

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики



**Побудова семантичної моделі зображення за допомогою
машинного навчання**

**Текстова частина до курсової роботи
за спеціальністю “Комп’ютерні науки”**

Керівник курсової роботи

к.ф.-м.н., доц. Глибовець А. М.

(підпис)

“ ____ ” _____ 2020 р.

Виконав

студент I курсу МП

факультету інформатики

Кладько Я.Т.

Київ 2020

Вступ

Можливість знаходити зображення за пошуковим запитом – щоденна проблема інформаційного пошуку. Щодня мільйони людей, шукаючи інформації в інтернеті, прагнуть знайти не лише текстові співпадиння, але й мультимедійну інформацію. Причина виникнення такого пошуку досить проста: є колекція мультимедійних документів та потреба користувача в інформації.

Пошук на основі контенту або ж content-based – ефективний метод пошуку інформації у мультимедійних колекціях документів, ріст яких невинно збільшується. Ці дані зазвичай зберігаються в електронних бібліотеках. Цей спосіб пошуку чудово доповнює традиційний і розширює його можливості, тому є актуальним, зараз стрімко розвивається і точно буде розвиватися у найближчі роки.

Мета дослідження: опанувати архітектурні підходи проектування систем призначених для генерації описів до зображень, розглянути методи машинного навчання, зокрема згорткові та рекурентні нейронні мережі, застосувати набуті знання для генерації опису до фотографій.

Об’єкт дослідження: автоматична генерація людською мовою опису до зображення.

Предмет дослідження: застосування методів машинного навчання з метою створення автоматично генерованого опису зображення на основі аналізу вмісту самих зображень.

РОЗДІЛ 1. Потреба у описі зображень

Проблема пошуку інформації завжди залишається актуальною, перші спроби структурувати швидкий доступ до інформації з'явився майже одночасно з виникненням фізичних пристроїв перенесення інформації – книжок. Найпростіший індекс це зміст у книжці. Можливість розпізнавати потреби користувача та шукати відповідний контент застосовується у багатьох галузях – медичній, туристичній, в бізнес цілях, в розробці систем штучного інтелекту та у багатьох інших. У людей часто з'являється потреба знайти інформацію за певним зображенням, не лише текстовим запитом. Також, це є важливим аспектом комп'ютерного зору, який у поєднанні з програмами штучного інтелекту дає вражаючі результати:

- Автоматичне керування автомобілями, галузь, яка швидко розвивається, якщо комп'ютерна програма матиме можливість розпізнати об'єкти на дорозі, безпілотні автомобілі зможуть з'явитися, як запит суспільства суспільства
- Камери спостереження, зокрема системи схожі до «Безпечного міста», матимуть змогу не лише транслювати, що відбувається у різних областях, за якими ведеться спостереження, а й вчасно самим ідентифікувати небезпечні ситуації і спрямовувати правоохоронців для проведення превентивних заходів
- Туристичні програми, у яких людина за фотографією зможе отримати інформацію про архітектурний об'єкт чи споруду.
- Медична галузь, де система може допомогти пацієнту за його рентгенівським знімком, причому стан пацієнта буде визначений з більшою точністю та швидкістю, це допоможе справити з нестачею кадрів у медичній сфері

Насамперед хочеться нагадати, що розпізнавання зображень – одна з важливих задач комп'ютерного зору. Проблема генерації семантичного опису зображення включає в себе зокрема задачу розпізнавання зображень та обробку мови, і, як результат, дає можливість не витрачати час та людські ресурси для індексації мультимедійного контенту.

Можливості комп'ютерного зору можна перераховувати як завгодно довго, однак потрібно розуміти, які задачі ставить перед собою людина і які з них може вирішити машинне навчання. Одна з таких фундаментальних задач це задача класифікації. Зазвичай у такому випадку існує певна множина об'єктів кожен з яких належить якомусь класу. Причому множина класів скінченна. Також наявності – тренувальна вибірка, яка однозначно встановлює приналежність певного об'єкта якомусь класу. Задача полягає у побудові алгоритму, який на основі тренувальних даних будує модель, яка вміє приймати на вхід незнайоме їй зображення та встановлювати йому у відповідність певний клас. Цей вид машинного навчання носить назву «навчання з учителем», але існує також інший вид навчання «навчання без вчителя», яке передбачає розділ об'єктів на класи, без їх попереднього визначення, однак це вже задача кластеризації, яка не розглядається в цій роботі.

1.2 Нейронні мережі

Відомо, що нейронна мережа – математична модель, складна функція, спроба людей відтворити роботу людського мозку. Таким чином визначається конекціоністський підхід до побудови систем штучного інтелекту, тобто їх моделювання на основі нейронів та зв'язків між ними схожих до людських. Зазвичай такі системи характеризуються дещо повільнішою роботою, ніж звичайні обчислювальні системи, однак вони можуть приймати рішення на основі їхнього визначення подібності об'єктів, тобто їх близькості у побудованому просторі із застосуванням обраної під задачу метрики. Цей принцип лежить в основі людського мислення – встановлення асоціацій між об'єктами, визначення спільних рис між подіями чи явищами. Здатність людини розпізнати та інтерпретувати певне явище зумовлене можливістю нейронів активуватися, коли до них приходить інформація про зовнішній світ та передавати цей сигнал далі, поширюючи його по нервовій системі, внаслідок чого людина може у якийсь спосіб реагувати на зміну середовища.

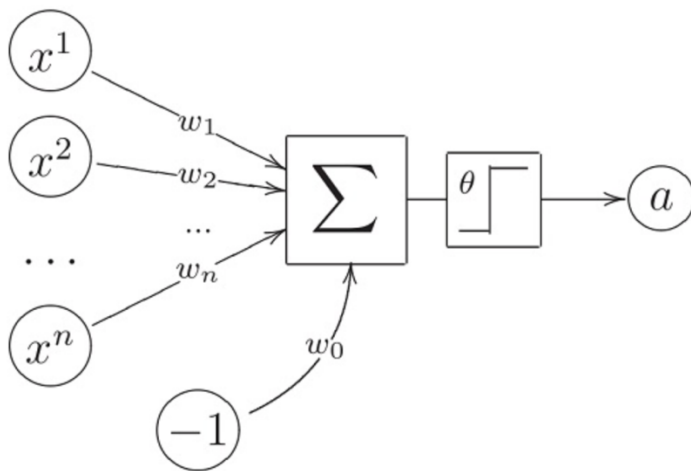
Важливою особливістю системи штучного інтелекту є їхня здатність до ефективного самонавчання. Штучна нейронна мережа – система, яка складається із певної кількості нейронів, які зв'язані між собою. Зрозуміло, що поведінка нейронної мережі, залежить від поведінки кожної з її елементів. Є сенс розглянути штучний нейрон як окрему підсистему. В нього може бути один або кілька входів. На вході у нейрон поступає інформація, яка виражається числами. Існують різні способи перетворення всіх цих чисел у одне, але зазвичай ШІ формує мережевий вхід, при цьому кожне число на вході має свій ваговий коефіцієнт. У моделі, представлений у 1943р. Мак-Каллоком та Пітсом пропонується наступна схема розгляду штучного нейрона:

$$y = f(u)$$

Тобто нейрон відповідальний за перетворення інформації на вході у інформацію на виході, де u визначається у такий спосіб

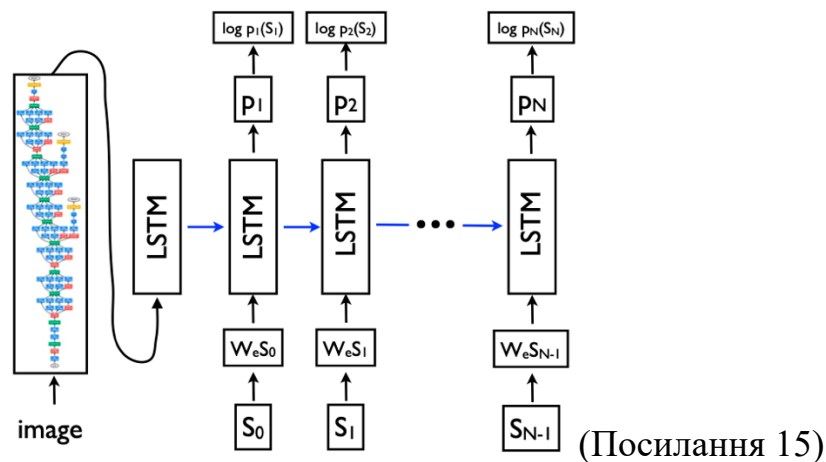
$$u = \sum_{i=1}^n w_i x_i + w_0 x_0$$

При цьому зазначається, що ваги можуть мати як додатні, так і від'ємні значення. На наступному малюнку зображена найпростіша модель штучного нейрона Мак-Каллока Пітса, який виступає в ролі зваженого сумматора. Функція $f(S)$ називається активаційною. У найпростішому випадку вона – порогова, як і зображено нижче, тобто дорівнює 0, якщо $S \leq 0$, та 1, якщо $S > p$, де p – деякий фіксований поріг.



РОЗДІЛ 2. Архітектура «Show And Tell»

Архітектурний підхід, який буде використовуватися для вирішення проблеми генерація підпису до зображення, називається “Show and Tell” або ж “CNN-RNN”. Він передбачає таку послідовність обробки інформації, що на вхід моделі подається зображення, далі це зображення трансформується у вектор ймовірностей за допомогою згорткової нейронної мережі. Після отримання вектора ймовірностей, він подається на вхід до рекурентної нейронної мережі, яка генерує послідовність деяку слів, що описує певне зображення. Архітектура “Show and Tell” може бути представлена в такий спосіб



Ця архітектурна модель реалізує паттерн encoder-decoder, у якому згорткова нейронна мережа – це encoder, а рекурентна – decoder. На цьому зображенні голубі стрілки відображають рекурентні зв'язки архітектури LSTM, суть якої викладена у цій роботі пізніше.

2.1 Згорткові нейронні мережі

Як відомо, комп'ютери теж вміють “бачити”, однак їхній “зір” дещо відрізняється від людського – вони бачать числами. На відміну від людей, зображення для комп'ютера це набір пікселів, кожен з яких описується трьома характеристиками. Три характеристики – це значення кольорів від 0 до 255, якщо мова йде про RGB-зображення, або ж квадратна матриця, якщо зображення чорно-біле. Основним механізмом розпізнавання комп'ютером зображень є згорткова нейронна мережа (Convolutional Neural Network CNN). Отож, що це?

Згорткова нейронна мережа – це один з абстрактних алгоритмів глибокого навчання, математична модель, яка приймає на вхід дані у вигляді зображення. За допомогою фільтрів модель може виокремити із зображення характеристики, які дозволяють однозначно ідентифікувати певний об’єкт. Кожен об’єкт володіє своїм набором характеристик, тому може бути інтерпретований системою як окремий – з легкістю відрізнити kota від собаки. Однією із важливих ознак, про які хочеться нагадати є те, що навчання відбувається автоматично і модель може обробляти дані самостійно, їй не потрібен попередній опис того, з чим вона працює.

Припустимо що в нас є мультимедійний документ – зображення з нашого фотоапарату 1920 на 1080. Наша ідея полягає в тому, щоб дати моделі на вхід масив з чисел, які описують зображення, а на виході отримати ймовірність приналежності моделі до певного класу. Така робота схожа до роботи мозку – ми дивимося на предмет, ідентифікуємо його, та за певним набором характеристик можемо сказати, що це. Модель працює за схожим способом і низькорівневі властивості, наприклад вигини по контуру зображення, може потім перетворити у високорівневі характеристики, наприклад “крило літака”, “лапа собаки”, “колесо автомобіля”. Архітектура класичної нейронної мережі складається з таких пунктів:

- Згортковий шар
- Активаційна функція після кожного згорткового шару
- Шар об’єднання
- Шар зв’язування

2.1.1 Згортковий шар.

Після попередньої обробки зображення, наприклад перетворення його до необхідного розміру, зазвичай це квадратна форма, але необов’язково, зображення подається на вхід згорткового шару. Існує деяке ядро або фільтр, який проходиться по зображенню та виконує операцію згортки. Припустимо, ми розглядаємо певну частину зображення, тобто один із шарів трьохвимірної матриці кольору. Умовно, нехай це буде червоний.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

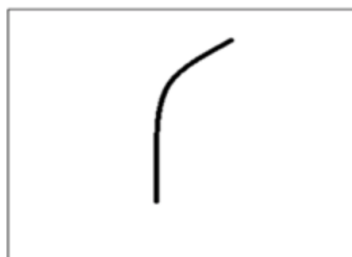
Є менша матриця – це ядро згортки. Після проведення операції згортки ми отримаємо таку матрицю:

4	3	4
2	4	3
2	3	4

Легко бачити, що результат операції – матриця, у якій кожен елемент – це сума всіх елементів з зображення, які співпали з елементами фільтру. Застосування різних фільтрів чи ядер може дати різні результати – наприклад визначення країв на зображенні, розмиття зображення чи навпаки «наведення різкості». Процес «згортки» відбувається для кожного шару з глибини кольору – нагадаю, це червоний, синій та зелений, якщо говорити про RGB-зображення. Після проведення згортки ми отримуємо трьохвимірний фільтр, на кожному шарі якого існує двохвимірна матриця або ж матриця характеристик (посилання).

Раніше вже згадувалося про низькорівневі та високорівневі характеристики. Постає питання як комп'ютеру зрозуміти, що він “бачить” якусь характеристику? Ядра або ж фільтри обробки відповідають за виокремлення характеристик. Припустимо, у нас є такий фільтр, зображення чорно-біле.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0



(Посилання 5)

На другому зображенні візуалізація цього фільтру.

Розглянемо таке зображення. Цілком зрозумілим є те, що у певний момент часу фільтр знайде на зображенні вигин, схожий до своєї характеристики, яку він описує.



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0

Pixel representation of filter

Знову ж таки, результат застосування фільтру до цієї частини зображення – сума добутків чисел, які знаходяться на однакових позиціях у матриці, відповідно, результуюче число буде великим (6600). Якщо цим ж самим ядром пройти по іншій частині зображення, результат суми добутків перетину елементів матриць буде нульовий або у іншому випадку близький до нуля, що характеризує число обробки малим.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0

Pixel representation of filter

Мапа характеристик показує, що у певній частині зображення з високою ймовірністю може знаходитися характеристика, за яку відповідає ядро.

Правило діє у очевидний спосіб, де ми бачимо більші числа – там, швидше за все, і знаходиться ця характеристика. Зрозуміло, що для кожного фільтру необхідно пройти по зображенню і зрозуміти, чи зображена його характеристика. Значення у шарі активаційної матриці визначається як скалярний добуток значення фільтру та значення пікселя у зображенні помножене на кількість каналів, до якого додається корекційний параметр (bias)

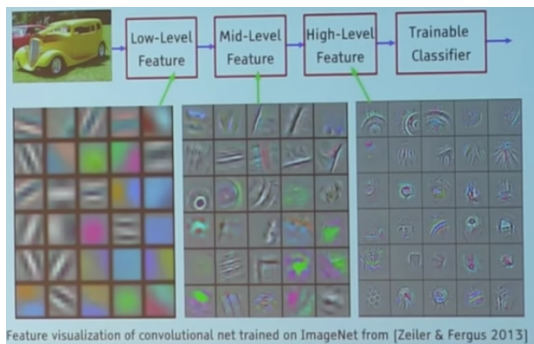
$$S = wt * x + b$$

Для кольорового зображення наприклад 32x32x3 фільтр буде наприклад 5x5x3. Результат матриця характеристик 28x28x1, оскільки існує 28 позицій, куди міг

стати фільтр. Згортковий шар – найбільш складна частина в сенсі обчислень. Цей шар складається з фільтрів, кожен з яких описує свою характеристику. Як результат застосування всіх n фільтрів, ми отримаємо n матриць характеристик, розміром $28 \times 28 \times 1$. Отже, розмір нашого першого згорткового шару буде $n \times 28 \times 28 \times 1$. Нехай $n \times n$ – це розмір зображення, $f \times f$ – розмір фільтра. Якщо не користуватися методом нульової обгортки (zero-padding), розмір матриці активації буде визначатися такою формулою:

$$O = (n - f + 1) * (n - f + 1)$$

Однак часто у нейронній мережі може бути послідовність із кількох згорткових шарів. Фільтри – параметри, які спочатку визначаються випадковим чином і згодом будуть підбиратися все краще з процесом навчання моделі.

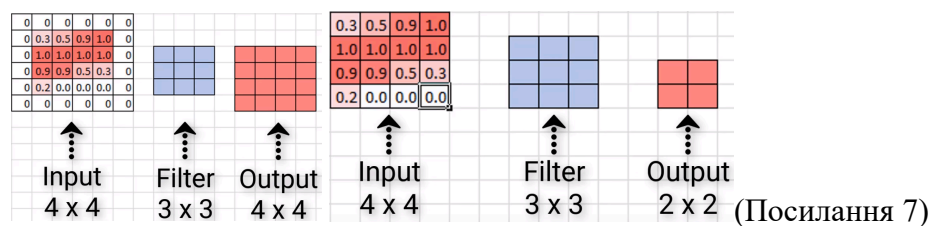


(Посилання 7)

Низькорівневі характеристики починають перетворюватися у високорівневі за рахунок автоматичної корекції значень фільтрів. Автоматична корекція зумовлена алгоритмом зворотнього проходу для навчання моделі. Кожен наступний згортковий шар отримує на вхід матрицю характеристик з попереднього згорткового шару і перемножують її з фільтром, значення якого помінялися, в наслідок чого виходить більш чітке виокремлення певної характеристики. (дописати більш детально?)

Про матрицю характеристик ми можемо думати як про об'єднання 28×28 нейронів, кожен з яких відповідає тільки за маленьку частину зображення і має однакову вагу, в порівнянні з іншими. Варто також згадати про локальну зв'язаність нейронів, суть якої полягає в тому, що на відміну від звичайної нейронної мережі, у згортковій моделі кожен нейрон відповідає тільки за певну частину вхідних даних. Це допомагає знизити кількість параметрів у системі.

Також існує проблема, що при множенні матриці характеристик на фільтр, результат постійно буде зменшуватися, тобто виходить, що певну частину інформації на краях згортки ми будемо втрачати. Ця інформація може бути важливою характеристикою, тому в такому випадку користуються методом нульової обгортки (zero padding). Її суть - в додаванні периметру з нулів навколо матриці характеристик, і в такий спосіб результат множення залишиться такої самої ж розмірності, як і сама матриця. У протилежному випадку розмір активаційної матриці буде зменшуватися.



Проходження через більшу кількість шарів дає можливість знаходити характеристики на більш високому рівні. Із кожним застосуванням фільтру характеристику на зображенні стає видно все краще, і з більшою ймовірністю можна сказати, що вона там присутня.

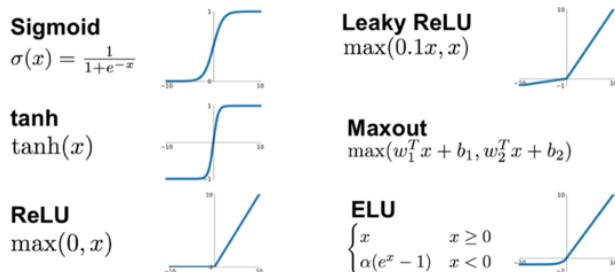
2.1.2 Активаційна функція.

Активаційний шар – наступний крок обробки фільтрів.

“The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.” — [Analytics Vidhya](#)

Популярні активаційні функції:

Activation Functions

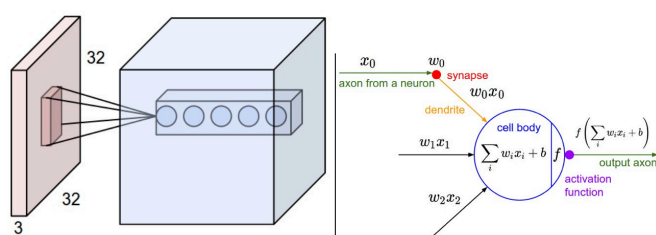


Найбільш часто серед них використовується leaky Rectified Linear Unit (ReLU).

Один з її наріжних каменів це те, що вона не активує одночасно всі нейрони, тобто, як можна бачити з графіку, негативні значенні матриці характеристик вона перетворює на значення близькі до нуля. Це значно пришвидшує

обчислення, в середньому у шість раз, за даними (посилання 6), в порівнянні із сигмоїдною, яка теж була тривалий час дуже популярною. Навідміну від своєї першої версії, тобто звичайної ReLU, вона не перетворює негативні значення матриць характеристик у нулі, що означає, що під час зворотнього проходження, тобто навчання, нульові значення у матриці не будуть коригуватися. Також, важливо відмітити, що ця функція не є центрована навколо нуля (zero-centered), що означає, що для досягнення оптимальної точки під час зворотнього проходження, тобто під час пошуку найкращої конфігурації для ваг, і, як наслідок, найкращих результатів моделі, при градієнтному спуску буде використовуватися зигзагоподібний пошук оптимальної точки, що робить обчислення довшими.

Вимоги до активаційних функцій: вони повинні бути неперервними, тобто у кожній точці можна знайти похідну, також функція повинна бути нелінійною – лінійну використовувати непрактично, в основному вона не прикладна для даних, які зазвичай подаються на вхід нейронній мережі, її областю значень є множина від мінус нескінченності до плюс нескінченності. Похідна ж нам потрібна для визначення як саме міняти значення на фільтрах у пошуку оптимальної точки. Виникає питання чому ж лінійні функції нам не підходять? Об'єкти на зображенні зазвичай мають нелінійні контури. Ціллю використання нелінійної функції є введення нелінійності у нейронну мережу. Нелінійність означає, що вихідні дані не можуть бути отримані за допомогою лінійної комбінації вхідних даних. Тобто, при використанні лінійної функції як активаційної, ми все одно отримаємо одношаровий перцептрон, тому що сума всіх внутрішніх шарів дасть нам лінійне перетворення. В такому випадку наша мережа це не що інше, як звичайне множення матриць.



(Посилання 9)

2.1.3 Шар об'єднання

Цей елемент нейронної мережі передбачає собою функцію зменшення розмірності кількості параметрів у моделі, у такий спосіб запобігаючи перенавчанню. На малюнку нижче представлені перетворення за допомогою Max Pooling-підходу, який використовується значно частіше, у порівнянні з, наприклад, Average Pooling-підходом, та є найбільш ефективним, враховуючи те, що виділяє найсильніше виражені характеристики зображення.

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

->

6	8
3	4

Якщо більш формально, то $W1 \times H1 \times D1$ – розмір вхідного шару, відповідно до позначення ширини, висоти та глибини. Характеризується двома гіперпараметрами F та S , які позначають ширину та висоту фільтру об'єднання. Як вихід передає шар розміром $W2 \times H2 \times D2$, який характеризується в такий спосіб:

$$W2 = (W1 - F) / S + 1$$

$$H2 = (H1 - F) / S + 1$$

$$D2 = D1$$
 - як бачимо, глибина зберігається

2.1.4 Шар зв'язування

На цьому етапі відбувається визначення класів характеристик, які присутні на зображенні. На вхід подається одновимірний вектор характеристик, з шару об'єднання. У цьому векторі знаходяться ймовірності належності характеристики певному класу. Після цього за допомогою softmax-функції визначаються слова, які можуть міститися на зображенні.

2.2 Рекурентні нейронні мережі

Якщо абстрактно, у цій роботі використовується модель, де на вхід приходить зображення I , а сама модель натренована видати результат з максимальною ймовірністю $p(S|I)$, для виведення послідовності $S = \{S_1, S_2\}$, де кожне слово S_t , міститься у словнику, який був складений попередньо. Ця задача, може сприйматися як узагальнення задачі, де послідовність S , написана на мові оригіналу, повинна бути перекладена на потрібну мову T , з ймовірністю $p(T|S)$. Підхід у моделі цієї роботи полягає у використанні глибокої згорткової нейронної мережі, у цьому випадку – Хсертіон, у якості «розкодувальника» (encoder). Відомо, що глибокі згорткові нейронні мережі чудово справляються із задачею представлення зображення як вектора характеристик на ньому. Потім, цей вектор передається у рекурентну нейронну мережу типу Long Short Term Memory (LSTM). Рекурентна нейронна мережа у свою чергу тренується для збільшення точності правильної генерації тексту:

$$\theta^* = \arg \max_{\theta} \sum_{(I,S)} \log p(S|I; \theta) \quad (1)$$

,де тета – це параметри нашої моделі, I – зображення, яке подається на вхід, та S – правильний опис зображення. Враховуючи те, що S – це певна послідовність, у цьому випадку речення, її довжина може бути як завгодно довгою. Тому, широко застосовується саме підхід ланцюгового правила для моделювання ймовірності, для всіх S_0, \dots, S_n , де N – довжина конкретного прикладу.

$$\log p(S|I) = \sum_{t=0}^N \log p(S_t|I, S_0, \dots, S_{t-1}) \quad (2)$$

Під час навчання рекурентна мережа, намагається збільшити суму логарифму ймовірностей, яка представлена у формулі вище шляхом градієнтного спуску. Для моделі $p(S_t|I, S_0, \dots, S_{t-1})$ у рекурентних нейронних мережах, кількість слів ми задаємо числом $t-1$, як фіксовану довжину прихованих станів або пам'яті h_t , де X_t – вхідне значення, а H_t – вихідне значення.

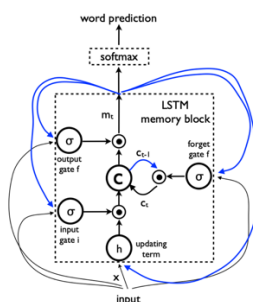
Ця пам'ять оновлюється після отримання нових даних на вхід x_t за допомогою нелінійної функції f :

$$h_{t+1} = f(h_t, x_t) . \quad (3)$$

Для побудови моделі генерації автоматичного опису до зображень, потрібно визначити два наступні елементи: яка конкретно функція f буде використовуватися, та як зображення та слова будуть подаватися на вхід, як вхідні дані X_t . У роботі було використано у ролі f , рекурентну нейронну мережу LSTM, яка добре працює з послідовностями тексту, та глибоку згорткову нейронну мережу для представлення зображень.

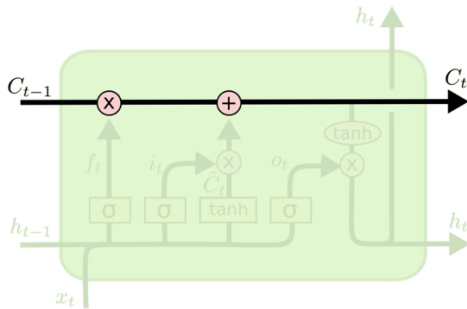
2.2.1 Архітектура LSTM

Модель LSTM було обрано через її можливість справлятися з проблемою зникання градієнту. Проблема полягає у швидкому сповільненні швидкості навчання або його повній зупинці, через надзвичайно мале значення градієнту, і, відповідно, вагові коефіцієнти, які на кожній ітерації отримують уточнення пропорційно до частинної похідної функції похибки, відносно свого значення на попередній ітерації, можуть практично не мінятися. Основа LSTM-моделі полягає у наступній схемі. Блок пам'яті містить підблок C , вплив на який мають три фільтри (gates). Синіми лініями зображені рекурентні зв'язки – вихідні дані m у момент часу $t-1$, подаються на вхід у блок пам'яті у момент часу t через три фільтри. У підблок C подається значення через фільтр «забування», а слово, яке передбачається у час $t-1$, подається на вхід, як додаткові вхідні дані до вихідних даних m , у час t , які передаються у нормовану експоненційну функцію softmax, яка і передбачає слово з найбільшою ймовірністю.



(Посилання 15)

На малюнку нижче показаний стан підблоку пам'яті C – горизонтальна лінія. Її стан можна порівняти з конвеєрною лентою, цей підблок пам'яті або комірка C , проходить через весь ланцюг і бере участь у кількох лінійних перетвореннях. Інформація може бути видалена з комірки C фільтрами, про які вже згадувалося вище. Сам фільтр складається із шару сигмоїдальної нейронної мережі та операції поточкового множення.



(Посилання 16)

Сигмоїдальний шар повертає числа від нуля до одиниці, які означають яку частку інформації з кожного блоку потрібно далі пропустити по мережі. Стан цієї комірки контролюється трьома фільтрами, мова про які піде далі.

Перший крок здійснюється фільтром «забування», він дивиться на h_{t-1} та x_t , і повертає число у проміжку $[0,1]$ для кожного числа з комірки C_{t-1} , де 1 означає повне збереження інформації, а нуль, відповідно, повне її видалення. У нашому випадку генерації тексту до зображення, стан іменника, який знайдений на фотографії, повинен зберігатися до моменту узгодження з дієсловом, після чого інформацію про нього мережа може «забути».

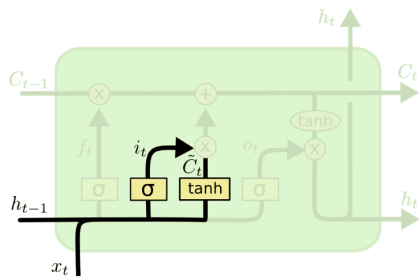
Фільтри «забування» визначається такою формулою:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Далі, мережа визначає, яка саме нова інформація повинна зберігатися у стані комірки. Це розбивається на два етапи: спершу «шар вхідного фільтру»

визначає, які значення потрібно оновити, потім \tanh -шар будує вектор нових

значень кандидатів \tilde{C}_t , які можна додати в стан комірки. Припустимо, що на зображенні було знайдено інший об'єкт, який теж щось робить. При цьому ми додаємо новий іменник, міняючи інформацію про старий.



(Посилання 16)

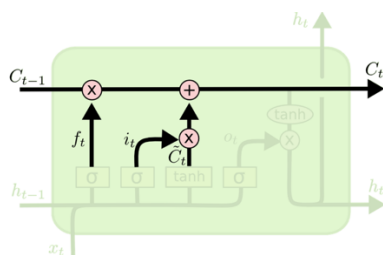
Це виражається такою формулою:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Тепер ми міняємо старий стан комірки C_{t-1} на новий стан C_t . Ми множимо старий стан на f_t (фільтр «забування»), в такий спосіб мережа забуває інформацію, яку було прийнятою вважати непотрібною. Після обчислення результату множення, до отриманого числа додається результат множення

$i_t * \tilde{C}_t$. Це нові значення-кандидати, що помноженні на число t , тобто величину того, на скільки ми хочемо оновити кожне із значень станів.



(Посилання 16)

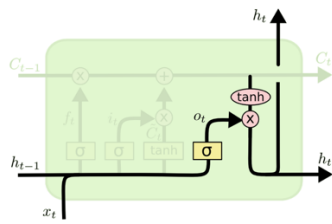
У нашому випадку ця формула описує обчислення на той момент часу, коли ми «забуваємо» інформацію про старий іменник і додаємо інформацію про новий.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

В решті-решт, нам потрібно вирішити, яку інформацію ми хочемо отримати на виході. Ця рішення відбувається внаслідок застосування кількох фільтрів до комірки C – спочатку ми застосовуємо сигмоїдальний шар, який вирішує, яку частину інформації зі стану комірки ми хочемо вивести. Після цього значення із комірки C проходять через \tanh -шар, який видає на виході значення в діапазоні $[-1;1]$, в результаті чого значення з \tanh -шару та сигмоїдального шару перемножуються, залишаючи тільки ту інформацію, яка необхідна. У нашому випадку це може бути інформація про кількість дітей, які граються на пляжі, з

метою акценту на множині, щоб потім застосувати правильну форму дієслова.

Схематично це виглядає так:



(Посилання 16)

І описується такими формулами:

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

LSTM-модель тренується передбачати кожне слово з речення після того, як вона побачила зображення у представленні вектора ймовірностей

$p(S_t | I, S_0, \dots, S_{t-1})$. LSTM, як рекурентна нейронна мережа, створює свою окрему копію для кожного слова з послідовності і передає туди значення h_{t-1} , обчислене у момент часу $t-1$. Якщо у загальному представити зображення як $I, s = (S_0, \dots, S_N)$ - речення, які описують кожне зображення, то процедурі розгортки рекурентної нейронної мережі передаються такі значення:

$$x_{t-1} = \text{CNN}(I);$$

$$x_t = W_e S_t, t \in \{0 \dots N-1\};$$

$p_{t+1} = \text{LSTM}(x_t), t \in \{0 \dots N-1\}$; де ми представляємо кожне слово, як вектор S_t , а W_e – векторне представлення слів із словника. Варто відмітити, що S_0 та S_N – це спеціальні слова, які позначають початок та кінець послідовності, зокрема останнє використовується для позначення того, що послідовність згенерована.

Функція втрат для LSTM визначається формулою $L(I, S) = - \sum_{t=1}^N \log p_t(S_t)$ - від'ємне значення логарифму ймовірності підбору коректного слова для опису на кожному кроці.

РОЗДІЛ 3. Реалізація

3.1 Інструменти реалізації

В якості мови програмування був обраний Python3.8. Із залучення розроблених для машинного навчання бібліотек, розробка на цій мові полегшує і спрощує багато процесів. Як показує практика у машинному навчанні, як і будь-якій іншій області, дуже багато підводних каменів, тому за допомогою бібліотечних функцій написання коду суттєво спрощується. Основні бібліотеки, які використовувалися у цій роботі – Keras і Tensorflow.

Про кожну з них детальніше: Keras – дуже популярна бібліотека із відкритим вихідним кодом для глибокого навчання та швидкої побудови нейронних мереж. Вона надає простий інтерфейс для створення різноманітних моделей. Для створення нейронної мережі Keras можна використати два типи моделей – послідовні і з використанням функційного API, у даному випадку використовувався функційний. Після обрання одного з методів потрібно додати вхідні, приховані та вихідні шари та задати деякі параметри, зокрема dropout, який захищає модель від перенавчання. Також на кожному шарі використовується функція dense для з'єднування шарів між собою. Також один із параметрів, це функція активації. Як показала практика при побудові згорткової нейронної мережі, зазвичай є сенс використовувати Rectified Linear Unit, однак у Keras також є можливість використати LeakyReLU з модуля `tf.keras.layers`, хоча вона і не йде як базова активаційна функція класу Dense.

Tensorflow – фреймворк для машинного навчання. Її робота базується на графі обчислень, конструкції, яка описує як саме будуть проводитися обчислення. Всі обчислення відбуваються у сесії. Її об'єкт містить у собі всі змінні та ресурси, необхідні для виконання коду. Tensorflow складається з двох частин - задання структури порядку обрахунків і виконання частин коду у попередньо визначених структурах.

Також використовувалися інші бібліотеки Python, зокрема numpy – бібліотека для роботи з багатовимірними масивами, Pillow (стара назва PIL) – бібліотека для роботи із зображенням, pickle – бібліотека для серіалізації об'єктів та деякі інші. Розробка велась у середовищі Jupyter Notebook та Visual Studio Code.

3.2 Набір даних та моделі

Для тренування моделей було обрано датасет Flickr_8k. Він включає у себе набір з восьми тисяч фотографій та по п'ять підписів для кожної фотографії англійською мовою. У якості тренувальної вибірки було використано шість тисяч зображень, у якості тестової – дві тисячі. Це не найбільший набір даних, яким можна було скористатися, однак був обраний саме він.

Як згадувалося раніше, для побудови архітектури «Show and Tell» потрібно було скористатися згортковою та рекурентною нейронними мережами. Вибір згорткової нейронної мережі впав на мережу Xception. Ця модель базується на архітектурному підході Inception. Його суть полягає в тому, щоб замість одного розміру фільтра, наприклад, 3×3 , зразу використати декілька, наприклад ще 5×5 і 1×1 , останній для зменшення розміру сигналу, який передається у наступний шар, оскільки кількість операцій суттєво зростає, якщо замість одного фільтра обчислювати декілька. Якщо ми подаємо на вхід згорткового шару частину зображення розміром $W \times H \times C$, відповідно до ширини, висоти та кількості каналів, і маємо звичайний фільтр розміром 3×3 . Після застосування фільтра, всі канали вихідного сигналу будуть згорнуті різними фільтрами, виходить активаційна матриця розміром $(W - 2) * (H - 2) * C^2$. Натомість, ця модель згортає всю матрицю точковим фільтром 1×1 , в результаті отримується матриця розміром $W * H * C^2$. Далі ми згортаємо кожен канал по черзі, навідміну від класичної згортки, фільтром розміру 3×3 , при цьому вихідна розмірність не міняється, як при звичайній згортці. Чому це

пришвидшує роботу? Це зменшує кількість ваг. Якщо у нас буде 16-канальне зображення і 32 фільтри, сумарно згортковий шар буде складатися з $16 * 32 * 3 * 3 = 4608$ ваг. У підході Inception в нас залишається тільки $16 * 32$ ваг, враховуючи те, що розмір фільтру точковий – 1×1 . Також, однією з переваг Xception було те, що вона може обробляти зображення більшої, у порівнянні з конкурентами, розмірності – 299×299 пікселів. У процесі розробки, модель використовувалася для отримання характеристик із зображень, з подальшої серіалізацією їх файл. У наведеному лістингу наведена функція отримання вектору характеристик із зображень, які знаходяться у директорії. Тут також вказується використання функції max-pooling, що означає, що лише найбільші значення на у шарі об'єднання будуть відбиратися. Ми видаляємо останній класифікаційний шар і, як результат, модель повертає 2048-вимірний вектор характеристик. Функція get_features() витягує характеристики для всіх зображень та серіалізує мапу, яка містить ім'я файлу та вектор характеристик у файл.

```
def get_features(img_dir):
    model = Xception(include_top=False, pooling='max')
    features = {}
    for item in tqdm(os.listdir(img_dir)):
        filename = img_dir + "/" + item
        img = Image.open(filename)
        img = img.resize((299,299))
        img = np.expand_dims(img, axis=0)
        img = img/127.5
        img = img - 1.0
        feature = model.predict(img)
        features[item] = feature
    return features
features = get_features(dataset_images)
dump(features, open("features_file_512.p", "wb"))
```

Попередньо ми проводимо парсинг файлу з описами зображень, складаючи їх всіх у мапу «ім'я зображення – масив описів». Функція text_cleanup() чистить описи від «'s» та “-” символів, перетворює слова у слова у нижньому регістрі, видаляє знаки пунктуації та видаляє номери з описів. Вона представлена у наступному лістингу.

```
def text_cleanup(descr):
    table = str.maketrans(' ', '', string.punctuation)
    for img, caps in descr.items():
        for i, img_d in enumerate(caps):
            img_d.replace("-", " ")
            description = img_caption.split()
            description = [word.lower() for word in description]
            description = [word.translate(table) for word in description]
            description = [word for word in description if (len(word)>1)]
            description = [word for word in description if (word.isalpha())]

            img_d = ' '.join(desc)
            descr[img][i] = img_d
    return descr
```

У лістингу нижче показане об'єднання двох моделей у архітектуру «Show And Tell», визначаються параметри, призначення яких описувалося вище. Тут ми додаємо ще один шар, на якому визначаємо активаційною функцією Rectifier Linear Unit, на виході з цього шару у цьому прикладі ми отримуємо тільки 512-вимірний вектор характеристик. Раніше для інших моделей використовувалися також 256-вимірні вектори та dropout = 0.5.

```
def model_definition(voc_lenght, max_seq_lenght):
    cnn_inputs = Input(shape=(2048,))
    cnn_drop = Dropout(0.7)(cnn_inputs)
    cnn_dense = Dense(512, activation='relu')(cnn_drop)

    lstm_inputs = Input(shape=(max_seq_lenght,))
    lstm_embedding = Embedding(vocab_size, 512, mask_zero=True)(lstm_inputs)
    lstm_propout = Dropout(0.7)(lstm_embedding)
    lstm = LSTM(512)(lstm_propout)

    decoder1 = add([cnn_dense, lstm])
    decoder2 = Dense(512, activation='relu')(decoder1)
    outputs = Dense(voc_lenght, activation='softmax')(decoder2)

    model = Model(inputs=[cnn_inputs, lstm_inputs], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model
```

Код запуску тренування, де voc_lenght – кількість слів у словнику, який зберігається як серіалізована колекція, max_seq_lenght – максимальна довжина терміну в колекції (обмежена значенням 32). Ця модель тренувалася протягом 5 епох. Створюється генератор або ж об'єкт класу Sequence, який передається у в одну з трьох функцій бібліотеки Keras, яка тренує отримані дані.

```
model = model_definition(voc_lenght, max_seq_lenght)
epochs = 5
train_len = len(descr_train)
os.mkdir("custom_models_512")
for i in range(epochs):
    generator = create_generator(descr_train, features_train, tokenizer, max_seq_lenght)
    model.fit_generator(generator, epochs=1, steps_per_epoch=train_len, verbose=1)
    model.save("custom_models_512/model_" + str(i) + ".h5")
```

Висновки

В ході виконання курсової роботи розглядалася архітектура «Show and Tell», також були розглянуті принципи роботи та побудови згорткових нейронних мереж та рекурентні нейронні мережі. Як відомо, згорткові чудово підходять для розпізнавання об'єктів на зображенні, а рекурентні – для генерації та роботи з послідовностями. Для експерименту було створено три моделі, результати яких порівнювалися – перша тренувалася 10 епох та мала задекларовану похибку у 2.726, що є досить непоганим результатом. Деколи їй вдавалося розпізнати та детально описати вміст зображення, однак досить часто результат був не таким, який б хотілося отримати – опис до зображення був сформульований некоректно. З цим все ж краще справлялася друга модель, час її тренування був більшим – 20 епох, а похибка складала 2.5740. Для обох цих моделей розмір вихідного вектору зі згорткової нейронної мережі складав 256 значень, та dropout був 0.5. Модель після 20 епохи було протестовано на 10 зображеннях, серед яких тільки у 3-ох підпис був коректний. Приклади наведені нижче. Видається, що модель добре генерувала підпис для яскравих та «типових» зображень – як-от «собака, який біжить у траві» або частково коректно «діти граються на пляжі». Зазвичай це контрастні зображення, на них можна було помітити кращі результати. У випадку зображення з групою людей, модель могла помітити частково «людину у голубій сорочці», однак це було єдине, що відповідало семантиці зображення, решта підпису була некоректною. Схожий результат спостерігався з більшістю зображень. В середньому на 10 зображень було 2-3 зображення, на яких підпис повністю або майже повністю відображав зміст зображення, приблизно ще 3 зображення, де згенерований текст хоча б якось стосувався зображення, наприклад що на зображенні «існує чоловік» або на зображенні «вулиці», і в середньому ще 4 зображення, де підпис генерувався взагалі неправильно. Для тесту була створена ще одна модель, яка використовувала 512-розмірний вектор характеристик. Її тренування було значно довшим, однак вже за 4 епохи вона показала більш точний результат під час тестів. Також, як підбір параметрів для її шару об'єднання було встановлено pooling='max', що,

на мою думку, і дало цей більший приріст у точності, оскільки характеристики відбиралися більш чітко і не розмивали усередненням. Це демонструє приклад із дітьми на пляжі, де попередні дві моделі не змогли розпізнати кількох дітей, що зробила третя. Також важливу роль відіграв більший вектор характеристик, який, зрозуміло, підвищив точність. Також параметр dropout, який пропускав більше інформації у третій моделі, суттєво збільшив кількість параметрів, які довелося тренувати, тобто ваг. Trainable params: 5,002,649 у другій моделі, на відміну від Trainable params: 11,177,369 у другій. Це збільшило час тренування, але заодно покращилася і точність опису. Якість створення підписів можна покращити емпіричним шляхом тестування моделей із різними параметрами та збільшенням розміру навчальної вибірки.

Частково правильне розпізнання:

Перша модель:

Boy is walking along the beach

Друга модель:

Boy is black swimsuit is walking along the beach

Третя модель:

Two children are playing in the water



Повністю неправильне розпізнання:

Перша модель:

Two children are playing on the carpet

Друга модель:

Two children are playing on the carpet

Третя модель:

Boy is standing in front of building



Повністю правильне розпізнання:

Перша модель:

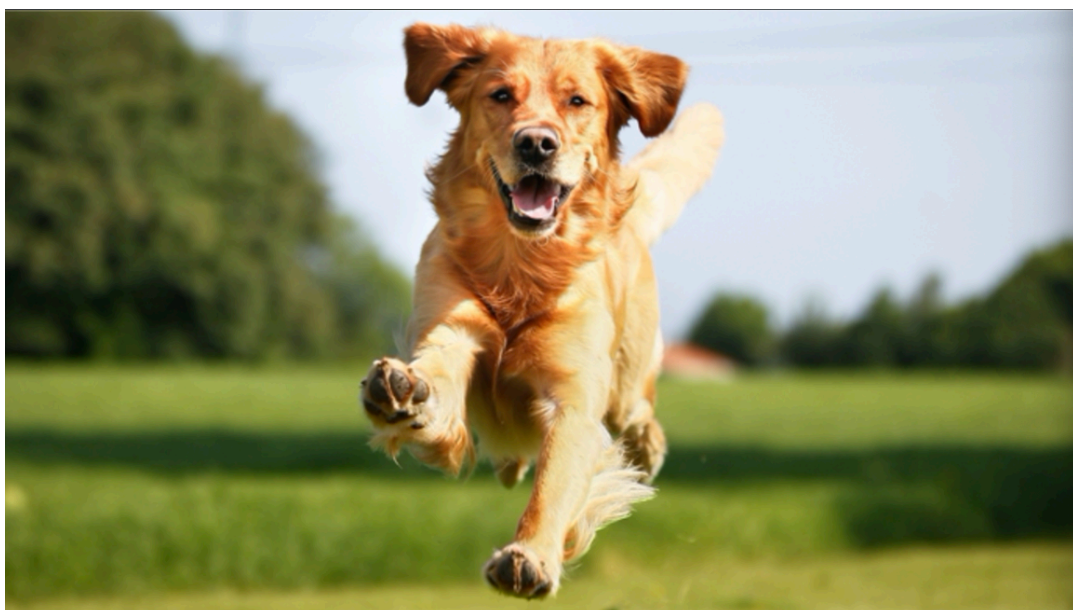
Dog is running through the grass

Друга модель:

Dog is running through the grass

Третя модель:

Dog is running through the grass



Список використаної літератури:

1. <https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8> [Електронний ресурс]
2. <https://data-flair.training/blogs/convolutional-neural-networks-tutorial/> [Електронний ресурс]
3. <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add> [Електронний ресурс]
4. <https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8> [Електронний ресурс]
5. https://deeplizard.com/learn/video/qSTv_m-KFk0 [Електронний ресурс]
6. <https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17> [Електронний ресурс]
7. <https://stats.stackexchange.com/questions/237169/why-are-non-zero-centered-activation-functions-a-problem-in-backpropagation> [Електронний ресурс]
8. <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> [Електронний ресурс]
9. <https://cs231n.github.io/convolutional-networks/> [Електронний ресурс]
10. <https://arxiv.org/pdf/1502.03044.pdf> [Електронний ресурс]
11. <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks> [Електронний ресурс]
12. <https://nplus1.ru/material/2016/11/04/recurrent-networks> [Електронний ресурс]
13. https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg [Електронний ресурс]
14. https://www.researchgate.net/figure/Model-architecture-of-Show-and-Tell-image-captioning-system-22-The-tokens-in-green-and_fig4_311299402 [Електронний ресурс]
15. <https://arxiv.org/pdf/1609.06647.pdf> [Електронний ресурс]
16. <https://habr.com/ru/company/wunderfund/blog/331310/> [Електронний ресурс]
17. <http://datareview.info/article/issleduem-lstm-seti-chast-1/> [Електронний ресурс]
18. Глибовець М. М. Штучний інтелект / М. М. Глибовець, О. В. Олецький. – Київ: Видавничий дім "КМ Академія", 2002. – 358 с.

19. <https://www.bioinf.jku.at/publications/older/2604.pdf> [Электронный ресурс]
20. http://www.machinelearning.ru/wiki/index.php?title=Модель_МакКаллока-Питтса [Электронный ресурс]