнозування. Спочатку ми передаємо попередній прихований стан та поточний вхід до сигмоїдної функції. Потім ми передаємо новий змінений стан клітини у функцію tanh. Ми примножуємо вихід tanh на вихід сигмоїду, щоб вирішити, яку інформацію має нести прихований стан. На виході ми отримуємо прихований стан. Новий стан клітини та новий прихований стан передаються на наступний тимчасовий крок.

Підсумовуючи, ворота забування вирішують, що з попередніх кроків є актуальним, а що ні. Вхідні ворота вирішують, яку інформацію слід додати із поточного кроку. Вихідні ворота визначають, яким має бути наступний прихований стан.

2.3 Методи реконструкції після кодування

Крім використання різних типів рекурентних блоків, існують різні підходи до створення остаточної реконструкції зображення з виходів нашого декодера. Порівняємо ці підходи в даному підрозділі, а також необхідні зміни у функції втрат.

Одномоментна реконструкція: ми обробляємо повне зображення після кожної ітерації декодера. Кожна наступна ітерація має доступ до більшої кількості бітів, згенерованих кодером, що дозволяє покращити реконструкцію. Цей підхід можна назвати " реконструкція в один раз". Хоча ми намагаємося відновити вихідне зображення на кожній ітерації, ми передаємо тільки залишок від попередньої ітерації до наступної. Це зменшує кількість ваги, і експерименти показують, що передача як вихідного зображення, так і залишку не покращує реконструкцію.

Адитивна реконструкція: В адитивній реконструкції, яка ширше використовується в традиційному кодуванні зображень, кожна ітерація намагається відновити лише залишок попередніх ітерацій. Остаточне відновлення зображення є сумою виходів всіх ітерацій.

Масштабування залишку: як при адитивній, так і при одномоментній реконструкції, залишок починається з великого значення, і ми очікуємо, що він зменшуватиметься з кожною ітерацією. Однак кодер і декодер може ефективно працювати в широкому діапазоні значень. Крім того, швидкість зменшення залишку залежить від змісту. У деяких ділянках (наприклад, в однорідних областях) згасання буде значно більшим, ніж в інших (наприклад, у ділянках з високою текстурою).

Щоб зважити на ці відмінності, ми розширили нашу архітектуру адитивної реконструкції, включивши в неї коефіцієнт посилення, що залежить від змісту та ітерації. Концептуально, ми розглядаємо реконструкцію попереднього залишкового зображення, і виводимо множник посилення для кожного набору.

Потім ми множимо цільовий залишок, що надходить на поточну ітерацію, коефіцієнт посилення, отриманий при обробці виходу попередньої ітерації.

Рівняння набуває такого вигляду:

$$g_t = G(\hat{x}_t), \quad b_t = B(E_t(r_{t-1} \odot \mathrm{ZOH}(g_{t-1}))), \tag{4}$$

$$\hat{r}_{t-1} = D_t(b_t) \oslash \mathrm{ZOH}(g_{t-1}), \tag{5}$$

$$\hat{x}_t = \hat{x}_{t-1} + \hat{r}_{t-1}, \quad r_t = x - \hat{x}_t, \tag{6}$$

$$g_0 = 1, \quad r_0 = x.$$

де $\odot$ по-елементне розбиття, а *ZOH* - просторовий апсемплінг за нульовим порядком утримання. *G()* оцінює коефіцієнт посилення, g_t, за допомогою п'ятишарової конволюційної мережі зворотного зв'язку, кожен шар з кроком два. Останній шар дає на виході глибину 1, використовуючи конволюційне ядро з нелінійністю ELU. Оскільки ELU має діапазон (-1, ∞), до виходу цієї мережі додається константа 2, щоб отримати g_t у діапазоні (1, ∞).

2.4 Ентропійне кодування

У більшості зображень сусідні пікселі корельовані і тому містять надмірну інформацію. Наше завдання – знайти менш корельоване уявлення зображення, потім виконати скорочення надмірності та скорочення незв'язаності. Скорочення надмірності усуває дублювання джерела сигналу (наприклад, цифрового зображення). Зменшення нерегулярності опускає частини сигналу, які помітні візуальної системою людини. Ентропійне кодування додатково стискає квантовані значення без втрат, забезпечуючи найкращий загальний стиск. Він використовує модель для точного визначення ймовірностей для кожного квантованого значення та виробляє відповідний код на основі цих ймовірностей таким чином, щоб результуючий потік вихідного коду був меншим за вхідний потік.

Ентропія кодів, що генеруються в процесі виведення, не є максимальною, оскільки мережа явно не призначена для максимізації ентропії у своїх кодах, і модель не обов'язково використовує переваги візуальної надмірності у великому просторовому масштабі. Додавання шару ентропійного кодування може поліпшити коефіцієнт стиснення, як це зазвичай робиться в стандартних кодеках для стиснення зображень. При цьому кодер зображень є зрозумілим і використовується тільки як генератор двійкового коду. Схеми ентропійного кодування без втрат є повністю конволюційними. Вони обробляють двійкові коди в прогресивному порядку і для даної ітерації кодування в порядку кадрової розгортки. Всі наші архітектури кодерів зображень генерують двійкові коди виду c (y, x, d) розміру H×W×D, де H та W – цілі частки висоти та ширини зображення, а D – m × кількість ітерацій. Ми розглядаємо стандартну схему кодування без втрат, яка поєднує умовну ймовірну модель фактичного двійкового коду c (y, x, d) з арифметичним кодером для виконання фактичного стиснення.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ КОМПЮТЕРНОГО ЕКСПЕРИМЕНТУ

3.1 Методика чисельного експерименту

У цьому розділі описано архітектури моделі на високому рівні, яку ми будемо досліджуємо. У підрозділах представлені додаткові відомості про різні компоненти мережі, які використовуються у роботі.

Мережі стиснення складаються з мережі кодування E, бінаризатора B та мережі декодування D, де D та E містять рекурентні компоненти мережі. Вхідні зображення спочатку кодуються, а потім перетворюються на двійкові коди, які можуть бути збережені або передані декодеру. Мережа декодера створює оцінку вихідного зображення на основі отриманого двійкового коду. Ми повторюємо цю процедуру з залишковою помилкою – різницею між вихідним зображенням та реконструкцією декодера. Коли ваги мережі розподіляються між ітераціями, стани рекурентних компонентів передаються на наступну ітерацію. Тому залишки кодуються та декодуються у різних етапах та в різних ітераціях. Також слід зазначити, що бінаризатор B у нашій системі є статичним компонентом. Можемо компактно уявити одну ітерацію нашої мережі таким чином:

$$b_t = B(E_t(r_{t-1})), \quad \hat{x}_t = D_t(b_t) + \gamma \hat{x}_{t-1}, \tag{7}$$

$$r_t = x - \hat{x}_t, \quad r_0 = x, \quad \hat{x}_0 = 0 \tag{8}$$

Dt і *Et* - декодер і кодер зі своїми станами на ітерації t відповідно;

bt - прогресивне двійкове уявлення;

x^t – прогресивна реконструкція вихідного зображення *x*, де *γ* = 0 для одномоментної реконструкції або 1 для адитивної реконструкції;

rt - залишок між x та реконструкцією *x^t*.

На кожній ітерації B вироблятиме бінаризований бітовий потік $bt \in \{-1, 1\}^m$, де m - кількість бітів, що виробляються після кожної ітерації. Після k ітерацій мережа виробляє загалом m помножити на k біт.

Оскільки наші моделі повністю конволюційні, m є лінійною функцією розміру вхідних даних. Рекурентні блоки, що використовуються для створення кодера та декодера, включають два згорткові ядра: одне у вхідному векторі, який надходить у блок із попереднього шару, а інше у векторі стану, який забезпечує рекурентний характер блоку . Ми будемо називати згортку по вектору стану та його ядру "прихованою згорткою" та "прихованим ядром".

Під час навчання втрати розраховуються на зважених залишках, що генеруються на кожній ітерації, тому наші загальні втрати для мережі складають:

$$\beta \sum_t |r_t| \tag{9}$$

У наших мережах кожне вхідне зображення приблизно 500×500×3 зменшується до бінаризованого потоку ітерація за ітерацію. В результаті, кожна ітерація становить 1/8 біта на піксель (bpp). Якщо використовувати тільки першу ітерацію, це буде стиснення 192: 1, навіть до ентропійного кодування. Ми досліджували комбінацію варіантів рекурентних блоків та схем реконструкції для наших схем стиснення. В подальшому ми порівнюємо ці результати стиснення з результатами первинного зображення.

3.1.1 LSTM

Рекурентний елемент нейронної мережі який буде доданий для покращення мережі - це LSTM. Нехай *xt*, *ct* і *ht* - вхідний, клітинний і прихований стан, відповідно, на ітерації *t*. Враховуючи поточний вхід *xt*, попередній стан клітини *ct* - 1 та попередній прихований стан *ht* - 1, новий стан клітини *ct* та новий прихований стан *ht* обчислюються як

$$[f, i, o, j]^T = [\sigma, \sigma, \sigma, \tanh]^T \left((W x_t + U h_{t-1}) + b \right) \quad (10)$$

$$c_t = f \odot c_{t-1} + i \odot j, \quad (11)$$

$$h_t = o \odot \tanh(c_t), \quad (12)$$

де $\odot$ позначає множення на елементи, а *b* – байас. Функція активації σ - сигмоїдна функція, буде виглядати так:

$$\sigma(x) = 1/(1 + \exp(-x)) \quad (13)$$

Виходом LSTM шару на ітерації *t* буде *ht*. Перетворення *W* і *U*, що застосовуються до *xt* і *ht* - 1, є лінійними згортковими перетвореннями. Тобто, вони є композитами матриць. Як зазначалося раніше в цьому розділі, згортки *U* мають ту ж глибину, що і згортки *W*.

3.1.2 Одноітераційний ентропійний кодер

Ми використовуємо архітектуру схожу до PixelRNN та застосовуємо аналогічну архітектуру BinaryRNN для одношарового стиснення двійкового коду. У цій архітектурі (Рис. 3) оцінка ймовірності умовного коду для рядка у залежить безпосередньо від деяких сусідніх кодів, але також опосередковано від раніше декодованих двійкових кодів через лінію станів S розміром $1 \times W \times k$, яка відображає деякі короткострокові, так і довгострокові залежності.

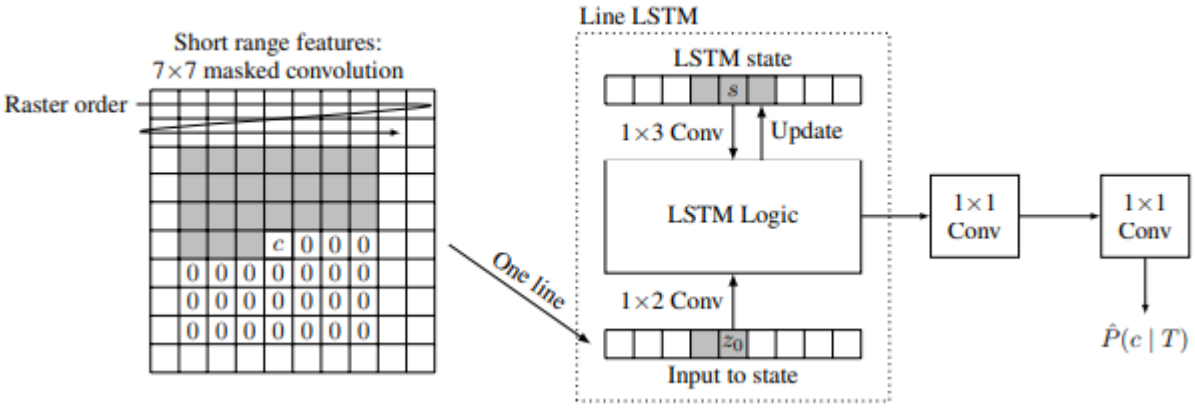

Рис.3 – Схематичне зображення одноітераційного ентропійного кодера

Головна лінія є сумою попередніх ліній. Можливості оцінюються, і стан оновлюється рядково за допомогою згортки 1×3 LSTM. Оцінка наскрізної ймовірності включає 3 етапи. По-перше, початкова згортка - це згортка, що використовується для збільшення рецептивного поля стану LSTM, рецептивне поле - це набір кодів $c\ (i, j, \cdot)$, які можуть впливати на оцінку ймовірності кодів $c\ (y, x, \cdot)$. Ця початкова згортка є маскою, щоб уникнути залежності від майбутніх кодів. На другому етапі лінія LSTM приймає на вхід результат z_0 цієї початкової згортки та обробляє по одному рядку сканування за один раз. Оскільки приховані стани LSTM створюються шляхом обробки попередніх рядків сканування, LSTM лінія захоплює як короткострокові, так і довгострокові залежності. З цієї причини перетворення LSTM від входу до стану також є згорткою з маскою. Нарешті, додаються дві

згортки *1 × 1*, щоб збільшити здатність мережі запам'ятовувати більше шаблонів бінарного коду. Оскільки ми намагаємось передбачити двійкові коди, параметр розподілу Бернуллі можна оцінити безпосередньо, використовуючи сигмоїдну активацію в останній згортці. Ми хочемо мінімізувати кількість бітів, які використовуються після ентропійного кодування, що, природно, призводить до втрати перехресної ентропії. У разі двійкових кодів {0, 1}, втрата перехресної ентропії може бути записана як:

$$\sum_{y,x,d} -c \log_2(\hat{P}(c \mid T)) - (1-c) \log_2(1-\hat{P}(c \mid T)) \qquad (14)$$

Більш формально, враховуючи контекст *T(y, x, d)*, який залежить тільки від попередніх бітів у порядку потоку, ми оцінимо *P(c(c(y, x, d) | T(y, x, d))* так, що очікувана ідеальна кодована довжина *c(y, x, d)* дорівнює перехресній ентропії між *P(c|T)* та *Pˆ(c|T)*. Ми не розглядаємо невеликий штраф, пов'язаний з використанням практичного арифметичного кодера, який вимагає квантованої версії *Pˆ(c|T)*.

3.1.3 Кількість шарів на нейронів

Порівнюючи різні моделі та різні принципи роботи нейронних мереж, підбір кількості внутрішніх шарів та нейронів в них є найголовнішим компонентом для навчання. В цій роботі будуть порівняні 4 схеми роботи (Рис. 4). Оскільки кодер та декодер працюють однаково, різниця лише в напрямку проходження мережі, розберемо лише процес кодування.

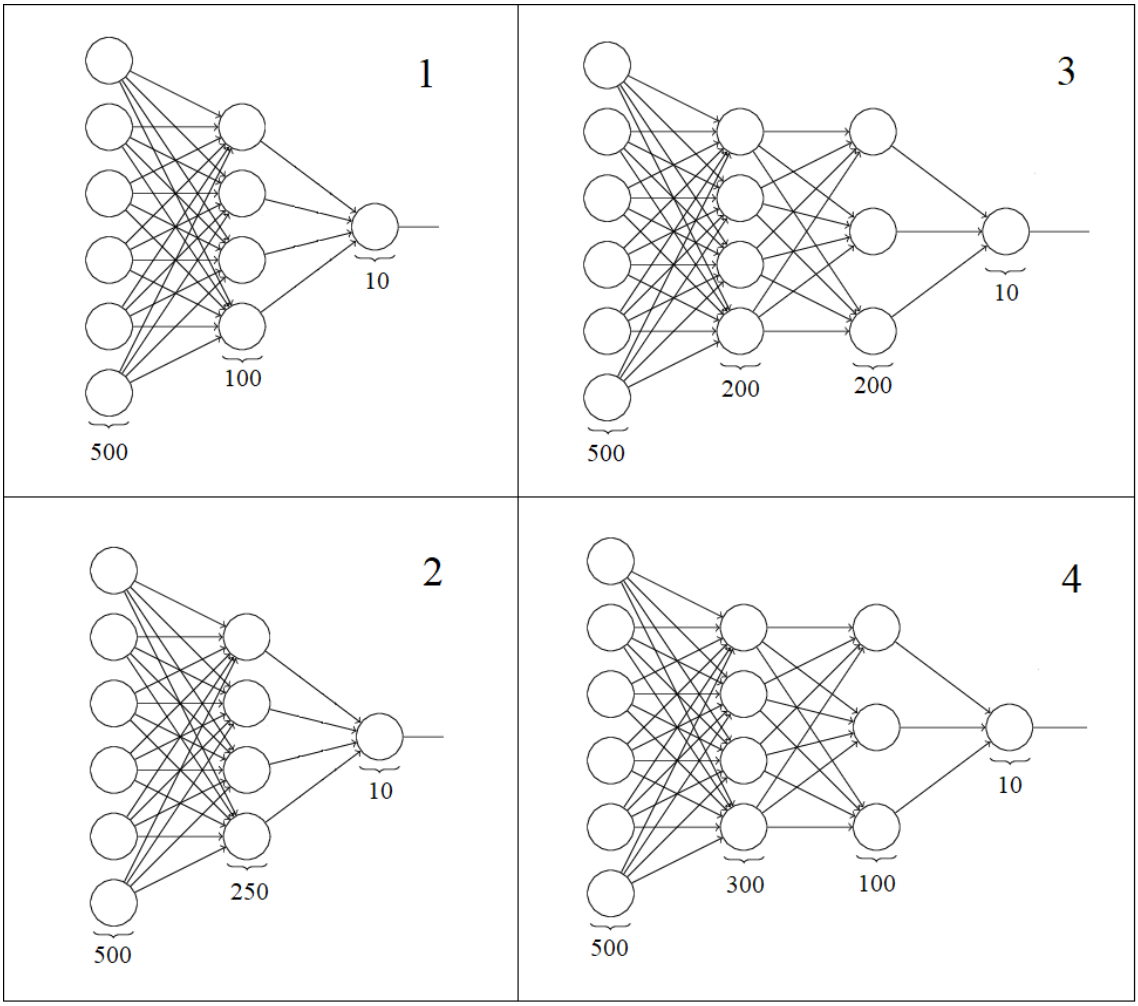

Рис.4 – Схематичне зображення моделей для реалізації

Перша та друга схема містять по одному внутрішньому шару, різниця лише у плавності стиснення. На третій та четвертій схемі був доданий ще один внутрішній шар. Різниця в них полягає лише в кількості нейронів, четверта схема має більш закріплюючий характер у другому шарі.

3.1.4 Багатомасштабний індекс SSIM (MS-SSIM)

Сприйняття деталей зображення залежить від щільності дискретизації сигналу зображення, відстані від площини зображення до спостерігача та перцептивних можливостей зорової системи спостерігача. Насправді суб'єктивна оцінка цього зображення змінюється, коли ці чинники варіюються. Метод однієї шкали, може підійти лише конкретних конфігурацій. Багатомасштабний метод - це зручний спосіб увімкнення деталей зображення з різною роздільною здатністю. Цей метод SSIM буде використаний для оцінки якості зображення. Приймаючи на вхід еталонний та спотворений сигнали зображення, система ітеративно застосовує фільтр низьких частот та зменшує відфільтроване зображення у 2 рази. Ми позначаємо вихідне зображення як масштаб 1, а найвищий масштаб як масштаб M, який виходить після *M - 1* ітерацій. На *j*-му масштабі обчислюються контрастне порівняння та структурне порівняння, які позначаються як *cj (x, y)* та s_j *(x, y)* відповідно. Порівняння яскравості розраховується лише за шкалою M і представляється як lM(x, y).

$$l(\mathbf{x},\mathbf{y}) = \frac{2\,\mu_x\,\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \tag{15}$$

$$c(\mathbf{x},\mathbf{y}) = \frac{2\,\sigma_x\,\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{16}$$

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\,\sigma_y + C_3}, \tag{17}$$

Загальна оцінка SSIM виходить шляхом об'єднання вимірів у різних масштабах за формулою:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [l_M(\mathbf{x},\mathbf{y})]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(\mathbf{x},\mathbf{y})]^{\beta_j} [s_j(\mathbf{x},\mathbf{y})]^{\gamma_j} \tag{18}$$

3.2 Результат розрахунків

3.2.1 Данні для навчання

Для коректної роботи нашої мережі, нам необхідний великий набір даних для навчання. Набір має містити різноманітні фото, на будь-яку тематику. Оскільки така мережа не має можливості розпізнавання, то немає різниці наповнення фото. Але важливими залишаються кількість фото та розмірність. Щодо кількості чим більше тим краще, але і не варто перенавчати мережу, тому що це не має сенсу. А стосовно розмірності, то вона має бути приблизно 500 на 500 пікселів. За всіх вище згаданих пунктів там буде підходити набір фото від компанії Microsoft “MSCoco”. Набір містить близько 40 тисяч фотографій (Рис. 5) із різним наповненням та розмірність від 400 на 400 до 600 на 600 пікселів.

Рис.5 – Приклад фото для навчання

3.2.2 Модель №1

Перша модель буде найпростіша, вона буду слугувати основою для подальшої роботи. Після тривалого процесу навчання, кодування та декодування були отримані такі результати (Рис. 6).

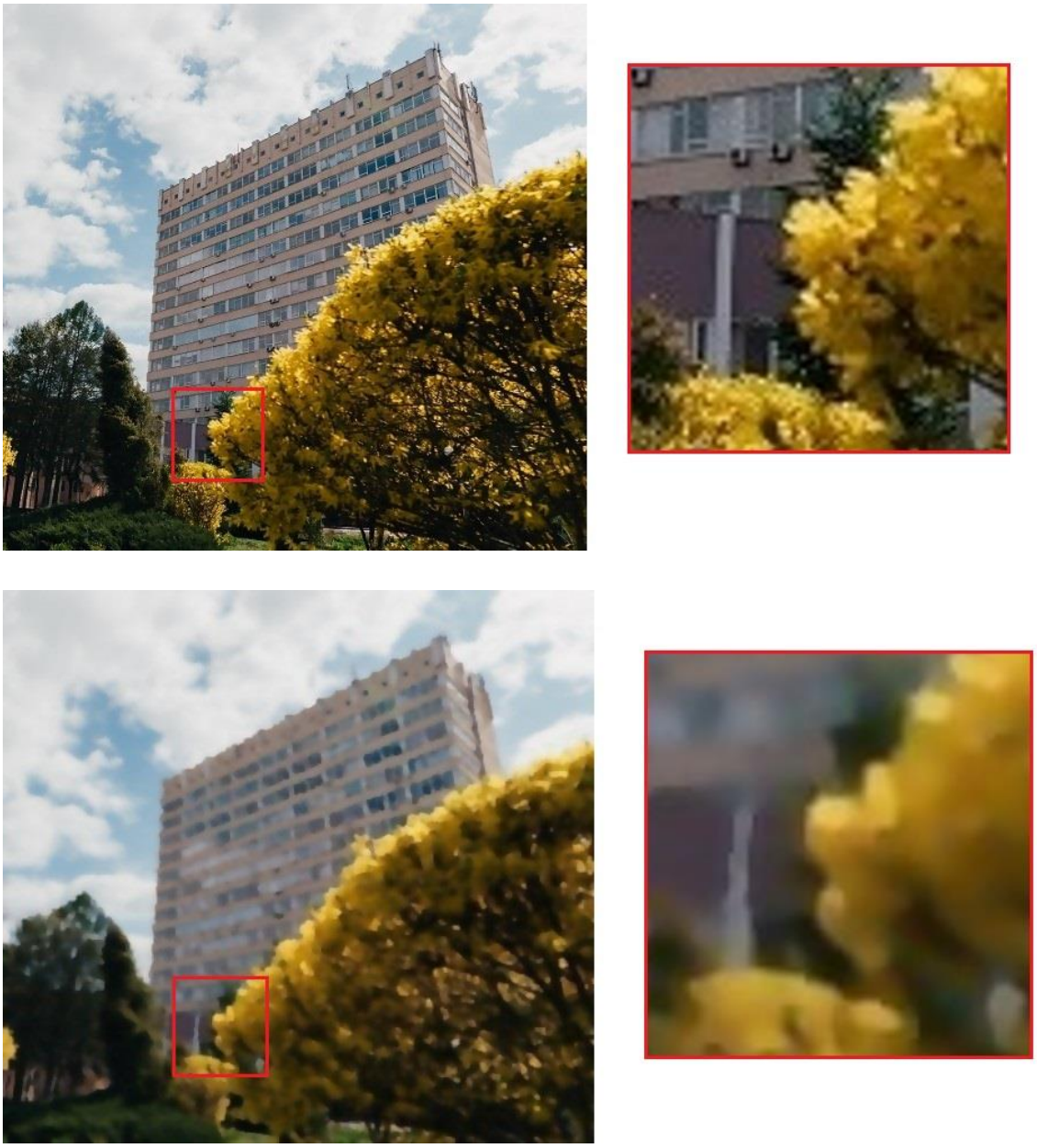

Рис.6 – згори оригінальне зображення; знизу вихідне зображення моделі №1

3.2.3 Модель №2

Для покращення роботи із одним внутрішнім шаром збільшимо кількість нейронів в ньому до кількості 250 одиниць. Це дасть нам змогу більш плавно проходити процес навчання. Отримані такі результати (Рис. 7).

Рис.7 – згори оригінальне зображення; знизу вихідне зображення моделі №2

3.2.4 Модель №3

Тепер розглянемо модель із двома внутрішніми шарами однакової кількості, а саме по 200 нейронів у кожному шарі. В цьому випадку перший прихований шар слугує для навчання, а другий більше для закріплення та для уточнення. Це має збільшити точність, але також і суттєво збільшити час навчання моделі. Отримані такі результати (Рис. 8).

Рис.8 – згори оригінальне зображення; знизу вихідне зображення моделі №3

3.2.5 Модель №4

Розглянемо останню побудовану модель, вона буде складатися із двох внутрішніх шарів, але тепер різної кількості, а саме 300 та 100 нейронів відповідно. Це дасть нам змогу більш плавно проходити процес навчання, а збільшення кількості шарів має збільшити точність. Отримані такі результати (Рис. 9).

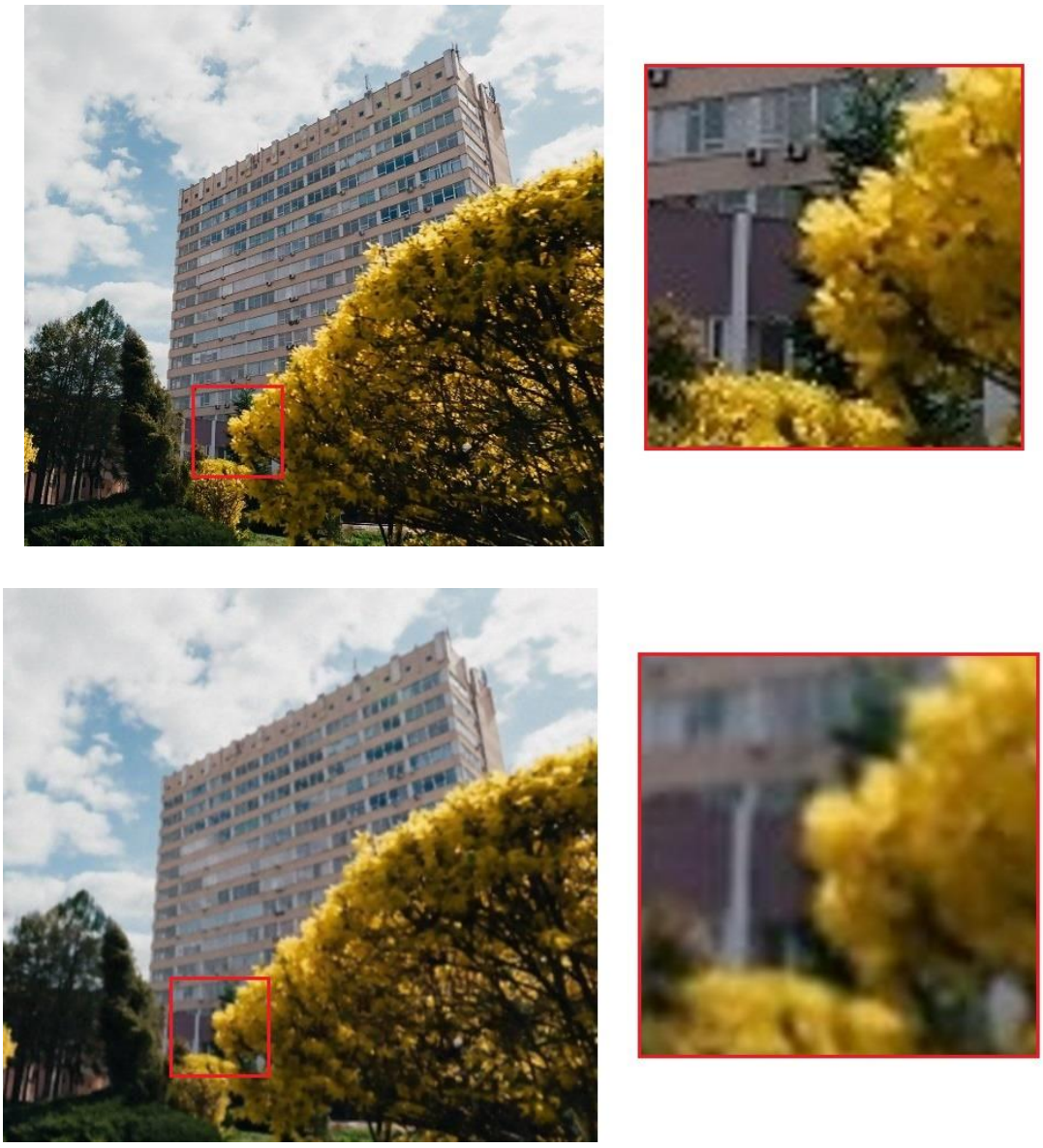

Рис.9 – згори оригінальне зображення; знизу вихідне зображення моделі №4

3.2.6 Порівняння із іншими методами

Щоб краще можна було помітити різницю, порівняємо наші моделі із іншими методиками. Візьмемо JPEG та JPEG2000, які широко використовуються у стисненні зображень. Отримані результати (Рис.10).

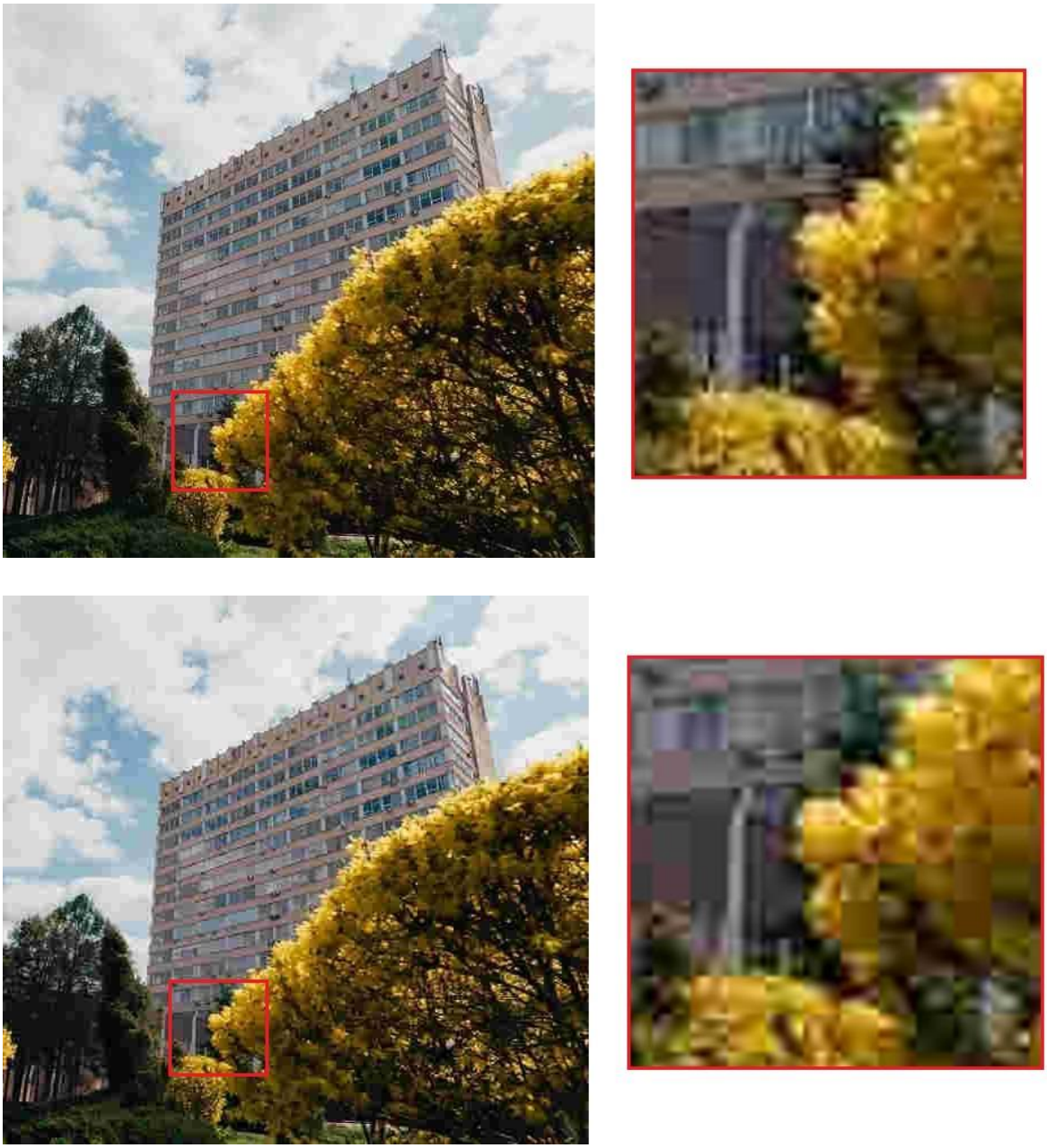

Рис.10 – згори стиснуте зображення jpeg2000; знизу стиснуте зображення jpeg

3.3 Узагальнення та оцінка результатів

Для порівняння та кількісної оцінки всіх вище згаданих методів будемо використовувати метрику для попіксельного порівняння, а саме багатомасштабний індекс SSIM (MS-SSIM). Тому що людське око не здатне порівняти різні зображення та одноголосно прийняти рішення, що краще.

Рис.11 – Результати роботи всіх моделей

Модель 1 MS-SSIM: 0.794	Модель 2 MS-SSIM: 0.819	Модель 3 MS-SSIM: 0.823
Модель 4 MS-SSIM: 0.863	Модель jpeg MS-SSIM:872	Модель jpeg2000 MS-SSIM: 906

Таб.1 – Оцінка всіх моделей

З отриманих результатів в Таблиці 1 можна затвердити, що всі побудовані моделі в тій чи іншій мірі справляються із поставленим завданням, а саме стиснення зображення. При більш детальному порівняння можна сказати, що моделі з одним внутрішнім шаром є менш точними, через менший термін навчання. Але в той же час вони є більш швидкими у порівнянні із моделями в яких вже два внутрішніх шара. Порівнюючи кількість нейронів всередині шару, можна точно сказати, що для даної задачі більше підходить плавний перехід від вхідного шару до вихідного. Кожна наступна модель починаючи з першої має все більше точності.

Порівнюючи із іншими методами (JPEG, JPEG2000) можна сказати, що наша найкраща модель може конкурувати із JPEG на високому рівні. Але з точки зору людини, зображення достатньо сильно відрізняються. В нашій моделі вихідне зображення є менш яскравим, але в той же час воно є більш згладженим і на ньому присутня менша кількість артефактів та шумів. Зображення JPEG в свою чергу є більш поділене на обласні, що збільшує кількість артефактів, але не сильно змінює кольорову гамму. JPEG2000 є модифікованою версією JPEG тому і результати є кращими. Менша кількість артефактів та висока оцінка поки, що залишає цей метод найкращим у порівнянні із іншими.

ВИСНОВКИ

В даній роботі було проведене дослідження стосовно різних моделей стиснення зображення за допомогою нейронних мереж. Були проаналізовані джерела в яких описується дана тематика. Більш детально була розглянута модель рекурентної нейронної мережі для стиснення зображень без втрат.

Була зібрана інформація, яка дає смогу проаналізувати різні підходи в даній тематиці. Після цього було проведено підготовка та обрання потрібних даних. Після проведення навчання мережі було, для прикладу, стиснуте та збільшене до первинних розмірів початкове зображення, що дало змогу порівняти наші моделі.

В даній роботі були побудовані моделі з різними значеннями внутрішніх шарів та різною кількістю нейронів в цих шарах. Було розглянуто, як змінюється вихідне зображення в залежності від цих параметрів. Був зроблений аналіз результатів у порівнянні із іншими способами стиснення зображення.

Всі побудовані моделі є варіаціями із можливих додаткових внутрішніх компонентів. Але обрання саме таких внутрішніх шарів, параметрів та кількості нейронів у вхідному, вихідному та внутрішніх шарах дало нам можливість достатньо близько чисельно наблизитися до іншим методів з якими відбувалося порівняння. Це лише певні варіанти модифікації мережі, яка можливо допоможе покращити модель в майбутньому для більш точної та якісної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. D. A. Huffman, “A method for the construction of minimumredundancy codes,” Proceedings of the IRE, том. 40, стр. 1098– 1101, 1952
2. H. Andrews and W. Pratt, “Fourier transform coding of images,” in Proc. Hawaii Int. Conf. System Sciences, 1968, стр. 677–679.
3. G. K. Wallace, “Overview of the JPEG (ISO/CCITT) still image compression standard,” in Image Processing Algorithms and Techniques, vol. 1244. International Society for Optics and Photonics, 1990, стр. 220–234.
4. C. Christopoulos, A. Skodras, and T. Ebrahimi, “The JPEG2000 still image coding system: an overview,” IEEE trans. on consumer electronics, том. 46, стр. 1103–1127, 2000.
5. X. Zhang, R. Xiong, S. Ma, and W. Gao, “Adaptive loop filter with temporal prediction,” in IEEE Picture Coding Symposium (PCS), 2012, стр. 437–440
6. X. Zhang, S. Wang, Y. Zhang, W. Lin, S. Ma, and W. Gao, “HighEfficiency Image Coding via Near-Optimal Filtering,” IEEE Signal Processing Letters, том. 24, стр. 1403–1407, 2017
7. G. K. Wallace, “The JPEG still picture compression standard,” Communications of the ACM, том. 34, стр. 30–44, 1991.
8. M. W. Gardner and S. Dorling, “Artificial neural networks (the multilayer perceptron)-a review of applications in the atmospheric sciences,” Atmospheric environment, том. 32, стр. 2627–2636, 1998.
9. R. D. Dony and S. Haykin, “Neural network approaches to image compression,” Proceedings of the IEEE, том. 83, стр. 288–303, 1995.
10. C. Manikopoulos, “Neural network approach to DPCM system design for image coding,” IEE Proceedings I (Communications, Speech and Vision), стр. 501–507, 1992.

11. Namphol, S. H. Chin, and M. Arozullah, “Image compression with a hierarchical neural network,” IEEE Trans. on Aerospace and Electronic Systems, том. 32, стр. 326–338, 1996.
12. E. Gelenbe and M. Sungur, “Random network learning and image compression,” in IEEE International Conference on Neural Networks (ICNN), том. 6, 1994, стр. 3996–3999.
13. J. Balle, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Vari- ´ational image compression with a scale hyperprior,” in International Conference on Learning Representations, 2018.
14. E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool, “Soft-to-hard vector quantization for endto-end learning compressible representations,” in Advances in Neural Information Processing Systems, 2017, стр. 1141–1151.
15. E. Ahanonu, M. Marcellin, and A. Bilgin, “Lossless Image Compression Using Reversible Integer Wavelet Transforms and Convolutional Neural Networks,” in IEEE Data Compression Conference, 2018.
16. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, том. 9, стр. 1735–1780, 1997
17. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” arXiv preprint arXiv:1412.3555, 2014
18. K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, “Draw: A recurrent neural network for image generation,” arXiv preprint arXiv:1502.04623, 2015
19. Y. Li, D. Liu, H. Li, L. Li, Z. Li, and F. Wu, “Learning a Convolutional Neural Network for Image Compact-Resolution,” IEEE Transactions on Image Processing, том. 28, стр. 1092–1107, 2019.

20. M. M. Alam, T. D. Nguyen, M. T. Hagan, and D. M. Chandler, “A perceptual quantization strategy for HEVC based on a convolutional neural network trained on natural images,” in Applications of Digital Image Processing XXXVIII. International Society for Optics and Photonics, 2015
21. Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” IEEE Trans. on image processing, том. 13 , стр. 600–612, 2004.
22. R. Song, D. Liu, H. Li, and F. Wu, “Neural network-based arithmetic coding of intra prediction modes in HEVC,” in Visual Communications and Image Processing (VCIP), 2017, стр. 1–4.
23. C. Jia, S. Wang, X. Zhang, S. Wang, and S. Ma, “Spatial-temporal residue network based in-loop filter for video coding,” in Visual Communications and Image Processing (VCIP), 2017, стр. 1–4.
24. W.-S. Park and M. Kim, “CNN-based in-loop filtering for coding efficiency improvement,” in Image, Video, and Multidimensional Signal Processing Workshop (IVMSP), 2016, стр. 1–5.
25. C. Dong, Y. Deng, C. Change Loy, and X. Tang, “Compression artifacts reduction by a deep convolutional network,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, стр. 576–584
26. K. Li, B. Bare, and B. Yan, “An efficient deep convolutional neural networks model for compressed image deblocking,” in International Conference on Multimedia and Expo (ICME), 2017, стр. 1320–1325
27. L. Zhu, Y. Zhang, S. Wang, H. Yuan, S. Kwong, and H. H.-S. Ip, “Convolutional neural network-based synthesized view quality enhancement for 3d video coding,” IEEE Transactions on Image Processing, том. 27, стр. 5365–5377, 2018.
28. R. Soltani and H. Jiang. Higher order recurrent neural networks. arXiv preprint arXiv:1605.00064, 2016.

29. P. Gupta, P. Srivastava, S. Bhardwaj, and V. Bhateja. A modified psnr metric based on hvs for quality assessment of color images. IEEEXplore, 2011
30. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, стр. 1097–1105.
31. Y. Chen and T. Pock, “Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration,” arXiv preprint arXiv:1508.02848, 2015
32. C. Dong, C. C. Loy, and X. Tang, “Accelerating the super-resolution convolutional neural network,” in European Conference on Computer Vision. Springer, 2016, стр. 391–407.
33. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, стр. 1097–1105.

ДОДАТКИ

Додаток А

```
import argparse
import numpy as np
from scipy.misc import imread, imresize, imsave

import torch
from torch.autograd import peremena

encod.load_state_dict(torch.load(args.model))
bina.load_state_dict(
    torch.load(args.model.replace('encod', 'bina')))
deckod.load_state_dict(torch.load(args.model.replace('encod', 'deckod')))

rozkod_vus_1 = (peremena(
    torch.zeros(batch_size, 256, height // 4, width // 4), volatile=True),
                peremena(
                    torch.zeros(batch_size, 256, height // 4, width // 4),
                    volatile=True))
rozkod_vus_2 = (peremena(
    torch.zeros(batch_size, 512, height // 8, width // 8), volatile=True),
                peremena(
                    torch.zeros(batch_size, 512, height // 8, width // 8),
                    volatile=True))
rozkod_vus_3 = (peremena(
    torch.zeros(batch_size, 512, height // 16, width // 16), volatile=True),
                peremena(
                    torch.zeros(batch_size, 512, height // 16, width // 16),
                    volatile=True))
```

```
zazkod_vus_1 = (peremena(
    torch.zeros(batch_size, 512, height // 16, width // 16), volatile=True),
            peremena(
                torch.zeros(batch_size, 512, height // 16, width // 16),
                volatile=True))
zazkod_vus_2 = (peremena(
    torch.zeros(batch_size, 512, height // 8, width // 8), volatile=True),
            peremena(
                torch.zeros(batch_size, 512, height // 8, width // 8),
                volatile=True))
zazkod_vus_3 = (peremena(
    torch.zeros(batch_size, 256, height // 4, width // 4), volatile=True),
            peremena(
                torch.zeros(batch_size, 256, height // 4, width // 4),
                volatile=True))
zazkod_vus_4 = (peremena(
    torch.zeros(batch_size, 128, height // 2, width // 2), volatile=True),
            peremena(
                torch.zeros(batch_size, 128, height // 2, width // 2),
                volatile=True))


parser = argparse.ArgumentParser()
parser.add_argument(
    '--model', '-m', required=True, type=str, help='path to model')
parser.add_argument(
    '--input', '-i', required=True, type=str, help='input image')
parser.add_argument(
    '--output', '-o', required=True, type=str, help='output kod')
parser.add_argument('--cuda', '-g', action='store_true', help='enables cuda')
```

```
image = imread(args.input, mode='RGB')
image = torch.from_numpy(
    np.expand_dims(
        np.transpose(image.astype(np.float32) / 255.0, (2, 0, 1)), 0))
batch_size, input_channels, height, width = image.size()
assert height % 32 == 0 and width % 32 == 0

image = peremena(image, volatile=True)
import network

encod = network.encodCell()
bina = network.bina()
deckod = network.deckodCell()

encod.eval()
bina.eval()
deckod.eval()

if args.cuda:
    encod = encod.cuda()
    bina = bina.cuda()
    deckod = deckod.cuda()

    image = image.cuda()

    rozkod_vus_1 = (rozkod_vus_1[0].cuda(), rozkod_vus_1[1].cuda())
    rozkod_vus_2 = (rozkod_vus_2[0].cuda(), rozkod_vus_2[1].cuda())
    rozkod_vus_3 = (rozkod_vus_3[0].cuda(), rozkod_vus_3[1].cuda())

    zazkod_vus_1 = (zazkod_vus_1[0].cuda(), zazkod_vus_1[1].cuda())
    zazkod_vus_2 = (zazkod_vus_2[0].cuda(), zazkod_vus_2[1].cuda())
```

```
    zazkod_vus_3 = (zazkod_vus_3[0].cuda(), zazkod_vus_3[1].cuda())
    zazkod_vus_4 = (zazkod_vus_4[0].cuda(), zazkod_vus_4[1].cuda())
kod = []
res = image - 0.5
for iters in range(args.iterations):
    encoded, rozkod_vus_1, rozkod_vus_2, rozkod_vus_3 = encod(
        res, rozkod_vus_1, rozkod_vus_2, rozkod_vus_3)

    code = bina(encoded)

    output, zazkod_vus_1, zazkod_vus_2, zazkod_vus_3, zazkod_vus_4 = deckod(
        code, zazkod_vus_1, zazkod_vus_2, zazkod_vus_3, zazkod_vus_4)
    res = res - output
    kod.append(code.data.cpu().numpy())

kod = (np.stack(kod).astype(np.int8) + 1) // 2
export = np.packbits(kod.reshape(-1))
np.savez_compressed(args.output, shape=kod.shape, kod=export)
```