НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи та методи штучного інтелекту» спеціальності 122 «Комп'ютерні науки та інформаційні технології»
на тему: «Модель генерації тексту на основі візуальної інформації»

Виконав:
студент IV курсу, групи КА-65
Корень Віталій Олександрович _____

Керівник:
Професор, д.т.н.
Зайченко Олена Юріївна _____

Консультант з економічного розділу:
доцент, к.е.н., доцент кафедри ТТПЕ
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
доцент, к.т.н., доцент кафедри ММСА
Коваленко Анатолій Єпіфанович _____

Рецензент:
к.т.н, с.н.с. кафедри ПЗКС ФПМ.
Вішталь Дмитро Михайлович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _________

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Кореню Віталію Олександровичу

1. Тема роботи «Модель генерації тексту на основі візуальної інформації», керівник роботи Зайченко Олена Юріївна, професор, д.т.н.,, затверджені наказом по університету від «25» травня 2020 р. № 1143-с

2. Термін подання студентом роботи 8.06.2020.

3. Вихідні дані до роботи програмне забезпечення для генерації підписів на основі вхідного зображення.

4. Зміст роботи аналіз існуючих архітектур для розв'язку даної задачі, вибір архітектури із перечислених, реалізація процесу навчання мережі, реалізація програмного продукту для взаємодії з нейронною мережею, проведення порівняльного аналізу різних конфігурацій моделі.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) зображення архітектур нейронної мережі та її компонентів, зображення прикладів роботи нейронної мережі, приклади зображень навчальної вибірки, презентація до виступу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О.А., к.е.н., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування теми БДР	03.09.2019	Виконано
2	Огляд літератури за тематикою роботи та її опрацювання	03.09.2019-01.03.2020	Виконано
3	Програмна реалізація моделі	11.03.2020-10.04.2020	Виконано
4	Програмна реалізація взаємодії з моделлю	10.04.2020-20.04.2020	Виконано
5	Написання першого розділу БДР	20.04.2020-27.04.2020	Виконано
	Узгодження теми БДР з керівником	28.04.2020	Виконано
6	Написання другого розділу БДР	01.05.2020-07.05.2020	Виконано
7	Написання третього розділу БДР	14.05.2020-27.05.2020	Виконано
8	Написання четвертого розділу БДР	27.05.2020-29.05.2020	Виконано

Студент Віталій КОРЕНЬ

Керівник Олена ЗАЙЧЕНКО

РЕФЕРАТ

Дипломна робота: 80 ст., 4 ч., 9 табл., 48 рис., 3 дод., 26

джерел.

РОЗРОБКА СИСТЕМИ ДЛЯ ГЕНЕРАЦІЇ ТЕКСТУ БАЗУЮЧИСЬ НА ВІЗУАЛЬНІЙ ІНФОРМАЦІЇ

Об’єкт даного дослідження – Навчальні вибірки зображень та відповідних описів або підписів до них.

Мета та цілі роботи – Дослідити існуючі підходи систем генерації тексту на основі візуальної інформації.

Предмет дослідження – Нейронна мережа, що складається зі згорткової у якості енкодера та рекурентної у якості декодера.

Результатом роботи є система, що здатна генерувати підписи базуючись на зображенні у соціальну мережу. Новизною є створена система зі зручною взаємодією з вищезгаданою штучною нейронною мережею.

ABSTRACT

Thesis contains: 80 p., 4 sections, 9 tabl., 48 fig., 3 appendixes, 26 souces.

DEVELOPMENT OF A TEXT GENERATION SYSTEM BASED ON VISUAL INFORMATION

This study uses the dataset of images and corresponding captions to evaluate the system's performance.

The main aim of this paper is to explore the architectures for image captioning and conditioning language models.

The encoder-decoder architecture that consists of convolution and recurrent neural nets is the purpose of this work and is explored in this work.

The result is the system that uses encoder-decoder architecture to solve the task of image captioning for social networks. The novelty of this study is created system with convenient interaction with the architecture.

Зміст

ВСТУП

На сьогоднішній день розвиток глибинного навчання дав змогу створювати різноманітні системи для розв’язку різних задач замінюючи при цьому людське втручання. Яскравими прикладами можуть бути такі системи як розпізнавання облич, сегментація зображень в комп’ютерному зорі, або машинний переклад чи абстрактний підсумок тексту в обробці людської мови. Крім того є задачі, наприклад відповідання на запитання, де нейронні мережі перевершують людський результат на певних наборах даних. Існують же такі задачі, які стоять на межі комп’ютерного зору та обробки людської мови. Прикладом такої є опис зображення (image captioning), ціллю якої є описати людською мовою, що відбувається на зображенні. Існує безліч застосувань для цієї задачі. Метою даної роботи буде створення системи для генерації підпису до картинок в соціальні мережі.

Розділ 1. ОСНОВНІ ВІДОМОСТІ ПРО АРХІТЕКТУРИ ДЛЯ ОПИСУ ЗОБРАЖЕНЬ.

1.1 Загальний опис архітектури

Зазвичай для Image Captioning використовується архітектури, які включають в себе нейронну мережу, що слугує як енкодер (encoder) та умовної мовної моделі, задача якою є генерувати текст, за умови закодованої енкодером візуальної інформації.

1.2 Енкодер

Енкодером може слугувати багатошаровий парцептрон або як його ще називають – повнозв’язна мережа, на вхід якій подається розгорнуте зображення в вигляді вектору як показано на Рисунку 1.1.

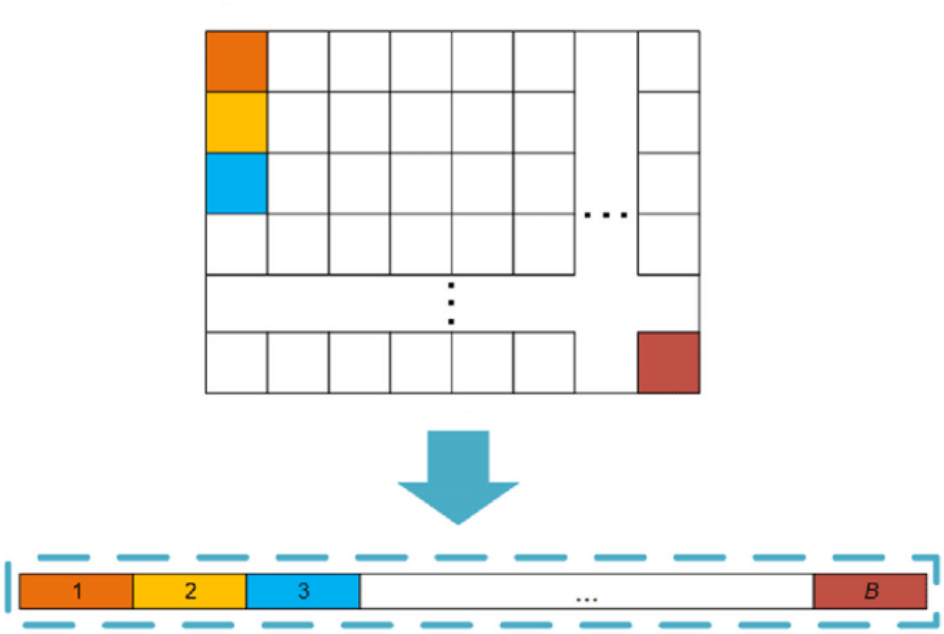


Рисунок 1.1 – Представлення зображення в вигляді одномірного вектору.

Проте це застарілий підхід, який вже не використовується в ком’ютерному зорі по причині своєї неефективності. Причиною неефективності є тип зв’язку такої нейронної мережі – «кожний з кожним», тобто кожний нейрон попереднього шару з’єднаний з нейроном поточного. Натомість, зараз зазвичай використовують згорткові нейронні мережі для декодування візуальної інформації зображення у вектор, чи тензор.

Також є роботи, що використовують заздалегідь претреновані нейронні мережі для виявлення об’єктів в якості енкодера. Основою подібних нейронних мереж зазвичай слугує також згорткова мережа. Виходом такої є вектори для кожного об’єкту, що присутній на зображенні (Рисунок 1.2).

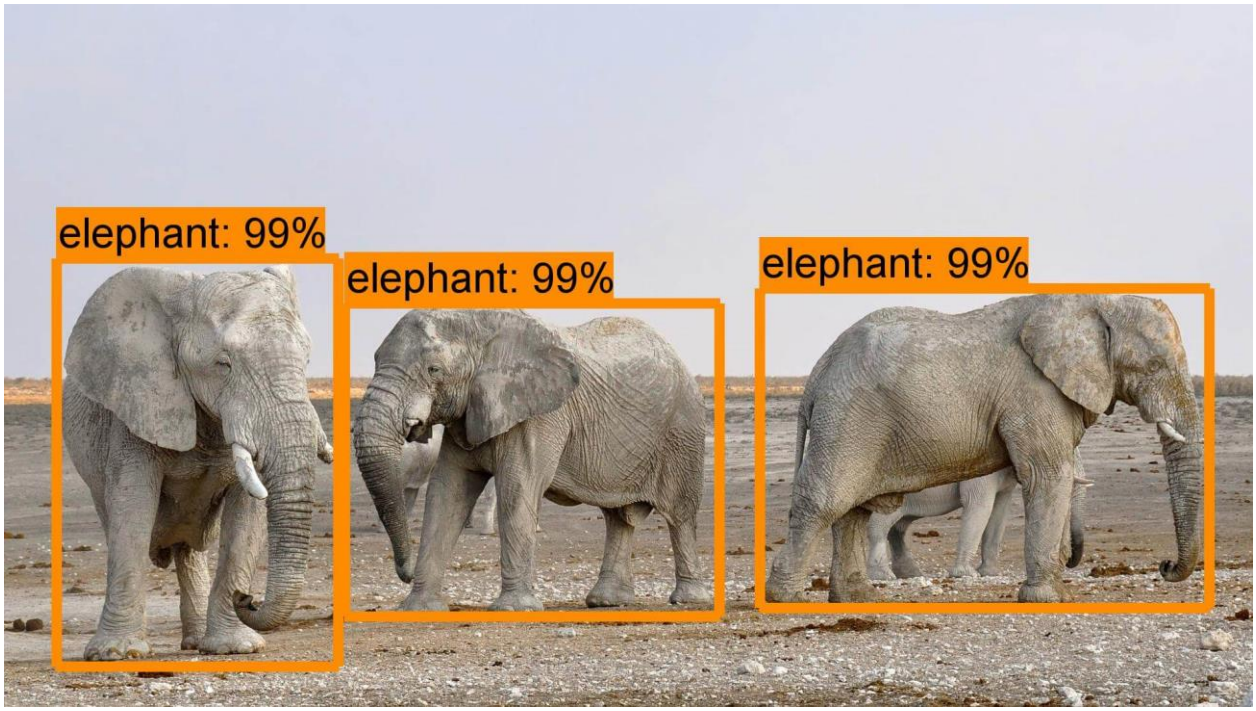


Рисунок 1.2 – Приклад виявлення об’єктів зображення.

1.3 Декодер

Умовною мовною моделлю часто слугують рекурентні нейронні мережі, оскільки вони добре підходять для обробки послідовностей. Проте за останні роки в світі обробки людської мови значно кращі результати показують нейронні мережі на основі архітектури Трансформер, тому результатом цього є поява нових робіт, що використовують такі архітектури в якості декодера (Рисунок 1.3).

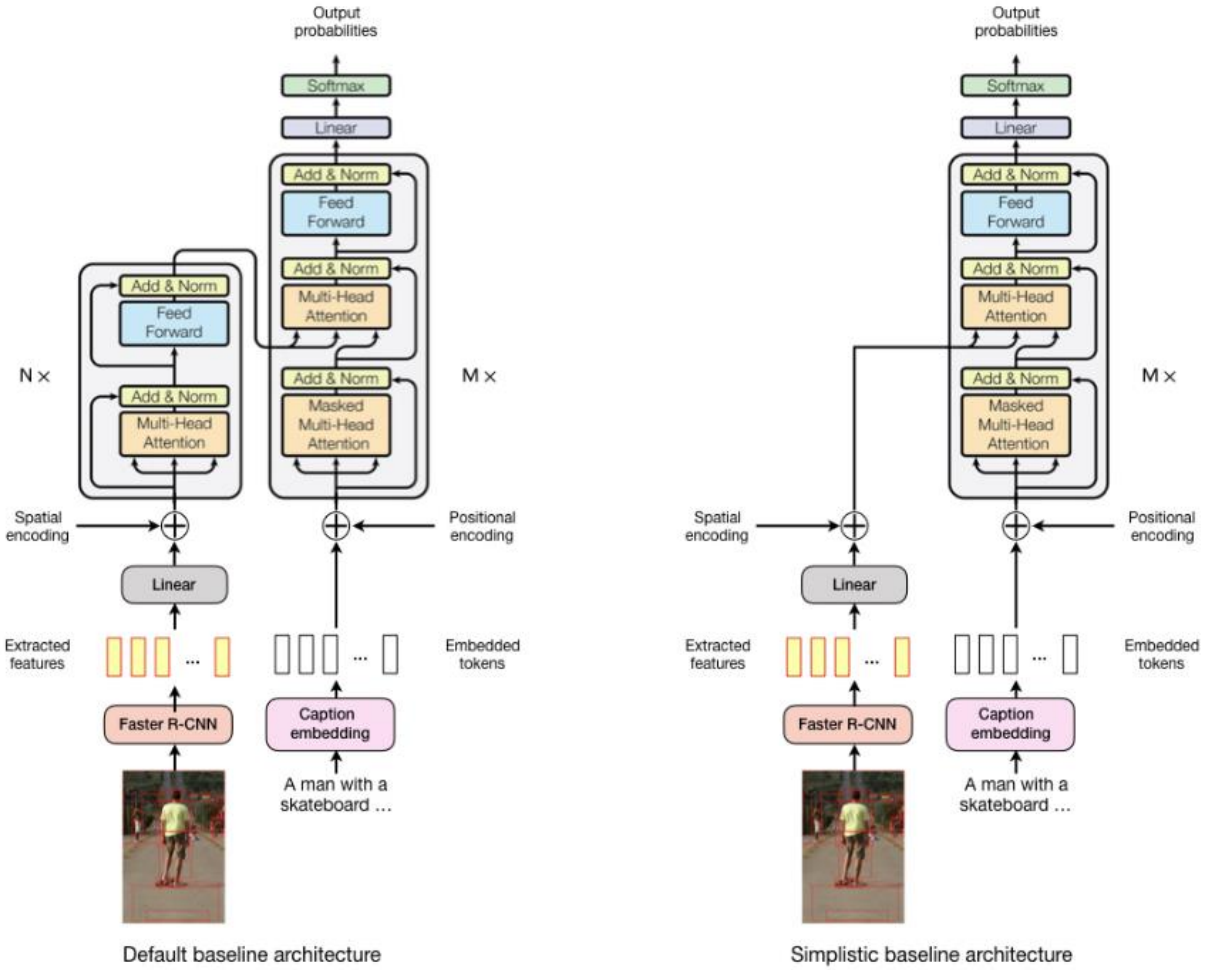


Рисунок 1.3 – Приклад схематичних зображень архітектур, що використовують Трансформер моделі.

1.4 Висновки

В даній роботі я зупинився на використанні претренованої згорткової мережі в якості енкодеру та рекурентної мережі в якості умовної мовної моделі. Даний вибір буде пояснений далі.

Розділ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ.

2.1. Згорткова мережа

Згорткові (конволюційні) нейронні мережі дуже схожі на всім відомі звичайні повнозв’язні нейронні мережі: вони складаються з нейронів, які мають ваги, що навчаються. Кожен нейрон отримує деякі входи, після чого виконується лінійне перетворення і потім слідує нелінійна функція активації. Вся мережа все ще виражає єдину диференційовану функцію оцінок: від пікселів зображення на одному кінці (вході) до оцінок класу на іншому.

Проте архітектури згорткових мереж припускають, що вхід в мережу –це зображення, що дозволяє задати певні особливості в архітектурі. Ці особливості роблять її більш ефективною та значно скорочують кількість параметрів у мережі.

Повнозв’язні нейронні мережі отримують на вхід один вектор, який підлягає перетворенню проходячи через низку прихованих шарів. Кожен прихований шар складається з набору нейронів, де кожен нейрон повністю зв’язаний зі всіма нейронами в попередньому шару, і де нейрони в одному й тому ж шару функціонують незалежно один від одного та не мають між собою зв’язків

(Рисунок 2.1). Якщо розв’язується задача класифікації, то останній шар має кількість нейронів відповідну кількості класів.

Але такі мережі погано маштабуються для обробки зображень. Наприклад, в відкритому датасеті CIFAR-10, зображення мають розмір 32x32x3 (32 в ширину, 32 в висоту, 3 кольорових канали), тому перший повнозв’язний шар буде мати 32 * 32 * 3 = 3072 параметра. При збільшенні зображення до 200x200x3 – кількість параметрів вже стає 200 * 200 * 3 = 120000. Ця повнозв’язність є надлишковою та може легко призвести до перенавчання.

Використовуючи той факт, що на вхід поступає зображення, з’являється можливість розумно обмежити архітектуру. На відміну від шарів парцептрона, згорткові шари мають три виміри: ширина, висота та глибина (кількість кольорових каналів). При розв’язанні задачі класифікації останній шар або декілька все ж лишаються повнозв’язними аналогічно вищезгаданій архітектурі.

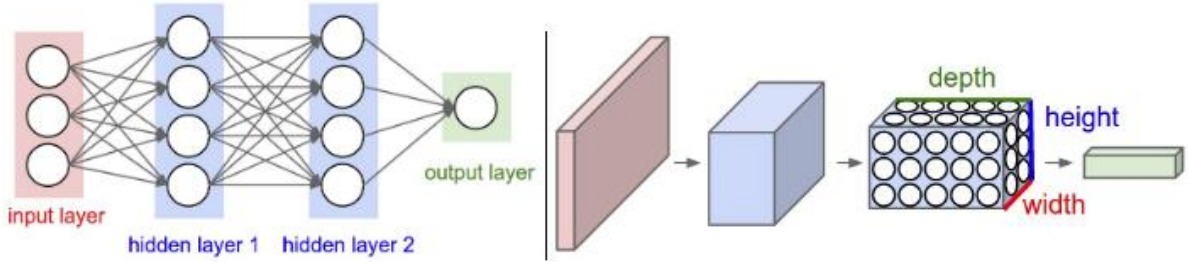


Рисунок 2.1 – Схематичне порівнняння архітектури повнозв’язної та згорткової нейронних мереж.

Проста архітектура згорткової мережі – це послідовність таких шарів як: безпосередньо згортковий шар, пулінг (pooling) шар та звичайний повнозв’язний.

2.1.1. Згортковий шар

Згортковий шар є основним у архітектурі, який виконує більшу частину обчислювальної роботи. Опишемо його особливості.

1) Локальне з’єднання. Як було вказано вище, якщо входом до нейронної мережі слугує зображення, то є недоцільним робити повне зв’язання нейронів з нейронами на попередньому шару. Натомість нейрон на наступному шару має зв’язки тільки з локальною областю входу. Просторова розмірність цієї зв’язності є гіперпараметром – розміром фільтру. Розмірність зв’язків по осі глибини завжди є рівною глибині входу. Важливо також відмітити асиметрію по відношенню до просторових вимірів (ширина та висота) і розмірністю глибини: зв’язки є локальними в просторі висоти та ширини, але є повнов’язними в просторі глибини входу (Рисунок 2.2).

Наприклад, нехай вхід має таку ж розмірність як зображення датасету CIFAR-10. Припустимо, що розмірність фільтру є 5x5, тоді кожен нейрон в згортковому шару буде мати ваги розмірністю 5x5x3 по відношенню до входу, тобто загальна кількість зв’язків – 5 * 5 * 3 = 75 (+1 для параметру зміщення).

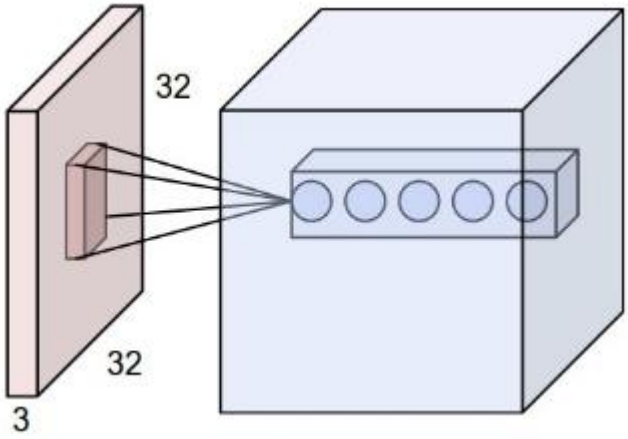


Рисунок 2.2 – Приклад локальної зв’язності нейронів по відношенню до висоти і ширини та повної зв’язності по відношенню глибини входу в згортковому шару.

2) Вихід згорткового шару. Три гіперпараметри контролюють розмірність виходу: глибина, крок і нульовий відступ.

Глибина – це кількість фільтрів. Кожен фільтр вивчає певні силуети для входу. Часто фільтри перших згорткових шарів таких мереж вивчають прямі лінії, наприклад, вертикальні на вхідних зображеннях.

Крок – крок з яким ми зсуваємо фільтр. Відповідає кількості пікселів на які зміститься фільтр за один раз.

Нульовий відступ потрібен для контролю розмірності виходу. Це просто нульові пікселі, що додаються навколо вхідного зображення, якщо це перший згортковий шар.

Розмір виходу можна вважати функцією, що залежить від розмірності входу, кроку, розміру фільтра та нульових відступів:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

де n_{in} ширина чи висота входу, p – кількість нульових відступів, k – ширина чи висота фільтру, а S – крок. Наприклад, нехай вхід має висоту та ширину 5, тоді щоб отримати такий же вихід ми можемо зробити одиничний відступ та пройтись по входу фільтром розмірністю 3x3 з кроком 1. З кроком 2 ми б отримали б вже вихід розмірністю 3x3 (Рисунок 2.3):

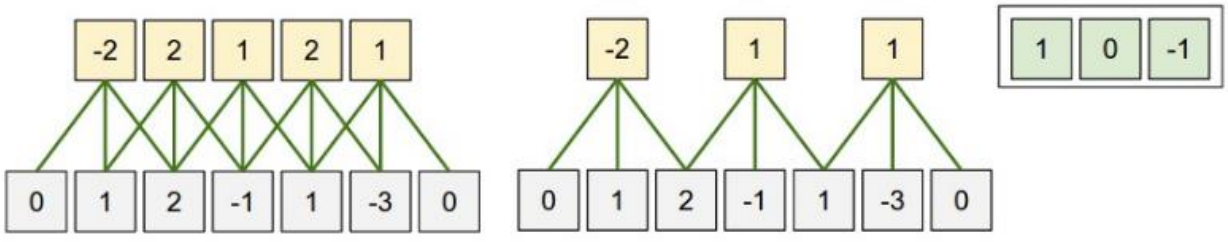


Рисунок 2.3 – Схематичне зображення роботи згорткового шару.

3) . Розділення параметрів. Для зменшення кількості параметрів використовують механізм їх розподілення між собою. З реального прикладу нейронна мережа в роботі Алека Крижевського, Ільї Суцькевера та Геофрея Е. Хінтона ImageNet Classification with Deep Convolutional Neural Networks, їх мережа мала вхід 227x227x3, а перший згортковий шар мав розмірність 55x55x96. Тобто 55 * 55 * 96 = 290 400 нейронів і кожний має 11 * 11 * 3 = 363 параметрів (тобто розмір фільтру був 11)(Рисунок 2.4). Разом виходить 290400 * 364 = 105 705 600 параметрів на першому тільки шарі нейронної мережі. Зрозуміло, що ця цифра є дуже високою.

Виявляється що ми можемо значно скоротити кількість параметрів зробивши певне припущення, що якщо один об'єкт є корисним для обчислення в деякій просторовій позиції (x, y) то він може бути корисним для обчислення в іншій позиції (x2, y2). Тобто позначивши кожен двовимірний зріз (ширина на висоту) як зріз глибини або ще поширеною назвою є фічемапа, ми можемо використовувати одні й ті ж параметри для кожного з цих зрізів. В такому випадку перший шар мережі приведеної прикладом вище буде мати 96 наборів унікальних параметрів (для кожного зрізу глибини) і в загальній складності 96 * 11 * 11 * 3 = 34 848 унікальних параметра (+96 параметрів зміщення). Крім того всі 55*55 нейронів в кожному зрізі глибини будуть використовувати одні й ті ж параметри. При зворотньому розповсюдженні помилки (backpropagation) для кожного нейрона буде обчислений градієнт для своїх параметрів, але ці градієнти будуть сумуватись по кожній фічемапі і буде оновлюватись один набір параметрів на зріз. Через те, що всі нейрони в одному зрізі мають один і той же набір параметрів, то при прямому проході обчислення можуть проводитись як згортка параметрів для входу.

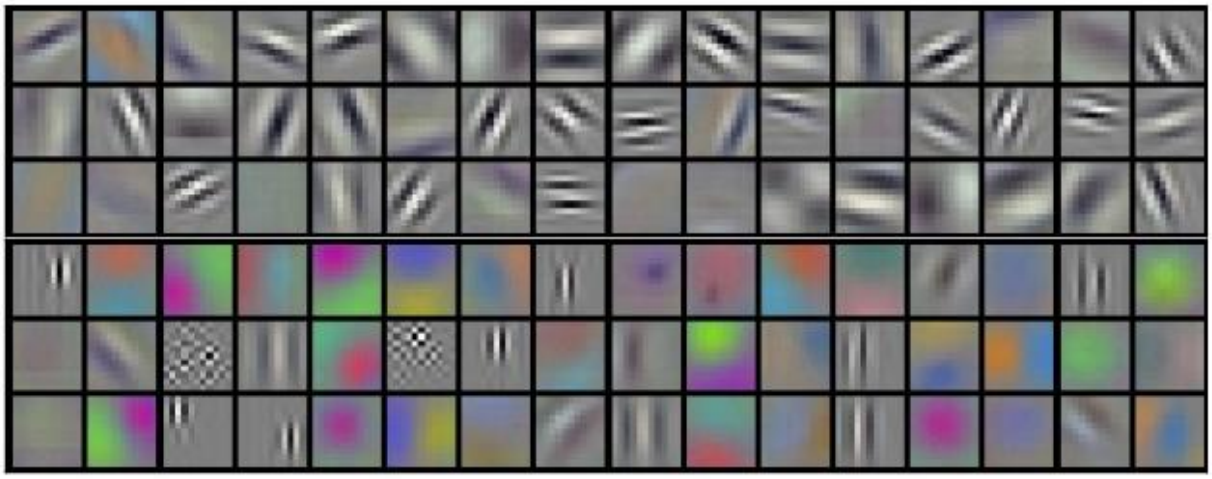


Рисунок 2.4 – Приклад фільтрів, що були взяті з першого шару натренованої нейронної мережі з роботи згаданої вище. Кожний з 96 фільтрів має розмірність 11x11x3 і кожний використовується кожним нейроном для свого зрізу глибини.

Можна звернути увагу на те, що розподілення параметрів є достатньо розумним: якщо виявлення горизонтального краю в деякому місці зображення є корисним, воно може бути корисним і в іншому місці зображення через трансляційнно-інваріантну структуру зображення.

Отже, згортковий шар можна описати наступним чином:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

де $\star$ – операція згортки. Її ж можна розписати наступним чином:

Нехай дана матриця A з розмірністю m на k, та фільтр згортки B розмірністю K на K, тоді в результаті операції згортки A $\star$ B, отримуємо матрицю C:

$$C_{ij} = \sum_{k,l}^{K,K} a_{k+i,l+j} * b_{k,l}\,, \text{ де } i = \overline{0, m-k+1},\ j = \overline{0, n-k+1},$$

де a, b – відповідні елементи матриць A та B. Дана рівність виконується для кроку S = 1.

Проте іноді припущення щодо спільного використання тих же параметрів в деяких випадках все ж може не мати сенсу. Це може трапитись тоді, коли, наприклад, слід очікувати, що на одній стороні зображення повинні бути вивчені абсолютно інші функції, ніж інші. Одним із практичних прикладів є те, що вхідні дані – це обличчя, які були в центрі зображення. Очікується, що різні особливості, пов'язані з очима або волосся, можуть (і повинні) вчитися в різних просторових місцях. У цьому випадку прийнято розслабляти схему спільного використання параметрів, а замість цього просто називати шар – лоскально повнозв’язним шаром.

2.1.2. Пулінг (Pooling) шар

Поширеним є використання цього типу шарів між згортковими. Його основна функція полягає в зменшенні розмірностей шарів, що в свою чергу приводить до зменшення кількості параметрів моделі. Також він допомагає боротися з перенавчанням. Пулінг шар працює незалежно на кожному зрізі і зменшує їх розмірності використовуючи такі операції як max, рідше mean або min (Рисунок 2.5). Дуже поширеним є використання такого шару з розміром 2x2 (ширина, висота) та кроком 2. Такий шар дозволяє викинути 75% активацій, при цьому глибина залишається незмінною. Інтуїтивно саме max pooling шар має вибирати тільки найважливішу інформацію, отриману з згорткового шару.

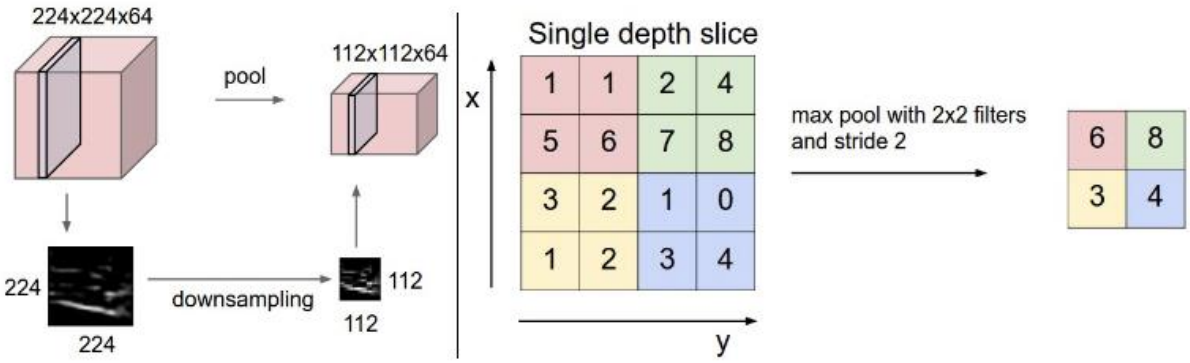


Рисунок 2.5 – Схематичне зображення роботи пулінг (pooling) шару.

Проте зараз є тенденція щодо відмови використання таких типів шарів. Натомість для зменшення розмірності пропонують просто використовувати більше значення гіперпараметру кроку S. Відмова від об'єднання шарів також виявилася важливою при навчанні хороших генеративних моделей, таких як варіативні автокодери (VAE) або генеративні змагальні мережі (GAN).

2.1.3. Поєднання згорткового шару та повнозв’язного

Варто зазначити, що єдиною відмінністю повнозв’язних шарів та згорткових є те, що нейрони в згортковому шарі з'єднані лише з локальною областю на вході, що дозволяє скоротити кількість параметрів. Однак нейрони в обох шарах все ще обчислюють скалярний добуток, тому їх функціональна форма залишається такою ж. Тому виходить, що можливе перетворення між цими шарами.

Для будь-якого згорткового шару існує такий повнозв’язний, який реалізує ту саму функцію при проходженні вперед. Матриця параметрів була б великою матрицею, яка здебільшого дорівнює нулю, за винятком певних блоків (через локальну сполучуваність), де ваги у багатьох блоках рівні (за рахунок спільного використання параметрів).

І навпаки, будь-який повнозв’язний шар може бути перетворений у згортковий. Наприклад, шар з кількістю нейронів K = 4096, який приймає вхідний об'єм розміром 7 × 7 × 512, може бути еквівалентно виражений у вигляді згорткового з розміром фільтру (ширина, висота) F = 7, нульовим відступом P = 0, кроком S = 1 та кількістю фільтрів K = 4096. Іншими словами, ми встановлюємо розмір фільтра точно таким чином, як розмір вхідного об'єму, а значить, вихід буде просто 1 × 1 × 4096.

Тож це дає змогу поєднати ряд згорткових та пулінг (pooling) шарів з повнозв’язним для розв’язку, наприклад, задачі класифікації. Тобто кожен нейрон останнього повнозв’язного шару буде відповідати результату входу в мережу, щодо класів.

Найпоширеніша форма архітектури для класифікації складається з декількох згорткових шарів з активацією RELU, за якими слідує пулінг (pooling) шар, ця комбінація повторюється декілька разів поки зображення не буде невеликого розміру проте матиме значно більшу глибину ніж на вході. Потім слідує повнозв’язний шар.

2.1.4. Архітектура ResNet

Глибокі згорткові мережі досягли значних успіхів в класифікації зображень. Варто відзначити, що згорткові шарі здатні вивчати низькорівневі, середньорівневі та високорівневі ознаки. Як було показано Матью Д. Зейлером та Роб Фергусом, збільшення глибини нейронної мережі дає змогу вивчати більш високорівневі ознаки (Рисунок 2.6).

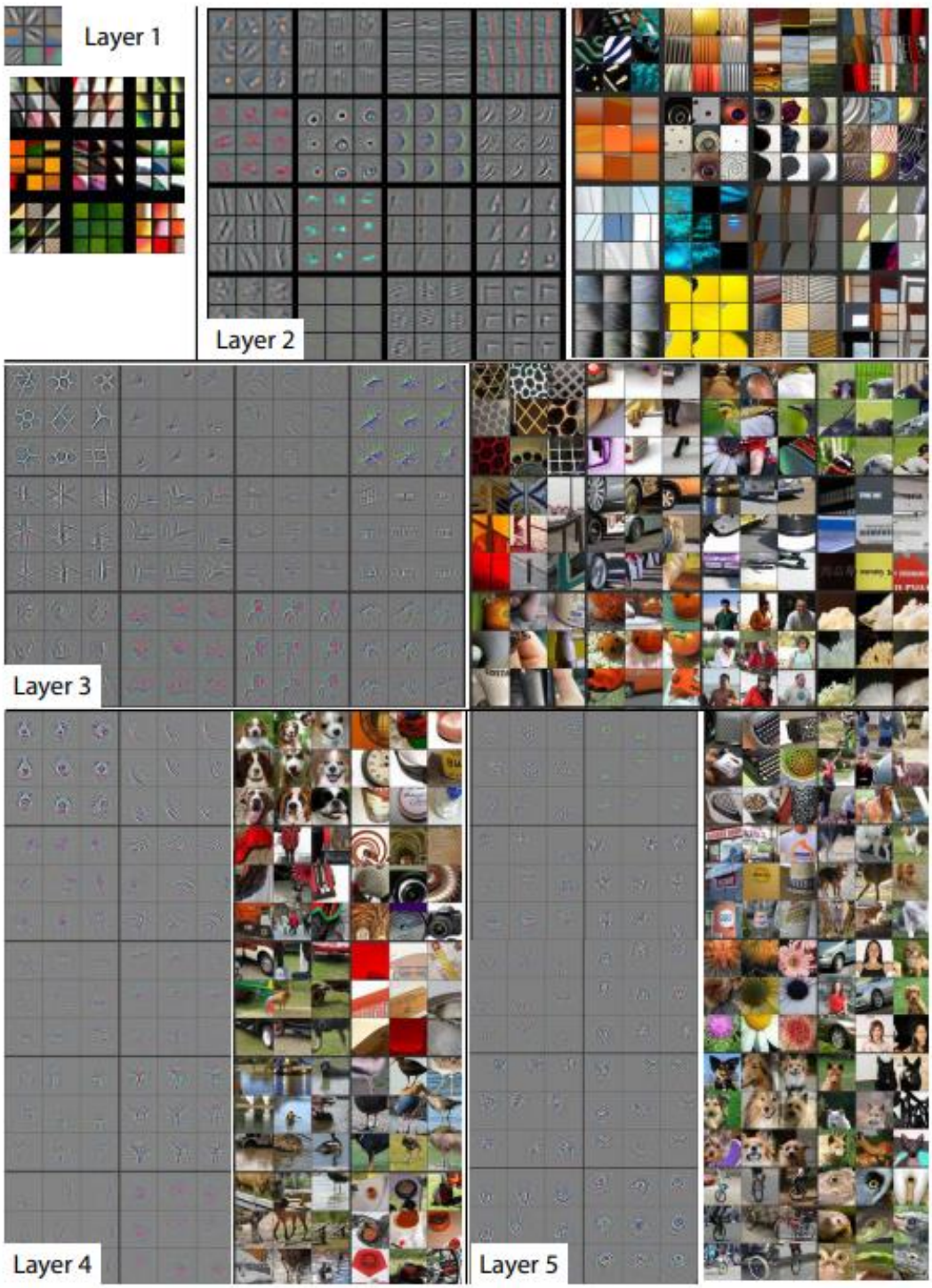


Рисунок 2.6 – Візуалізація ознак, що вивчає нейронна мережа на різних шарах.

Як можна побачити з Рисунка 2.6, коли на перших шарах мережа вивчає прості лінії, на фічемапі четвертого, вже можна побачити силуети морди собаки. Це дає інтуїтивне розуміння про те, що збільшення глибини може дати приріст у

точності класифікації. Проте як виявилося, просте збільшення глибини не дає очікуваний результат (Рисунок 2.7).

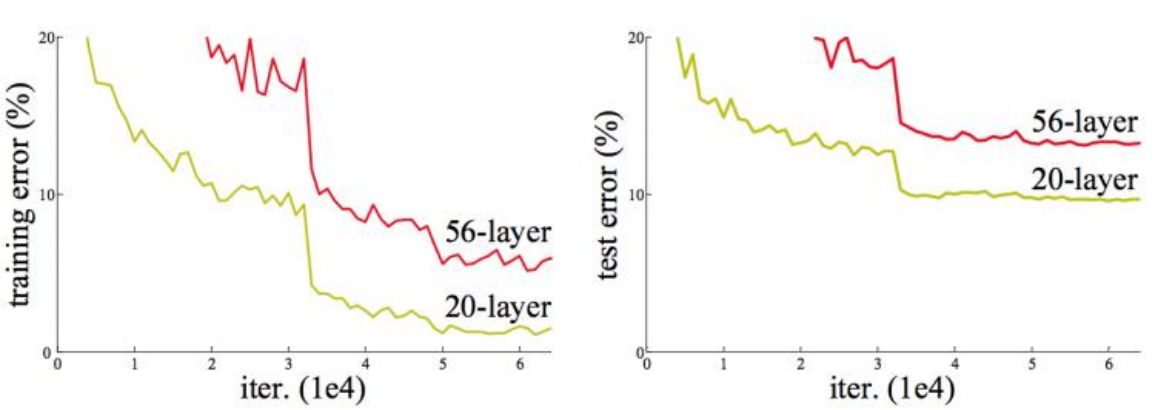


Рисунок 2.7 – Помилка навчння на навчальній вибірці (зліва) та тестовій вибірці (зправа) 56 шарової та 20 шарової мережі на датасеті SIFAR-10.

Як можна зрозуміти з графіків – причиною збільшення помилки при використанні глибшої нейронної мережі не є перенавчання. Виявилося, що причиною цього є нестабільність глибшої мережі, яку важче оптимізувати. Тому було запропоновано Кеймінгом Хі, Ксянгуї Чангом, Шаукінг Ренгом та Джуан Саном використання пропускних (residual) зв'язків, що роблять її більш стійкою у навчанні та дають змогу робити її глибшою. Ідея цих зв'язків полягає в наступному: нехай шар мережі можна описати як функцію від її входу: y = F(x), тоді шар з пропускним (residual) зв'язком буде виглядати наступним чином: y = F(x) + x (Рисунок 2.8). Це дає можливість під час навчання мережі зануляти параметри згорткового шару, якщо, наприклад, такий є надлишковим. При цьому вихід даного шару залишиться таким же як і був вхід, натомість без цього зв'язку мережі доводилося б вивчати функцію x = F(x), що є значно важчим, оскільки згорткові шари мають нелінійні функції активації. Крім того, в їх роботі було

запропоновано використовувати такі зв’язки до блоків згорткових шарів, а не до окремих.

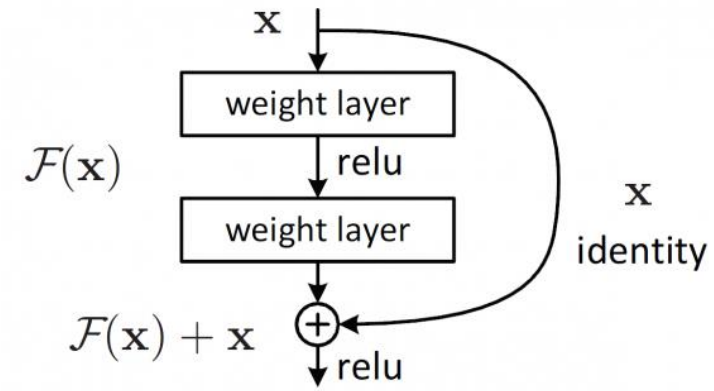


Рисунок 2.8 – Схематичне зображення пропускного (residual) зв’язку.

Використання даного підходу вирішило проблему глибших нейронних згорткових мереж та сприяло отриманню кращих результатів класифікації (Рисунок 2.9).

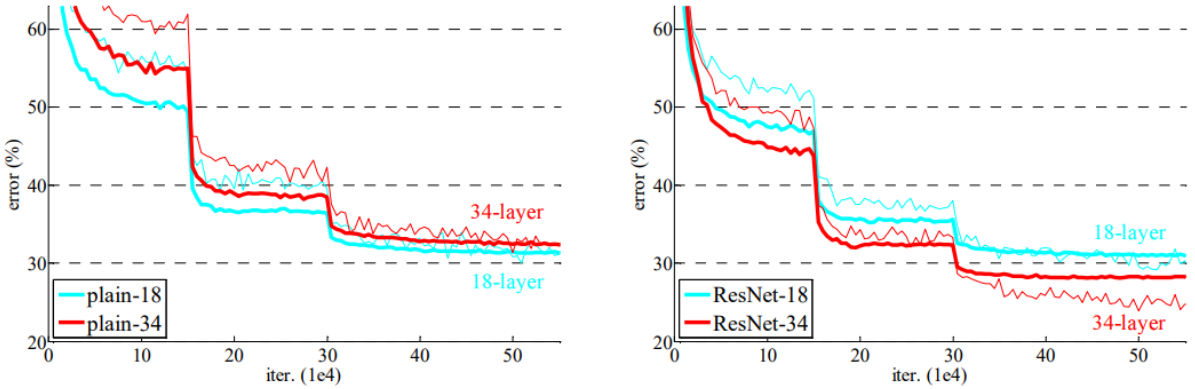


Рисунок 2.9 – Порівняння 18 та 34 шарових згорткових мереж на датасеті ImageNet з використанням residual зв’язків (зправа) та без їх використання (зліва). Як можна побачити, коли збільшення кількості шарів вдвічі призвело до збільшення помилки у першому випадку, в другому ж навпаки – збільшивши кількість шарів вдвічі дало зменшення помилки.

З даною ідеєю було створені архітектури ResNet 18, ResNet 34.

Також, в даній роботі було запропоновано використання блоків з «вузькими місцями». Ідея таких блоків полягала в тому, щоб зменшити кількість парметрів блоці та збільшити їх кількість. Приклад такого блоку зображений на Рисунку 2.10.

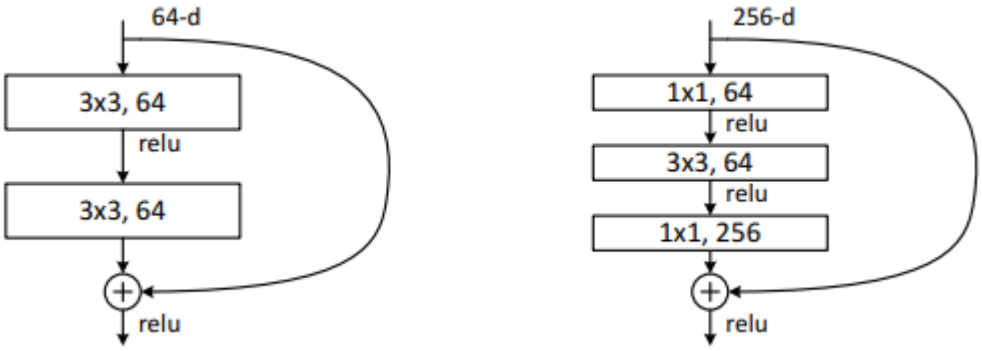


Рисунок 2.10 – Порівняння звичайного блоку з residual зв’язками та блоку з «вузьким місцем» та residual зв’язками.

Тож збільшивши кількість блоків було створено архітектури ResNet 50, ResNet 101 та ResNet 152 (Таблиця 2.1) і було досягнено ще кращих результатів класифікації.

Таблиця 2.1 – Порівняння архітектур, які було створені в вищезгаданій роботі Deep Residual Learning for Image Recognition.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

2.2. Рекурентні мережі

Результатом роботи нашої мережі має бути згенерований текст на основі вхідного зображення. Так як Image Captioning це задача навчання з учителем, це означає, що ми маємо вибірку зображень і кожне зображення має відповідні описи. Тож ми маємо визначитись з цілевою функцією нашої моделі, яку вона буде апроксимізувати під час навчання.

2.2.1. Мовна модель

Статистична мовна модель – це ймовірнісний розподіл над послідовністю слів. Зазвичай мовні моделі працюють в правому напрямку, тобто їх цільовою функцією є максимізація правдоподібності наступного слова (u_i) за умови попередніх:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

де k – контекстне вікно (кількість слів, що впливають на вибір наступного), а θ – параметри мовної моделі. В нашому ж випадку вибір наступного слова буде залежати не тільки від попередніх, а ще від візуальної інформації, що ми отримуватимемо з картинки. Як було вказано в попередньому розділі, глибокі згорткові мережі здатні вивчати високорівневі ознаки зображень, тому ми будемо використовувати останні шари згорткової мережі для отримання векторного представлення входу. Також, оскільки підписи в навчальній вибірці не є дуже довгими, ми можемо не використовувати фіксоване контекстне вікна, натомість враховувати всі попередньо згенеровані слова. Отже нам потрібно дещо модифікувати дану функцію:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_c \, , \, u_{image}; \Theta)$$

де u_c – попередньо згенеровані слова, а u_{image} – векторне представлення зображення.

Як можна побачити з цілевої функції, результат генерується авторегресивно і вхід в мовну модель змінюється впродовж генерації тексту. Це робить неможливим використання звичайних повнозв’язних мереж, оскільки вони мають приймати на вхід вектор фіксованої розмірності. Натомість існують рекурентні мережі, що вдало підходять для розв’язку даної задачі (Рисунок 2.11).

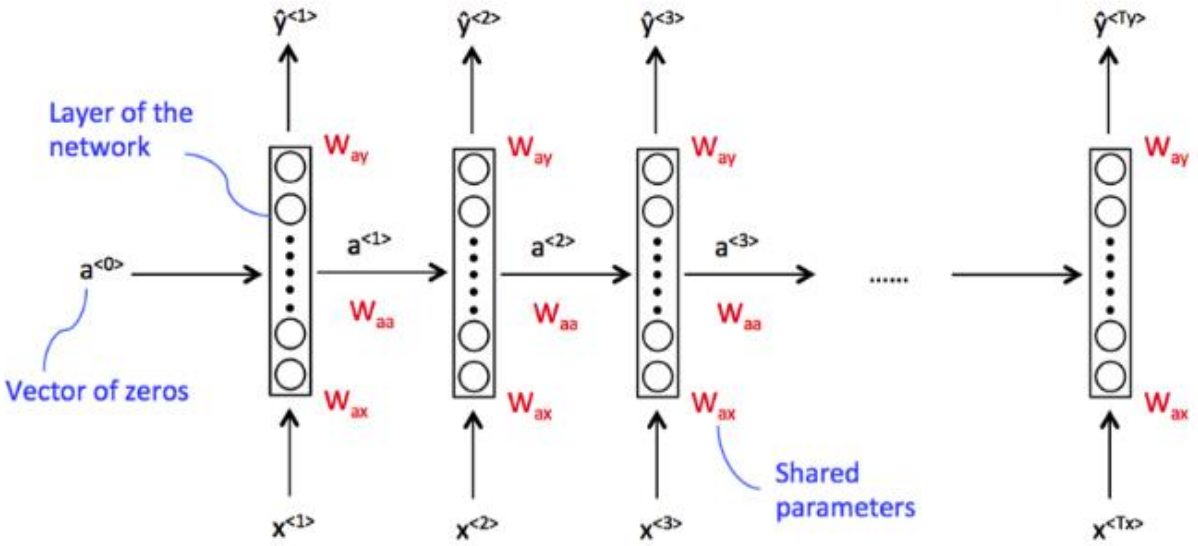


Рисунок 2.11 – Розгорнутий вигляд рекурентної мережі.

Позначимо через W_{aa} – матрицю парметрів, що буде слугувати для переходу стану мережі з одного часового кроку в наступний, W_{ax} – матрицю параметрів для входу в мережу, W_{ay} – матриця параметрів виходу, $x^{<t>}$– вхід в мережу на t - тому кроці, $a^{<t-1>}$ – стан отриманий з попереднього кроку та $\hat{y}^{<t>}$ – вихід мережі на t - тому кроці. Тоді стан мережі на t – му кроці може бути обчислений як:

$$a^{<t>} = f_a(w_{ax}x^{<t>} + w_{aa}a^{<t-1>} + b_a)$$

А вихід відповідно:

$$\hat{y}^{<t>} = f_y\left(w_{ay}a^{<t>} + b_y\right)$$

На практиці зазвичай для обчислення $a^{<t>}$ матриці W_{aa} та W_{ax}, а також вектора $a^{<t-1>}$ і $x^{<t>}$ конкатенуються, як зображено на Рисунку 2.12:

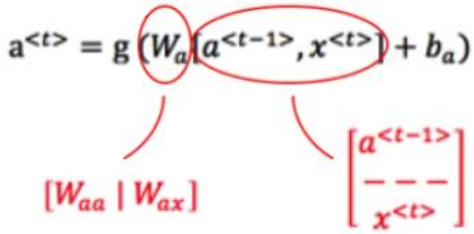


Рисунок 2.12.

Отже, маючи таку архітектуру, ми будемо використовувати техніку teacher forcing під час навчання. Нехай $\hat{y}^{<1>}, \hat{y}^{<2>}, \ldots \hat{y}^{<Ty>}$ відповідний опис зображення з навчальної вибірки, тоді першим входом в рекурентну мережу буде векторне представлення зображення отримане з згорткової мережі, позначимо його $x^{<1>}$. Тоді наступним входом в мережу ми подамо перше слово цільової послідовності, тобто $x^{<2>} = \hat{y}^{<1>}$. Останнім же токеном цільової послідовності буде спеціальний токен, що буде символізувати закінчення. Таким чином ми будемо вчити нашу мережу саму зупиняти генерацію, отримавши на виході з певного часового кроку цей токен. Позначимо його як $\hat{y}^{<Ty>}$. Початковий вектор стану $a^{<0>}$ зазвичай ініціалізується нулями (Рисунок 2.13).

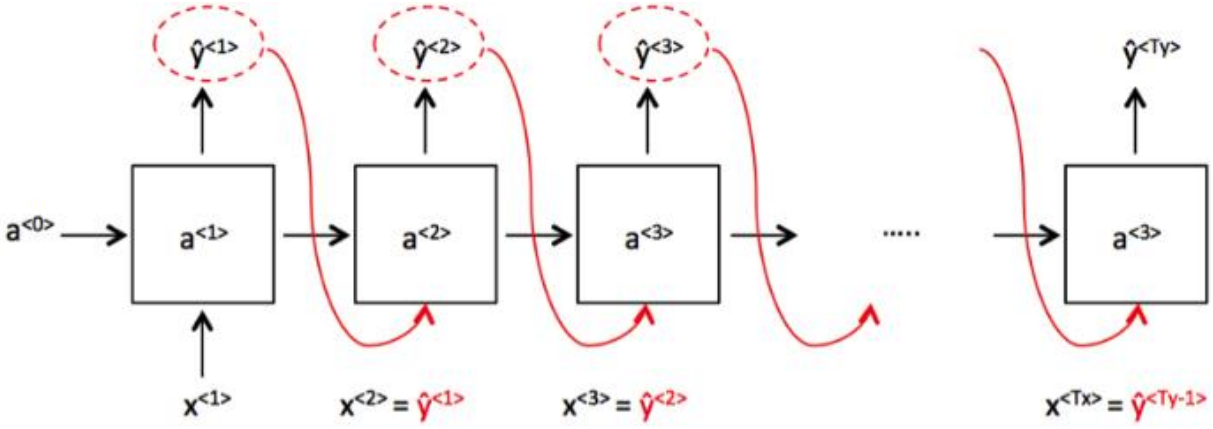


Рисунок 2.13 – Візуалізація техніки навчання teacher forcing в рекурентній нейронній мережі, зображеній в розгорнутому вигляді.

2.2.2 Метод зворотного поширення помилки в рекурентних мережах

Також варто відзначити, що при проходженні послідовності матриці параметрів рекурентної нейронної мережі W_{aa}, W_{ax}, W_{ay} лишаються одними й тими ж (Рисунок 2.14). Це означає, що ми маємо спільні параметри для всієї послідовності, що інтуїтивно має сенс і чимось нагадує спільне використання параметрів у згорткових мереж, тільки в даному випадку замість того, щоб виявляти подібні ознаки на різних частинах зображення, ми маємо змогу вивчати певні закономірності, які можуть трапитись в тексті в різних місцях.

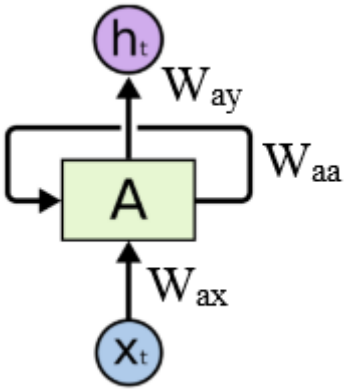


Рисунок 2.14 – Згорнутий вигляд рекурентної нейронної мережі.

Проте цей факт призводить до того, що мережа стає циклічною і є не очевидним як використовувати звичайний метод зворотного поширення помилки (backpropagation) для навчання такої. Тому існує дещо модифікована версія цього методу, яка називається метод зворотного поширення помилки в часі (backpropagation through the time).

Перш за все, маючи певну функцію втрат (лосс функцію) $L_t(y^{<t>}, \hat{y}^{<t>})$, де $y^{<t>} -$ правильне слово, а $\hat{y}^{<t>} -$ передбачене моделлю, ми рахуємо її на

кожному часовому кроці, а потім сумуємо отримуючи загальне значення лосс функціїї для послідовності: $L = \sum_i L_i(y^{<i>}, \hat{y}^{<i>})$ (Рисунок 2.15)

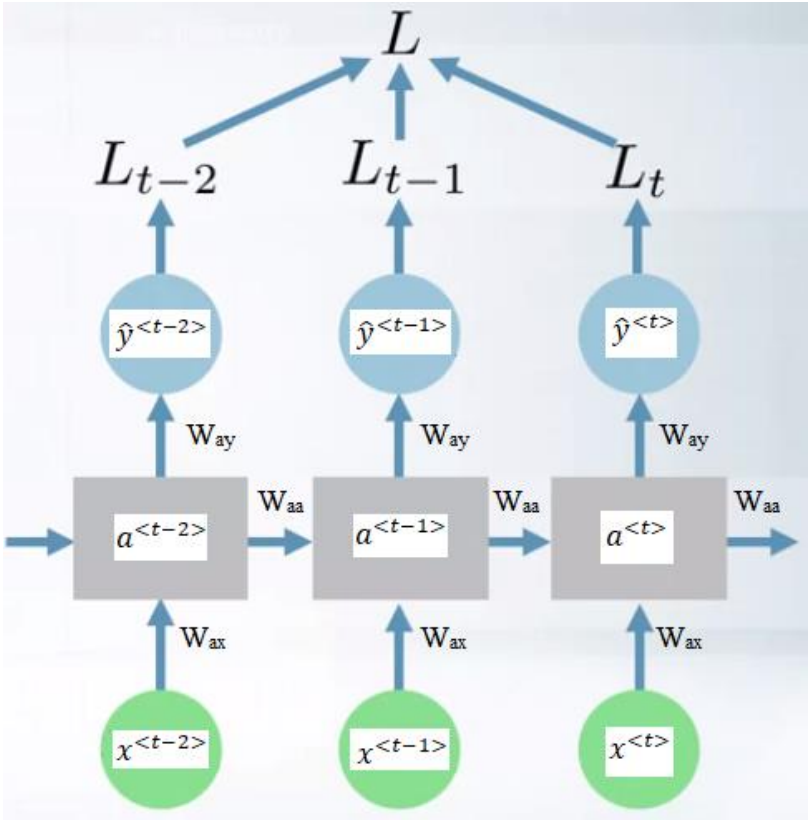


Рисунок 2.15 – Схематичне зображення підрахунку лосс функції всієї послідовності.

Градієнт по відношенню до матриці W_{ay} на часовому кроці t буде обчислюватись наступним чином:

$$\frac{\partial L_t}{\partial w_{ay}} = \frac{\partial L_t}{\partial \hat{y}^{<t>}} \frac{\partial \hat{y}^{<t>}}{\partial w_{ay}}$$

Проте у випадку з матрицею W_{aa} ми вже не зможемо скористатись правилом ланцюга, оскільки на відміну від $\hat{y}^{<t>}$, який залежить тільки від поточного стану мережі $a^{<t>}$:

$$\hat{y}^{<t>} = f_y\left(w_{ay}a^{<t>} + b_y\right)$$

де $b_y -$ вектор зміщення, $a^{<t>}$ вже залежить від попереднього стану $a^{<t-1>}$:

$$a^{<t>} = f_a(w_{ax}x^{<t>} + w_{aa}a^{<t-1>} + b_a)$$

Тому градієнт по відношенню до матриці W_{aa} на кроці t буде розраховуватись наступним чином:

$$\frac{\partial L_t}{\partial w_{aa}} = \frac{\partial L_t}{\partial \hat{y}^{<t>}} \cdot \frac{\partial \hat{y}^{<t>}}{\partial a^{<t>}} \sum_{k=0}^{t} \frac{\partial a^{<t>}}{\partial a^{<t-1>}} \cdots \frac{\partial a^{<k+1>}}{\partial a^{<k>}} \frac{\partial a^{<k>}}{\partial w_{aa}} =$$

$$= \frac{\partial L_t}{\partial \hat{y}^{<t>}} \cdot \frac{\partial \hat{y}^{<t>}}{\partial a^{<t>}} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{<i>}}{\partial a^{<i-1>}} \right) \frac{\partial a^{<k>}}{\partial w_{aa}}$$

Через таку особливість, використовуючи нелінійні функції активації такі як гіперболічний тангенс, сигмоїда, при обчисленні метод зворотного поширення помилки (backpropagation), чим далі прихований стан знаходиться від останнього, тим його градієнти будуть меншими в силу перемноження матриць, значення яких близьке до нуля. Цей ефект називається затуханням градієнту. Тому оновлюватись такі стани будуть значно менше. Це означає, що, наприклад, контекст, який йшов раніше, має менший вплив за попередньо згенерований токен. Проте в людській мові часто бувають складні речення, які не підпорядковуються такій логіці. Також існує протилежна проблема - вибуху градієнтів. Через це було запропоновано удосконалити дану архітектуру.

2.2.3 Архітектура LSTM

Для вирішення вищезгаданих проблем, було запропоновано використовувати додатковий «канал пам’яті» (Рисунок 2.16). На відміну від звичайної рекурентної мережі, у якій вихід прихованого стану обчислювався як:

$$a^{<t>} = f_a(w_{ax}x^{<t>} + w_{aa}a^{<t-1>} + b_a)$$

в дану архітектуру також додаються нові матриці параметрів, що слугуватимуть для контролю «кількості інформації, що ми запам’ятовуємо і забуваємо відповідно»:

$$i^{<t>} = \sigma(w_{ix}x^{<t>} + w_{ia}a^{<t-1>} + b_i)$$

$$o^{<t>} = \sigma(w_{ox}x^{<t>} + w_{oa}a^{<t-1>} + b_o)$$

$$f^{<t>} = \sigma\left(w_{fx}x^{<t>} + w_{fa}a^{<t-1>} + b_f\right)$$

де $\sigma -$ функція сігмоїди. Причиною вибору саме такої є те, що вона переводить значення з $-\infty$ до $+\infty$ в значення від 0 до 1, де, наприклад 1 означає повністю відкритий канал, а 0 – повністю закритий.

Таким чином ми отримати вхідний (input), вихідний (output) та забувальний гейти (forget gates). Інтуїтивно, input gate контролює кількість інформації, що ми беремо від входу на поточному часовому кроці, output gate, контролю те, скільки ми візьмемо з каналу пам’яті для того, щоб передати в наступний прихований стан, а forget gate контролює скільки потрібно забути з того, що було в каналі пам’яті.

Виходом такого прихованого стану буде:

$$h^{<t>} = o^{<t>} \cdot f_a(c^{<t>})$$

$$c^{<t>} = f^{<t>} \cdot c^{<t-1>} + i^{<t>} \cdot a^{<t>}$$

де $c^{<t>} -$ вихід каналу пам'яті з даного часового кроку.

Проте все ж якобіан $\frac{\partial c^{<t>}}{\partial c^{<t-1>}} = diag(f^{<t>})$, а $f^{<t>}$ приймає значення від 0 до 1, тому проблема затухання градієнтів лишається актуальною. Вирішується вона ініціалізацією вектору зміщення b_f високим додатнім значенням, вищим за 1. Тоді на перших ітераціях значення $f^{<t>}$ є не близьким до нуля і дана архітектура «не забуває» і може знайти довгі залежності в даних.

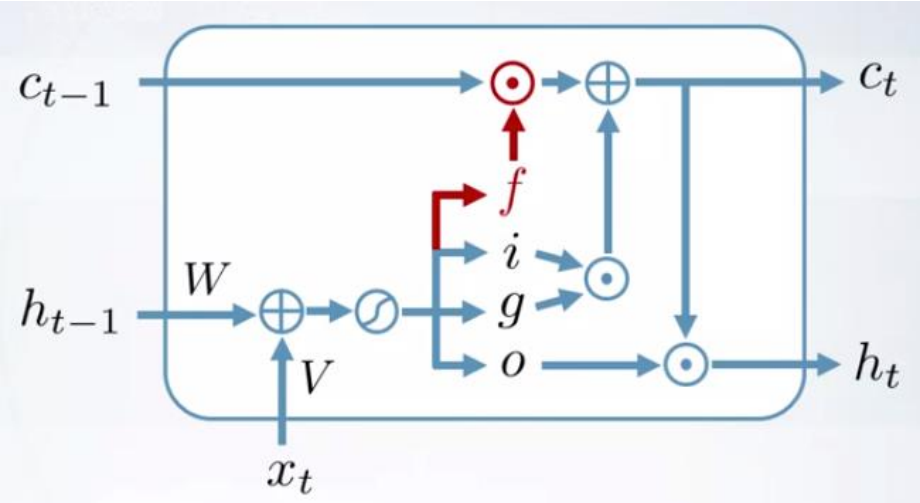


Рисунок 2.16 – Схематичне зображення архітектури рекурентної нейронної мережі LSTM.

Також варто відзначити, що рекурентні нейронні мережі можуть мати декілька шарів. В такому випадку одна мережа приймає на вхід виходи попередньої, як показана на Рисунку 2.17.

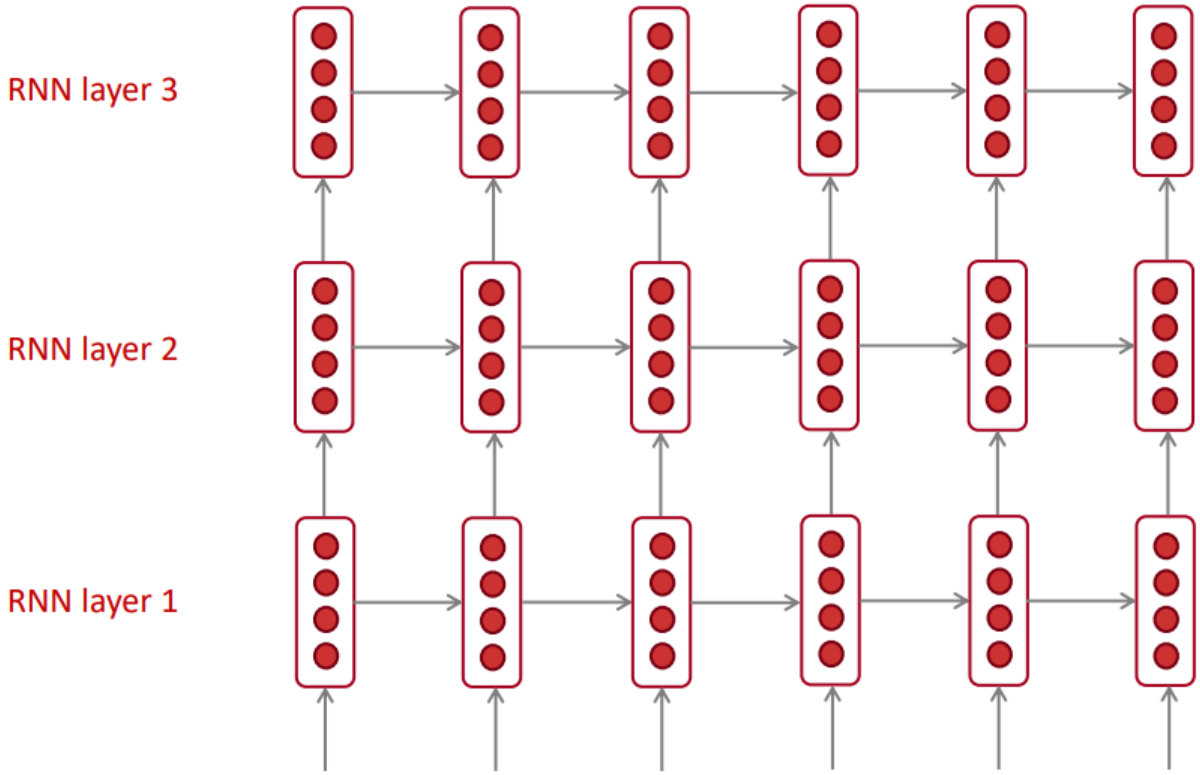


Рисунок 2.17 – Схематичне зображення багатошарової рекурентної мережі.

2.3 Трансформери

2.3.1. Актуальність архітектури

У 2017 році Ашиш Весвані, Ноам Шазір, Нікі Пармар, Якоб Юзкеревич, Ліон Джонс, Адіан Ню Гомез, Лукаш Кайзер та українець Ілля Полосухін опублікували роботу, що змінила світ обробки людскої мови. Вони запропонували архітектуру мережі, що не використовує рекуренність під час тренування, що дає змогу значно збільшити модель, оскільки з’являється можливість розпаралелити обчислювання, яка скорочує час цього процесу. Їх архітектура складається з енкореру та декодеру, проте після появи їх архітектури

з’явилася низка модифікацый, що використовують тілки декодер або тільки енкодер.

2.3.2. GPT

Однією з таких є модель GPT, яка базується на декодері Трансформера. Ця модель використовує мульти хед селф етеншин (multi-head self-attention) операції над вхідними токенами, за чим слідує повнозв’язний шар, який видає розподілення над всіма токенами в словнику:

$$h_0 = UW_e + W_p$$
$$h_l = \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n]$$
$$P(u) = \texttt{softmax}(h_n W_e^T)$$

де $U = (u_{-k}, \ldots, u_{-1})$ вектор контекстних токенів, n – кількість слоїв, W_e – матриця токен ембеддінгу, W_p – матриця позиційного ембеддінгу.

2.3.2.1 Ембеддінг

Перш за все, перед тим як текст потрапить до нейронної мережі потрібно представити його в вигляду набору векторів, де кожен вектор буде відповідати кожному токену. Попередньо текст токенізується використовуючи Byte Pair Encoding що зазвичай розбиває одне слово на частинки. Для співставлення токен – вектор використовується матриця ембеддінгу (Рисунок 2.18).

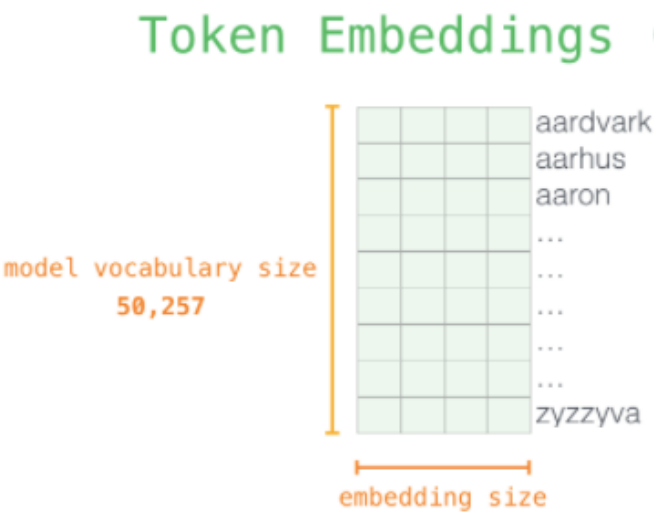


Рисунок 2.18 – Візуалізація ембеддінг матриці.

де embedding size – розмірність вектору, model vocabulary size – кількісь токенів в словнику. Проте цього ще недостатньо, оскільки архітектура Трансформер не є рекурентною, текст потрапляє до нейронної мережі одночасно, тому використовується позиційне кодування, що дає змогу моделі «зрозуміти» порядок слів (Рисунок 2.19).

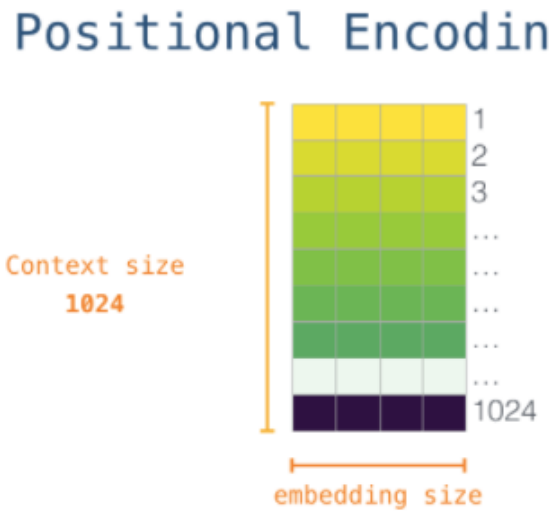


Рисунок 2.19 – Візуалізація позиційного ембеддінгу.

де context size – максимальна кількість токенів, що потрапляє в мережу. Далі значення вектору з матриці токен ембеддінгу сумується зі значенням позиційної ембеддінг матриці поелементно та потрапляє безпосередньо в декодер (Рисунок 2.20).

Рисунок 2.20 – Схематичне зображення потрапляння токенів до декодер-блоку.

2.3.2.2 Маскед Селф Етеншн

Наступним кроком вектора потрапляють до декодеру, що складається зі спеціальних декодер-блоків. Кожен декодер блок складається з masked self-attention блоку, та feed-forward блоку. Self-attention блок є одним з найважливіших в даній архітектурі. Його головна ціль – вивчати відносні взаємозв’язки між токенами в реченні. Цей блок складається з декількох голів. В кажній голові токен представляється в вигляді трьох векторів: ключ (key), запит (query) та значення (value). Ці вектора є результатом матричного множення токену на відповідні матриці, ваги якої оновлюються під час методу зворотного поширення помилки (backpropagation). Наступним кроком query вектор множиться на кожний транспонований key вектор, отримані скаляри (attention scores) діляться на квадратний корінь з розмірності key, query, value векторів та до них застосовується функція softmax. Після цього, відповідне значення в отриманому векторі множиться на value вектор цього токена. Потім отримані ветора сумуються:

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

Дані операції повторюються для кожного query вектору. Оскільки для того, щоб відправити результати цих обчислень в повнозв'язний шар нам потрібно отримати тільки один ветор для кожного слова, вектори кожної голови конкатенуються в один (Рисунок 2.21):

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

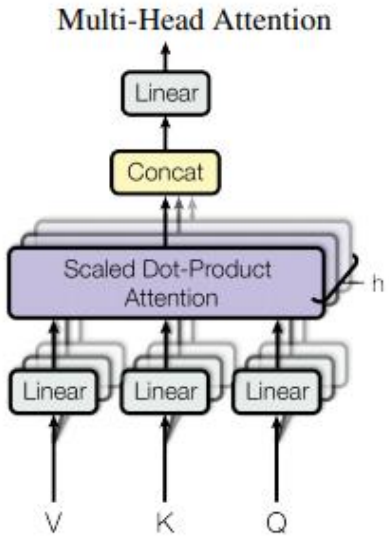


Рисунок 2.21 – Схематичне зображення Multi-Head Attention блоку.

Masked self-attention потрібен для того, щоб модель не могла рахувати attention scores для майбутніх токенів. Тому ключам майбутніх токенів присвоюється значення -∞.

Після self attention блоку слідують два звичайних повнозв'язних шари. Також після кожного self attention блоку та повнозв'язного шару застосовується нармалізація шару (layer normalization) та пропускні зв'язки (residual connection).

2.3.2.3 Нормалізація шару(layer normalization)

Нехай маємо на вхід x, розміром m, де m – розмір підвибірки (міні батчу) під час навчання стохастичним градієнтним спуском. Кожен елемент x має розмірність K, тобто має K елементів. Для кожного елементу x обчислюється середнє та дисперсія:

$$\mu_i = \frac{1}{K}\sum_{k=1}^{K} x_{i,k}$$

$$\sigma_i^2 = \frac{1}{K}\sum_{k=1}^{K}(x_{i,k} - \mu_i)^2$$

Наступним кроком кожен елемент нормалізується (z-нормалізація):

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}$$

Нарешті застосовується маштабування та зсув, де γ і β оновлюються під час методу зворотного поширення помилки (backpropagation):

$$y_i = \gamma\hat{x}_i + \beta \equiv \mathrm{LN}_{\gamma,\beta}(x_i)$$

2.3.2.4 Пропускні (residual) зв’язки в архітектурі трансформер

Оскільки для досягнення хорошої результативності моделі потрібно зробити її глибокою (GPT складається з 12 Трансформер блоків), з’являється таке поняття як затухання та вибухання градієнтів під час зворотного поширення помилки (backpropagation). Для усунення такої проблеми використовуються пропускні зв’язки (residual connections), що були запропоновані в роботі, згаданій вище. В даному випадку, нехай ми маємо на вхід self-attention блоку x, тоді результатом проходження даного входу через блок буде F(x) + x, де F(x) – результат multi-head self-attention операції над x. Те ж саме відбувається при проходженні feed-forward блоку (Рисунок 2.22).

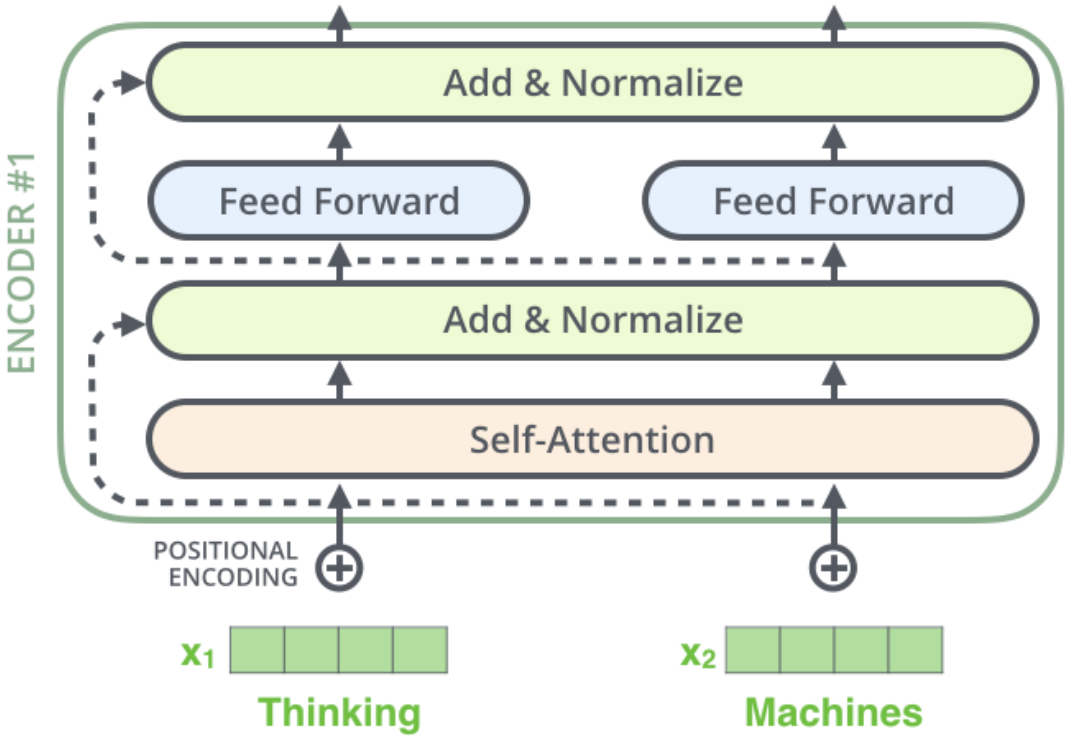


Рисунок 2.22 – Схематичне зображення Residual зв’язків.

В кінці всіх блоків слідує повнозв’язний шар з розмірністю в розмір словника та softmax активацією який дає розподіл над всіма токенами в словнику. Тобто вибирається токен з найбільшим значенням.

2.4. Алгоритми декодування

2.4.1. Вступ

Для взаємодії з безпосередньо натренованою моделлю потрібен алгоритм декодування. Вони використовуються при виборі наступного токену при авторегресивній генерації тексту. Найбільш поширеними алгоритмами є жадібний пошук та beam-пошук.

2.4.2. Жадібний пошук

Оскільки на кожному кроці модель будує розподілення над словником токенів, то ідея жадібного алгоритму заключається в виборі максимально ймовірного токену. Такий же метод використовується зазвичай при тренуванні моделі. Це тренування називається teacher forcing і воно полягає в тому що вибираючи максимально ймовірний токен, ми співставляємо його з реальним словом навчальної вибірки та рахуємо кросс етропійний лосс. Це допомагає швидше сходитись моделі, проте під час генерації тексту такий підхід може привести до різкої зміни теми (виділено червоним) і причиною цього є «ненадійний хвіст» розподілення. Згенерувавши невірний токен, модель ризикує продовжувати генерувати нові слова вже не пов’язані з початковим контекстом (Рисунок 2.23).

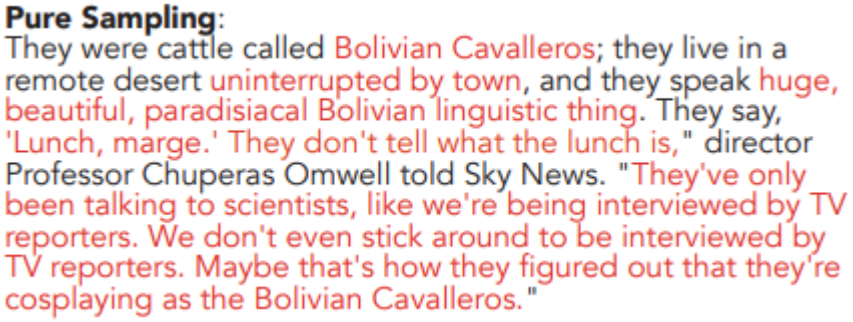


Рисунок 2.23 – Приклад згенерованого тексту з використанням жадібного алгоритму для декодування(червоним виділено нерелевантний контекст).

2.4.2. Алгоритм beam search

За для усунення таких ситуації було запропоновано використовувати beam-search алгоритм (Рисунок 2.24). Його ідея полягає в тому, що на першому кроці вибирається топ n максимально ймовірних токенів, після чого для вибраних токенів знову вибирається топ n найбільш ймовірних - гіпотез. Наступним

кроком для кожного вибраного токену та його топ n гіпотез рахується сума логарифму ймовірностей, та нормується на довжину вже згенерованої послідовності:

$$\frac{1}{t}\sum_{i=1}^{t}\log P_{\mathrm{LM}}(y_i|y_1,\ldots,y_{i-1},x)$$

та вибирається послідовність з найменшою сумою. Така послідовнісь є максилально правдоподібною серед наявних гіпотез.

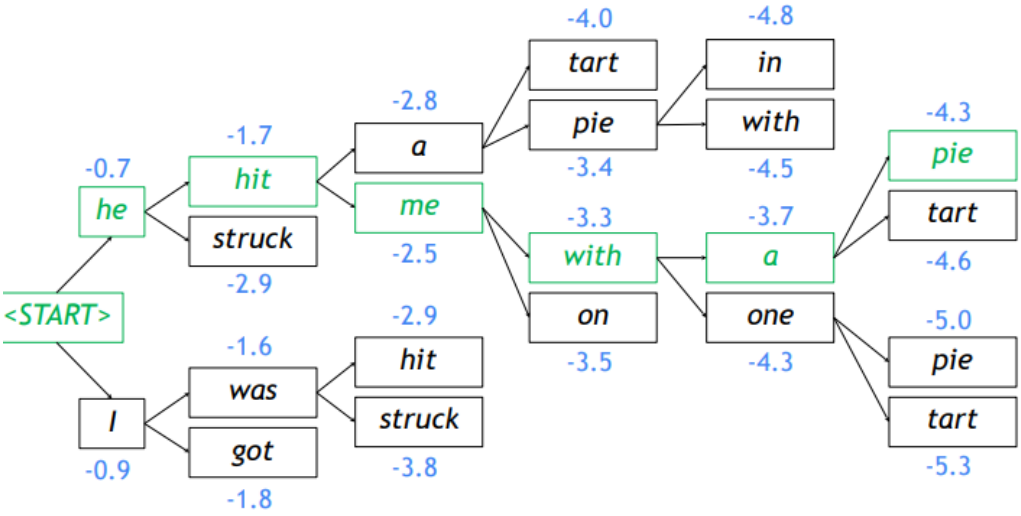


Рисунок 2.24 – Візуалізація роботи beam-search алгоритму з параметром beam size = 2.

Проте було показано, що навіть при нормуванні суми на довжину послідовності, цей алгоритм є дуже чутливим до її довжини. Також такий аглгоритм зазвичай дає узагальнений та нерізноманітний текст, розподіл якого не схожий на текст написаний людиною (Рисунок 2.25).

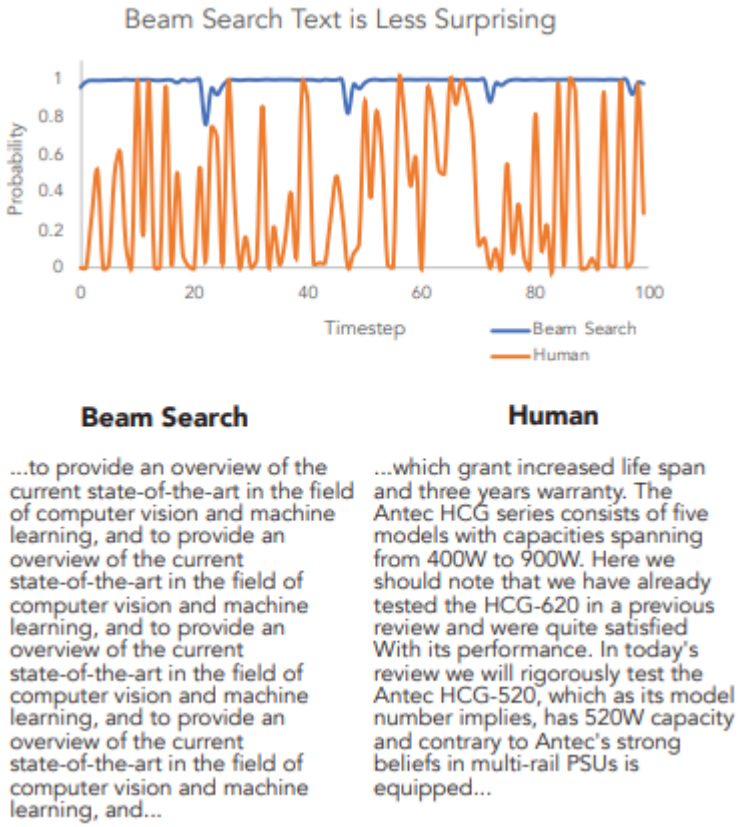


Рисунок 2.25 – Приклад розподілу при генерації тексту з beam пошуком в порівнянні з людським текстом.

2.5. Висновки

В даному розділі було досліджено низку архітектур нейронних мереж, а також було досліджено два декодуючих алгоритма. Вивчивши ці теоретичні відомості, можна зробити осмислений вибір архітектури та алгоритму, що будуть використані у створенні системи.

3. ВИБІР АРХІТЕКТУРИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Вибір архітектури

Тож дослідивши дані архітектури, було прийняте рішення вибрати архітектуру ResNet101 в якості енкодера та LSTM в якості декодера. Причиною вибору рекурентної мережі став той факт, що дана архітектура має значно меншу кількість параметрів, через що вона не займає багато місця в пам’яті комп’ютера. ResNet101 використовувався претренований на вибірці ImageNet.

Причиною вибору саме претренованої мережі є тенденція, яка широко використовується як в комп’ютерному зорі так і обробці людської мови. Ясоном Йосінськи, Джефом Клуном. Йоша Бенгіо та Ход Ліпсоном було показано, що використання претренованої мережі дає значний приріст в результатах. В своїй роботі вони розбили вибірку ImageNet, що має 1000 класів на 2 частини по 500. Нехай модель, що навчається на першій підвибірці буде позначена як baseA, а на другій – baseB. Тоді BnB – модель, яка використовує n шарів моделі, яка була натренована на другій підвибірці, тобто baseB, і дотреновується на ній же, а AnB – модель модель, що дотреновується на другій частині датасету, але використовує параметри моделі baseA для ініціалізації свої перших n шарів. Позначення «+» означає, що під час дотреновування, всі параметри моделі оновлюються. Тоді залежність точності класифікації від значення n зображена на Рисунку 3.1.

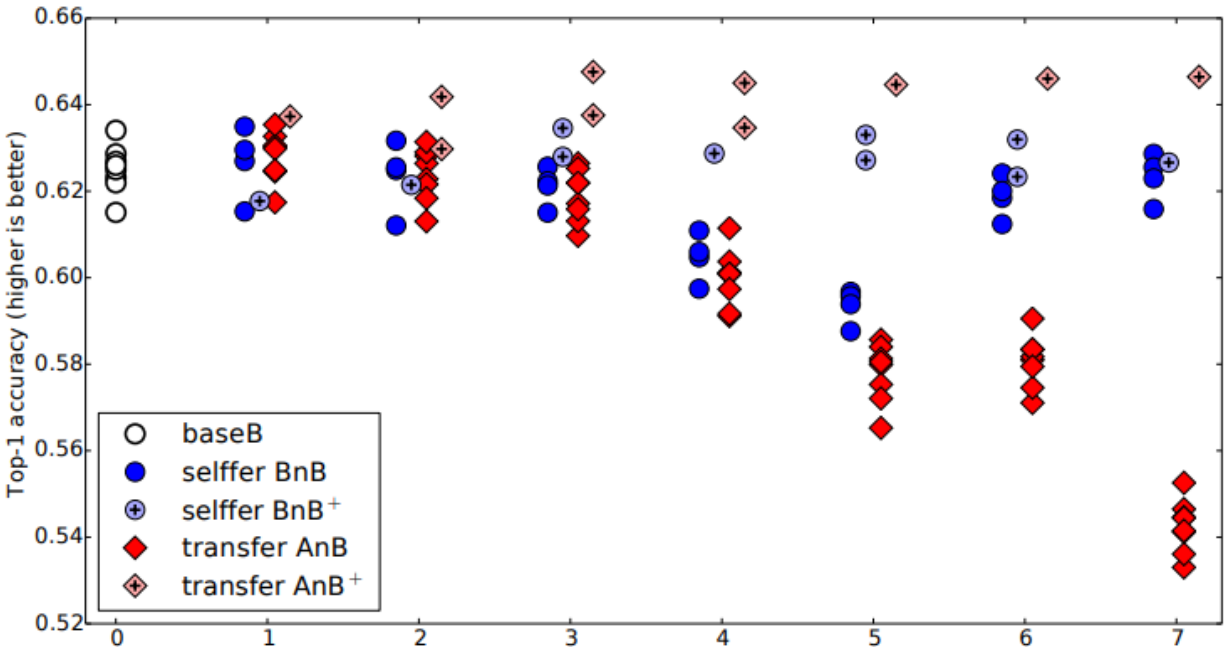


Рисунок 3.1.

Як бачимо, найвищу точність було отримано при використанні параметрів моделі, що була натренована на іншій частині ImageNet. Як було вказано в попередніх розділах, згорткові мережі здатні «вивчати» ознаки зображень. Тому використовуючи параметри натренованої моделі можна перенести певні «знання», що отримала мережа з попереднього навчального набору.

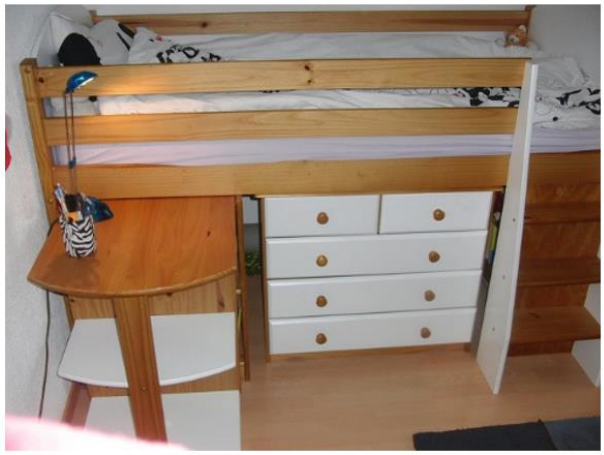
3.2. Навчальна вибірка

Спершу в якості навчальної вибірки був вибраний відомий датасет для даної задачі – Coco2014 (Рисунок 3.2). Даний датасет містить понад 80 тисяч зображень в навчальній вибірці та 40 тисяч валідаційної. Кожне зображення має 5 підписів.

The man at bat readies to swing at the pitch while the umpire looks on.

A large bus sitting next to a very tall building.

A horse carrying a large load of hay and two people sitting on it.



Bunk bed with a narrow shelf sitting underneath it.

Рисунок 3.2 – Приклади з датасету Coco2014.

3.2. Метрика оцінки якості згенерованого тексту.

В якості метрики оцінки якості було вибрано BLEU 4. BLEU – одна з перших оцінок якості згенерованого тексту, яка спершу була придумана для задачі машинного перекладу, а потім стала однією з основних для більшості задач обробки людської мови. Вона приймає значення від 0 до 1, де 1 означає точне співпадіння між згенерованим текстом та цільовим, тобто написаним людиною. Це є модифікованою версією широковикористованої метрики precision:

$$\text{Precision} = \frac{tp}{tp + fp}$$

де tp – кількість правильно передбачених прикладів, а fp – кількість помилок першого роду. Помилка першого роду в нашому випадку буде інтепретуватись як згенерований токен, який не зустрічається в цільовому реченні. Розглянемо приклад, нехай ми маємо два можливих варіанти згенерованого тексту (reference 1 та reference 2) та той, що передбачила модель (candidate) (Таблиця 3.1).

Таблиця 3.1

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

В даному випадку precision = $\frac{7}{7}$ = 1, тобто максимальне значення метрики, оскільки ми маємо ми маємо слово the, котре зустрічається як в reference1 так і в reference2. Проте зрозуміло, що дане передбачення не є бажаним. Тому модифікація полягає в наступному: для кожного слова в передбаченому реченні рахується їх кількість (m_w), потім для відповідних слів рахується їх кількість в реченнях, написаних людиною (m_{max}). Наступним кроком значення m_w обрізається до відповідного m_{max} . Для нашого прикладу m_w = 7, m_{max} = 2. Тоді значення модифікованої метрики буде: precision = $\frac{2}{7}$. Проте на практиці зазвичай використовується дана метрика по відношенню до n-грам аніж до окремих слів, де n-грамма – це послідовність слів довжиною n. Високе значення BLEU при використанні окремих слів може свідчити про приріст в збереженні інформації, коли використання n-грам свідчить про більш природній текст, схожий на людський. Було показано, що найбільш оптимальним значенням n = 4.

Проте дана метрика є чутливою до довжини згенерованої послідовності. Тоді можна поєднати дану метрику з метрикою recall:

$$\text{Recall} = \frac{tp}{tp + fn}$$

де fn – кількість помилок другого роду. Але високе значення даної метрики може бути отримане при генерації всіх токенів із цільових речень, якщо таких декілька, що також не є бажаним.

Тому для отримання BLEU для всього передбачення, рахується геометричне середнє модифікованого recall усіх n-грам та множиться на штраф за короткість. Нехай r – довжина цілевого тексту, а c – довжина згенерованого. Тоді, якщо c ≤ r, то штраф за короткість визначається як $e^{\left(1-\frac{r}{c}\right)}$. У випадку, коли цілевих речень декілька – використовується довжина найкоротшого.

3.3. Оптимізація

В якості алгоритму оптимізації був вибраний Adam (Рисунок 3.4), який є модифікацією стохастичного градієнтного спуску. Оскільки розмір навчальної вибірки є значно більшим за розміри, які можуть поміститись в операційну пам’ять, ми маємо навчатись групами (batches) – підвибірками датасету. Проте, оскільки за центральною граничною теоремою відомо, що стандартна помилка середнього обернено пропорційно залежить від розміру підвибірки з генеральної сукупності. Це означає те, що кожна група може бути дещо зміщена по відношенню до розподілу повної навчальної вибірки. Тому напрямки у якому ми будемо рухатись оптимізуючи цільову функцію, обчислюючи градієнти для кожної підвибірки (mini-batch) не будуть завжди вести до глобального мінімуму. Через це, збільшується час тренування, та ризик потрапити до локального мінімуму (Рисунок 3.3).

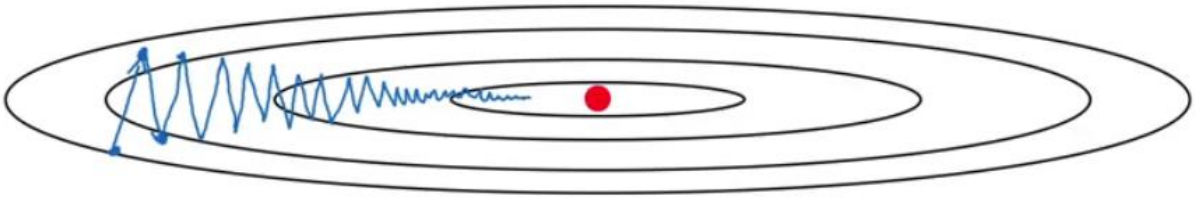


Рисунок 3.3 – Схематичне зображення наближення до глобального мінімуму використовуючи навчання підвибірками (mini-batches).

Алгоритм стохастичного mini-batch градієнтного спуску виглядає наступним чином:

Algorithm 1 Minibatch SGD

Require: Global learning rate η
Require: Batch size m
Require: Training set $\mathcal{X} = \{x_1, x_2, ...x_n\}$
Require: Initial model parameter θ_0
1: **while** θ_k not converged **do**
2: Update iteration: $k \leftarrow k + 1$
3: Get batch: select batch $\mathcal{B}$ of size m from training set $\mathcal{X}$ by SRS
4: Compute gradient : $\nabla J_{\mathcal{B}}(\theta_k) = \sum_{i \in \mathcal{B}} \nabla J_i(\theta)/m$
5: Apply update: $\theta_{k+1} = \theta_k - \eta * \nabla J_{\mathcal{B}}(\theta_k)$
6: **end while**

Натомість, алгоритм Adam поєднує в собі дві інші модифікації градієнтного спуску такі як градієнтний спуск з моментумом та середньоквадратичного розповсюдження (RMSProb). Він використовує техніку ковзного середнього для градієнтів та для їх L2 норми (норма Фробеніуса), за допомогою якої потім відбувається нормалізація, для адаптивного значення кроком навчання (learning rate).

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

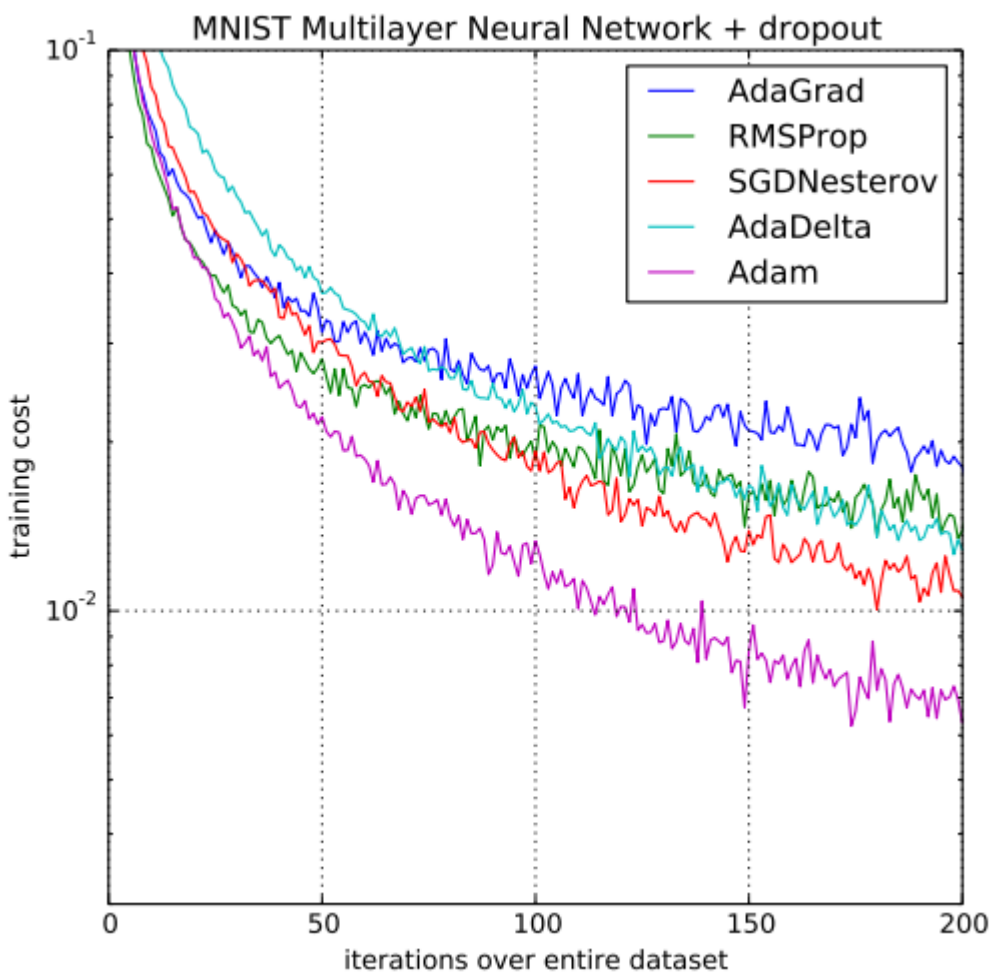


Рисунок 3.4 – Порівняння різних модифікацій стохастичного градієнтного спуску згорткових мереж на вибірці MNIST.

Крім цього, була використана стратегія Reduce Learning Rate On Plateau, ідея якої полягає в тому, що коли модель протягом вказаної кількості епох не покращує метрику, гіперпараметр крок навчання (learning rate), який відповідає за степінь оновлення параметрів моделі зменшується у попередньо вказану кількість разів.

3.5 Регуляризація

Були використані такі методи регуляризації як L2 регуряризація та дропаут. Ідея першого полягає в тому, що в лосс функцію додається доданок, з

регуляризаційним параметром λ. Нехай функція втрат (лосс функція) нашої мережі визначається як:

$$L(w) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)$$

де w – параметри моделі, m – кількість навчальних прикладів, $\hat{y}^{(i)}$ – значення, що передбачила модель, а $y^{(i)}$ – правильні значення для відповідного прикладу. Тоді при регуляризації:

$$L(w) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l-1}}\left(W_{ij}^{(l)}\right)^2$$

де L – кількість шарів мережі, u_l – кількість нейронів на шару l.

Ідея техніки дроп-ауту полягає в тому, що при тренуванні для кожної підвибірки (batch) деякі нейрони «вимикаються», тобто вини не впливають на результат проходу вперед і результат проходу назад (Рисунок 3.5). Нехай $h(x) = wx + b$ –лінійна комбінація вектору воду, $a(h)$ - функція активації деякого шару мережі, тоді за використанням даної техніки виходом цього шару буде вектор:

$$f(h) = D * a(h) \,/\, \mathrm{p}$$

де « $*$ » – поелементне множення, « / » – поелементне ділення, $D = (X_1 \dots X_n)$ – вектор випадкових величин X_i, розподілених за законом Бернуллі з параметром p.

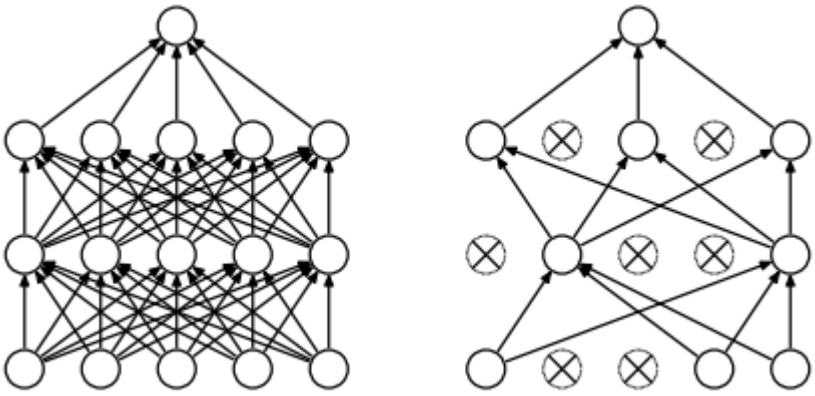


Рисунок 3.5 – Схематичне зображення використання дроп-ауту (справа) в порівнянні з звичайним проходом в нейронній мережі (зліва).

3.5 Експерименти

Було проведена низка експериментів в продовж якої змінювалися гіперпараметри моделі. Перша конфігурація використовувала ResNet50 в якості енкодера та одношарову LSTM в якості декодера. Розмір підвибірки був 512 (batch size). Нажаль, через власну помилку, графіки метрик були втрачені проте інформація щодо помилки на валідаційній та навчальній вибірці залишилися. Дана модель мала мінімальне значення помилки (лоссу) на валідаційній вибірці 2.24. Приклади роботи даної мережі (Рисунок 3.6.):

BotVitalika

a man in a blue shirt is standing on a beach holding a surfboard

BotVitalika

a bike parked in front of a brick wall .

BotVitalika

a bird sitting on a chair next to a bird .

BotVitalika

a black bear walking in the grass near a forest .

BotVitalika

a close up of a donut and a drink on a counter .

Рисунок 3.6.

Наступною конфігурація вже використовувала ResNet101, було використаний різний крок навчання (learning rate) для batch normalization шару, який зв’язував енкодер та декодер, та для декодеру, інші параметри залишитися такими ж. Результати даної мережі (Рисунок 3.7):

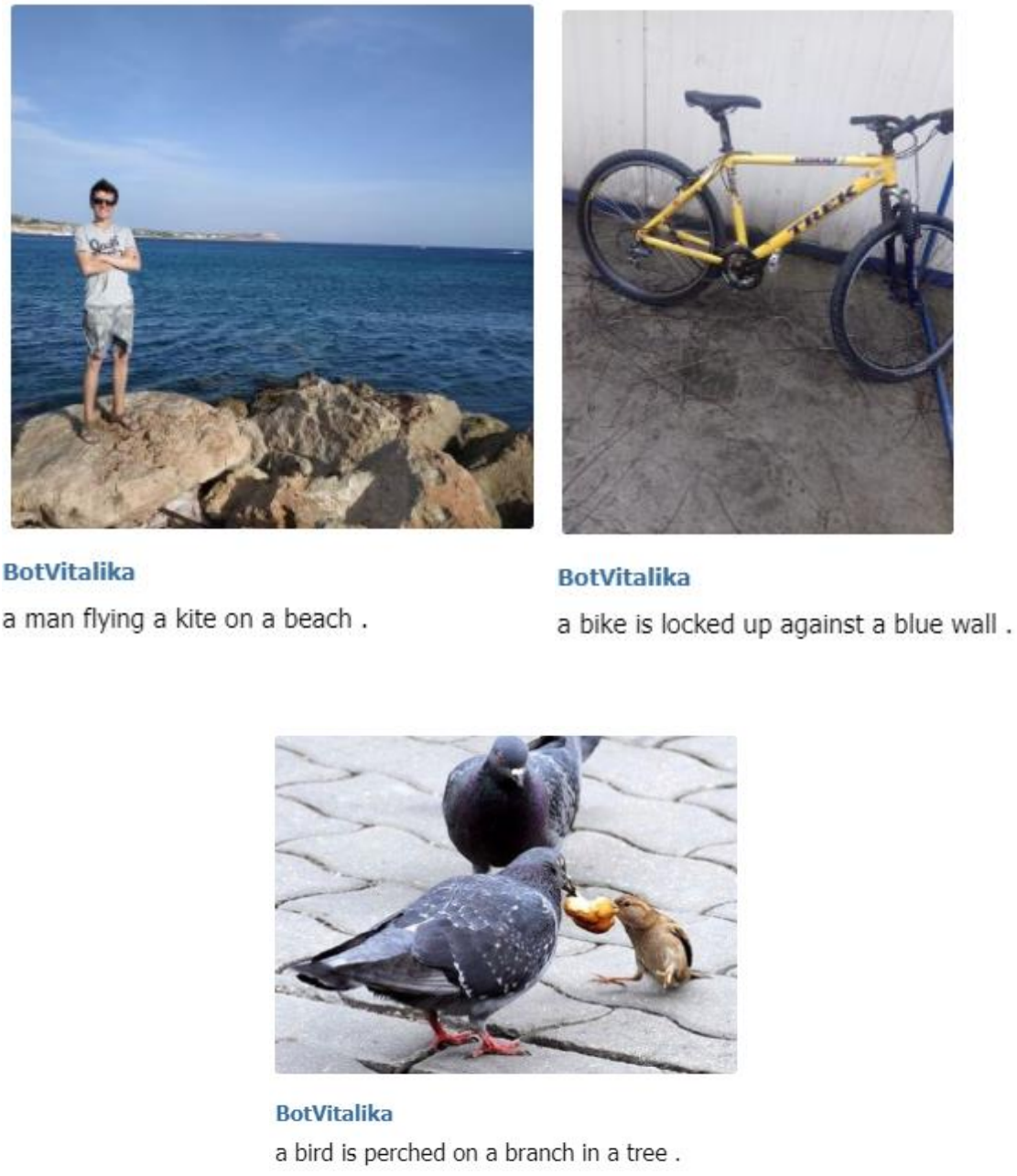


Рисунок 3.7.

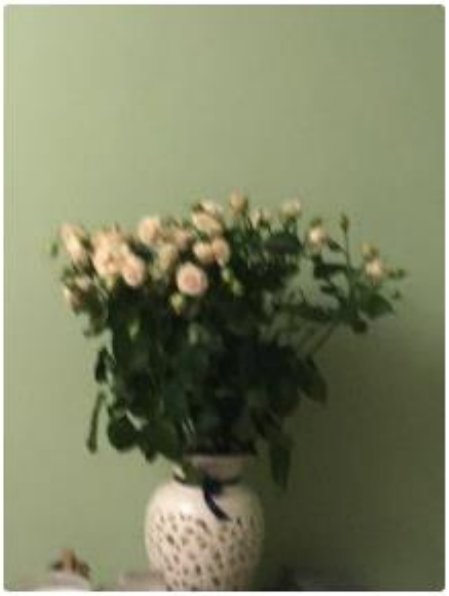
Потім глибина рекурентної мережі була збільшена до 2. А також була використана техніка акумулювання градієнтів. В даній конфігурації помилка (лосс) на валідаційній вибірці зменшилася до 2.16. Результати (Рисунок 3.8):

BotVitalika

A dog that is sitting in the grass

BotVitalika

A bowl of fruit sitting on a wooden table



BotVitalika

A bouquet of flowers in a vase on a table

BotVitalika

A clock hanging from the side of a wall

Рисунок 3.8.

В фінальній конфігурації кількість шарів LSTM був збільшений до 3, а також використаний дроп-аут та L2 регуляризація, щоб уникнути перенавчання. Також, після декількох епох декілька шарів енкодеру було «розморожено», тобто вони параметри тих шарів бути задіяні в проході назад. Через це розмір підвибірки (batch-size) був зменшений до 256, оскільки обчислення стали вимагати більше оперативної пам’яті. Це дозволило зменшити помилку (лосс) до 2.01 на валідації і отримати значення BLEU-4 близько 17.5. Варто відзначити, що метрика обчислювалася не на основі 5 можливих підписів, а тільки до одного. Результати даної конфігурації (Рисунок 3.9):

BotVitalika

A bunch of flowers that are in the grass .

BotVitalika

There is a bird that is sitting in the water

BotVitalika

A man standing next to a black dog on a beach .

BotVitalika

A bike parked next to a parking meter

Рисунок 3.9.

Далі остання конфігурація була дотренована на вибірці, яка була зібрана з соціальної мережі Інстаграм та опублікована в роботі Attend to You: Personalized Image Captioning with Context Sequence Memory Networks Байончанг Інга та Гахі Кіном. Цей датасет був зібраний по 270 ключевим словам, які складаються з 10 найпоширеніших хеш-тегів для кожної з 27 категорій соціального інтернет сервісу Pinterest. Публікації були відфільтровані за довжиною підпису (коротші за 3 та довші за 15 не потрапляли у вибірку), мовою (тільки ті, що містили більше 80% англійських слів). Максимальна кількість публікацій для одного користувача – 1000, мінімальна – 50 (Таблиця 3.2).

Таблиця 3.2 – Статистичні дані, щодо вибірки з Інстаграму. В дужках зазначена медіана.

Dataset	# posts	# users	# posts/user	# words/post
caption	721,176	4,820	149.6 (118)	8.55 (8)

Тож дотренована мережа давала такі приклади (Рисунок 3.10):

BotVitalika

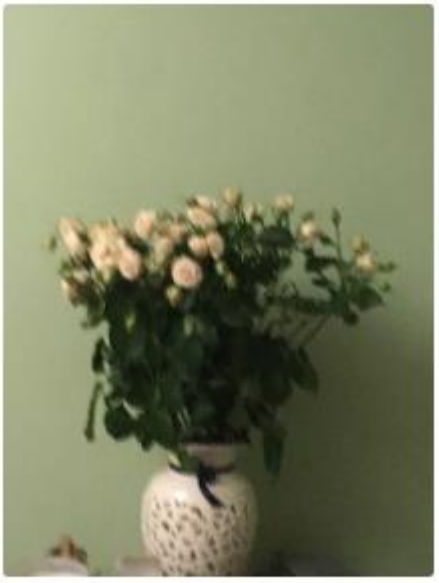
Spring is in the air

BotVitalika

Happy birthday to me

BotVitalika

Beautiful day for a walk in the park



BotVitalika

Flowers from my 😉 😉

BotVitalika

So true

Рисунок 3.10.

Варто відзначити, що смайли замінюють <unk> токени. Такі були отримані по тій причині, що моя архітектура не передбачала розширення словника моделі і всі слова, що їй раніше не зустрічалися замінювались на цей токен, а в другому датасеті була значна кількість слів, що не зустрічалися в першому.

Також, всі вищевказані приклади були згенеровані використовуючи алгоритм декодування beam-search зі значенням параметру k = 10. Було проведене невелике порівняння щодо використання різного значення k (Рисунки 3.11 – 3.14).

Рисунок 3.11 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 1 до 5 з кроком 1 моделі натренованої на Coco2014.

Рисунок 3.12 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 6 до 10 з кроком 1 моделі натренованої на Coco2014.

Рисунок 3.13 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 1 до 5 з кроком 1 моделі натренованої на Coco2014.

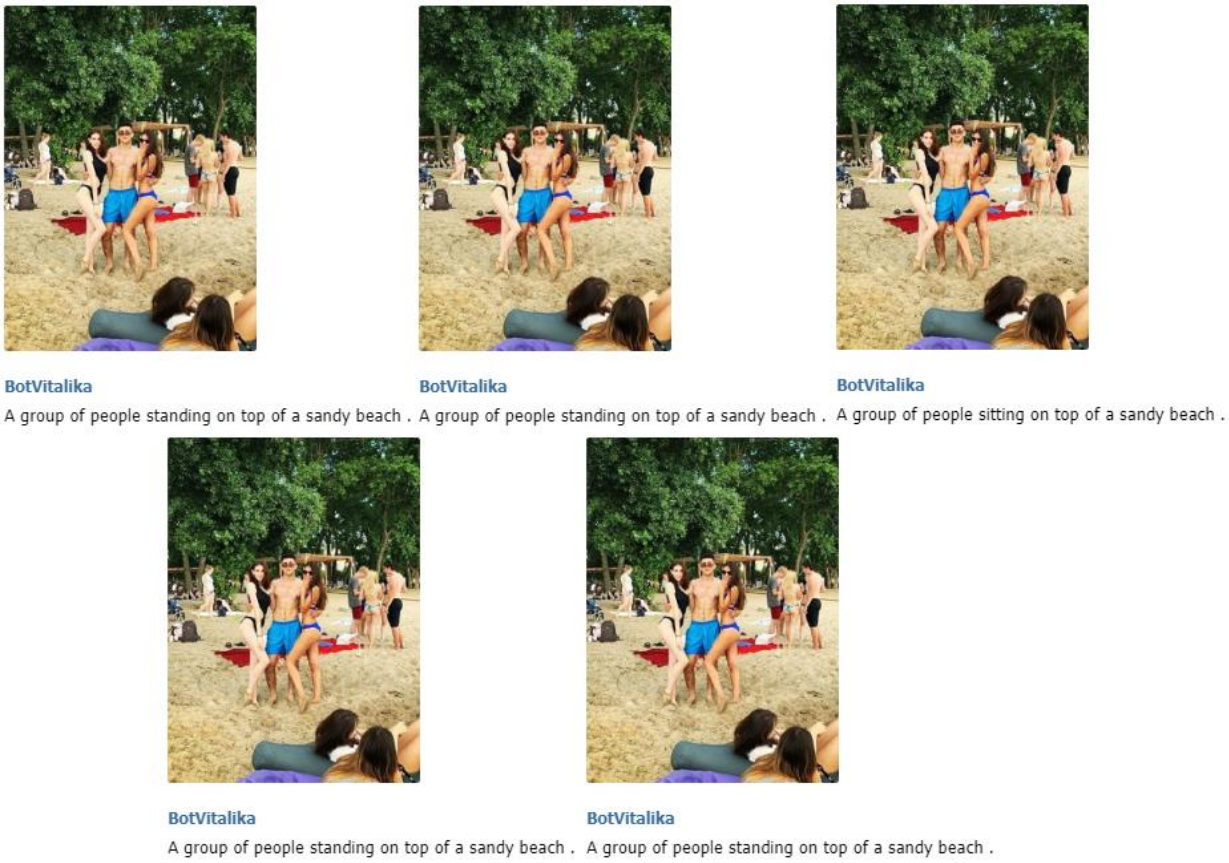


Рисунок 3.14 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 6 до 10 з кроком 1 моделі натренованої на Coco2014.

Можна побачити, що при збільшенні параметру k, різноманітність підпису зменшується. Іноді можна спостерігати, як при зменшенні параметру модель генерує нерелевантні токени такі як «boat» для зображення з пташкою на воді та «sitting» для зображення, де люди все ж стоять. Проте, можливо, через більшу варіативність даних, більше значення даного параметру може бути рішенням проблеми генерації поганих підписів моделлю, що натренована на Instagram датасеті (Рисунок 3.15, Рисунок 3.16).

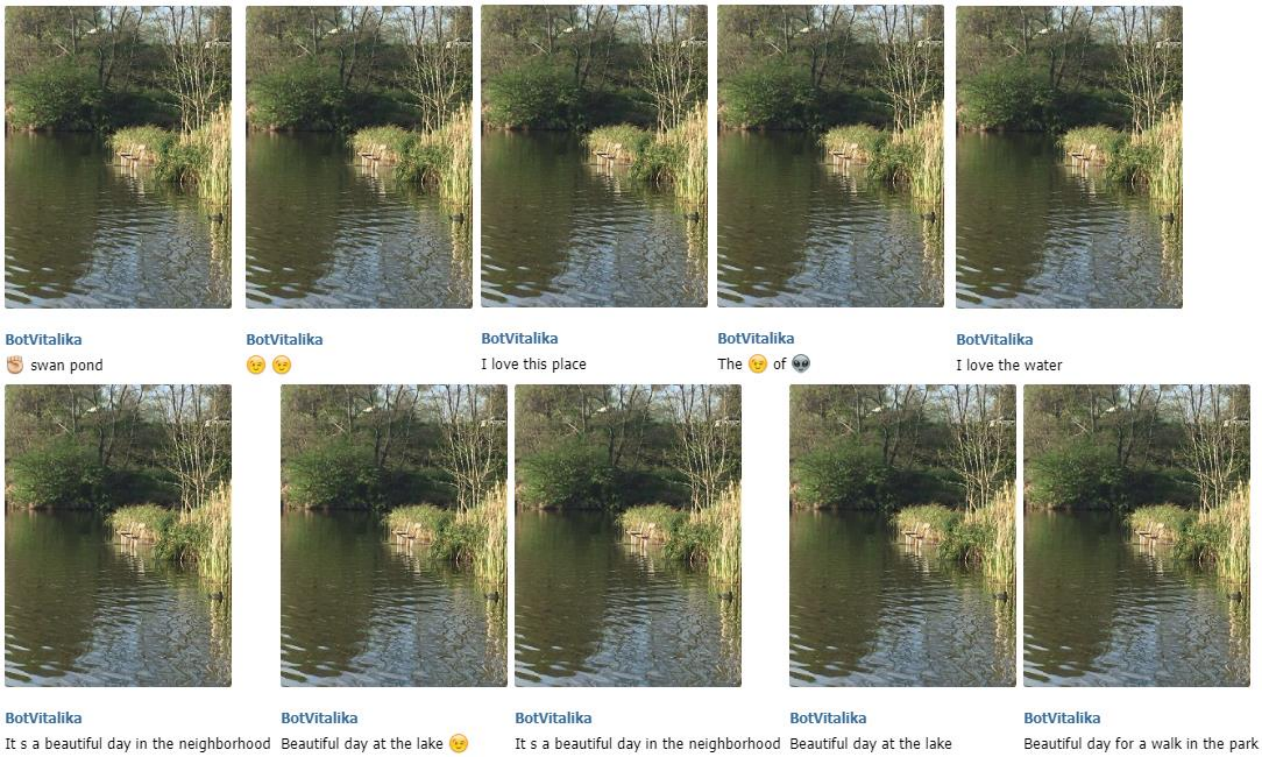


Рисунок 3.15 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 1 до 10 моделі натренованої на Instagram датасеті.

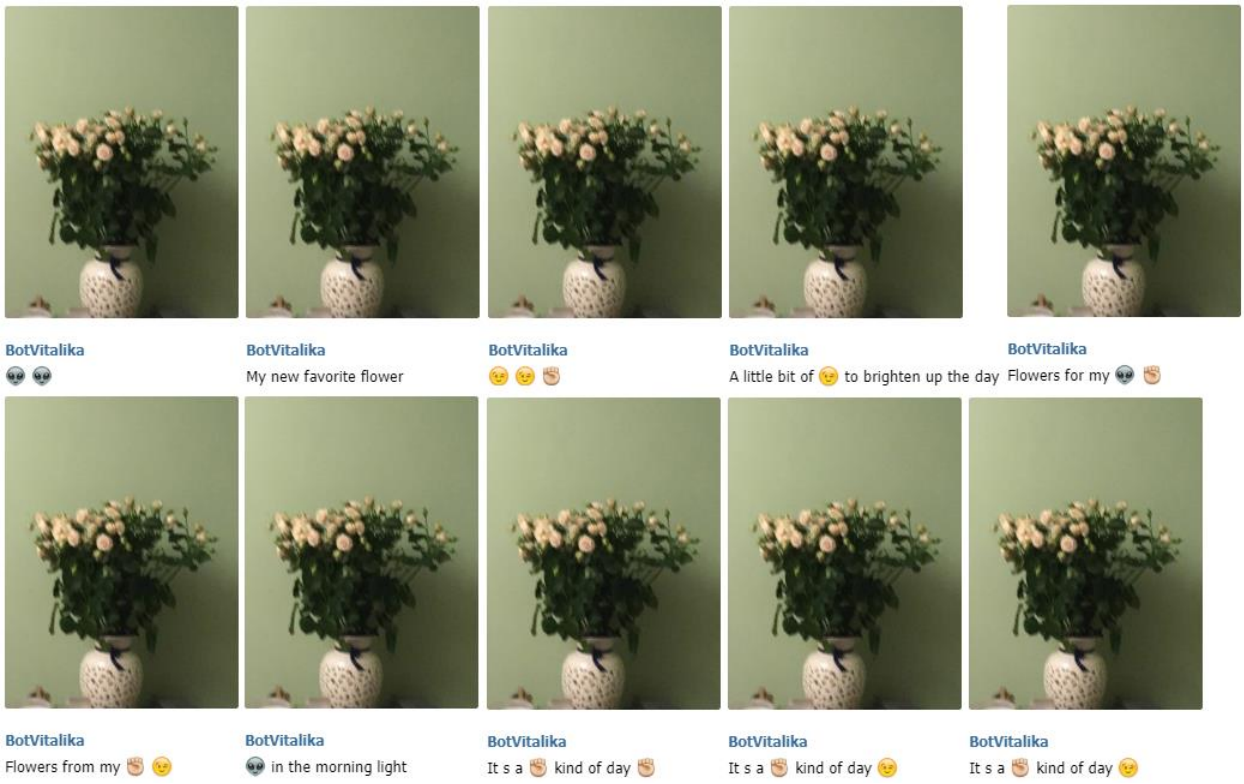


Рисунок 3.16 – Приклад згенерованого тексту для одного ж того зображення для параметру k від 1 до 10 моделі натренованої на Instagram датасеті.

З прикладів вище можна помітити схоильність моделі генерувати більш узагальнюючі фрази, які б могли підійти до великої кількості зображень.

3.6 Висновок

Тож в даному розділі були описані особливості навчання вибраної мережі. А також був проведений аналіз результатів та порівняння як деяких конфігурацій мережі так і декодуючих алгоритмів.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

4.1. Постановка задачі проектування

Проектований програмний продукт є системою, що здатна забезпечити зручну взаємодію з нейронною мережею для генерування тексту на основі візуальної інформації для отримання підписів для їх фото.

4.2. Обґрунтування функцій та параметрів програмного продукту

Виходячи з конкретних цілей, які реалізуються:

F_1 – хмарний сервіс: а) Google Cloud б) Heroku.

F_2 – вибір формату програмного забезпечення: а) телеграм – бот б) кросс – платформений мобільний додаток

F_3 – мова програмування: а) Python, б) C#

Виходячи з представлених варіантів будуємо морфологічну карту (рис. 4.1)

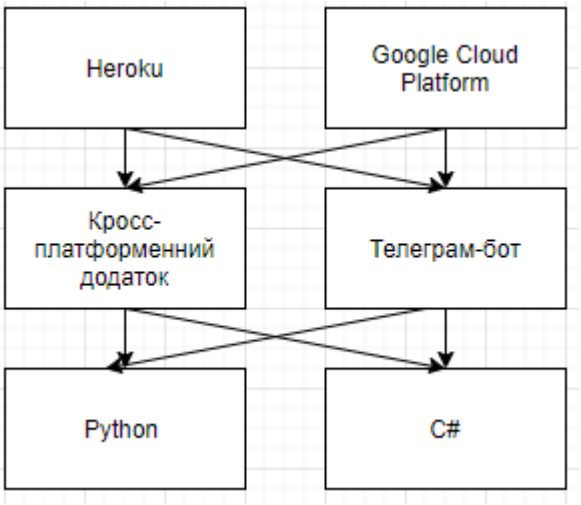


Рисунок 4.1 – Морфологічна карта

Дана морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів програмного продукту.

Спираючись на морфологічну карту, була побудована позитивно-негативна матриця (табл. 4.1)

Таблиця 4.1. Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	a	Має більше можливостей та вільного місця на диску	Не безкоштовний
	b	Повністю безкоштовний	Має мало вільного місця
F2	a	Простий в розробці	Не гнучкий інтерфейс
	b	Зручність використання з мобільного пристрою	Непростий в розробці
F3	a	Зручний при розробці як і кросс-платформенного додатку так і боту, зручний для роботи з глибинним навчанням	Повільніший
	b	Зручний при розробці кросс-платформенного додатку	Менше можливих бібліотек для глибинного навчання

Для характеристики прототипу програмного додатку використовуємо параметри X1 – X4. На основі даних, що представлені у літературі, визначаємо мінімальні, середні отримуванні та максимально допустимі значення(табл. 4.2)

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одинці виміру	Значення параметра		
			гірші	середні	кращі

Швидкодія мови програмування	X1	Оп/мс	18000	10000	1000
Об’єм коду	X2	К-ть строк	2100	1600	1100
Час обробки запитів користувача	X3	Мс	550	275	50
Об’єм пам’яті для збереження даних	X4	Мб	64	32	16

За даними, наведеними у таблиці 4.2, будуються графічні характеристики параметрів – рисунки 4.2 – 4.5.

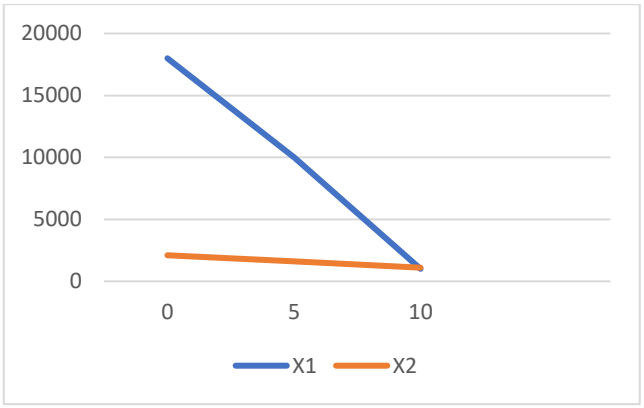


Рис. 5.2 Значення параметрів X1, X2

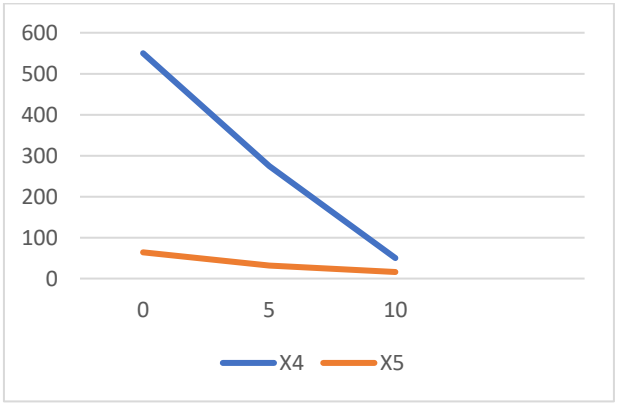


Рис.5.4 Значення параметрів X3 X4

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Рани варіюються від 1 до 4 (вищий – нижчий). Результати наведені в табл. 4.3.

Таблиця 4.3 – Результати ранжування параметрів

параметр	Ранг параметра за оцінкою експерта							Сума рангів в R_i	Відхилення Δi	Δ_{i^2}
	1	2	3	4	5	6	7			
X1	1	3	3	3	1	3	3	17	1,5	2,25
X2	4	4	4	4	3	4	4	27	9,5	90,25
X3	2	1	1	1	2	1	1	9	-8,5	72,25
X4	3	2	2	2	4	2	2	17	-2,5	6,25
Разом	10	10	10	10	10	10	10	70	0	171

Таблиця 4.4 Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1.5
X1 і X3	>	<	<	<	>	<	<	<	0.5
X1 і X4	>	<	<	<	>	<	<	<	0.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	>	<	<	<	0.5
X3 і X4	>	>	>	>	>	>	>	>	1.5

Визначимо коефіцієнт конкординації:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 171}{49 * (4^3 - 4)} = 0{,}69 > W_k = 0{,}67$$

Так як коефіцієнт конкординації більше нормативного, результати вважають достовірними.

Розрахунок вагомості параметрів наведено в табл. 4.5

Таблиця 4.5 розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$\text{К}_{\text{ві}}^1$	b_i^2	$\text{К}_{\text{ві}}^2$
X1	1	1.5	0.5	0.5	3.5	0.219	12.25	0.2	48.25	0.211
X2	0.5	1	0.5	0.5	2.5	0.156	11.5	0.188	36.375	0.159
X3	1.5	1.5	1	1.5	5.5	0.344	21.25	0.347	81.25	0.356
X4	1.5	1.5	0.5	1	4.5	0.281	16.25	0.265	62.5	0.247
Разом:					16	1	61.25	1	228.375	1

Враховуючи дані з порівнянь варіантів реалізацій функцій можна виключити з реалізацій функцій наступні варіанти: F1(б), F3(б)

Залишаються наступні:

1) F1а-F2а-F3а
2) F1а-F2б-F3а

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1 (X1)	А	10000	5	0.274	1.37
F2 (X2)	А	1300	3.5	0.356	1.246
F2 (X2)	Б	1100	3.2	0.356	1.14
F2 (X3)	А, Б	400	2.5	0.159	0.39
F3 (X4)	А	64	10	0.211	2.11

Обрахуємо коефіцієнти якості кожного з варіантів розробки:

$$K_{я1} = 1.37 + 0.39 + 1.246 + 2.11 = 5.116$$

$$K_{я2} = 1.37 + 0.39 + 1.14 + 2.11 = 5.01$$

Оскільки варіант 2 має найменше коефіцієнт якості, він є найкращим.

4.3 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Деплой продукту на Google Cloud Platform;
2. Розробка на мові Python;

Кожний з варіантів має додаткове завдання, які є реалізаціями розгалужених варіантів розробки незалежного модуля. Далі наведено варіанти додаткових завдань:

3.1 Реалізація продукту в вигляді Телеграм-Боту

3.2 Реалізація в вигляді кросс-платформенного мобільного додатку

У варіанті 1 присутнє наступне додаткове завдання під номером 3.1.

У варіанті 2 присутнє наступне додаткове завдання під номером 3.2.

За ступенем новизни завдання 1 відноситься до групи А, завдання 2, 3.1 відноситься до групи В, завдання 3.2 відноситься до групи Б.

За складністю алгоритми, які використовуються в завданні 1 належать до 1 групи, завдання 2, 3.1, 3.2 – до 3 групи.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_п$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної

інформації для всіх семи завдань рівний 1: $К_{ск} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{ст}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 * 1.7 * 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для інших завдань. Для другого завдання (використовується алгоритм третьої групи складності , ступінь новизни В), тобто $Т_р$ =12 людино-днів, $К_п$ =0.6, $К_{ск}$ = 1, $К_{ст}$ = 0.8. Отже:

$T_2 = 12 * 0.6 * 0.8 = 5.76$ людино-днів.

Аналогічно для завдання 3.1 та 3.2:

$$T_3 = 12 * 0.6 * 0.8 = 5.76$$

$Т_р$ =19 людино-днів, $К_п$ =0,9, $К_{ск}$ =1, $К_{ст}$ =0,8

$$T_4 = 19 * 0.6 * 0.8 = 9{,}12$$

Визначимо повну трудомісткість варіантів (людино-днів):

$$T_1 = 122{,}4 + 5{,}76 + 5{,}76 = 133{,}92$$

$$T_2 = 122{,}4 + 5{,}76 + 9{,}12 = 137{,}28$$

Найбільш трудомістким завданням є 2, найбільш трудомістким варіантом є 2. Далі вважається, що робочий день складає 8 годин, в тижні п’ять робочих днів. В розробці бере участь два програміста з окладом 17000 грн та аналітик з окладом 19500 грн. Визначимо середню заробітну плату за годину:

$$С_ч = \frac{2 * 17000 + 19500}{3 * 21 * 8} = 106{,}15$$

Тоді заробітну плату визначимо за формулою:

Заробітна розробників за варіантами становить:

$$С_{ЗП} = 106{,}15 * 8 * 133{,}92 = 113724{,}86$$

$$С_{ЗП} = 106{,}15 * 8 * 137{,}28 = 116578{,}176$$

Відрахування на єдиний соціальний внесок становить 22%:

$$С_{ВІД} = 113724{,}86 * 0{,}22 = 25019.46$$

$$С_{ВІД} = 116578{,}176 * 0{,}22 = 25647.19$$

Визначимо витрати на оплату однієї машино-години. Так як одна машина обслуговує двох програмістів з окладом 17000 грн. з коефіцієнтом зайнятості 0.6 та одного аналітика з окладом 19500, то для трьох машин отримаємо:

$С_г = 12 * 17000 * 2 * 0{,}6 + 12 * 19500 * 0{,}6 = 385200.0$(грн)

Враховуючи додаткову заробітну плату 40%

$С_{ЗП} = 385200 * (1 + 0{,}4) = 539280$ (грн)

Відрахування на соціальне страхування 22%

$С_{ВІД} = 539280 * 0{,}22 = 118641.6$ (грн)

Розрахуємо амортизаційні підрахунки (амортизація 25%, вартість ЕОМ 21000 грн)

$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1{,}15 * 0{,}25 * 21000 = 6037{,}5$ (грн)

Розрахуємо витрати на ремонт та профілактику:

$$С_Р = К_{ТМ} * Ц_{ПР} * К_Р = 1{,}15 * 21000 * 0{,}05 = 1207{,}5 \text{ (грн)}$$

Розрахуємо ефективний годинний фонд часу ПК за рік:

$$Т_{ЕФ} = (365 - 142 - 16) * 8 * 0{,}8 = 1324{,}8 \text{ год}$$

Розрахуємо витрати на електроенергію

$$С_{ЕЛ} = 1324{,}8 * 0{,}6 * 0{,}8 * 1{,}75 = 1112{,}83 \text{ грн}$$

Накладні витрати рівні:

$С_Н = 21000 * 0{,}67 = 14\,070$ (грн)

Отже експлуатаційні витрати:

$С_{ЕКС} = 118641.6 + 539280 + 6037{,}5 + 1207{,}5 + 1112{,}83 + 14\,070 = 680349$ (грн)

Собівартість однієї машино-години буде:

См-г = $\frac{680349}{1324{,}8}$ = 513.54 грн/год.

Оскільки в даному випадку всі роботи, які пов’язані з розробкою програмного продукту ведуться на ЕОМ, то витрати на оплату машинного часу в залежності від обраного варіанту, складають:

См = 513.54 *8* 133,92= 550186 грн.

См = 513.54*8* 137,28= 563990 грн.

Накладні витрати складають 67% від заробітної плати:

$С_Н = 550186 * 0{,}67 = 368624.62$ (грн)

$С_Н = 563990 * 0{,}67 = 377873.3$ (грн)

Отже, вартість розробки програмного продукту за варіантами становить:

$С_{ПП} = 113724{,}86 + 25019.46 + 550186 + 368624.62 = 1057554.94$ (грн)

$С_{ПП} = 116578{,}176 + 25647.19 + 563990 + 377873.3 = 1084088.66$ (грн)

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня:

$$К_{ТЕР1} = \frac{5.116}{1057554.94} = 4.83 * 10^{-6}$$

$$К_{ТЕР2} = \frac{5.01}{1084088.66} = 4.62 * 10^{-6}$$

4.5 Висновок

Отже враховуючи всі дослідження, що описані вище, можна сказати, що 1 варіант реалізації є найбільш оптимальним зі сторони якісно-економічної оцінки. Його коефіцієнт техніко-економічного рівня складає $4.83 * 10^{-6}$.

Цей варіант включає

– хмарний сервіс – Google Cloud Platform;

– продукт в вигляді телеграм боту;

– розробка на мові програмування Python.

ВИСНОВКИ

В даній роботі було досліджено низку архітектур, якими можна розв’язати задачу опису зображення. Наступним кроком були розібрані детально можливі складові вибраної системи. Та було показано, що навіть така, досить проста нейронна мережа, може генерувати релевантні описи. Під час розробки та тренування мережі було вивчено деякі передові техніки навчання. Тож можна вважати, що дана задача була розв’язана в даній роботі. Також мережу було використано в додатку для створення підписів у соціальну мережу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.

1. Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering. Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, Lei Zhang. URL: https://arxiv.org/ pdf/1707.07998.pdf
2. Image Captioning Transformer. Martin Krasser. URL: https://github.com/krasserm/fairseq-image-captioning
3. Convolutional Neural Networks. CS231N. URL: https://cs231n.github.io/convolutional-networks/
4. ImageNet Classification with Deep Convolutional Neural Networks. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks
5. PyTorch Documentation. URL: https://pytorch.org/docs/master/generated/torch.nn.Conv2d.html
6. Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. URL: https://arxiv.org/ pdf/1512.03385.pdf
7. Visualizing and Understanding Convolutional Networks. Matthew D Zeiler, Rob Fergus. URL: https://arxiv.org/ pdf/1311.2901.pdf
8. Natural Language Processing with Deep Learning (CS224N/Ling284). Abigail See, John Hewitt. URL: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf
9. Simple RNN and Backpropagation. Ekaterina Lobacheva. URL: https://www.coursera.org/lecture/intro-to-deep-learning/simple-rnn-and-backpropagation-zGHtr

10. Modern RNNs: LSTM and GRU. Ekaterina Lobacheva. URL: https://de.coursera.org/lecture/intro-to-deep-learning/modern-rnns-lstm-and-gru-WpduX

11. DeepLearning series: Sequence Models. Michele Cavaioni. URL: https://medium.com/machine-learning-bites/deeplearning-series-sequence-models-7855babeb586

12. THE CURIOUS CASE OF NEURAL TEXT DeGENERATION by Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi. URL: https://arxiv.org/pdf/1904.09751.pdf

13. The Illustrated GPT-2 (Visualizing Transformer Language Models) by Jay Alammar. URL: https://jalammar.github.io/illustrated-gpt2/

14. Improving Language Understanding with Unsupervised Learning by Alec Radford. URL: https://openai.com/blog/language-unsupervised/

15. Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

16. Attention Is All You Need by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. URL: https://arxiv.org/pdf/1706.03762.pdf

17. The Illustrated Transformer by Jay Alammar. URL: https://jalammar.github.io/illustrated-transformer/

18. Layer Normalization Explained by Lei Mao. URL: https://leimao.github.io/blog/Layer-Normalization/

19. Illustrated: Self-Attention by Raimi Karim. URL: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

20. Natural Language Processing with Deep Learning CS224N/Ling284, Lecture 8: Machine Translation, Sequence-to-sequence and Attention by Abigail See,

Matthew Lamm slides: URL: https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

21. How transferable are features in deep neural networks? Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson. URL: https://arxiv.org/ pdf/1411.1792

22. Microsoft COCO: Common Objects in Context. Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár. URL: https://arxiv.org/ pdf/1405.0312.pdf

23. BLEU. Wikipedia. URL: https://en.wikipedia.org/wiki/BLEU

24. Adam: A Method for Stochastic Optimization. Diederik P. Kingma, Jimmy Ba. URL: https://arxiv.org/ pdf/1412.6980.pdf

25. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. URL: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

26. Attend to You: Personalized Image Captioning with Context Sequence Memory Networks. Cesc Chunseong Park, Byeongchang Kim, Gunhee Kim. URL: https://arxiv.org/pdf/1704.06485.pdf

ДОДАТОК А

Модель генерації тексту на основі візуальної інформації

Виконав: Корень Віталій Олександрович
Науковий керівник: д.т.н. проф. Зайченко Олена Юріївна

Актуальність

за даними маркетингової компанії Ipsos Mori британці, в період карантину почали проводити на 47% більше часу в соціальних мережах.

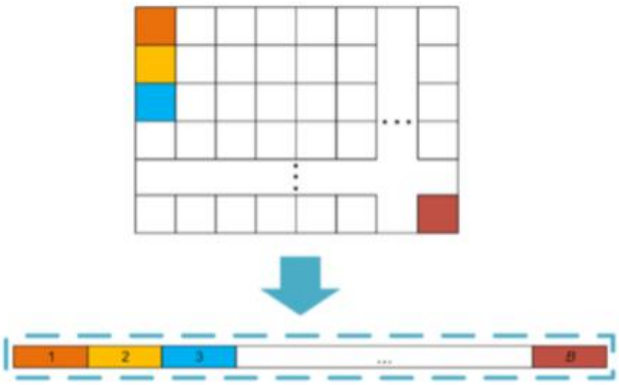
існуючі підходи

Постановка задачі

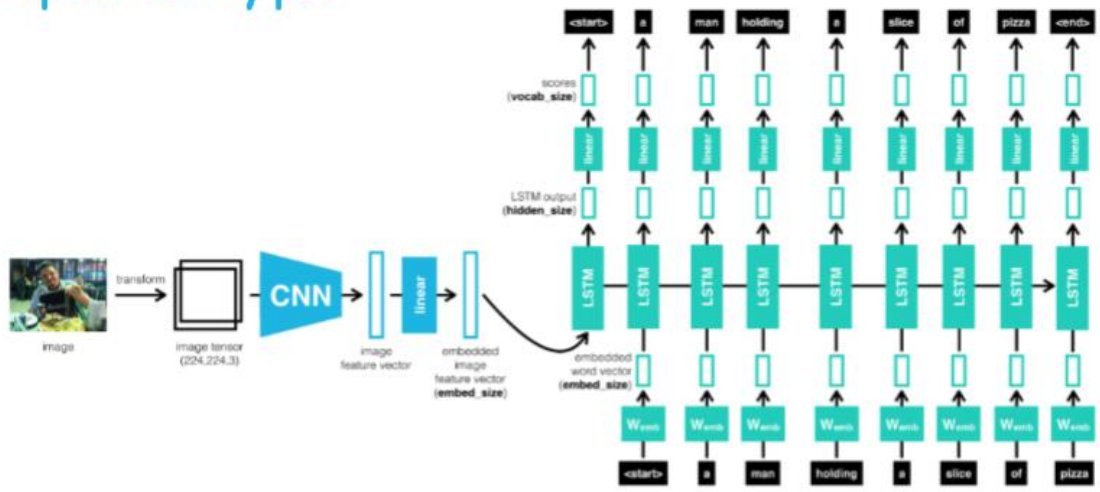
Створити систему, що буде використовувати підходи глибинного навчання для автоматичної генерації підписів до зображень у соціальні мережі

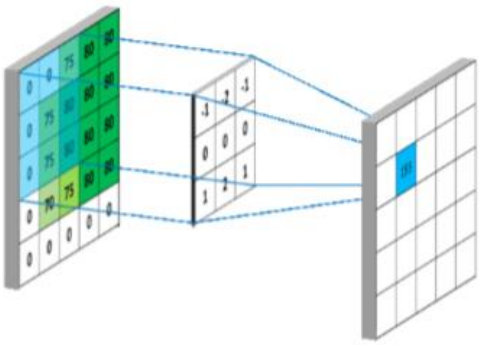
Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Apr 06, 2020	SA-Net on Albert (ensemble) *QIANXIN*	**90.724**	**93.011**
2 May 05, 2020	SA-Net-V2 (ensemble) *QIANXIN*	90.679	92.948
2 Apr 05, 2020	Retro-Reader (ensemble) *Shanghai Jiao Tong University* http://arxiv.org/abs/2001.09694v2	90.578	92.978
3 May 04, 2020	ELECTRA+ALBERT+EntitySpanFocus (ensemble) *SRCB_DML*	90.442	92.839
4 Mar 12, 2020	ALBERT + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	90.386	92.777
5 Jan 10, 2020	Retro-Reader on ALBERT (ensemble) *Shanghai Jiao Tong University* http://arxiv.org/abs/2001.09694v2	90.115	92.580

- Мета: Дослідити існуючі підходи систем генерації тексту на основі візуальної інформації
- Предмет дослідження: Нейронна мережа, що складається зі згорткової у якості енкодера та рекурентної у якості декодера.
- Об'єкт дослідження: Навчальні вибірки зображень та відповідних описів або підписів до них

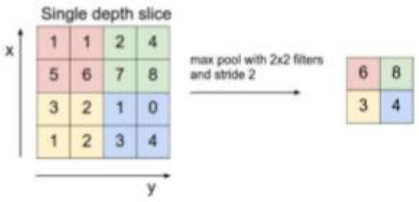


Архітектура

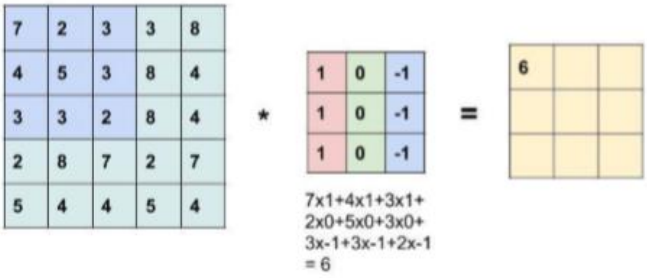




pooling



convolution

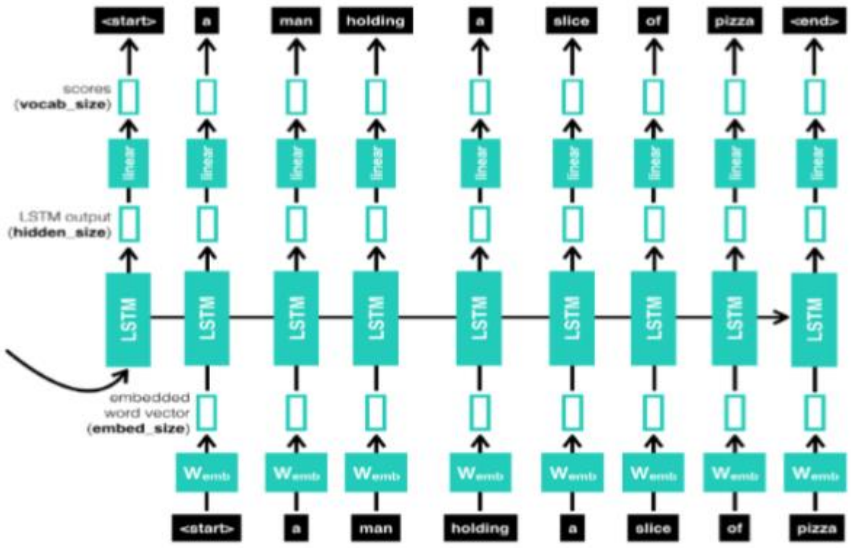


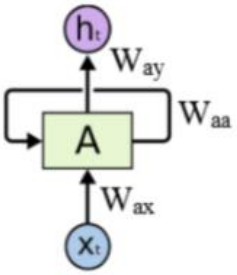
Згортковий шар:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

де ★ - операція згортки:

$$C_{ij} = \sum_{k,l}^{K,K} a_{k+i,l+j} * b_{k,l}\,, \text{ де } i = \overline{0, m-k+1},\ j = \overline{0, n-k+1},$$





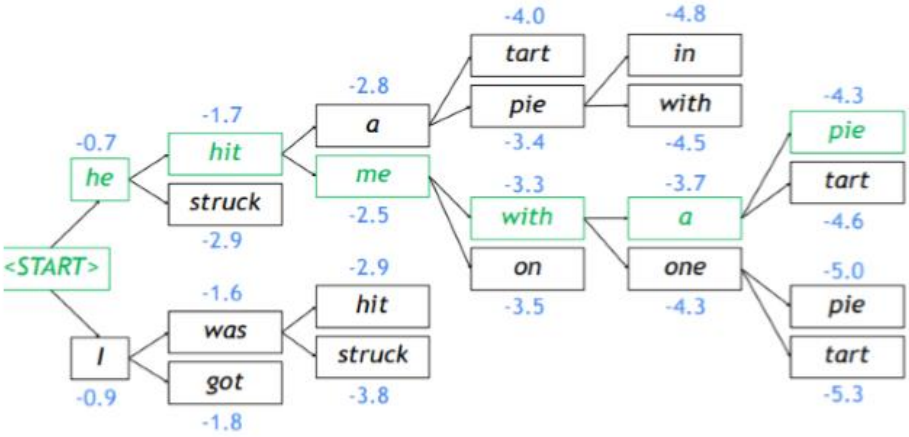
Цілева функція нашої умовної мовної моделі:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_c\,,\, u_{image}; \Theta)$$

де u_c - попередньо згенеровані токени,
а u_{image} – векторне представлення зображення.

Beam-search | k = 2

Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Coco2014

The man at bat readies to swing at the pitch while the umpire looks on.

A large bus sitting next to a very tall building.

Instagram Dataset

(GT) pool pass for the summer ✔

(GT) the face in the woods

(GT) awesome view of the city

Вдалі приклади, згенеровані нейронною мережею натренованою тільки на Coco2014:

BotVitalika

A bunch of flowers that are in the grass .

BotVitalika

There is a bird that is sitting in the water

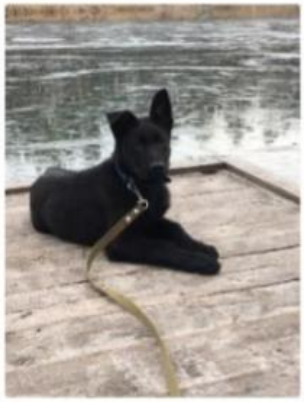
BotVitalika

A group of people standing on top of a sandy beach .

Невдалі приклади, згенеровані нейронною мережею натренованою тільки на Coco2014:

BotVitalika

A couple of birds standing on top of a wooden table .



BotVitalika

A man standing next to a black dog on a beach .

Приклади, згенеровані нейронною мережею натренованою на Coco2014 та дотренованою на Instagram вибірці:

BotVitalika

Spring is in the air

BotVitalika

Fun at the park

BotVitalika

Happy birthday to me

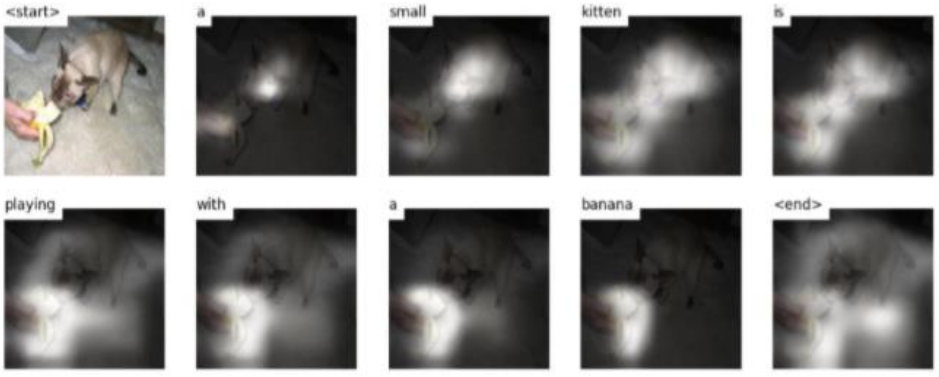
Висновок

- В даній роботі була запропонована нейронна мережа, що здатна створювати достатньо хороші підписи під зображення в соціальні мережі.

- Варто відзначити, що дана архітектура може бути застосована для вирішення більш важливих проблем, наприклад, допомога незрячим людям.

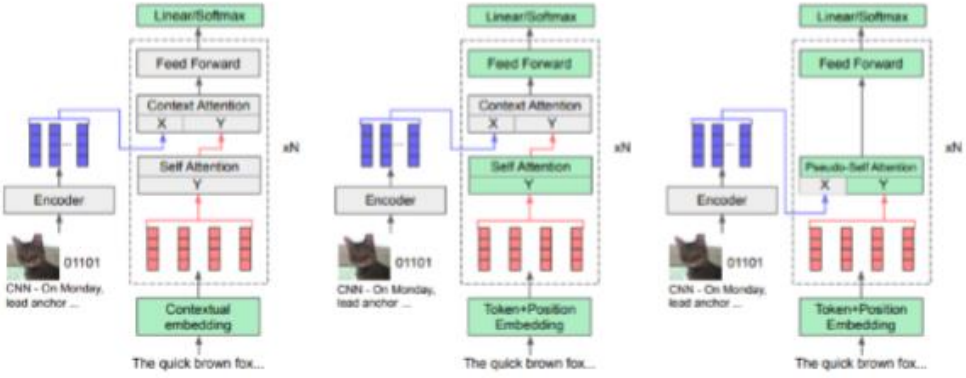
Шляхи подального розвитку

- Реалізація роботи Show, Attend and Tell: Neural Image Caption Generation with Visual Attention



Шляхи подального розвитку

- Реалізація роботи Encoder-Agnostic Adaptation for conditional language generation



(a) Repr-Transformer (b) Context-Attn (c) Pseudo-Self

ДОДАТОК Б

data_loader.py

```
import os
import nltk
import pickle
import torch
import torch.utils.data as data
from vocabulary import Vocabulary
from PIL import Image
from pycocotools.coco import COCO
import numpy as np
from tqdm import tqdm
import random
import json

from nltk.tokenize import RegexpTokenizer
regex_tokenizer = RegexpTokenizer(r'\w+')

def get_loader(transform,
               mode='train',
               batch_size=1,
               vocab_threshold=None,
               vocab_file='./vocab.pkl',
               glove_file='./glove.pkl',
               start_word="<start>",
               end_word="<end>",
```

```
        unk_word="<unk>",
        vocab_from_file=True,
        num_workers=0,
        dataset = 'coco'):
    """Returns the data loader.
    Args:
      transform: Image transform.
      mode: One of 'train' or 'test'.
      batch_size: Batch size (if in testing mode, must have batch_size=1).
      vocab_threshold: Minimum word count threshold.
      vocab_file: File containing the vocabulary.
      start_word: Special word denoting sentence start.
      end_word: Special word denoting sentence end.
      unk_word: Special word denoting unknown words.
      vocab_from_file: If False, create vocab from scratch & override any existing vocab_file.
                       If True, load vocab from from existing vocab_file, if it exists.
      num_workers: Number of subprocesses to use for data loading
    """

    assert mode in ['train', 'test', 'val'], "mode must be one of 'train', 'val' or 'test'."
    if vocab_from_file==False: assert mode=='train' or mode=='val', "To generate vocab from captions file, must be in training  or val mode."

    # Based on mode (train, val, test), obtain img_folder and annotations_file.
    if mode == 'train':
```

```
    if vocab_from_file==True: assert os.path.exists(vocab_file), "vocab_file does not exist.  Change vocab_from_file to False to create vocab_file."
    if dataset=='coco':
        img_folder = './train2014/train2014/'
        annotations_file = './captions/annotations/captions_train2014.json'
    elif dataset=='insta':
        img_folder = './Insta/images/'
        annotations_file = './captions/annotations/insta-caption-train.pkl'
  # validation
  if mode == 'val':
    if vocab_from_file==True: assert os.path.exists(vocab_file), "vocab_file does not exist.  Change vocab_from_file to False to create vocab_file."
    if dataset=='coco':
        img_folder = './val2014/val2014/'
        annotations_file = './captions/annotations/captions_val2014.json'
    elif dataset=='insta':
        img_folder = './Insta/images/'
        annotations_file = './captions/annotations/insta-caption-test1.pkl'

  if mode == 'test':
    assert batch_size==1, "Please change batch_size to 1 if testing your model."
    assert os.path.exists(vocab_file), "Must first generate vocab.pkl from training data."
    assert vocab_from_file==True, "Change vocab_from_file to True."
    annotations_file = None

  # COCO caption dataset.
```

```
if dataset=='coco':
    dataset = CoCoDataset(transform=transform,
                          mode=mode,
                          batch_size=batch_size,
                          vocab_threshold=vocab_threshold,
                          vocab_file=vocab_file,
                          glove_file=glove_file,
                          start_word=start_word,
                          end_word=end_word,
                          unk_word=unk_word,
                          annotations_file=annotations_file,
                          vocab_from_file=vocab_from_file,
                          img_folder=img_folder)
elif dataset=='insta':
    dataset = InstaDataset(transform=transform,
                          mode=mode,
                          batch_size=batch_size,
                          vocab_threshold=vocab_threshold,
                          vocab_file=vocab_file,
                          glove_file=glove_file,
                          start_word=start_word,
                          end_word=end_word,
                          unk_word=unk_word,
                          annotations_file=annotations_file,
                          vocab_from_file=vocab_from_file,
                          img_folder=img_folder)
```

```
    if mode == 'train' or mode == 'val':
        # Randomly sample a caption length, and sample indices with that length.
        indices = dataset.get_train_indices()
        # Create and assign a batch sampler to retrieve a batch with the sampled indices.
        initial_sampler = data.sampler.SubsetRandomSampler(indices=indices)
        # data loader for COCO dataset.
        data_loader = data.DataLoader(dataset=dataset,
                                      num_workers=num_workers,

batch_sampler=data.sampler.BatchSampler(sampler=initial_sampler,
                                                        batch_size=dataset.batch_size,
                                                        drop_last=False))
    else:
        data_loader = data.DataLoader(dataset=dataset,
                                      batch_size=dataset.batch_size,
                                      shuffle=False,
                                      num_workers=num_workers)

    return data_loader


class CoCoDataset(data.Dataset):


    def __init__(self, transform, mode, batch_size, vocab_threshold, vocab_file,
glove_file, start_word,
        end_word, unk_word, annotations_file, vocab_from_file, img_folder):
        self.transform = transform
        self.mode = mode
```

```
        self.batch_size = batch_size
        self.vocab = Vocabulary(vocab_threshold, vocab_file, glove_file, start_word,
            end_word, unk_word, annotations_file, vocab_from_file, dataset='coco')
        self.img_folder = img_folder
        self.sel_length = None

        if self.mode == 'train' or self.mode == 'val':
            self.coco = COCO(annotations_file)
            self.ids = list(self.coco.anns.keys())
            print('Obtaining caption lengths...')
            all_tokens =
[nltk.tokenize.word_tokenize(str(self.coco.anns[self.ids[index]]['caption']).lower())
for index in tqdm(np.arange(len(self.ids)))]
            self.caption_lengths = [len(token) for token in all_tokens]



    def __getitem__(self, index):

        # obtain image and caption if in training/val mode
        if self.mode == 'train' or self.mode == 'val':
            ann_id = self.ids[index]
            caption = self.coco.anns[ann_id]['caption']
            img_id = self.coco.anns[ann_id]['image_id']
            path = self.coco.loadImgs(img_id)[0]['file_name']
            image = Image.open(os.path.join(self.img_folder, path)).convert('RGB')
            image = self.transform(image)
```

```
            # Convert caption to tensor of word ids.
            tokens = nltk.tokenize.word_tokenize(str(caption).lower())
            caption = []
            # forming input tensor
            caption.append(self.vocab(self.vocab.start_word))
            caption.extend([self.vocab(token) for token in tokens])
            caption.append(self.vocab(self.vocab.end_word))
            caption = torch.Tensor(caption).long()

            # return pre-processed image and caption tensors
            return image, caption

        # obtain image if in test mode
        else:

            # Convert image to tensor and pre-process using transform
            PIL_image = Image.open('image.jpg').convert('RGB')
            orig_image = np.array(PIL_image)
            image = self.transform(PIL_image)

            # return original image and pre-processed image tensor
            return orig_image, image

    def get_train_indices(self):
        """In this way we get captures in batch with the same length"""
        sel_length = np.random.choice(self.caption_lengths)
        all_indices = np.nonzero(self.caption_lengths == sel_length)[0]
```

```
        indices = list(np.random.choice(all_indices, size=self.batch_size))
        return indices #, sel_length + 1

    def __len__(self):
        if self.mode == 'train' or self.mode == 'val':
            return len(self.ids)


class InstaDataset(data.Dataset):

    def __init__(self, transform, mode, batch_size, vocab_threshold, vocab_file,
glove_file, start_word,
        end_word, unk_word, annotations_file, vocab_from_file, img_folder):
        self.transform = transform
        self.mode = mode
        self.batch_size = batch_size
        self.vocab = Vocabulary(vocab_threshold, vocab_file, glove_file, start_word,
                                end_word, unk_word, annotations_file, vocab_from_file,
dataset='insta')
        self.img_folder = img_folder
        self.sel_length = None
        if self.mode == 'train' or self.mode == 'val':
            import time
            start = time.time()
            print('Start reading...')
            self.insta = pickle.load(open(annotations_file, 'rb'))
            print('Done: ', time.time()-start)
```

```
        self.ids = list(self.insta.keys())
        print('Obtaining caption lengths...')
        all_tokens = [regex_tokenizer.tokenize(str(self.insta[index]['caption']).lower()) for index in tqdm(self.ids)]
        #all_tokens = [nltk.tokenize.word_tokenize(str(self.insta[index]['caption']).lower()) for index in tqdm(self.ids)]
        self.caption_lengths = [len(token) for token in all_tokens]


    def __getitem__(self, index):
        # obtain image and caption if in training/val mode
        if self.mode == 'train' or self.mode == 'val':
            ann_id = self.ids[index]
            caption = self.insta[ann_id]['caption']
            path = ann_id
            image = Image.open(os.path.join(self.img_folder, path)).convert('RGB')
            image = self.transform(image)

            # Convert caption to tensor of word ids.
            tokens = regex_tokenizer.tokenize(str(caption).lower())
            caption = []
            # Forming input tensor
            caption.append(self.vocab(self.vocab.start_word))
            caption.extend([self.vocab(token) for token in tokens])
            caption.append(self.vocab(self.vocab.end_word))
            caption = torch.Tensor(caption).long()
            return image, caption
```

```
        else:

            PIL_image = Image.open('image.jpg').convert('RGB')
            orig_image = np.array(PIL_image)
            image = self.transform(PIL_image)
            return orig_image, image

    def get_train_indices(self):
        """In this way we get captures in batch with the same length"""
        sel_length = np.random.choice(self.caption_lengths)
        all_indices = np.nonzero(self.caption_lengths == sel_length)[0]
        indices = list(np.random.choice(all_indices, size=self.batch_size))
        return indices #, sel_length + 1

    def __len__(self):
        if self.mode == 'train' or self.mode == 'val':
            return len(self.ids)
```

eval.py

```
import torch
from torchvision import transforms
from data_loader import get_loader
import matplotlib.pyplot as plt
import numpy as np

from model import EncoderCNN, DecoderRNN
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

def clean_sentence(output):
    sentence = ''
    for x in output[1:]:
        if x == 1:
            break
        sentence += ' ' + data_loader.dataset.vocab.idx2word[x]
    return sentence

def get_prediction():

    orig_image, image = next(iter(data_loader))
    #plt.imshow(np.squeeze(orig_image))
    #plt.title('Sample Image')
    #plt.show()
    image = image.to(device)
    features = encoder(image).unsqueeze(1)
    print('Shape of feat: ', features.shape)
    output = decoder.greedy_sample(features)
    sentence = clean_sentence(output)
    print(sentence)



if __name__ == "__main__":
```

```
transform_test = transforms.Compose([
                transforms.Resize(256),
                transforms.RandomCrop(224),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize((0.485, 0.456, 0.406),
                                     (0.229, 0.224, 0.225))])

data_loader = get_loader(transform=transform_test,
                         mode='test',
                         cocoapi_loc='')



embed_size=512
vocab_threshold = 5
hidden_size = 1024



encoder=EncoderCNN(embed_size)
decoder=DecoderRNN(embed_size=512, hidden_size=1024 , vocab_size=8856,
num_layers=1)


if torch.cuda.is_available():
    encoder.load_state_dict(torch.load('./models/encoder-1.pkl'))
    decoder.load_state_dict(torch.load('./models/decoder-1.pkl'))
else:
    encoder.load_state_dict(torch.load('./models/encoder-1.pkl',
```

```
                                    map_location=torch.device('cpu')))
        decoder.load_state_dict(torch.load('./models/decoder-1.pkl',
                                    map_location=torch.device('cpu')))


    encoder=encoder.to(device)
    decoder=decoder.to(device)

    encoder.eval()
    decoder.eval()

    print("Start making prediction ...")

    get_prediction()
```

get_glove_dict.py

```
import bcolz
import pickle
import numpy as np
from tqdm import tqdm

if __name__ == "__main__":

    glove_path = 'glove_txt/'

    words = []
    idx = 0
    word2idx = {}
```

```
    vectors = bcolz.carray(np.zeros(1), rootdir=f'{glove_path}/6B.300.dat', mode='w')


    with open(f'{glove_path}/glove.6B.300d.txt', 'rb') as f:
        for l in tqdm(f):
            line = l.decode().split()
            word = line[0]
            words.append(word)
            word2idx[word] = idx
            idx += 1
            vect = np.array(line[1:]).astype(np.float)
            vectors.append(vect)


    vectors = bcolz.carray(vectors[1:].reshape((400000, 300)),
rootdir=f'{glove_path}/6B.300.dat', mode='w')
    vectors.flush()
    # pickle.dump(words, open(f'{glove_path}/6B.300_words.pkl', 'wb'))
    # pickle.dump(word2idx, open(f'{glove_path}/6B.300_idx.pkl', 'wb'))
    vectors = bcolz.open(f'{glove_path}/6B.300.dat')[:]


    glove = {w: vectors[word2idx[w]] for w in words}
    pickle.dump(glove, open('glove.pkl', 'wb'))
```

instadata_parser.py

```
import json
import pickle
from tqdm import tqdm
```

```
def parse(annotations_file):
    print('Parsing the ', annotations_file)
    insta = json.load(open(path+annotations_file, 'r'))
    parsed_insta={}
    for up_key, up_val in tqdm(insta.items()):
        for sub_key, sub_val in up_val.items():
            parsed_insta.update({up_key+'_@_'+sub_key: sub_val})
    annotations_file = annotations_file[:-4]
    print('Num examples: ', len(parsed_insta))
    return parsed_insta, annotations_file

if __name__ == "__main__":
    path = './captions/annotations/'
    annotations_files = ['insta-caption-train.json',
                         'insta-caption-test1.json',
                         'insta-caption-test2.json']

    parsed_insta, annotations_file = parse(annotations_files[0])
    pickle.dump(parsed_insta, open(path+annotations_file+'pkl', 'wb'))

    parsed_insta, annotations_file = parse(annotations_files[1])
    parsed_insta2, _ = parse(annotations_files[2])

    parsed_insta.update(parsed_insta2)
    pickle.dump(parsed_insta, open(path+annotations_file+'pkl', 'wb'))
```

learner.py

```
import os
import sys
import math
import copy
import torch
import numpy as np
import torch.utils.data as data
from torch.utils.tensorboard import SummaryWriter
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

chencherry = SmoothingFunction()

class Learner():

    def __init__(self, model, criterion, optimizer, dataloader_dict,
        num_epochs, device, stage, hyper_params, scheduler=None, last_epoch=None,
grad_acumulation_step = 1):

        self.model = model
        self.criterion = criterion
        self.optimizer = optimizer
        self.dataloader_dict = dataloader_dict
        self.num_epochs = num_epochs
        self.device = device
        self.stage = stage
        self.hyper_params = hyper_params
        self.scheduler = scheduler
```

```
        self.last_epoch = last_epoch
        self.grad_acumulation_step = grad_acumulation_step
        self.tb = SummaryWriter(comment=self.stage)

    def fit(self):
        # tb.add_graph(model['encoder'])
        # tb.add_graph(model['decoder'])
        best_loss = float('inf')
        train_loss = []
        valid_loss = []
        train_bleu = []
        valid_bleu = []

        for i in range(self.num_epochs):
            print('Epoch {}/{}'.format(i+1, self.num_epochs))
            print('*'*48)
            for phase in ['train', 'val']:

                epoch_dict = self.epoch(phase)

                print('\n{} epoch loss: {:.4f} '.format(phase , epoch_dict['epoch_loss']))
                print('{} epoch metric: {:.4f} '.format(phase , epoch_dict['epoch_bleu']))

                self.tb.add_scalar('{} Epoch Loss'.format(phase.title()),
epoch_dict['epoch_loss'], i)
                self.tb.add_scalar('{} Epoch BLEU'.format(phase.title()),
epoch_dict['epoch_bleu'], i)
```

```
            # probably it is unnessesury lists
            if phase == 'train':
                train_loss.append(epoch_dict['epoch_loss'])
                train_bleu.append(epoch_dict['epoch_bleu'])
            else:
                valid_loss.append(epoch_dict['epoch_loss'])
                valid_bleu.append(epoch_dict['epoch_bleu'])

                model_name = '_{}_{:.2f}_val_{:.2f}_tr_{}.pth'.format(i+self.last_epoch
if self.last_epoch else i,
                                                      valid_loss[-1],
                                                      train_loss[-1],
                                                      self.stage)
                torch.save(epoch_dict['decoder'].state_dict(),
                           os.path.join('./models', 'decoder'+model_name))
                torch.save(epoch_dict['encoder'].state_dict(),
                           os.path.join('./models', 'encoder'+model_name))

            # not implemented loading best model weights callback
            # if phase == 'val' and best_loss>epoch_dict['epoch_loss']:
            #     best_encoder_wts = copy.deepcopy(epoch_dict['encoder'].state_dict())
            #     best_decoder_wts = copy.deepcopy(epoch_dict['decoder'].state_dict())
            #     best_loss = epoch_dict['epoch_loss']
        print()
    print('Batches skipped', epoch_dict['batches_skiped'])
    metrics = {'Train_Loss': train_loss[-1], 'Val_Loss': valid_loss[-1],
```

```
            'Train_Bleu': train_bleu[-1], 'Valid_Bleu': valid_bleu[-1]}

        self.tb.add_hparams(self.hyper_params, metrics)
        self.tb.close()

    def epoch(self, phase):

        encoder = self.model['encoder']
        decoder = self.model['decoder']

        if phase=='train':
            encoder.train()
            decoder.train()
        else:
            encoder.eval()
            decoder.eval()

        running_loss = 0.0
        running_bleu = 0.0
        batches_skiped = 0
        captionss = np.array([])
        raw_preds = np.array([])

        vocab_size = len(self.dataloader_dict[phase].dataset.vocab)
        total_steps = math.ceil(len(self.dataloader_dict[phase].dataset.caption_lengths)\
                                / self.dataloader_dict[phase].batch_sampler.batch_size)
```

```
#optimizer.zero_grad()
for step in range(total_steps):

    indices = self.dataloader_dict[phase].dataset.get_train_indices()
    new_sampler = data.sampler.SubsetRandomSampler(indices=indices)
    self.dataloader_dict[phase].batch_sampler.sampler = new_sampler

    try:
        images, captions = next(iter(self.dataloader_dict[phase]))
    except:
        batches_skiped+=1
        continue

    images = images.to(self.device)
    captions = captions.to(self.device)

    with torch.set_grad_enabled(phase == 'train'):

        features = encoder(images)
        features = features.to(self.device)
        outputs = decoder(features, captions)

        loss = self.criterion(outputs.view(-1, vocab_size), captions.view(-1))

        if phase == 'train':
            loss.backward()
            torch.nn.utils.clip_grad_norm_(decoder.parameters(), 1.0)
```

```
                if (step+1)%self.grad_acumulation_step == 0:
                    self.optimizer.step()
                    self.optimizer.zero_grad()
                # writting weights and grads to tensorboard's histogram
                for name, weight in decoder.named_parameters():
                    self.tb.add_histogram(name, weight, step)
                    self.tb.add_histogram(f'{name}.grad', weight.grad, step)
            elif step%(total_steps//5)==0:
                examples = self.add_examples(captions, outputs, phase)
                self.tb.add_text(f'{phase}:ground_truth/predictions', examples, step)

        running_loss += loss.item() * features.size(0)
        bleu4 = self.compute_metric(captions, outputs)
        running_bleu+=bleu4

        stats = 'Step [{}/{}], Loss: {:.4f}, BLEU-4: {:.4f}'.format(step, total_steps,
loss.item(), bleu4)

        print('\r' + stats, end="")
        sys.stdout.flush()

        self.tb.add_scalar('{} Batch Loss'.format(phase.title()), loss.item(), step)
        self.tb.add_scalar('{} Batch BLEU'.format(phase.title()), bleu4, step)

    epoch_loss = running_loss /
len(self.dataloader_dict[phase].dataset.caption_lengths) #len of the data
    epoch_bleu = running_bleu / total_steps
```

```
        if self.scheduler and phase=='val': self.scheduler.step(epoch_loss)


        epoch_dict = {'batches_skiped':batches_skiped,
                      'epoch_loss': epoch_loss,
                      'epoch_bleu': epoch_bleu,
                      'encoder': encoder,
                      'decoder': decoder}
        return epoch_dict


    def add_examples(self, captions_out, outputs, phase, num_examples=4):
        examples = ''
        for ground_truth, prediction in zip(captions_out[:num_examples,:], outputs[:num_examples,:,:]):
            prediction = torch.argmax(prediction, axis=1)
            truth_text = [self.dataloader_dict[phase].dataset.vocab.idx2word[int(idx)] for idx in ground_truth]
            preds_text = [self.dataloader_dict[phase].dataset.vocab.idx2word[int(idx)] for idx in prediction]
            examples += ' '.join(truth_text) + '/' + ' '.join(preds_text) + '  \n'
        return examples


    def compute_metric(self, captions, preds):
        preds = torch.argmax(preds, axis=2)
        assert preds.shape == captions.shape
        bleu_score = 0.0
        for pred, ground_truth in zip(preds, captions):
```

```
            bleu_score += sentence_bleu([ground_truth.tolist()], pred.tolist(),
smoothing_function=chencherry.method1)
        return bleu_score/len(captions)
```

model.py

```
import torch
import torch.nn as nn
import torchvision.models as models
import numpy as np
import math
import copy
import torch.nn.functional as F

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

class EncoderCNN(nn.Module):
    def __init__(self, embed_size, cnn):
        super(EncoderCNN, self).__init__()
        if cnn == 'resnet101':
            encoder = models.resnet101(pretrained=True)
        elif cnn == 'vgg19':
            encoder = models.vgg19(pretrained=True)
        for param in encoder.parameters():
            param.requires_grad_(False)

        modules = list(encoder.children())[:-1]
        self.encoder = nn.Sequential(*modules)
        if cnn == 'resnet101':
```

```
            self.linear = nn.Linear(encoder.fc.in_features, embed_size)
        elif cnn == 'vgg19':
            self.linear = nn.Linear(encoder.classifier[6].in_features, embed_size)
        self.bn1 = nn.BatchNorm1d(embed_size)

    def forward(self, images):
        features = self.encoder(images)
        features = features.view(features.size(0), -1)
        features = self.linear(features)
        features = self.bn1(features)

        return features



    def freeze_encoder(self):
        for param in self.encoder.parameters():
            param.requires_grad = False

    def unfreeze_encoder(self, num_freezed):
        for l, child in enumerate(self.encoder.children()):
            if l > num_freezed:
                for param in child.parameters():
                  param.requires_grad = True

class DecoderRNN(nn.Module):
    def __init__(self, weights_matrix, hidden_size, vocab_size, num_layers=1,
dropout=0, non_trainable=False):
```

```
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size
        self.num_layers=num_layers
        self.word_embeddings, num_embeddings, embedding_dim =
create_emb_layer(weights_matrix, non_trainable)
        #self.word_embeddings = nn.Embedding(vocab_size, embed_size)
        self.linear = nn.Linear(hidden_size, vocab_size)
        self.lstm = nn.LSTM(input_size=embedding_dim,
                            hidden_size=hidden_size,
                            num_layers=num_layers,
                            dropout=dropout,
                            batch_first=True,
                            bidirectional=False)


    def init_hidden(self, batch_size):
        return torch.zeros(self.num_layers, batch_size, self.hidden_size).to(device), \
                               torch.zeros(self.num_layers, batch_size,
self.hidden_size).to(device)


    def forward(self, features, captions):
        captions = captions[:, :-1]
        self.batch_size = features.shape[0]
        self.hidden = self.init_hidden(self.batch_size)
        embeds = self.word_embeddings(captions)
        inputs = torch.cat((features.unsqueeze(dim=1),embeds), dim=1)
        lstm_out, self.hidden = self.lstm(inputs,self.hidden)
```

```
    outputs=self.linear(lstm_out)
    return outputs

def greedy_sample(self, inputs):
    cap_output = []
    batch_size = inputs.shape[0]
    hidden = self.init_hidden(batch_size)

    max_len = 0

    while True:
        lstm_out, hidden = self.lstm(inputs, hidden)
        outputs = self.linear(lstm_out)
        outputs = outputs.squeeze(1)
        _, max_idx = torch.max(outputs, dim=1)
        cap_output.append(max_idx.cpu().numpy()[0].item())
        if (max_idx == 1):
            break

        inputs = self.word_embeddings(max_idx)
        inputs = inputs.unsqueeze(1)

        max_len += 1
        if (max_len) == 20:
            break

    return cap_output
```

```
    def beam(self, inputs):

        k = 10
        cap_output = []
        batch_size = inputs.shape[0]
        hidden = self.init_hidden(batch_size)

        # generating words next after CNN's vector
        lstm_out, hidden = self.lstm(inputs, hidden)
        outputs = self.linear(lstm_out)
        outputs = outputs.squeeze(1)
        outputs = F.log_softmax(outputs, dim=1)
        top_first_k = torch.topk(outputs, k, dim=1)

        # we will store scores, indexes (in vocab), their embeddings
        # and hiddens states in separate lists and we will use their orders
        scores = top_first_k[0].squeeze(0)
        indexes = top_first_k[1].squeeze(0)
        embeddings = [self.word_embeddings(idx.unsqueeze(0)).unsqueeze(1) for idx in
indexes]

        # hiddens are the same for k generated words but it will more
        # convinient to use them in the same 'style' as other objects
        hiddens = [hidden]*k

        # collecting sentences
```

```
sentences = [[index] for index in indexes]

# startin length of each sentence is 1 now
length = 1

while True:

    length += 1

    current_scores = torch.tensor([])
    current_indexes = torch.tensor([], dtype=int)
    current_hiddens = []

    for i in range(k):

        # we get embedds and hiddens for each child
        h = hiddens[i]
        e = embeddings[i]

        # the same steps
        lstm_out, h_out = self.lstm(e, h)
        outputs = self.linear(lstm_out)
        outputs = outputs.squeeze(1)
        outputs = F.log_softmax(outputs, dim=1)
        top_k = torch.topk(outputs, k, dim=1)

        temp_scores = top_k[0].squeeze(0)
```

```
            temp_indexes = top_k[1].squeeze(0)

            # for each child we add score of their parent score
            temp_scores += scores[i]

            current_scores = torch.cat((current_scores, temp_scores))
            current_indexes = torch.cat((current_indexes, temp_indexes))
            current_hiddens.extend([h_out]*k)


        candidates = torch.topk(current_scores, k)[1] # indexes in arrays for best childs
        best_candidates_indexes = current_indexes[candidates] # indexes in vocab -||-
        best_candidates_scores = current_scores[candidates] # their scores
        best_hiddens = [current_hiddens[candidate] for candidate in candidates]

        scores = best_candidates_scores # updating scores
        indexes = best_candidates_indexes # updating indexes (to generate next words)
        embeddings = [self.word_embeddings(idx.unsqueeze(0)).unsqueeze(1) for idx in indexes]
        hiddens = best_hiddens

        # extending current sentences by new words
        temp = []
        for i, idx in enumerate(candidates):
            sts = copy.deepcopy(sentences[idx//k])
```

```
            temp.append(sts)
            temp[i].append(current_indexes[idx])

        # updatinf current sentences
        sentences = temp

        if length == 20:
            break

    # deviding score to length of the sentence
    normalized_score = []
    for i, score in enumerate(scores):
        try:
            score /= sentences[i].index(1)
        except:
            score /= len(sentences[i]) # if we don't get by generating <eos> token
        normalized_score.append(float(score))



    print('\nMEAN: ', np.mean(normalized_score))
    print('STD: ', np.std(normalized_score))
    print('Normalized score:', max(normalized_score))

    # choosing the best
    best_score = np.argmax(np.array(normalized_score))
    best_sentence = sentences[best_score]
```

```
        # returning truncated sentence
        try:
            best_sentence = best_sentence[:best_sentence.index(1)]
        except:
            best_sentence = best_sentence
        best_sentence =  [int(word) for word in best_sentence]

        return best_sentence



def create_emb_layer(weights_matrix, non_trainable=False):
    num_embeddings, embedding_dim = weights_matrix.shape
    emb_layer = nn.Embedding.from_pretrained(torch.Tensor(weights_matrix), freeze=False)
    if non_trainable:
        emb_layer.weight.requires_grad = False

    return emb_layer, num_embeddings, embedding_dim
```

train.py

```
import os
import sys
import argparse

import nltk
from nltk.translate.bleu_score import sentence_bleu
from data_loader import get_loader
from torchvision import transforms
```

```
import math
import numpy as np

import torch.utils.data as data

import torch
import torch.nn as nn
import torchvision.models as models

from model import EncoderCNN, DecoderRNN
# from train_utils import *
from learner import Learner

# Define a transform to pre-process the training images.
transform_train = transforms.Compose([
    transforms.Resize(256),                          # smaller edge of image resized to 256
    transforms.RandomCrop(224),                      # get 224x224 crop from random location
    transforms.RandomHorizontalFlip(),               # horizontally flip image with probability=0.5
    transforms.ToTensor(),                           # convert the PIL Image to a tensor
    transforms.Normalize((0.485, 0.456, 0.406),      # normalize image for pre-trained model
                         (0.229, 0.224, 0.225))])
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

if __name__  == "__main__":

    nltk.download('punkt')

    parser = argparse.ArgumentParser()
    parser.add_argument("num_epochs", type=int,
                        help="num of epoches")
    parser.add_argument("--dataset", default='coco')
    parser.add_argument("--encoder_lr", type=float)
    parser.add_argument("--decoder_lr", type=float)
    parser.add_argument("--accum_step", type=int, default=1)
    parser.add_argument("--stage", default='default_stage')
    parser.add_argument("--cnn", default='resnet101')
    parser.add_argument("--vocab_from_file", action="store_true")
    parser.add_argument("--hidden_size", type=int, default=768)
    parser.add_argument("--num_layers", type=int, default=2)
    parser.add_argument("--dropout", type=float, default=0.0)
    parser.add_argument("--weight_decay", type=float, default=0.0)
    parser.add_argument("--freeze_glove", action="store_true")
    parser.add_argument("--adam", action="store_true")
    parser.add_argument("--scheduler", action="store_true")
    parser.add_argument("--load_model", action="store_true")
    parser.add_argument("--unfreeze_encoder", type=int)
    parser.add_argument("--batch_size", type=int, default=512)
    args = parser.parse_args()
```

```
embed_size=512
batch_size = args.batch_size
num_layers = args.num_layers
hidden_size = args.hidden_size
dropout = args.dropout
vocab_threshold = 3
vocab_from_file = args.vocab_from_file

train_data_loader = get_loader(transform=transform_train,
                               mode='train',
                               batch_size=batch_size,
                               vocab_threshold=vocab_threshold,
                               vocab_from_file=vocab_from_file,
                               dataset=args.dataset)

val_data_loader = get_loader(transform=transform_train,
                               mode='val',
                               batch_size=batch_size,
                               vocab_from_file=True,
                               dataset=args.dataset)

dataloader_dict = {'train':train_data_loader, 'val': val_data_loader}
vocab_size = len(train_data_loader.dataset.vocab)
weight_matrix = train_data_loader.dataset.vocab.weight_matrix
_, embed_size = weight_matrix.shape
```

```
cnn = args.cnn

encoder=EncoderCNN(embed_size, cnn)
encoder=encoder.to(device)

decoder=DecoderRNN(weight_matrix,
                   hidden_size=hidden_size,
                   vocab_size=vocab_size,
                   num_layers=num_layers,
                   dropout=dropout,
                   non_trainable = args.freeze_glove)

decoder=decoder.to(device)

if args.load_model:
    name = input('Type name of encoder/decoder')
    last_epoch = int(name[1])
    if torch.cuda.is_available():
        encoder.load_state_dict(torch.load('./models/encoder'+name+'.pth'))
        decoder.load_state_dict(torch.load('./models/decoder'+name+'.pth'))
    else:
        encoder.load_state_dict(torch.load('./models/encoder'+name+'.pth',
                                map_location=torch.device('cpu')))
        decoder.load_state_dict(torch.load('./models/encoder'+name+'.pth',
                                map_location=torch.device('cpu')))

num_epochs = args.num_epochs
```

```
    grad_acumulation_step = args.accum_step
    decoder_lr = args.decoder_lr
    encoder_lr = args.encoder_lr
    stage = args.stage
    last_epoch = None

    if args.unfreeze_encoder:
        encoder.unfreeze_encoder(args.unfreeze_encoder)
        encoder_params = []
        for name,param in encoder.named_parameters():
            if param.requires_grad == True:
                encoder_params.append(param)
                print("\t",name)
    else:
        encoder_params = list(encoder.linear.parameters()) +
list(encoder.bn1.parameters())

    criterion = nn.CrossEntropyLoss()

    if args.adam:
        optimizer_name = 'Adam'
        optimizer = torch.optim.Adam(
            [
                {"params":decoder.parameters(),"lr": decoder_lr},
                {"params":encoder_params, "lr": encoder_lr},
            ], weight_decay = args.weight_decay)
    else:
```

```
    optimizer_name = 'ASGD'
    optimizer = torch.optim.ASGD(
        [
            {"params":decoder.parameters(),"lr": decoder_lr},
            {"params":encoder_params, "lr": encoder_lr},
        ], weight_decay = args.weight_decay)

if args.scheduler:
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                           patience=1,
                                                           verbose=True)
else:
    scheduler = None


model = {'encoder' : encoder, 'decoder' : decoder}
hyper_params = {'embed_size':embed_size,
                'batch_size':batch_size,
                'num_layers':num_layers,
                'dropout':dropout,
                'weight_decay': args.weight_decay,
                'optimizer': optimizer_name,
                'hidden_size':hidden_size,
                'cnn':cnn,
                'decoder_lr':decoder_lr,
                'encoder_lr':encoder_lr,
                'grad_acumulation_step':grad_acumulation_step
```

```
                }


    # total_step = math.ceil(len(train_data_loader.dataset.caption_lengths) /
train_data_loader.batch_sampler.batch_size)
    # total_val_step = math.ceil(len(val_data_loader.dataset.caption_lengths) /
val_data_loader.batch_sampler.batch_size)
    learner = Learner(model, criterion, optimizer, dataloader_dict,
                      num_epochs, device, stage, hyper_params,
                      scheduler = scheduler, last_epoch=last_epoch, grad_acumulation_step =
grad_acumulation_step)


    print('Start Training!')
    learner.fit()
```

vocabulary.py

```
import nltk
import pickle
import os.path
import numpy as np
from pycocotools.coco import COCO
from collections import Counter


class Vocabulary(object):


    def __init__(self,
        vocab_threshold,
        vocab_file='./vocab.pkl',
        glove_file='./glove.pkl',
```

```
        start_word="<start>",
        end_word="<end>",
        unk_word="<unk>",
        annotations_file='./cocoapi/annotations/captions_train2014.json',
        vocab_from_file=False,
        embedd_dimention = 300,
        dataset = 'coco'):
        """Initialize the vocabulary.
        Args:
          vocab_threshold: Minimum word count threshold.
          vocab_file: File containing the vocabulary.
          start_word: Special word denoting sentence start.
          end_word: Special word denoting sentence end.
          unk_word: Special word denoting unknown words.
          annotations_file: Path for train annotation file.
          vocab_from_file: If False, create vocab from scratch & override any existing
vocab_file
                           If True, load vocab from from existing vocab_file, if it exists
        """
        self.vocab_threshold = vocab_threshold
        self.vocab_file = vocab_file
        self.glove_file = glove_file
        self.start_word = start_word
        self.end_word = end_word
        self.unk_word = unk_word
        self.num_special_words = 3
        self.annotations_file = annotations_file
```

```
    self.vocab_from_file = vocab_from_file
    self.embedd_dimention = embedd_dimention
    self.dataset = dataset
    self.get_vocab()

def get_vocab(self):
    """Load the vocabulary from file OR build the vocabulary from scratch."""
    if os.path.exists(self.vocab_file) & self.vocab_from_file:
        with open(self.vocab_file, 'rb') as f:
            vocab = pickle.load(f)
            self.word2idx = vocab.word2idx
            self.idx2word = vocab.idx2word
            self.weight_matrix = vocab.weight_matrix
        print('Vocabulary successfully loaded from vocab.pkl file!')
    else:
        self.build_vocab()
        with open(self.vocab_file, 'wb') as f:
            pickle.dump(self, f)

def build_vocab(self):
    """Populate the dictionaries for converting tokens to integers (and vice-versa)."""
    self.init_vocab()
    self.load_glove_dict()
    self.add_captions()
    self.add_word(self.start_word)
    self.add_word(self.end_word)
    self.add_word(self.unk_word)
```

```
    def init_vocab(self):
        """Initialize the dictionaries for converting tokens to integers (and vice-versa)."""
        self.word2idx = {}
        self.idx2word = {}
        self.idx = 0


    def load_glove_dict(self):
        with open(self.glove_file, 'rb') as f:
            self.glove = pickle.load(f)


    def add_word(self, word):
        """Add a token to the vocabulary."""
        if not word in self.word2idx:
            self.word2idx[word] = self.idx
            self.idx2word[self.idx] = word
            try:
                self.weight_matrix[self.idx] = self.glove[word]
            except:
                self.weight_matrix[self.idx] = np.random.normal(scale=0.6, size=(self.embedd_dimention,))
            self.idx += 1


    def add_captions(self):
        """Loop over training captions and add all tokens to the vocabulary that meet or exceed the threshold."""
```

```
        counter = Counter()
        if self.dataset=='coco':
            coco = COCO(self.annotations_file)
            ids = coco.anns.keys()
            data = coco.anns
        elif self.dataset=='insta':
            insta = pickle.load(open(self.annotations_file, 'rb'))
            ids = list(insta.keys())
            data = insta
        for i, id in enumerate(ids):
            caption = str(data[id]['caption'])
            tokens = nltk.tokenize.word_tokenize(caption.lower())
            counter.update(tokens)
            if i % 100000 == 0:
                print("[%d/%d] Tokenizing captions..." % (i, len(ids)))

        words = [word for word, cnt in counter.items() if cnt >= self.vocab_threshold]
        self.weight_matrix =
np.zeros((len(words)+self.num_special_words,self.embedd_dimention))
        for i, word in enumerate(words):
            self.add_word(word)
    def __call__(self, word):
        if not word in self.word2idx:
            return self.word2idx[self.unk_word]
        return self.word2idx[word]
    def __len__(self):
        return len(self.word2idx)
```

ДОДАТОК В

bot.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import telebot
import logging
import ssl
from emoji import emojize

from aiohttp import web

API_TOKEN = 'xxxxx'

WEBHOOK_HOST = 'xx.xx.xx.xx'
WEBHOOK_PORT = 8443  # 443, 80, 88 or 8443 (port need to be 'open')
WEBHOOK_LISTEN = '0.0.0.0'  # In some VPS you may need to put here the IP add
r

WEBHOOK_SSL_CERT = '../ssl/url_cert.pem'  # Path to the ssl certificate
WEBHOOK_SSL_PRIV = '../ssl/url_private.key'  # Path to the ssl private key

# Quick'n'dirty SSL certificate generation:
#
# openssl genrsa -out webhook_pkey.pem 2048
# openssl req -new -x509 -days 3650 -key webhook_pkey.pem -
out webhook_cert.pem
#
# When asked for "Common Name (e.g. server FQDN or YOUR name)" you should r
eply
# with the same value in you put in WEBHOOK_HOST

WEBHOOK_URL_BASE = "https://{}:{}".format(WEBHOOK_HOST, WEBHOOK
_PORT)
WEBHOOK_URL_PATH = "/{}/".format(API_TOKEN)

logger = telebot.logger
telebot.logger.setLevel(logging.INFO)

bot = telebot.TeleBot(API_TOKEN)
```

```
app = web.Application()

# Process webhook calls
async def handle(request):
    if request.match_info.get('token') == bot.token:
        request_body_dict = await request.json()
        update = telebot.types.Update.de_json(request_body_dict)
        bot.process_new_updates([update])
        return web.Response()
    else:
        return web.Response(status=403)

app.router.add_post('/{token}/', handle)


import torch
from torchvision import transforms
from data_loader import get_loader
import numpy as np
from model import EncoderCNN, DecoderRNN

keyboard1 = telebot.types.ReplyKeyboardMarkup(1)
keyboard1.row('Simple Capture', 'Insta Capture')

insta_capture = True
beam_size = 10

start_text = "Hi, I'm a bot that generates captions for pictures, so send me some photo
:)\n\n"\
             "(If after this command you can't see buttons on the keyboard, "\
             "send me this command or something else one more time)\n\n"\
             "To get info about extra feature use command /help"

help_text = "So, You can play with beam search size.\n\n"\
            "It's a hyperparameter that gives you the ability to control the generalizability
of output text\n\n"\
            "The range is from 1 to 10 - higher value can make your capture more generali
ze, "\
            "but lower can produce a poor result, "\
            "so before sending the picture send the number in this range"

@bot.message_handler(commands=['start'])
```

```
def start_message(message):
    bot.send_message(message.chat.id,
                     start_text,
                     reply_markup=keyboard1)

@bot.message_handler(commands=['help'])
def start_message(message):
    bot.send_message(message.chat.id,
                     help_text,
                     reply_markup=keyboard1)


@bot.message_handler(content_types=['text'])
def send_text(message):
    global insta_capture
    global beam_size
    if message.text == 'Simple Capture':
        insta_capture = False
    elif message.text == 'Insta Capture':
        insta_capture = True
    elif message.text.isdigit():
        if int(message.text)>10:
            beam_size = 10
        elif int(message.text)<1:
            beam_size = 1
        else:
            beam_size = int(message.text)
    else:
        bot.send_message(message.chat.id, 'Не зрозумів..', reply_markup=keyboard1)

@bot.message_handler(content_types=['photo'])
def photo(message):

    emoji_dict = {0: ':wink:',
                  1: ':alien:',
                  2: ':fist:'}

    def get_prediction():
        orig_image, image = next(iter(data_loader))
        image = image.to(device)
        features = encoder(image).unsqueeze(1)
        output = decoder.beam(features, k=beam_size)
```

```
    sentence = clean_sentence(output)
    return sentence

def clean_sentence(output):
    sentence = ''
    for x in output[1:]:
        word = data_loader.dataset.vocab.idx2word[x]
        if word == '<end>':
            break
        if word == '<unk>':
            em = np.random.choice(3)
            word = emoji_dict[em]
        sentence += ' ' + word
    return sentence.strip().capitalize()


device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

fileID = message.photo[-1].file_id
file_info = bot.get_file(fileID)

downloaded_file = bot.download_file(file_info.file_path)

with open("image.jpg", 'wb') as new_file:
    new_file.write(downloaded_file)

transform_test = transforms.Compose([
                transforms.Resize(256),
                transforms.RandomCrop(224),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize((0.485, 0.456, 0.406),
                                     (0.229, 0.224, 0.225))])

data_loader = get_loader(transform=transform_test,
                         mode='test')


vocab_size = len(data_loader.dataset.vocab)
embed_size= 300
hidden_size = 768
num_layers = 3
```

```
    encoder=EncoderCNN(embed_size)
    decoder=DecoderRNN(embed_size=embed_size,
                       hidden_size=hidden_size,
                       vocab_size=vocab_size,
                       num_layers=num_layers)

    if insta_capture:
        if torch.cuda.is_available():
            encoder.load_state_dict(torch.load('./models/encoder_insta.pth'), strict=False)
            decoder.load_state_dict(torch.load('./models/decoder_insta.pth'), strict=False)
        else:
            encoder.load_state_dict(torch.load('./models/encoder_insta.pth',
                                    map_location=torch.device('cpu')), strict=False)
            decoder.load_state_dict(torch.load('./models/decoder_insta.pth',
                                    map_location=torch.device('cpu')), strict=False)
    else:
        if torch.cuda.is_available():
            encoder.load_state_dict(torch.load('./models/encoder_coco.pth'), strict=False)
            decoder.load_state_dict(torch.load('./models/decoder_coco.pth'), strict=False)
        else:
            encoder.load_state_dict(torch.load('./models/encoder_coco.pth',
                                    map_location=torch.device('cpu')), strict=False)
            decoder.load_state_dict(torch.load('./models/decoder_coco.pth',
                                    map_location=torch.device('cpu')), strict=False)


    encoder=encoder.to(device)
    decoder=decoder.to(device)

    encoder.eval()
    decoder.eval()

    sentence = get_prediction()
    bot.send_message(message.chat.id, emojize(sentence, use_aliases=True), reply_ma
rkup=keyboard1)



# Remove webhook, it fails sometimes the set if there is a previous webhook
bot.remove_webhook()
```

```
# Set webhook
bot.set_webhook(url=WEBHOOK_URL_BASE + WEBHOOK_URL_PATH,
                certificate=open(WEBHOOK_SSL_CERT, 'r'))

# Build ssl context
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_cert_chain(WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV)

# Start aiohttp server
web.run_app(
    app,
    host=WEBHOOK_LISTEN,
    port=WEBHOOK_PORT,
    ssl_context=context,
)
```

model.py

```
import torch
import torch.nn as nn
import torchvision.models as models
import numpy as np
import copy
import torch.nn.functional as F

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

class EncoderCNN(nn.Module):
    def __init__(self, embed_size):
        super(EncoderCNN, self).__init__()
        resnet = models.resnet101(pretrained=True)
        for param in resnet.parameters():
            param.requires_grad_(False)

        modules = list(resnet.children())[:-1]
        self.resnet = nn.Sequential(*modules)
        self.linear = nn.Linear(resnet.fc.in_features, embed_size)
        self.bn1 = nn.BatchNorm1d(embed_size)

    def forward(self, images):
        features = self.resnet(images)
```

```
        features = features.view(features.size(0), -1)
        features = self.linear(features)
        features = self.bn1(features)

        return features

class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size
        self.num_layers=num_layers
        self.word_embeddings = nn.Embedding(vocab_size, embed_size)
        self.linear = nn.Linear(hidden_size, vocab_size)
        self.lstm = nn.LSTM(input_size=embed_size,
                            hidden_size=hidden_size,
                            num_layers=num_layers,
                            batch_first=True,
                            bidirectional=False)

    def init_hidden(self, batch_size):
        return torch.zeros(self.num_layers, batch_size, self.hidden_size).to(device), \
                               torch.zeros(self.num_layers, batch_size, self.hidden_size).to(devi
ce)

    def forward(self, features, captions):
        captions = captions[:, :-1]
        self.batch_size = features.shape[0]
        self.hidden = self.init_hidden(self.batch_size)
        embeds = self.word_embeddings(captions)
        inputs = torch.cat((features.unsqueeze(dim=1),embeds), dim=1)
        lstm_out, self.hidden = self.lstm(inputs,self.hidden)
        outputs=self.linear(lstm_out)
        return outputs

    def greedy_sample(self, inputs):
        cap_output = []
        batch_size = inputs.shape[0]
        hidden = self.init_hidden(batch_size)

        max_len = 0
```

```
        while True:
            lstm_out, hidden = self.lstm(inputs, hidden)
            outputs = self.linear(lstm_out)
            outputs = outputs.squeeze(1)
            _, max_idx = torch.max(outputs, dim=1)
            cap_output.append(max_idx.cpu().numpy()[0].item())
            if (max_idx == 1):
                break

            inputs = self.word_embeddings(max_idx)
            inputs = inputs.unsqueeze(1)

            max_len += 1
            if (max_len) == 20:
                break

        return cap_output

    def beam(self, inputs, k=10):

        cap_output = []
        batch_size = inputs.shape[0]
        hidden = self.init_hidden(batch_size)

        # generating words next after CNN's vector
        lstm_out, hidden = self.lstm(inputs, hidden)
        outputs = self.linear(lstm_out)
        outputs = outputs.squeeze(1)
        outputs = F.log_softmax(outputs, dim=1)
        top_first_k = torch.topk(outputs, k, dim=1)

        # we will store scores, indexes (in vocab), their embeddings
        # and hiddens states in separate lists and we will use their orders
        scores = top_first_k[0].squeeze(0)
        indexes = top_first_k[1].squeeze(0)
        embeddings = [self.word_embeddings(idx.unsqueeze(0)).unsqueeze(1) for idx in
indexes]

        # hiddens are the same for k generated words but it will more
        # convinient to use them in the same 'style' as other objects
        hiddens = [hidden]*k
```

```
# collecting sentences
sentences = [[index] for index in indexes]

# startin length of each sentence is 1 now
length = 1

while True:

    length += 1

    current_scores = torch.tensor([])
    current_indexes = torch.tensor([], dtype=int)
    current_hiddens = []

    for i in range(k):

        # we get embedds and hiddens for each child
        h = hiddens[i]
        e = embeddings[i]

        # the same steps
        lstm_out, h_out = self.lstm(e, h)
        outputs = self.linear(lstm_out)
        outputs = outputs.squeeze(1)
        outputs = F.log_softmax(outputs, dim=1)
        top_k = torch.topk(outputs, k, dim=1)

        temp_scores = top_k[0].squeeze(0)
        temp_indexes = top_k[1].squeeze(0)

        # for each child we add score of their parent score
        temp_scores += scores[i]

        current_scores = torch.cat((current_scores, temp_scores))
        current_indexes = torch.cat((current_indexes, temp_indexes))
        current_hiddens.extend([h_out]*k)


    candidates = torch.topk(current_scores, k)[1] # indexes in arrays for best child
s
    best_candidates_indexes = current_indexes[candidates] # indexes in vocab -||-
    best_candidates_scores = current_scores[candidates] # their scores
```

```
            best_hiddens = [current_hiddens[candidate] for candidate in candidates]

            scores = best_candidates_scores # updating scores
            indexes = best_candidates_indexes # updating indexes (to generate next words
)
            embeddings = [self.word_embeddings(idx.unsqueeze(0)).unsqueeze(1) for idx
 in indexes]
            hiddens = best_hiddens

            # extending current sentences by new words
            temp = []
            for i, idx in enumerate(candidates):
                sts = copy.deepcopy(sentences[idx//k])
                temp.append(sts)
                temp[i].append(current_indexes[idx])

            # updatinf current sentences
            sentences = temp

            if length == 20:
                break

        # deviding score to length of the sentence
        normalized_score = []
        for i, score in enumerate(scores):
            try:
                score /= sentences[i].index(11495)
            except:
                score /= len(sentences[i]) # if we don't get by generating <eos> token
            normalized_score.append(float(score))


        print('\nMEAN: ', np.mean(normalized_score))
        print('STD: ', np.std(normalized_score))
        print('Normalized score:', max(normalized_score))

        # choosing the best
        best_score = np.argmax(np.array(normalized_score))
        best_sentence = sentences[best_score]

        # returning truncated sentence
        try:
```

```
            best_sentence = best_sentence[:best_sentence.index(11495)]
        except:
            best_sentence = best_sentence
        best_sentence =  [int(word) for word in best_sentence]

        return best_sentence
```

dataloader.py

```
import nltk
import os
import torch
import torch.utils.data as data
from vocabulary import Vocabulary
from PIL import Image
import numpy as np
from tqdm import tqdm
import random
import json

from pathlib import Path


def get_loader(transform,
               mode='train',
               batch_size=1,
               vocab_threshold=None,
               vocab_file='./vocab.pkl',
               start_word="<start>",
               end_word="<end>",
               unk_word="<unk>",
               vocab_from_file=True,
               num_workers=0,
               img_folder = None):

    assert mode in ['train', 'test'], "mode must be one of 'train' or 'test'."
    if vocab_from_file==False: assert mode=='train', "To generate vocab from captions
 file, must be in training mode (mode='train')."

    if mode == 'test':
        assert batch_size==1, "Please change batch_size to 1 if testing your model."
```

```
        assert os.path.exists(vocab_file), "Must first generate vocab.pkl from training dat
a."
        assert vocab_from_file==True, "Change vocab_from_file to True."

    # COCO caption dataset.
    dataset = CoCoDataset(transform=transform,
                          mode=mode,
                          batch_size=batch_size,
                          vocab_threshold=vocab_threshold,
                          vocab_file=vocab_file,
                          start_word=start_word,
                          end_word=end_word,
                          unk_word=unk_word,
                          vocab_from_file=vocab_from_file,
                          img_folder=img_folder)

    if mode == 'train':
        # Randomly sample a caption length, and sample indices with that length.
        indices = dataset.get_train_indices()
        # Create and assign a batch sampler to retrieve a batch with the sampled indices.
        initial_sampler = data.sampler.SubsetRandomSampler(indices=indices)
        # data loader for COCO dataset.
        data_loader = data.DataLoader(dataset=dataset,
                                      num_workers=num_workers,
                                      batch_sampler=data.sampler.BatchSampler(sampler=initial_s
ampler,
                                                                              batch_size=dataset.batch_size,
                                                                              drop_last=False))
    else:
        data_loader = data.DataLoader(dataset=dataset,
                                      batch_size=dataset.batch_size,
                                      shuffle=True,
                                      num_workers=num_workers)

    return data_loader

class CoCoDataset(data.Dataset):

    def __init__(self, transform, mode, batch_size, vocab_threshold, vocab_file, start_
word,
        end_word, unk_word, vocab_from_file, img_folder):
        self.transform = transform
```

```
        self.mode = mode
        self.batch_size = batch_size
        self.vocab = Vocabulary(vocab_threshold, vocab_file, start_word,
            end_word, unk_word, vocab_from_file)
        self.img_folder = img_folder

    def __getitem__(self, index):
        # obtain image and caption if in training mode
        # Convert image to tensor and pre-process using transform
        #PIL_image = Image.open(os.path.join(self.img_folder, onlyfiles[0])).convert('R
GB')
        #PIL_image = Image.frombytes('RGB', (256, 256), self.img_folder)
        PIL_image = Image.open('image.jpg').convert('RGB')
        orig_image = np.array(PIL_image)
        image = self.transform(PIL_image)

        # return original image and pre-processed image tensor
        return orig_image, image

    def __len__(self):
        # as we always work with 1 picture
        return 1
```

vocabulary.py

```
import nltk
import pickle
import os.path
from collections import Counter

class Vocabulary(object):

    def __init__(self,
        vocab_threshold,
        vocab_file='./vocab.pkl',
        start_word="<start>",
        end_word="<end>",
        unk_word="<unk>",
        vocab_from_file=False):
        """Initialize the vocabulary.
        Args:
```

```
        vocab_threshold: Minimum word count threshold.
        vocab_file: File containing the vocabulary.
        start_word: Special word denoting sentence start.
        end_word: Special word denoting sentence end.
        unk_word: Special word denoting unknown words.
        vocab_from_file: If False, create vocab from scratch & override any existing vo
cab_file
                         If True, load vocab from from existing vocab_file, if it exists
        """
        self.vocab_threshold = vocab_threshold
        self.vocab_file = vocab_file
        self.start_word = start_word
        self.end_word = end_word
        self.unk_word = unk_word
        self.vocab_from_file = vocab_from_file
        self.get_vocab()

    def get_vocab(self):
        """Load the vocabulary from file OR build the vocabulary from scratch."""
        if os.path.exists(self.vocab_file) & self.vocab_from_file:
            with open(self.vocab_file, 'rb') as f:
                vocab = pickle.load(f)
                self.word2idx = vocab.word2idx
                self.idx2word = vocab.idx2word
            print('Vocabulary successfully loaded from vocab.pkl file!')
        else:
            print('Load the vocab to directory')

    def __call__(self, word):
        if not word in self.word2idx:
            return self.word2idx[self.unk_word]
        return self.word2idx[word]

    def __len__(self):
        return len(self.word2idx)
```