Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Генеративні змагальні мережі з використанням attention механізмів**»

Виконав: студент 4-го року навчання,

Спеціальності
121 Інженерія програмного забезпечення

Студента Залізного Антона Вячеславовича

Керівник Франків Олександр Олександрович,

магістр комп'ютерних наук, асистент

«____» __________________ 20____ р.

Київ – 2022

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“____” _________________20___року

ЗАВДАННЯ

ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Залізному Антону Вячеславовичу

1. Тема роботи «**Генеративні змагальні мережі з використанням attention механізмів**», керівник роботи Франків Олександр Олександрович, магістр комп’ютерних наук, асистент

2. Строк подання студентом роботи 20 травня 2022

3. План роботи

   Анотація

   Вступ

   Розділ 1. Механізм уваги та Трансформер архітектури

   Розділ 2. Генеративні змагальні мережі

   Розділ 3. Побудова та тренування власної генеративної змагальної мережі з використанням Трансформер архітектури

   Висновки

   Список використаних джерел

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	10 жовтня 2021			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	10 жовтня 2021 – 1 листопада 2021			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	1 листопада 2021			
4.	Написання розділів роботи або Постановка експерименту, аналіз отриманих результатів наукового дослідження	1 листопада 2021 – 1 березня 2022			
5.	Проміжний контроль виконання роботи	1 лютого 2022			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	1 січня 2022 – 1 березня 2022			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	3 лютого 2022			
	Розділ 2 (аналітично-дослідницька частина)	24 лютого 2022			
	Розділ 3 (проектно-рекомендаційна частина)	20 березня 2022			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	1 квітня 2022 – 15 травня 2022			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	20 травня 2022			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 10 жовтня 2021 р.

Науковий керівник Франків Олександр Олександрович

Виконавець курсової роботи Залізний Антон Вячеславович

Зміст

Анотація 1

Вступ 2

Розділ 1. Механізм уваги та Трансформер архітектури 4

1.1 Принципи роботи механізму уваги 4

1.2 Типи механізмів уваги 7

1.3 Принципи роботи Трансформера 8

1.4 Приклади нейронних мереж для обробки візуальних даних на основі Трансформера 10

1.4.1 Візуальний Трансформер 11

1.4.2 Фокальний Трансформер 12

1.4.3 Свін Трансформер 13

1.4.4 Сприймач 14

Розділ 2. Генеративні змагальні мережі 16

2.1 Принципи роботи генеративних змагальних мереж 16

2.2 Автокодери 17

2.3 Вектор квантована ГЗМ 19

2.4 Стиль ГЗМ 20

2.4.1 Перший Стиль ГЗМ 21

2.4.2 Другий Стиль ГЗМ 24

2.5 ГЗМ із використанням Трансформера 25

2.5.1 Транс ГЗМ 25

2.5.2 СТранс ГЗМ 26

2.5.3 Свін Стиль ГЗМ 27

Розділ 3. Побудова та тренування власної генеративної змагальної мережі з використанням Трансформер архітектури 29

3.1 Середовище розробки 29

3.2 Проблеми тренування генеративних змагальних мереж 31

3.3 Створення та тренування власної генеративної змагальної мережі на базі Трансформер архітектури.. 33

3.4 Виклик PyTorch моделей з C++ 35

3.5 Покращення 37

Висновки 39

Список використаних джерел 40

Анотація

У даній роботі досліджено застосування механізму уваги у генеративних змагальних мережах. Розглянуто механізм уваги і його типи, архітектуру Трансформер, генеративні змагальні мережі і архітектури, що базуються або використовують Трансформер. Побудовано та натреновано власну генеративну змагальну мережу на базі Трансформер та досліджено проблеми тренування мереж даного типу. Також розглянуто портування мереж для виклику з C++.

Вступ

Протягом багатьох років згорткові нейронні мережі (ЗНМ, Convolutional Neural Networks, CNNs[1]) були і залишаються стандартом для обробки візуальних даних. Проте у даного типу мереж є певна особливість: вони не враховують глобальну структуру зображення, оброблюючи його по частинах. Оскільки операція згортки (фільтр) накладається на маленьку частину зображення, у ЗНМ присутній локальний баєс (упередженість). Таким чином мережа на базі ЗНМ, оброблюючи лівий верхній куток зображення, ніяк не звертає увагу на те, що знаходиться, скажімо, у правому нижньому кутку, хоча це може бути важливо.

Вище розглянута особливість є одночасно і великою перевагою, і важливим недоліком. Вона накладає суттєві обмеження і критично впливає на результати, якщо віддалені частини зображення є пов’язаними між собою. Подібна проблема була і у задачах з обробки природньої мови, наприклад, у машинному перекладі. Так, значення слова у реченні може залежати від інших слів, які можуть бути розташовані у будь-який частині речення. Для вирішення цієї проблеми з’явився механізм уваги. За допомогою нього нейронна мережа може сама вибирати, на яку частину вхідних даних треба приділяти більшу вагу (увагу).

Ідея механізму уваги виявилася дуже вдалою, і згодом з’явився окремий вид мережі, який базується на цьому модулі – Трансформер. Дана архітектура не має жодного баєсу стосовно структури, що дозволяє їй гнучко підлаштовуватися під будь-які дані.

Незважаючи на величезний успіх, у механізмі уваги і, як наслідок, у Трансформера є дуже суттєвий недолік – квадратична складність. Алгоритм для розрахунку уваги займає багато часу, що робить модель повільною як під час тренування, так і під час використання. Проте найгіршим є те, що

механізм уваги неможливо використати, якщо розмір вхідних даних є великим, як наприклад, зображення. Тим не менш, існує достатня кількість досліджень, які доводять, що Трансформер архітектури та механізм уваги все ж таки можливо успішно застосовувати до задач машинного зору.

Генеративні змагальні мережі використовуються для генерації візуальних даних. Ідея архітектури проста: є генератор, що створює зображення, і є дискримінатор, що вирішує, чи є зображення справжнім, чи його створив генератор. Роль дискримінатора зазвичай виконує мережа на базі згорткових нейронних мереж. Через це ГЗМ, як і ЗНМ, має локальний баєс щодо структури даних. Для ГЗМ це особливо критично, адже це суттєво погіршує якість згенерованого зображення і швидкість тренування.

Мета цієї роботи - застосування механізму уваги та / або Трансформер архітектури у генеративних змагальних мережах. Для цього потрібно дослідити види уваг та Трансформер архітектур, а також наявні підходи їх використання у задачах машинного зору. Також необхідно побудувати власну архітектуру ГЗМ, натренувати модель і перевірити її на бенчмарк датасетах (CIFAR-10, ImageNet 64x64, FFHQ, STL-10 тощо). Це дозволить суттєво покращити якість згенерованих зображень та гнучкість моделі.

Варто зазначити, що у даній роботі не будуть розглядатися базові принципи машинного навчання. Якщо читача цікавлять основи машинного та глибинного навчання, пояснення принципів роботи та тренування нейронних мереж, детальний опис згорткових та рекурентних нейронних мереж, то наполегливо рекомендується звернутися до попередньої роботи автора «Розпізнавання об'єктів в режимі реального часу для мови жестів» [2]

Розділ 1. Механізм уваги та Трансформер архітектури

1.1 Принципи роботи механізму уваги

Ще з самого початку глибинне навчання в деякому сенсі намагалося емітувати роботу людського мозку. Тому ідея уваги з'явилася достатньо природньо. Коли ми дивимося на текст, то ми не концентруємося на кожній літері, а радше на співвідношенні між словами. Так, одне і теж саме слово може мати радикально різні значення в залежності від контексту, тобто інших слів у реченні. Теж саме із зображеннями. Людина не дивиться на кожний піксель або міліметр картини, а звертає увагу лише на певні частини, які її цікавлять.

Мета уваги – порахувати, яку інформацію потрібно враховувати більше всього, а яку варто відкинути. У тексті, наприклад, було б логічним, якщо вага сполучників або часток була значно меншою, ніж, скажімо, іменників і дієслів. Також необхідно якось зрозуміти і відрізняти значення слова в залежності від інших слів у реченні. Наприклад, дано такі речення: «у школі є уроки історії» та «мій друг полюбляє цікаві історії». Слово «історія» має різні значення, і потрібно, щоб мережа, вирішуючи сенс слова, звертала достатню увагу на слова «школі» і «уроки» у першому реченні, і «друг» та «цікаві» у другому. Отже, після застосування уваги потрібно, щоб абсолютне значення «важливих» слів залишилося достатньо великим або навіть збільшилося, а значення несуттєвих слів стало маленьким. Тобто для кожного слова треба вирішити, збільшити чи зменшити його значення в залежності від інших слів. Відповідно потрібно порахувати, як кожне слово корелюється (є пов'язаним) із іншими.

Механізм уваги найперше з'явився у дослідженні з обробки природньої мови (з англ. NLP) з назвою «Neural Machine Translation by jointly learning to align and translate»[3]. До цього у даній сфері було

популярним використовувати рекурентні нейронні мережі, адже вони дозволяють оброблювати послідовності даних будь-якої довжини, а текст, як відомо, це набір слів. Проте у цього типу мереж є певні недоліки, які буде розглянуто пізніше. Справжнього визнання механізм уваги досягнув завдяки дослідженню «Attention Is All You Need»[4], яке і запропонувало архітектуру Трансформер, яку також буде описано згодом.

Перед Трансформером у цьому домені природньої мови були поширені рекурентні нейронні мережі. РНМ можуть приймати на вхід або віддавати на виході послідовність даних. Мережі цього типу тримають у собі стан. Його ще називають контекстом або прихованим станом, але по суті це просто тензор. Хоча РНМ непогано справлялися із задачею, у них є суттєвий недолік. Вони дуже повільні й мають проблему зникаючих градієнтів[5]. Тому пізніше було запропоновано покращення – мережу із довгою короткочасною пам'яттю (ДКП, з англ. Long Short-Term Memory, LSTM). Проте навіть це покращення не змогло повністю вирішити проблему. Справа у тому, що з кожним новим словом мережа оновлює свій стан. Відповідно РНМ і ДКП схильні «забувати» (переписувати) інформацію, яка була достатньо давно. Більш того, тренування мереж, які послідовно приймають данні, є значно повільнішим, бо в такому випадку неможливо виконувати паралельні обчислення. За більш детальним описом проблеми автор знову рекомендує звернутися до його попередньої роботи[6]. Механізм уваги зміг вирішити цю проблему і показав феноменальні результати.

Розглянемо, як працює механізм уваги. Варто нагадати, що для обробки тексту речення розбиваються на слова, які потім перетворюються на масиви чисел. Ці операції відповідно називаються токенізацією та вбудовуванням (з англ. embedding). Оскільки ця робота більше спрямована

на комп'ютерний зір, автор утримується від детального пояснення цих функцій. Спершу, використовуючи вхідну послідовність, створюються три вектори: запити (queries), ключі (keys) та значення (values). Довжини усіх, тепер вже чотирьох векторів однакові. Тобто для кожного слова було створено його запит, ключ і значення. Варто розуміти інтуїцію за даними поняттями. Відношення запита одного слова до ключа іншого дозволить нам отримати числову репрезентацію того, як ці слова корелюються. Далі цей проміжний результат потрібно буде помножити на значення слова, і таким чином буде отримана його фінальна вага.

Заглибимося трішки більше у деталі. Для отримання векторів запит, ключ і значення використовують три мережі прямого поширення (з англ. feed forward networks). На вхід кожній подається початкова послідовність. Для того, щоб порахувати кореляцію, необхідно помножити вектор запитів на транспонований вектор ключів. Таким чином буде отримано тензор, який зображує, як кожне слово відноситься до усіх інших, включаючи себе. Також на даному етапі варто зробити масштабування даних (з англ. scaling), щоб уникнути занадто великих значень. Для цього прийнято поділити кожне значення тензора на корінь квадратний з довжини послідовності. Наступним етапом потрібно порахувати, яку частину значень потрібно залишити, а яку відкинути. Для цього використовують функцію softmax на кожному рядку тензора. Вона приймає масив чисел і масштабує його так, щоб кожне число було у проміжку від нуля до одного, і щоб сума усіх чисел дорівнювала одному. Наприклад, якщо на вході числа 7, 2 та 1, то softmax поверне значення 0.7, 0.2 та 0.1 відповідно. Часто дану функцію використовують, щоб перетворити масив чисел на масив ймовірностей. Фактично на даному етапі тензор тримає значення уваги або кореляції слів. Останнім етапом є множення тензора уваги на значення слів.

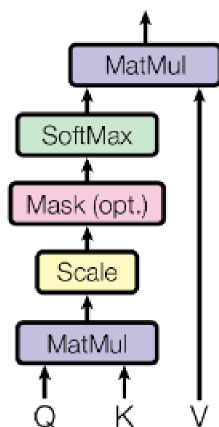

Рисунок 0. Реалізація механізму уваги.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})$$

Q, K та V – запити, ключі та значення

d_k – розмір (довжина) вхідної послідовності

1.2 Типи механізмів уваги

Механізми уваги поділяють на такі типи: м’які та жорсткі (soft and hard), глобальні та локальні (global and local), само-увагу, крос-увагу тощо. Розглянемо кожен на прикладі зображення.

У випадку м’якої уваги оброблюватися буде усе зображення. Із використанням жорсткої – лише частина. В такому разі буде значно менша складність обчислення, але модель неможливо буде диференціювати. Жорстка увага потребує більш складних технік, таких як навчання з підкріпленням для виділення частини зображення, на якому буде рахуватися увага.

Глобальна увага означає обробленням усього зображення за раз. У випадку локальної уваги зображення буде поділено на частини, і увага буде рахуватися на кожній із цих частин.

У випадку само-уваги запити, ключі та значення отримуються з однієї і тієї ж послідовності. Крос-увага (асиметрична увага) передбачає, що запити та ключі будуть отримані з одного вектору, а значення – з іншого.

Отже, приклад розглянутий у попередньому параграфі є м’якою, глобальною само-увагою. Приклад крос-уваги буде згадано пізніше у роботі на базі архітектури Сприймач (з англ. Perceiver).

1.3 Принципи роботи Трансформера

Не буде перебільшенням сказати, що Трансформер - це наразі найпопулярніша архітектура у всьому глибинному навчанні. Архітектури на базі Трансформер, такі як Берт (з англ. Bert), вже давно закріпилися, як найкращі рішення (state of the art) у сфері обробки природньої мови.

Трансформер - це набір із кодерів та декодерів. Спочатку кодери закодовують вхідні дані у проміжну репрезентацію. Декодери використовують її та опціонально свої вхідні дані для отримання результату. Розглянемо приклад машинного перекладу. Кодер отримує речення, яке потрібно перекласти, трансформує його та передає декодеру. Той у свою чергу поступово генерує нові перекладені слова, враховуючі попередні та репрезентацію кодера.

Кожний кодер-блок складається з багатоголового механізму уваги (з англ. multi-head attention) та мережі прямого поширення. Декодер, у свою чергу, - це набір: багатоголовий механізм уваги з маскою (з англ. masked multi-head attention), багатоголовий механізм уваги та мережі прямого поширення. Розглянемо кожний модуль більше детально.

Багатоголовий механізм уваги - це насправді набір з N-модулів простої уваги, яка вже була пояснена у попередньому розділі. Вхідні дані передаються до кожного модуля, який рахує власні запити, ключі, значення та обчислює увагу. Далі результати поєднуються за допомогою мережі прямого поширення. Багатоголова увага має дві переваги. По-перше, різні модулі уваги вивчають різні позиції, на які їм варто фокусувати увагу. По-

друге, так мережа може вивчити різні запити, ключі та значення, що дозволить їй створити різні репрезентації даних. Багатоголовий механізм уваги з маскою використовується лише у декодерах, щоб увага рахувалася лише до попередніх позицій. Наприклад, під час машинного перекладу декодер може використовувати лише слова, що він вже переклав.

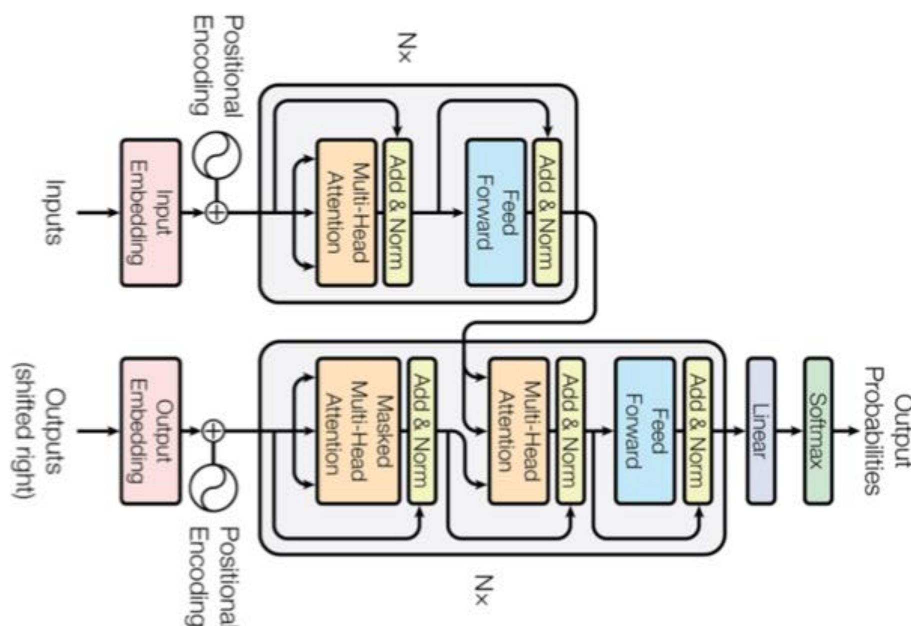

Рисунок 1 Архітектура Трансформера

Також після кожного модуля прийнято додавати нормалізацію. Це робить мережу більш стабільною, бо не дозволяє тензорам ставати занадто великими чи маленькими і до того ж спрощує оптимізацію за допомогою градієнтного спуску. Автори Трансформера використовують батч нормалізацію[7]. Проте на даний момент з'явилося багато альтернативних підходів, такі як нормалізація шару[8] (з англ. layer normalization), нормалізація екземпляру[9] (з англ. instance normalization), нормалізація по пікселю (з англ. pixelwise normalization) тощо. Вибір типу нормалізації залежить від конкретної ситуації та мети дослідника. Варто зазначити, що часто нормалізацію вибирають експериментальним шляхом.

Трансформер працює радше з набором (сетом) даних, ніж із послідовностями. Для того, щоб додати інформацію про позицію або порядок даних, часто застосовують позиційне кодування[10] (з англ. positional encoding) або позиційне вбудовування (з англ. positional embedding) перед першим кодером. Позиційне кодування використовується в оригінальному дослідженні про Трансформер. Цей тип, як і трансформація Фур'є[11], використовує синус та косинус для кодування інформації. Позиційне вбудовування - це просто параметри, які вивчає мережа. Як і у випадку з нормалізацією, часто цей модуль вибирають експериментально. У кінці декодера часто додають лінійний шар, щоб змінити розмір даних на той, що очікується на виході. Можливе також застосування softmax для отримання ймовірності, щоб у випадку машинного перекладу, отримати наступне слово.

Важливим уточненням на думку автора є те, що Трансформер може приймати дані різного розміру. Проте дуже часто розмір даних є фіксованим, бо це дозволяє проводити паралельну обробку батча. У батчі, як відомо, дані мають бути однакового розміру.

1.4 Приклади нейронних мереж для обробки візуальних даних на основі Трансформера

Шалений успіх у застосуванні Трансформера у сфері природньої мови змусив дослідників почати спроби адаптувати цю архітектуру під інші доменні частини, як наприклад, обробку візуальних даних.

Механізм уваги - це центральна та ключова частина Трансформера. Проте у нього є важливий недолік – квадратична складність. Навіть для обробки тексту, де кількість токенів може досягати тисяч, це вже становить проблему. А для візуальних даних, де зображення повної високої роздільної здатності має 1920×1080 пікселів, це стає критичним. Обчислення даних

такого розміру може займати значну кількість часу, а про тренування відповідної мережі годі і говорити. Тому мережі для комп'ютерного зору використовують певні техніки і прийоми, щоб зменшити часову складність без суттєвої втрати точності або іншої метрики. Далі буде розглянуто декілька мереж для задач машинного зору, які побудовані на базі Трансформер.

1.4.1 Візуальний Трансформер

Візуальний Трансформер[12] (з англ. Visual Transformer, ViT) - це одна з найперших успішних спроб застосування Трансформер архітектури у задачах машинного зору. Ідея мережі проста і прямолінійна – розбити зображення на частини (патчі) та використовувати їх, як послідовність «слів» у простому Трансформері.

Дослідження називалося «Зображення варте 16x16 слів» (з англ. An image is worth 16x16 words). Відповідно до назви зображення спершу розбивається на N-частин кожна 16x16 пікселів. Ці патчі розгортаються у вектори розміру 256. Далі використовується лінійна проекція – модуль, мета якого дуже схожа на вбудовування слів. Імплементація цього модуля теж дуже проста – операція згортки, де розмір фільтру дорівнює 256, а розмір каналів – бажаному розміру проекції. Після цього також додається позиційне вбудовування, що було розглянуто раніше. Нарешті цей проміжний результат передається в Трансформер, який складається тільки з кодер блоків. У кінці - багатошаровий перцептрон (мережа прямого поширення), щоб отримати передбачений клас.

Візуальний Трансформер показав непогані результати на багатьох бенчмарк датасетах. Проте даний тип мережі має важливий недолік – розбиття зображення на патчі. Ця операція додає індуктивний баєс, тобто певний формат, як мережі потрібно сприймати дані. Трансформери ж,

навпаки, є дуже гнучкими і здатні самі краще вирішувати, як варто дивитися на дані. До того ж ця архітектура все ще не здатна працювати із зображеннями великого розширення. Звісно, можливо збільшити розмір патчу і, таким чином, частково зменшити складність обчислення. Проте є задачі, для яких ця зміна може критично відобразитися на точності моделі. Візьмемо, наприклад, задачу знаходження людей на зображенні. Якщо людина буде значно меншою, ніж розмір патчу, то є ризик, що модель її не знайде.

1.4.2 Фокальний Трансформер

Архітектура Фокального Трансформера (з англ. Focal Transformer) більше сконцентрувалася на модулі уваги. Дослідження вважає, що спроби застосування локальної уваги або якоїсь простішої уваги, щоб зменшити складність обчислення, негативно впливають на якість моделі. Пропонується нова версія механізму уваги, яка називається фокальною. Ідея полягає у тому, щоб частини зображення звертали більше уваги на регіони ближчі до них і менше на ті, що знаходяться далеко. Таким чином можливо ефективно і вдало змоделювати близькі та далекі залежності.

Усього архітектура має 4 етапи. Кожний складається з лінійної проекції та N Трансформер блоків. Фокальний Трансформер, як і Візуальний, розбиває зображення на патчі і використовує лінійну проекцію. Трансформер також складається лише з кодер блоків і використовує фокальну увагу замість оригінальної. На першому етапі вхідне зображення розбивається на 4x4 патчі та використовується лінійна проекція. Результат передається на N-блоків Трансформера. На другому та наступних етапах відбувається те саме, проте лінійна проекція, яка у даному дослідженні називається вбудовуванням патча, зменшує кількість патчів удвічі, причому збільшуючи їх розмірність також у два рази.

Розглянемо детальніше фокальну увагу. Нехай дано зображення розмірністю 20x20 патчів. Спочатку воно розбивається на вікна (частини) 5x5 кожне розміром 4x4. Потрібно порахувати увагу для центрального вікна. Існує 3 рівня точності. На першому беруться сусідні 8x8 токенів, на другому – 12x12, третьому – 20x20. На другому та третьому рівнях також робиться агрегація (з англ. pooling), яка розповсюджено використовується у згорткових мережах. Таким чином отримуємо 8x8, 6x6 та 5x5 патчів, які розгортаються у одномірний масив та конкатенуються. Новий вектор використовується для створення ключів та значень, а центральний блок, для якого потрібно порахувати увагу, - для запитів. Результат передається у звичайний механізм уваги.

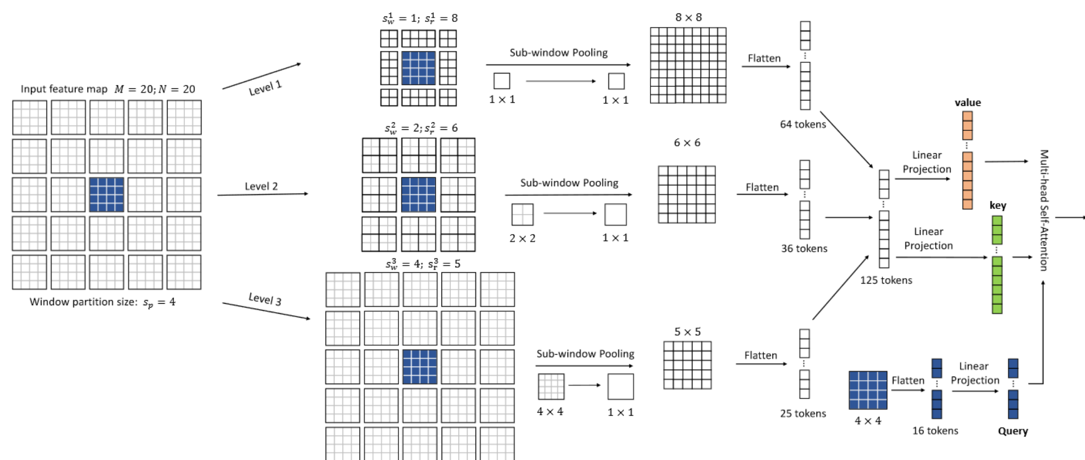

Рисунок 2 Фокальна увага

1.4.3 Свін Трансформер

Свін Трансформер (з англ. Swin Transformer) також пропонує власний механізм уваги. Дослідження вважає, що візуальні об’єкти можуть бути різного розміру, тому патчі фіксованої розмірності не є найбільш вдалим підходом.

Архітектура є дещо подібною на Фокальний Трансформер та має 4 етапи. Спочатку зображення розбивається на патчі 4x4. На першому етапі

виконується раніше згадана лінійна проекція. На наступних етапах замість неї виконується об’єднання патчів, що зменшує їх кількість удвічі й збільшує їх розмірність також у два рази. Далі патчі передаються в Свін Трансформер блоки. Ці блоки у свою чергу складаються з двох кодер блоків із новою свін увагою.

Щоб зменшити складність обчислення, свін увага є локальною. На першому етапі зображення розбивається на 8 вікон, другому – 4, третьому – 2, четвертому – 1. Далі увага обчислюється звичайним чином на цих локальних вікнах. Ключовим елементом у дизайні свін уваги є зміщення вікна. Наприклад, на другому етапі вікно було б розташовано прямо посередині зображення, і регіонів для розрахунку уваги виявилося б дев’ять. Як вже було сказано раніше, Свін Трансформер блок складається з двох кодерів. Саме у другому виконується зміщення вікна для розрахунку уваги. Це дозволяє порахувати увагу на межі декількох регіонів, що значно покращує ефективність моделі.

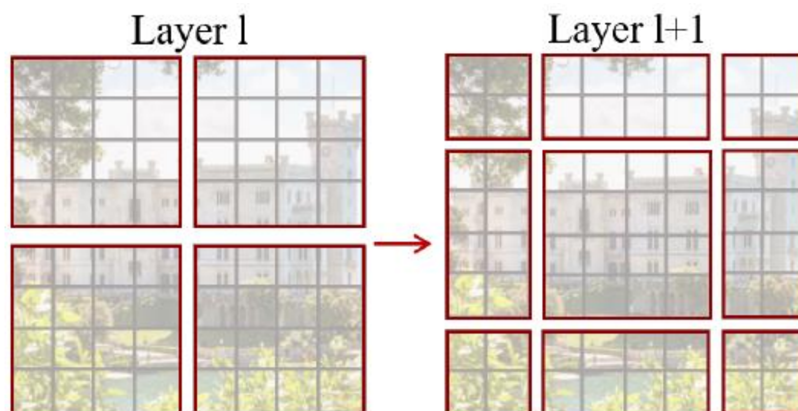

Рисунок 3 Зсув вікон у Свін увазі

1.4.4 Сприймач

Сприймач (з англ. Perceiver) здатен працювати із будь-якою модальністю: текст, зображення, звук тощо. Архітектура використовує крос увагу, щоб відфільтрувати найважливіші вхідні дані у невеликий тензор, що

дозволяє цій мережі масштабуватися до даних будь-якого розміру. Головним гаслом дослідження є цитата відомого сучасного дослідника Яна ЛеКуна: «But, given the increasing availability of large datasets, is the choice to bake such biases into our models with hard architectural decision the correct one? Or are we better off building in as much flexibility as possible, and encouraging the data to speak for itself».

Сприймач вирішує проблему квадратичної складності за допомогою латентного (з англ. latent) вектору. Усі вхідні дані сприймаються, як масив байтів. Із латентного масиву створюються запити, із вхідних даних – ключі та значення. Далі вони використовуються для розрахунку уваги. Це називається крос увага. Результат передається до оригінальних Трансформер блоків. Ці дії можуть повторюватися N разів. Тобто Сприймач ітеративно «опитує» вхідні дані, намагаючись знайти та дістати корисну інформацію. Ваги Трансформер блоків можливо зробити спільними, щоб ще більше зменшити часові витрати та витрати по пам'яті. Також до вхідних даних цілком можливо додати інформацію про позицію, час та навіть тип модальності.

Як можна побачити, дана архітектура не робить жодних припущень щодо структури (наприклад, грід пікселів) даних. Вона здатна працювати із будь-якою модальністю і даними будь-якого розміру. До того ж розмір Трансформер блоків і складність обчислення ніяк не залежать від розмірності вхідних даних.

Розділ 2. Генеративні змагальні мережі

2.1 Принципи роботи генеративних змагальних мереж

Генеративні змагальні мережі[13] (з англ. Generative Adversarial Networks, GAN) були вперше запропоновані Яном Гудфелоу у 2014. Це було одним із найуспішніших досліджень на той час у сфері генерації зображень. Хоча сьогодні існують певні альтернативи, такі як дифузійні моделі (з англ. diffusion models), ГЗМ продовжують розвиватися та залишатися стандартом для генерування візуальних даних.

Генеративна змагальна мережа - це насправді набір із двох мереж: генератора та дискримінатора. Другого інколи прийнято називати критиком. Генератор, не складно здогадатися, генерує зображення, а дискримінатор його оцінює. У оригінальному підході ця оцінка має лише два значення: справжнє або фальшиве. ГЗМ можна розглядати, як вчителя та учня, проте, як пізніше буде пояснено, така аналогія може ввести в оману. Краще думати про генеративні змагальні мережі, як про фальсифікатора (генератор) та контролера (дискримінатор). Контролер гарно розбирається у зображеннях і може з легкістю відрізнити оригінал від фальшивки. Фальсифікатор у свою чергу намагається навчитися обманювати контролера, створюючи все більш якісні імітації.

Ключовою особливістю генеративних змагальних мереж є те, що дискримінатор може «навчати» генератора. Генератор приймає на вхід вектор, який дуже часто є просто шумом і трансформує його на зображення. Далі це зображення йде на оцінку до дискримінатора, який намагається зрозуміти, чи прийшло це зображення із справжнього датасету або було згенеровано дискримінатором. Відповідно під час зворотного поширення помилки (з англ. backpropagation) генератор отримує градієнти, які

показують, як потрібно змінити ваги, щоб змінився результат (оцінка) генератора.

Метрику генеративних змагальних мереж легко придумати. Це точність дискримінатора. Відповідно генератор та дискримінатор грають у міні-макс гру. Мета генератора – мінімізувати метрику, тобто змусити дискримінатора просто вгадувати, де є справжнє зображення, а де - підробка. Дискримінатор у свою чергу намагається навчитися добре розпізнавати підробки. Тобто він намагається тримати точність близько до одиниці. Хоча ідея ГЗМ здається простою, на практиці тренування є надзвичайно складним. Детально проблеми тренування даного типу мереж буде розглянуто у наступному розділі.

У наступних розділах буде спершу розглянуто автокодери - генеративні мережі, які існували ще до ГЗМ. Далі буде описано найвпливовіші на даний момент генеративні змагальні мережі. На останок буде наведено приклад ГЗМ на базі Трансформера.

2.2 Автокодери

Перш ніж перейти безпосередньо до генеративних змагальних мереж, варто спершу розглянути альтернативні підходи до генерації зображень.

Автокодери[14] - це набір із кодера та декодера. Дуже часто ці мережі симетричні та достатньо прості, тобто складаються із декількох шарів мереж прямого поширення або згорткових нейронних мереж. Кодер кодує інформацію у проміжний латентний простір. Декодер розкодовує цю інформацію, щоб відновити зображення. Мережа тренується, щоб зображення на вході та на виході було максимально схожим. Природнє питання: у чому сенс такої мережі? Ідея у тому, щоб навчитися зменшувати розмірність даних до якогось малого латентного вектора так, щоб умістити

усю корисну та необхідну інформацію для подальшого відновлення оригіналу за допомогою декодування.

Автокодери мають великий набір застосувань. Їх використовують для семантичної сегментації, прибирання шуму з зображення, відновлення частини зображення, що було втрачено тощо. Стискання (компресія) даних, здавалося, мала б бути основним застосуванням, проте вона не дає жодних переваг у порівнянні з існуючими алгоритмами, які не використовують машинне навчання.

Автокодери, тим не менш, не можуть бути використанні для генерації. Цілком можливо передати на вхід декодеру якийсь випадково згенерований вектор, проте дуже часто декодер буде повертати зображення, схоже на шум. Справа у тому, що користувач не знає розподілу даних, з яким працює декодер. Відповідно, на вхід декодеру приходять дані, з якими він просто не знайомий і не може нормально працювати.

Для вирішення цієї проблеми було створено варіаційні автокодери[15] (з англ. variational autoencoders). У даному випадку метою мережі є перетворення даних до латентного вектора з визначеним середнім значенням і стандартним відхиленням. Тоді користувач може взяти випадковий вектор із цього розподілу й передати декодеру, який на цей раз вже зможе згенерувати адекватне зображення.

Іншим покращенням автокодерів є квантування. Квантовані варіаційні автокодери[16] (з англ. vector quantized variational autoencoders) вивчають набір можливих значень у латентному векторі. Цей фіксований набір називають кодовою книжкою (з англ. codebook). Після кодування даних кожен елемент вектору замінюється на найближчий згідно евклідової відстані з кодової книжки.

Автокодери наразі рідко використовуються для генерації зображень, бо продукують розмиті та нечіткі зображення.

2.3 Вектор квантована ГЗМ

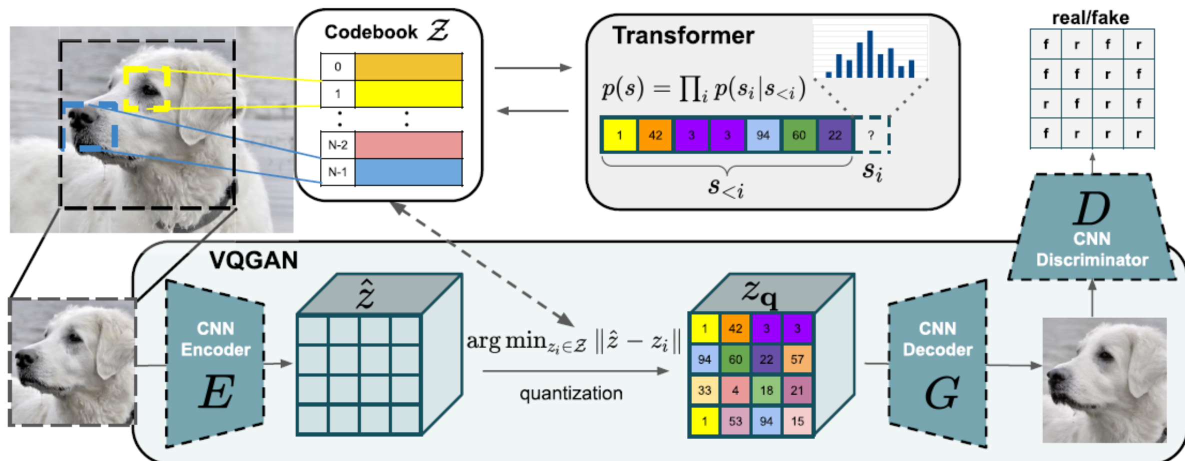

Рисунок 4 Архітектура Вектор Квантованої ГЗМ

Вектор квантована ГЗМ (з англ. Vector Quantized GAN, VQ GAN) - це одне з перших вдалих досліджень, яке якісно та ефективно генерує зображення великої розмірності. Варто зазначити, що генерація зображень великого розміру - це непроста задача, бо потрібно, щоб локально та глобально зображення виглядало реалістично. ВК ГЗМ використовує згорткові нейронні мережі, щоб вивчити кодову книжку візуальних частин, з яких потенційно може складатися зображення, та застосовує Трансформер щоб змоделювати їх композицію.

Спочатку ВК ГЗМ вивчає кодову книжку, використовуючи автокодер, побудований на базі згорткових нейронних мереж. Однією з цікавих особливостей даної мережі є те, що вона використовує змагальний підхід вже на цьому етапі. Функція помилки включає не тільки помилку реконструкції (з англ. – reconstruction loss), як у звичайному автокодері, а й оцінку дискрімінатора.

Після вивчення кодової книжки кодер прибирається і тренуються дискримінатор, Трансформер та декодер з автокодера, який виконує роль генератора. Задача Трансформера – авторегресивно згенерувати композицію зображення. Авторегресивно означає, що елементи генерується послідовно, один за одним, до того ж наступний елемент створюється, враховуючи усі попередні до нього. Композиція зображення - це масив візуальних частин із кодової книжки.

Дискримінатор ВК ГЗМ також побудований на базі ЗНМ. Його особливістю є те, що він розбиває зображення на частини та оцінює на правдоподібність кожну з них. По-перше, це зменшує складність обчислення, особливо на зображеннях великої розмірності. По-друге, це дозволяє локально оцінювати зображення.

2.4 Стиль ГЗМ

На суб’єктивну думку автора компанія Nvidia зробила найвпливовіші й найважливіші дослідження у генеративних змагальних мережах.

Першим ГЗМ Nvidia був Прогресивний ГЗМ[17] (з англ. Progressive GAN). Ідея була такою: спочатку навчити генератор створювати зображення розмірності 8x8, потім збільшити розмірність і поступово навчити генератор для розміру 16x16, використовуючи попередні знання. Коли згенеровані зображення теж досягнуть достатньої якості, знову збільшити розмірність удвічі. Таким чином ГЗМ прогресивно навчається генерувати зображення, починаючи від 4x4 і аж до 1024x1024. Саме у цьому дослідженні були запропоновані ідеї зрівняної швидкості навчання (з англ. equalized learning rate) та нормалізації по пікселю, які використовуються у наступних ГЗМ.

Наступними дослідженням у цій сфері стала серія Стиль ГЗМ. Третя версія не буде розглянута у даному дослідженні. Хоча це дослідження є безумовно успішним, воно більше стосується покращення існуючої архітектури і ідеї, запропоновані у ньому, важко застосувати до інших генеративних змагальних мереж. Стиль ГЗМ пропонував абсолютно новий та альтернативний підхід до генерації зображень. ГЗМ на стилях дуже швидко отримав визнання і шалену популярність. Вперше людина мала складнощі відрізнити зображення справжніх людей від тих, що створила нейромережа[18]. Навіть на сьогодні цей тип ГЗМ вважається ‘state of the art’ рішенням на багатьох бенчмарках, а ідея генерації зображень на стилях все ще активно розвивається.

Дослідження Стиль ГЗМ повністю сконцентрувалося на генераторі. Дискримінатор - це майже стандартна згорткова нейронна мережа. На відміну від усіх попередніх ГЗМ генератор починає з константного вектора, який корегується протягом навчання. Вхідні дані (випадковий латентний вектор) використовуються для того, щоб регулювати та налаштовувати стиль зображення. Фактично, генерація зображення базується на стилях, а не на контенті, як це було до цього.

2.4.1 Перший Стиль ГЗМ

Перший Стиль ГЗМ використовує ідеї Прогресивного і навчається генерувати зображення, поступово збільшуючи розмірність. Кожний етап складається з чотирьох модулів: збільшення розмірності, перші згорткові шари, перше додавання шуму, перше застосування (накладання) стилю, другі згорткові шари, друге додавання шуму та друге застосування (накладання) стилю.

Розмірність збільшується за допомогою підвищення дискретизації (з англ. upsampling). Є різни види і способи, як можна збільшити розмірність,

проте Стиль ГЗМ використовує стандартний - білінійна дискретизація. Далі зображення трансформується за допомогою згорткового шару та додається шум. Додавання шуму суттєво покращує метрики та дає моделі більше простору, гнучкості і «будівельних матеріалів» для генерації. Після цього нарешті накладається стиль. Це робиться за допомогою адаптивної нормалізації екземпляру[19] (з англ. adaptive instance normalization), яка замінює середнє значення та стандартне відхилення розподілу зображення на ті, що були запропоновані вхідним стилем. Додавання відмінних стилів на різних етапах дозволяє контролювати генерацію зображення на різних масштабах (розмірностях).

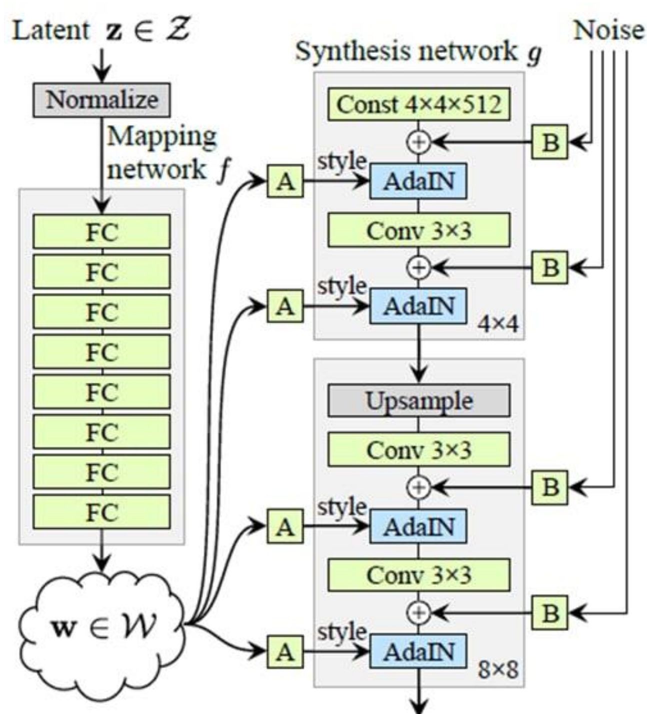

Рисунок 5 Архітектура Першого Стиль ГЗМ

Використання адаптивної нормалізації екземпляру для зміни стилю може здатися читачеві незрозумілим. Яким чином зміна середнього значення та стандартного відхилення може змінити стиль зображення? Варто дивитися на візуальні дані, як на розподіл. У кожного стилю буде свій розподіл. Тоді цілком логічно, що зміна середнього значення та

стандартного відхилення, які є основними характеристиками будь-якого нормального розподілу[20] (розподілу Гауса), призведуть до зміни стилю. Для кращого розуміння ідеї рекомендується прочитати дослідження про трансфер стилю (наприклад, «Передача довільного стилю в режимі реального часу з адаптивною нормалізацією»[21]) та власноруч поексперементувати із даною операцією.

Важливою ідеєю також є перетворення вхідного латентного вектору за допомогою мережі трансформації (з англ. mapping network) до того, як із нього буде створено стилі. Наприклад, генератор має створювати зображення чоловіків і жінок. Цілком природньо, що чоловіків із довгим волоссям у датасеті буде небагато, тому цей стиль буде зустрічатися рідко, і модель не буде вміти генерувати такі зображення. Тим не менш користувач вибирає випадковий вектор із нормального розподілу. Проте на прикладі, наведеному вище, явно видно, що розподіл чоловіків і жінок інший. Відповідно, модель має навчитися якось по-іншому сприймати вхідний латентний вектор, що сповільнює тренування та погіршує точність моделі. Мережа трансформації модифікує простір (розподіл) латентного вектору та прибирає комбінації характеристик, що зустрічалися рідко у датасеті. Таким чином інші модулі мережі не витрачають зайві сили і можуть легко сприймати вхідні стилі.

Окрім цього дослідження пропонує змішування стилів (з англ. style mixing) - спосіб регуляризації ГЗМ, який базується на стилях. Метою змішування стилів є зробити сусідні стилі незалежними і непов’язаними один з одним.

2.4.2 Другий Стиль ГЗМ

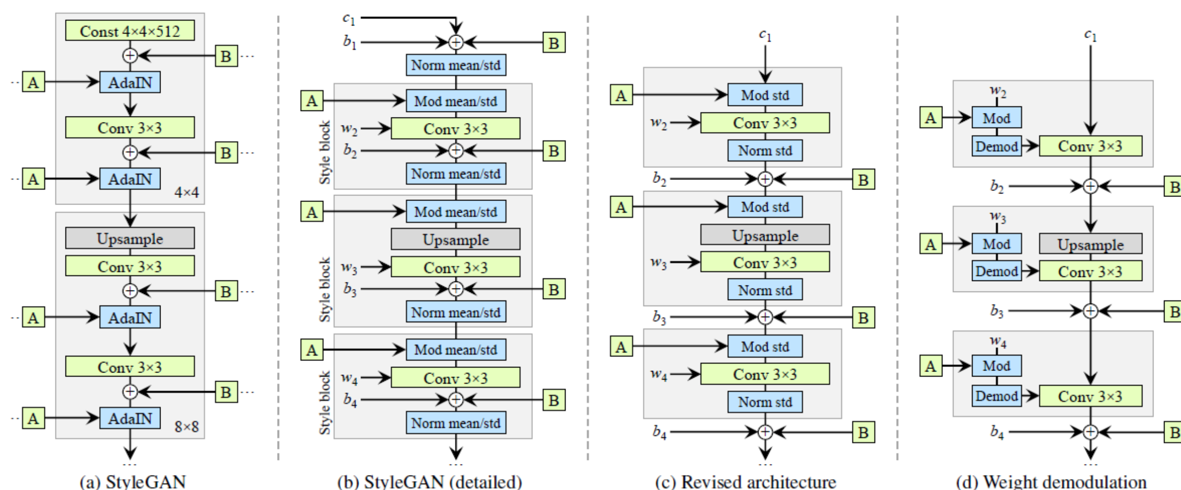

Рисунок 6 a) Перший Стиль ГЗМ b) Перший Стиль ГЗМ (Детально) c) Другий Стиль ГЗМ d) Накладання стилю у Другому Стиль ГЗМ

У Другому Стиль ГЗМ дослідники вирішили відмовитися від Прогресивного підходу. Вони стверджують, що це сильно впливає на локацію об'єктів на зображенні. Так, наприклад, на перших етапах модель вирішить створювати людину посередині, й змінити цю прив'язку на наступних може бути складно. Замість цього тепер використовуються залишкові зв'язки (з англ. – residual connections). Після того як генератор створить карту особливостей (англ. - feature map) для розміру 256x256, він також перетворить її у зображення, інформація з якого поєднується з іншими такими зображеннями, які були створені на інших розмірностях.

Також були вирішено проблему «водяної краплі», яка часто зустрічалася на зображеннях згенерованих Першим Стиль ГЗМ. Проблема була у неправильному застосуванні стилю за допомогою адаптивної нормалізації екземпляру. Дослідники вважають, що крапля - це своєрідний спосіб передачі інформації до наступних етапів. Накладання стилю може також збільшити значення певних карт особливостей у декілька разів, що є небажаним.

Також були зроблені важливі зміни у архітектурі. Кожен Стиль ГЗМ блок тепер спершу накладає стиль, трансформує зображення та нормалізує його. Шум додається після кожного блоку. Збільшення розмірності відбувається раз у три блоки після накладання стилю.

Другий Стиль ГЗМ також пропонує новий вид регуляризації – регуляризація довжини шляху (а англ. path length regularization). Мета: фіксована зміна у стилі має призводити до певної фіксованої зміни у зображенні. Так генератор є більш надійним та послідовним.

2.5 ГЗМ із використанням Трансформера

Трансформер архітектури показують хороший результат на задачах машинного зору. ГЗМ також у свою чергу все частіше починають спроби використати цю ідею. Багато вчених намагаються повністю замінити згорткові шари, щоб оцінити і порівняти, чи зможуть Трансформери краще впоратися із задачею. Інші ж досліджують, як можна поєднати локальний баєс ЗНМ із потужною здатністю Трансформера моделювати довготривалі залежності. Хоча всі ці експерименти наразі все ще є на дуже ранньому етапі, вже існує декілька архітектур, методів та підходів, які дозволили ефективно застосувати Трансформер у сфері генерування візуального контенту.

2.5.1 Транс ГЗМ

Транс ГЗМ намагається повністю замінити згорткові нейронні мережі на Трансформер блоки і в генераторі, і в дискримінаторі. Автори вважають це першою вдалою спробою застосування Трансформер у генеративних змагальних мережах і сподіваються, що їх праця покладе хорошу основу для майбутніх досліджень.

Проблему квадратичної складності механізму уваги було вирішено за допомогою зменшення кількості каналів та локальної уваги. Генератор починає створення зображення із розміру 8x8 та кількісті каналів C у карті особливостей. Кожен раз, коли розмірність збільшується вдвічі, кількість каналів зменшується у 4 рази. Таким чином генерація зображення 4x4 та 256x256 мають однакову часову складність. На великих розмірностях генератор також починає розбивати зображення на частини та рахувати увагу локально на кожній із них.

Автор даного дослідження, тим не менш, вважає це не повним вирішенням проблеми. Зменшення каналів є простою та елегантною ідеєю, проте вона все одно не дозволяє генерувати зображення великої розмірності і до того ж робить модель менш гнучкою. Використання локальної уваги також не є ідеальним, бо цільовий об’єкт генерації може знаходитися на перетині декількох частин. Тобто неможливо поєднати інформацію з різних регіонів. Хорошим покращенням цього підходу було б використання Свін уваги, розглянутої у попередніх розділах.

Дискримінатор Транс ГЗМ також складається лише з Трансформер блоків. Він розбиває зображення на патчі та працює на декількох рівнях. Наприклад, на першому розмір патча може бути PxP, а кількість каналів C/4, на другому - 2Px2P і C/2, третьому - 4Px4P і C тощо. Різний розмір патчів дозволяє отримати інформацію про зображення на різних масштабах і значно покращує ефективність моделі.

2.5.2 СТранс ГЗМ

СТранс ГЗМ показує важливість локальності у задачах із генерації зображень. Автори стверджують, що більшість пікселів звертали увагу на найближчих сусідів (10-20 навколо) або навпаки на занадто віддалених (відстань до яких більше 32), які не приносили жодної корисної інформації.

Відповідно дослідження вважає, що на великих розмірностях значно краще показує себе локальна увага, особливо з можливістю поєднання інформації з різних регіонів, як у свін увазі. Тому для генерації та оцінки зображень великої розмірності СТранс ГЗМ використовує Свін Трансформер.

Дослідження також доводить, що залишкові зв'язки механізму уваги у Трансформер архітектурі негативно впливають на модель. Автори стверджують, що більшість інформації проходить саме через них замість того, щоб вивчати та використовувати правильну увагу. Це значно сповільнює тренування і якість зображення. СТранс ГЗМ пропонує альтернативний модуль для вирішення цієї проблеми.

2.5.3 Свін Стиль ГЗМ

Як можна здогадатися з назви, Свін Стиль ГЗМ - це застосування Свін Трансформера у Стиль ГЗМ. У Свін Трансформері було замінено нормалізацію на адаптивну нормалізацію екземпляру, що дозволяє застосовувати стиль. Також було прибрано додавання шуму. Крім цього запропоновано застосування подвійної уваги, щоб збільшити рецептивне поле моделі.

Подвійна увага – це, насправді, застосування обох свін уваг – звичайної і посунутої. У Свін Трансформері їх застосовували по черзі. Це ж дослідження демонструє хороші результати, якщо застосувати обидві варіанти уваги одночасно і поєднати результат.

Автори також вважають, що додавання відносного позиційного кодування (з англ. relative positional encoding) до кожного патча під час рахування уваги є дуже важливим. Вони стверджують, що згорткові нейронні мережі неявно мали цю інформацію завдяки падінгу.

Дослідження також стверджує, що застосування локальної уваги на зображеннях високої розмірності призводить до появи блокових артефактів. Автори вважають, що причиною є втрата просторової єдності через застосування уваги на патчах. Щоб вирішити цю проблему було модифіковано дискримінатор, що помічає подібні недоліки і сильно штрафує модель за них.

Розділ 3. Побудова та тренування власної генеративної змагальної мережі з використанням Трансформер архітектури

Незважаючи на свою ідейну простоту, генеративні змагальні мережі дуже важко тренувати. Вони нестабільні і не мають чіткого визначення зупинки. Досягти рівноваги між генератором і дискримінатором (Nash Equilibrium[22]) буває складно. Деякі дослідники навіть слідкують за навчанням моделі і змінюють вручну гіперпараметри під час тренування.

У наступних розділах описана практична частина дослідження. Спершу пояснено вибір середовища розробки. Далі наведено головні проблеми тренування генеративних змагальних мереж та поради для їх вирішення. Після цього описано створення та тренування власної моделі автора. Також розглянуто способи виклику PyTorch моделей з C++. На завершення розглянуто можливі покращення та наступні дослідження.

3.1 Середовище розробки

Хоча тренування певних нейронних мереж цілком можливо без наявності хорошого логування, у випадку генеративних змагальних мереж, на суб’єктивну думку автора, це є абсолютно неможливим. Теж саме стосується тренування на CPU або слабкому GPU. Тому перед початком роботи необхідно налаштувати середовище, в якому модель буде тренуватися, вирішити, яку систему логування краще використовувати тощо. Автор цього дослідження рекомендує використовувати редактор коду або інтегроване середовище розробки для створення моделі, Kaggle[23] або Google Cloud[24] для її тренування та TensorBoard[25] або Weights and Biases[26] для логів.

Для даного дослідження було обрано Kaggle, як платформу для тренування моделі. Окрім того, що на Kaggle регулярно публікується велика кількість змагань із машинного навчання, платформа також

безкоштовно надає тридцять годин використання GPU на тиждень. Google Colab[27] у свою чергу також надає безкоштовний доступ до GPU, проте відсутня чітка інформація про квоту, і деякі користувачі повідомляють, що більше десяти днів не мають доступу. Тим не менш, найкращим варіантом на думку автора є оренда віртуальної машини з потужним GPU на хмарі. Існує достатня кількість сервісів, які пропонують великий вибір GPU за різними ціновими категоріями та потужностями.

Для логування у даній роботі був обраний TensorBoard. Хоча цей інструмент було створено для бібліотеки TensorFlow, існують також адаптації для PyTorch, який і використовується в даному дослідженні. TensorBoard дозволяє у реальному часі під час тренування моделі логувати скаляри, зображення, розподіли, гістограми, текст, часові ряди тощо. Усе це можна переглядати у зручному графічному веб-інтерфейсі. Наприклад, у даному дослідженні на сторінці текст логується конфігурація моделі з усіма параметрами. Також кожні 500 ітерацій зберігаються 32 зображення, створені генератором. Окрім цього також логуються помилки та ваги мережі кожні 100 та 500 ітерацій відповідно. TensorBoard відображає їх за допомогою зручних діаграм, що дозволяє розробнику краще розуміти поведінку моделі.

Варто звернути увагу, що на день написання роботи TensorBoard погано інтегрується з Kaggle. Деталі проблеми не стосуються теми роботи, проте труднощі можливо вирішити за допомогою тунелювання трафіку через ngrok. Приклад Kaggle зошита з імплементацією буде наведено пізніше.

Також дуже рекомендується якомога раніше написати клас або функцію для створення контрольної точки (з англ. checkpoint). Деякі нейронні мережі, як, наприклад, ГЗМ, можуть тренуватися кілька днів або

тижнів. Тому за один раз повністю натренувати мережу буває непросто. До того ж завжди є ризик, що тренування аварійно припиниться. Контрольна точка зберігає у собі усю необхідну інформацію про стан тренування, щоб його можна було швидко відновити. Він містить у собі ваги мереж, останній крок, метрики. До контрольної точки потрібно включати усе, що змінюється під час тренування: оптимізатори (у адаптивних є свій стан), класи, що динамічно змінюють швидкості навчання тощо.

3.2 Проблеми тренування генеративних змагальних мереж

Однією з головних проблем тренування ГЗМ є домінація однієї з мереж. Якщо генератор занадто добре навчиться створювати підробки, дискримінатор не зможе тренуватися, і навчання зупиниться. Так само, якщо дискримінатор занадто добре вміє відрізняти справжні зображення від фальшивих, навчання також зайде у глухий кут. Ця проблема пов’язана із зникаючими градієнтами[28] (з англ. vanishing gradients). Пояснити її можна достатньо просто. Наприклад, існує дуже хороший дискримінатор, який із сто відсотковою точністю може відрізнити справжнє зображення від підробки. Тоді маленька зміна у вагах генератора ніяк не впливає на результат – дискримінатор так само легко побачить фальшивку. Отже, градієнти будуть близькі до нуля, і ваги генератора не буде змінено – навчання зупинилося.

Інколи буває, що генератор та дискримінатор не можуть зрушити з місця, бо зайшли в локальний мінімум. Якщо хтось пробує змінити ваги – збільшується помилка, і в результаті навчання також зупиняються. На відміну від інших видів нейронних мереж, де така проблема також присутня, у ГЗМ вона є значно складнішою для вирішення.

Також можливо, що генератор знайде слабке місце дискримінатора і почне зловживати ним. Наприклад, генератор буде створювати дуже схожі

зображення, бо в них присутня якась характеристика, через яку дискримінатор буде вважати їх справжніми. Ця проблема називається обвал моди (з англ. mode collapse). Рішення її достатньо просте – потрібно порахувати статистику по батчу зображень та додати цю інформацію до вхідних даних дискримінатора. Тоді критик зможе легко побачити, що всі зображення занадто схожі, буде сильно штрафувати за це генератора, що змусить його генерувати більш різноманітні дані.

Оригінальна міні-макс помилка, запропонована Гудфелоу, має недолік – дискрімінатор оцінює зображення бінарно: реальне або фальшиве. Проте значно краще було б давати певну оцінку: наскільки зображення виглядає реальним. Це і робить помилка Васерштайна зі штрафом градієнтів (з англ. Wasserstein loss with gradient penalty[29]), яка на думку автора має використовуватися за замовчуванням замість оригінальної у всіх новітніх ГЗМ. Помилка Васерштайна дає кращу стабільність мережі. Проблеми зникаючих градієнтів та колапсу моди трапляються значно рідше.

Існує також ідея починати тренувати одну з мереж пізніше за іншу. Наприклад, було б логічно тренувати генератор на Трансформерах протягом декількох тисяч епох, перед тим як почати тренувати дискримінатор на згорткових нейронних мережах, оскільки Трансформери потребують значно більше часу і даних, щоб вивчити увагу і почати віддавати хоч якийсь адекватний результат.

Автор даного дослідження також експериментував із різними функціями активаціями і вважає, що Leaky ReLU[30] (Rectified Linear Unit) працює найкраще. Ця функція працює добре навіть наприкінці генератора, хоча здавалося природньо було б використовувати TanH[31], бо ця функція повертає значення у проміжку від мінус одного до одного, що легко

масштабується до значення пікселя. Новітні підходи як GELU[32] не дали позитивного результату.

Автор також рекомендує робити перевірку на дурня (sanity check) перед початком серйозного тренування – спробувати тренувати лише генератор без дискримінатора і навпаки. Якщо мережа нормально навчається і помилка падає, значить базова логіка правильна.

3.3 Створення та тренування власної генеративної змагальної мережі на базі Трансформер архітектури

За основу було взято Другий Стиль ГЗМ та Токен ГЗМ[33]. Варто зазначити, що Токен ГЗМ також базується на Другому Стиль ГЗМ й намагається його покращити. Також необхідно зауважити, що на день написання цієї роботи дослідження ще перевіряється і оцінюється експертами, і не було офіційно опубліковано.

Стиль ГЗМ використовує C++ та куда (з англ. cuda) код, щоб пришвидшити певні обчислення. На жаль, це робить неможливим запустити та протестувати цю архітектуру на машинах без Nvidia GPU. Відповідно, першим кроком було адаптування Другого Стиль ГЗМ для запуску на машині автора. Безумовно, зроблені зміни негативно вплинуть на модель, проте подальша легкість розробки, простота внесення змін та комфортний дебагінг того варті.

Головна ідея Токен ГЗМ - це починати генерацію зображення із вектору різних його частин (токенів). Далі за допомогою механізму уваги, який так вдало підходить до цієї задачі, стилі накладаються на токени. Запити створюються на основі токенів, значення - стилів, а ключі - це параметри, які модель вивчає сама. Генерація не тільки на базі стилів, а й токенів дозволяє краще локально контролювати синтез зображень.

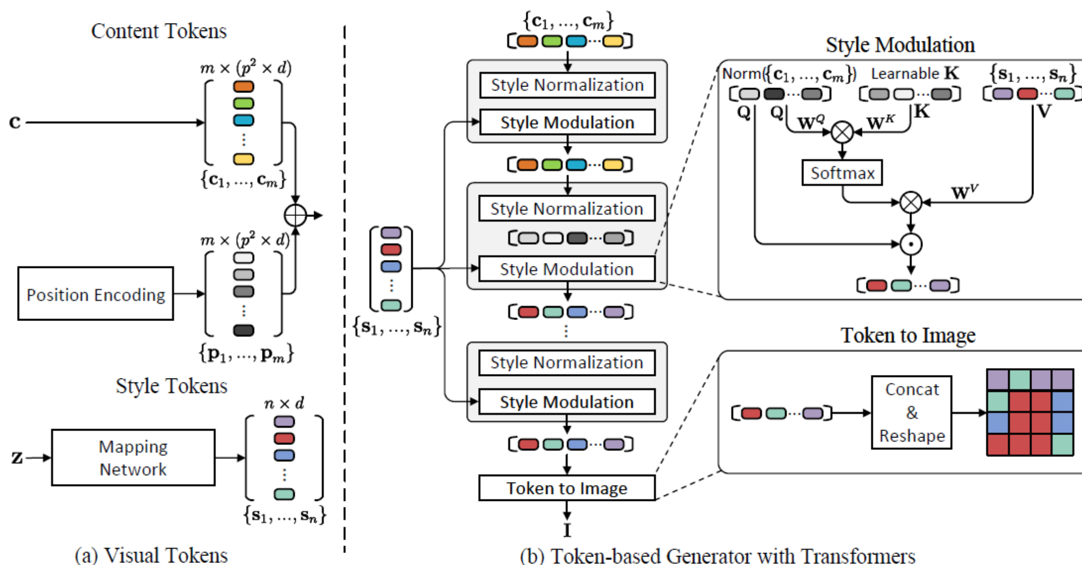

Рисунок 7 Архітектура Токен ГЗМ

Також дослідження стверджує, що застосування адаптивної нормалізації екземпляру знищує інформацію про контент і тому негативно впливає на якість моделі. Запропоновано альтернативний підхід із використанням нормалізації шару.

На жаль, Токен ГЗМ не публікує натренованої моделі, логів або параметрів для тренування. Автор даного дослідження поєднав ідеї, запропоновані у Другому Стиль ГЗМ, Токен ГЗМ та інших ГЗМ, які базується на Трансформерах. Більшість змін є достатньо незначними, тому рекомендується одразу звернутися до коду за деталями. Наприклад, щоб запобігти та унеможливити проблему колапсу моди дискримінатор рахує статистику по батчу. Як оптимізатор було обрано адам (з англ. adam, adaptive momentum). Як функцію активації завжди застосовується Leaky RELU з параметром негативного нахилу 0.2. Як вже було зазначено раніше автор поекспериментував із іншими функціями активацій на різних етапах, проте Leaky RELU, яка допускає негативні значення та не обмежує їх до певного проміжку, показала себе найкраще.

Код, Kaggle зошит, TensorBoard логи з усіма гіперпараметрами та натренована модель (контрольна точка) доступні за посиланням[34]. Модель навчалася генерувати людські обличчя на датасеті відомих людей CelebA[35]. Тренування тривало 10 годин та 70 000 ітерацій. Варто зазначити, що для більш якісних зображень варто дотренувати модель хоча б до 200 000 ітерацій.

Рисунок 8 Зображення, згенеровані після 58 000 (зліва) та 100 500 (справа) ітерацій

3.4 Виклик PyTorch моделей з C++

Python ідеально підходить для побудови та тренування моделі. Проте у продакшн середовищі може бути не найкращим вибором. По-перше, динамічна типізація і інтерпретованість фактично унеможливлює будь-яку оптимізацію. Глобальне блокування інтерпретатора (з англ. global interpreter lock, GIL) не дозволяє виконувати обчислення паралельно. Багато інших мов не дуже добре інтегруються з Python. C++ вирішує усі наведені раніше проблеми: оптимізація під час компіляції, паралельні обчислення і хороша інтеграція з будь-якою іншою мовою. Тому цілком природньо після розробки та тестування моделі спробувати портувати її на C++.

PyTorch має чудову інтеграцію з C++. Він дозволяє у декілька рядків коду портувати модель з Python на C++. В основному це робиться завдяки TorchScript[36] – формату, у який потрібно перевести модель, щоб ефективно запускати її з C++.

Є два способи перевести PyTorch модель у TorchScript формат: трейсінг (з англ. tracing) та скриптування (з англ. scripting). Ідея першого доволі проста – запам'ятати, як модель оброблює дані. Усе що потрібно, щоб перевести модель на TorchScript через трейсінг, - це зробити виклик відповідної функції, у яку передаються модель та приклад вхідних даних. Недоліком цього підходу є те, що трейсінг не враховує умовні оператори (if, elseif, else тощо). Він просто запам'ятовує шлях, обраний під час виклику функції, а всю іншу інформацію відкидає. Трейсінг також прибирає константі значення з майбутньої TorchScript моделі, що є насправді його перевагою. Якщо умовні оператори важливі, необхідно використовувати скріптинг. Проте у цього інструмента також є недолік – він не може працювати з рекурентними нейронними мережами. До того ж, скріптинг компілює код, щоб перевести його у новий формат. Це означає, що часто потрібно додавати анотації, щоб процес міг пройти успішно. Окрім цього скріптинг на відміну від трейсінгу не підтримує певні мовні конструкції Python.

Після отримання TorchScript моделі її нарешті можна запустити з C++. Але не все так просто. Спочатку потрібно підключити необхідні бібліотеки – LibTorch та OpenCV. Важливо пам'ятати, що LibTorch Debug та Release версії не є ABI[37] (Application Binary Interface) сумісними. Це означає, що програма у Debug режимі некоректно буде працювати із використанням Release версії LibTorch. У випадку автора це означало успішну компіляцію і лінкування, проте помилку «Файл не знайдено» під час завантаження моделі. Окрім цього на Windows машині автору вдалося зібрати проект лише з використанням MSVC (Microsoft Visual Compiler).

Проте зі зборкою проекту, ініціалізацією оточення та завантаженням моделі труднощі лише починаються. Щоб модель правильно працювала,

необхідно передати їй дані в очікуваному форматі. Це означає, зробити усю необхідну нормалізацію і попередні обчислення (з англ. preprocessing). У Python версії PyTorch є чудовий модуль, який широко використовується для таких цілей - torchvision. На жаль, C++ версія бібліотеки ще не підтримує цей модуль, що змушує користувача переписувати усю його логіку самому, що доволі непросто. Також автор хоче звернути увагу на те, що OpenCV завантажує зображення у форматі BGRA, хоча більшість PyTorch моделей очікують RGB, бо під капотом використовують бібліотеку Pillow для завантаження зображень. Окрім цього, канали у OpenCV на відміну від PyTorch йдуть на третьому вимірі (з англ. dimension) тензора, а не на першому.

Подолавши усі вищенаведені складнощі, користувач нарешті зможе запустити свою модель у C++. Приклад коду для запуску ResNet18 можна знайти за посиланням[38]. Автор роботи також наполегливо рекомендує дослідити інші інструменти та рішення, наприклад ONNX[39], для використання моделей машинного навчання у продакшн середовищі.

3.5 Покращення

Трансформери, у яких відсутній будь-який баєс, потребують величезної кількості даних та часу, щоб навчитись моделювати дані. У згорткових нейронних мережах певний час було популярно робити аугментацію даних (з англ. data augmentation). Тобто модифікувати справжнє зображення шляхом повороту, віддзеркалення, зміни кольору тощо, таким чином збільшуючи кількість даних. Безумовно, якість таких зображень залишає бажати кращого, і у ЗНМ ця ідея зазнала невдачі. Проте Трансформери можуть скористатися таким підходом. Тому одним із перших кроків у покращенні існуючої моделі на думку автора є застосування аугментації даних для навчання дискримінатора.

Токен ГЗМ вивчає токени контенту, проте цікавою ідеєю було б передавати їх разом зі стилями, як вхідні дані до мережі. Одним із таких досліджень є «Діагональна увага та ГЗМ на основі стилю для роз'єднання стилю вмісту в Генерація та переклад зображень[40]», яке, безперечно, варте уваги.

Цілком можливо також застосувати ідеї Вектор Квантованої ГЗМ, де кодова книжка фактично містить контент токени, які в подальшому використовуються у моделюванні структури зображень. Схожу ідею має дослідження «Вектор квантовоне моделювання зображення з покращеним ВК ГЗМ[41]»

«Вишенькою на торті» у дослідженні може бути створення Обумовленого ГЗМ. Цей тип ГЗМ використовує певні додаткові вхідні дані, окрім латентного вектору, для генерації зображень. Це може бути що завгодно: текст, інше зображення, музика тощо. Нейронна мережа CLIP[42] створена видатною компанією DeepMind[43] і здатна моделювати простір тексту та зображень так, що зображення та текст, що його описує, знаходяться близько у цьому просторі. Тому цю мережу дуже часто використовують у Обумовлених ГЗМ, щоб генерація зображень базувалась на тексті.

Під кінець дослідження також варто порахувати метрики, щоб зрозуміти наскільки якісні зображення генерує ГЗМ. Основними метриками для оцінки ГЗМ на момент дослідження є Inception Score[44] та Frechet Inception Distance[45]. Перша використовує нейромережу, на честь якої і названа, яка і оцінює зображення. Друга рахує статистику і порівнює згенеровані зображення із реальними. На GitHub[46-49] є достатня кількість хороших імплементацій обох цих метрик.

Висновки

Дослідження показало, що застосування Трансформер архітектур у поєднанні з генеративними змагальними мережами реальне і приносить хороші результати. Тим не менш, не варто повністю забувати і відкидати ідеї згорткових нейронних мереж, адже саме локальний індуктивний баєс та просторова інваріантність робить їх настільки потужними і успішними у задачах машинного зору. Відповідно, варто також пробувати поєднати Трансформер та згорткові нейронні мережі у генеративних змагальних мережах.

Генеративні змагальні мережі також продовжують розвиватися і їх навіть застосовують у серйозних продакшн середовищах. Наприклад, Nvidia створила декілька ГЗМ[50-52] для редагування зображення або відео. Здатність комп’ютера генерувати зображення здавалося фантастикою лише декілька десятиліть тому, а зараз все більше і більше досліджень спрямовані на бізнес-застосування цієї вражаючої ідеї.

Список використаних джерел

1. https://en.wikipedia.org/wiki/Convolutional_neural_network
2. http://ekmair.ukma.edu.ua/handle/123456789/21943
3. https://arxiv.org/pdf/1409.0473.pdf
4. https://arxiv.org/abs/1706.03762
5. https://en.wikipedia.org/wiki/Vanishing_gradient_problem
6. http://ekmair.ukma.edu.ua/handle/123456789/21943
7. https://en.wikipedia.org/wiki/Batch_normalization
8. https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html
9. https://pytorch.org/docs/stable/generated/torch.nn.InstanceNorm2d.html
10. https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/
11. https://en.wikipedia.org/wiki/Fourier_transform
12. https://arxiv.org/pdf/2010.11929.pdf
13. https://arxiv.org/pdf/1406.2661.pdf
14. https://en.wikipedia.org/wiki/Autoencoder
15. https://en.wikipedia.org/wiki/Variational_autoencoder
16. https://arxiv.org/pdf/1711.00937.pdf
17. https://arxiv.org/pdf/1710.10196.pdf
18. https://thispersondoesnotexist.com/
19. https://paperswithcode.com/method/adaptive-instance-normalization
20. https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D1%80%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB
21. https://arxiv.org/pdf/1703.06868.pdf
22. https://en.wikipedia.org/wiki/Nash_equilibrium
23. https://www.kaggle.com/
24. https://cloud.google.com/

25.https://www.tensorflow.org/tensorboard

26.https://wandb.ai/site

27.https://colab.research.google.com/

28.https://en.wikipedia.org/wiki/Vanishing_gradient_problem

29.https://paperswithcode.com/method/wgan-gp

30.https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html

31.https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html

32.https://pytorch.org/docs/stable/generated/torch.nn.GELU.html

33.https://openreview.net/forum?id=lGoKo9WS2A_

34.https://github.com/IronTony-Stark/StyleTransGAN

35.https://paperswithcode.com/dataset/celeba

36.https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html

37.https://en.wikipedia.org/wiki/Application_binary_interface

38.https://github.com/IronTony-Stark/ResNet18LibTorch

39.https://onnx.ai/

40.https://arxiv.org/pdf/2103.16146.pdf

41.https://openreview.net/pdf?id=pfNyExj7z2

42.https://openai.com/blog/clip/

43.https://www.deepmind.com/

44.https://arxiv.org/pdf/1801.01973.pdf

45.https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance

46.https://github.com/sbarratt/inception-score-pytorch

47.https://github.com/mseitzer/pytorch-fid

48.https://github.com/w86763777/pytorch-gan-metrics

49.https://github.com/toshas/torch-fidelity

50.https://www.youtube.com/watch?v=cS4jCvzey-4

51.https://www.youtube.com/watch?v=eaSTGOgO-ss

52.https://nv-tlabs.github.io/editGAN/