Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Нейромережеві моделі класифікації зображень
(тема)

Виконав:
студент II курсу, групи СПм-18-3
Шупилюк М.В.
(прізвище, ініціали)

Спеціальність
123 – Комп’ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Ільїна І.В
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ ______ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп’ютерна інженерія
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________________
(підпис)
“____” ________________ 20__ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Шупилюку Микиті Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Нейромережеві моделі класифікації зображень

затверджена наказом по університету від “ 30 ” березня 2020 р. № 478 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2020 р.

3. Вхідні дані до роботи
   1) наукова література на тему нейронних мереж
   2) сучасні архітектури згортальних нейронних мереж
   3) особливості організації попередньої обробки зображень

4. Перелік питань, що потрібно опрацювати в роботі
   Аналіз предметної області
   Попередня підготовка та вибір набору даних
   Розбір основних методів класифікації зображень
   Порівняння методів та формування рекомендації
   Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (слайдів)

Демонстраційні матеріали. Плакати – 10 арк. ф. А4

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасного стану проблеми	31.03.20 – 07.04.20	
2	Постановка задачі дослідження	07.04.20 – 10.04.20	
3	Аналіз літератури	10.04.20 – 21.04.20	
4	Пошук методів класифікації	22.04.20 – 26.04.20	
5	Порівняння знайдених методів	27.04.20 – 02.05.20	
6	Підсумок проведенної роботи	03.05.20 – 05.05.20	
7	Оформлення пояснювальної записки	06.05.20 – 12.05.20	
8	Підготовка до захисту	13.05.20 – 26.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент ____________________
(підпис)

Керівник роботи ____________________ доц. Ільїна І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 66 с., 30 рис., 3 табл., 1 дод., 17 джерел.

НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, INCEPTION, RESNET, MOBILENET, CIFAR10

Метою атестаційної роботи є отримання знань про те, як сучасні нейромережеві моделі справляються з проблемою класифікації зображень, які переваги дає попередня обробка зображень та які для цього використовують коефіцієнти та типи згорток. Розглянуті існуючі набори даних на яких проходить тренування нейронної мережі, є дуже важливим фактором при аналізі якості нейронних мереж та впливає на кінцевий результат.

У ході виконання атестаційної роботи були розглянуті існуючі нейромережеві моделі класифікації зображень, проаналізовані їх переваги та недоліки і вироблені рекомендації для покращення цих моделей.

ABSTRACT

Master's thesis: 66 pages, 30 figures, 3 tables, 1 appendices, 17 sources.

NEURAL NETWORKS, IMAGE CLASSIFICATION, INCEPTION, RESNET, MOBILENET, CIFAR10

The major goal of this thesis is to gain knowledge about how modern neural network models cope with the problem of image classification, what are the advantages of image pre-processing and what are the coefficients and types of convolutions. The considered existing data sets on which the neural network is trained are a very important factor in the analysis of the quality of neural networks and affect the final result.

During the attestation work, the existing neural network models of image classification were considered, their advantages and disadvantages were analyzed, and recommendations were made to improve these models.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7
ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Нейронна мережа, поняття та принцип роботи 9
1.2 Типи нейронних мереж 15
1.3 Глибинні нейронні мережі 20
1.3.1 Багатошаровий перцептрон 20
1.3.2 Згортальна нейронна мережа (CNN) 21
1.3.3 Рекурсивна нейронна мережа (Recursive neural network) 21
1.3.4 Рекурентна нейронна мережа (Recurrent neural network) 22
1.3.5 Довга короткострокова пам’ять (Long short-term memory) 23
2 ПІДГОТОВКА ДАНИХ 25
2.1 Попередня обробка зображень 25
2.2 Вибір та підготовка набору даних 30
3 ОПИС АЛГОРИТМІВ 34
3.1 Архітектура Inception_v1 35
3.2 Архітектура ResNet 40
3.3 Архітектура MobileNet_v1 43
4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ 47
4.1 Вибір технологій для реалізації програми 47
4.1.1 Вибір мови програмування 47
4.1.2 Вибір бібліотеки для машинного навчання 48
4.2 Аналіз результатів експериментів 52
ВИСНОВКИ 58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 59
ДОДАТОК А Графічний матеріал атестаційної роботи 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – штучний інтелект (англ., Artificial intelligence)

CNN – Згортальні нейронні мережі (англ., Convolutional Neural Networks)

GAP – Глобальне усереднене об’єднання (англ., Global Average Pooling)

ML – машинне навчання (англ., Machine Learning)

ReLU – (англ., Rectified Linear Unit)

RNN – Рекурентна нейронна мережа (англ., Recurrent Neural Network)

ВСТУП

Більшу кількість інформації про навколишній світ людина сприймає за допомогою зору. Комп’ютер зберігає і відображає таку інформацію за допомогою зображень. Кількість інформації, що зберігається за допомогою зображень, в сучасному світі, зростає щомиті, що призвело до появи таких понять як «Великі дані» і «Комп’ютерний зір». Джерелами інформації можуть бути як соціальні мережі так і внутрішня інформація підприємств і організацій зі сфер медицини та біоінформатики або з астрономічних спостережень. Різні техніки і методи аналізу, які застосовні до великих даних, включають в себе машинне навчання, імітаційне моделювання, штучні нейронні мережі, прогнозова аналітика, а також розпізнавання образів, просторовий аналіз, статистичний аналіз, краудсорсінг, методи класу Data Mining: класифікація, кластерний аналіз , регресійний аналіз.

Розпізнавання образів являє собою розділ, що містить методи класифікації та ідентифікації предметів, явищ, процесів, сигналів, ситуацій, які охарактеризовані деяким набором властивостей і ознак. Проблема розпізнавання образів зустрічається повсюдно, будь-то знаходження зондами або супутниками географічних об’єктів на Землі та в космосі, або діагностування захворювань за допомогою результатів з рентгенології, ультразвукоскопії, інфраскопії (обробка двох – і тривимірних зображень внутрішніх органів, аналіз розподілу температур тіла, томографія). Також можна згадати розпізнавання осіб, яке можна застосувати для ідентифікації особистості, виявлення людей і злочинців по знімках і по камерам відеоспостереження. Не випадково проблема розпізнавання образу входить в поле міждисциплінарних досліджень - в тому числі в зв’язку з роботою зі створення штучного інтелекту, а створення технічних систем розпізнавання образу привертає до себе все більшу увагу.

Розпізнавання образів може відбуватися за допомогою так званого комп’ютерного зору. Цей термін являє собою технологію створення штучних систем, які отримують інформацію з зображень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Нейронна мережа, поняття та принцип роботи

Під нейронними мережами маються на увазі обчислювальні структури, які моделюють прості біологічні процеси, які зазвичай асоціюються з процесами людського мозку. Вони являють собою розподілені і паралельні системи, здатні до адаптивного навчання шляхом аналізу позитивних і негативних впливів. Елементарним перетворювачем в даних мережах є штучний нейрон або просто нейрон, названий так за аналогією з біологічним прототипом.

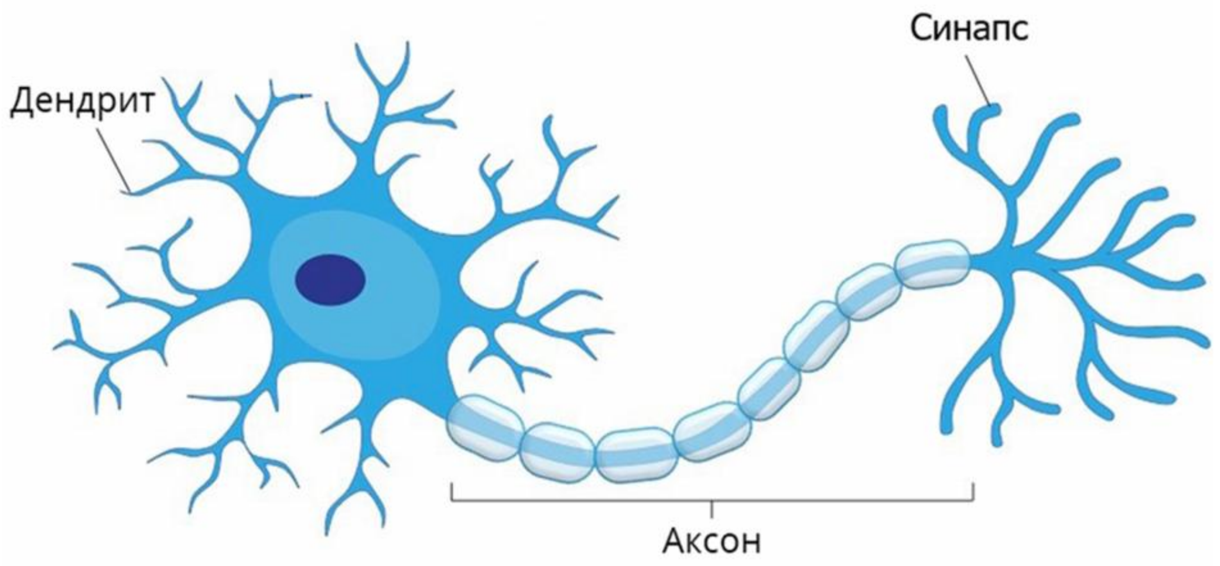

Рисунок 1.1 – Модель людського нейрону

Центральна нервова система має клітинну будову. Одиниця – нервова клітина, нейрон. Він складається з тіла і відростків, які з’єднують його з зовнішнім світом (рисунок 1.1). Відростки, за якими нейрон отримує збудження, називаються дендритами. Відросток, по якому нейрон передає збудження, називається аксон, причому аксон у кожного нейрона один. Дендрити і аксон мають досить складну гіллясту структуру. Місце з’єднання аксона нейрона - джерела збудження з дендритом називається синапсом. Основна функція нейрона полягає в передачі збудження з дендритів в аксон. Але сигнали, які надходять з різних дендритів, можуть впливати на сигнал в аксоні. Нейрон передає сигнал, якщо сумарне збудження перевищить деяке граничне значення, яке в загальному випадку змінюється в деяких межах. В іншому випадку на аксон сигнал виданий не буде: нейрон не відповість на збудження. У цієї основної схеми багато ускладнень і винятків, проте більшість нейронних мереж моделюють саме ці прості властивості.

В нейронній мережі, нейрон – це одиниця обробки інформації. В основі штучних нейронних мереж модель нейрона має три основних елемента:

a) набір синапсів або зв’язків, кожен з яких характеризується своєю вагою (weight) або силою (strength). На відміну від синапсів мозку синаптична вага штучного нейрону може приймати як позитивні, так и негативні значення;

b) суматор (adder) додає вхідні сигнали, звішені згідно з відповідними синапсами нейрона. Цю операцію можна описати як лінійну комбінацію;

c) функція активації (activation function) обмежує амплітуду вихідного сигналу нейрона. Цю функцію також називають функцією стиснення(squashing function). Зазвичай нормалізований діапазон амплітуд виходу нейрона лежить в інтервалі [0, 1] або [-1, 1]

Кожна функція активації додає нелінійність в навчанні нейронної мережі. Вона бере окреме число, що є результатом роботи попереднього шару і виконує фіксовану математичну операцію з ним. На практиці зустрічаються функції активації наведені в таблиці (таблиця 1.1) [2].

Функції активації, що у формулах представлені як F(s), визначають вихідний сигнал нейрона базуючись на локальне поле s. Хотілось би звернути увагу на три основні типи функції активації.

Таблиця 1.1 – Функції активації

Назва	Область значень	Формула
Лінійна	(-∞, ∞)	$F(s) = s$
Півлінійна	(-0, ∞)	$F(s) = \begin{cases} \mathrm{ks}, s > 0, \\ 0, s \leq 0. \end{cases}$
Логістична	(0, 1)	$F(s) = \dfrac{1}{1 + e^{-a}}$
Гіперболічний тангенс	(-1, 1)	$F(s) = \dfrac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$
Експоненціальна	(0, ∞)	$F(s) = e^{-a}$
Синусоїдна	(-1, 1)	$F(s) = \sin(s)$
Сигмоїдна	(-1, 1)	$F(s) = \dfrac{\mathrm{s}}{\mathrm{a} + \lvert\mathrm{s}\rvert}$
Лінійна з насиченням	(-1, 1)	$F(s) = \begin{cases} 0, & s \leq 0, \\ s, -1 < & s < 1, \\ 1, & s \geq 1 \end{cases}$
Порогова	(0, 1)	$F(s) = \begin{cases} 0, s < 0 \\ 1, s \geq 1 \end{cases}$
Модульна	(0, ∞)	$F(s) = \lvert s\rvert$
Знакова	(-1, 1)	$F(s) = \begin{cases} 1, s > 0, \\ -1, s \leq 0. \end{cases}$
Квадратична	(0, ∞)	$F(s) = s^2$

1. Функція одиничного стрибка, або порогова функція.

Цей тип функції відображено на (рисунок 1.2) та має наступний опис:

$$F(s) = \begin{cases} 0, s < 0 \\ 1, s \geq 1 \end{cases} \tag{1.1}$$

Відповідно до цього вихідний сигнал нейрона k такої функції можливо представити як:

$$y_k = \begin{cases} 0, s_k < 0, \\ 1, s_k \geq 0 \end{cases} \tag{1.2}$$

де s_k – це локальне поле нейрона, тобто:

$$s_k = \sum_{j=1}^{m} w_k \; x_j + b_k. \tag{1.3}$$

Ця модель ще називається моделлю Мак-Каллока-Питца [1]. В ній вихідний сигнал нейрона прийме значення 1, якщо індуковане локальне поле цього нейрона не від’ємне, та 0 – в іншому випадку.

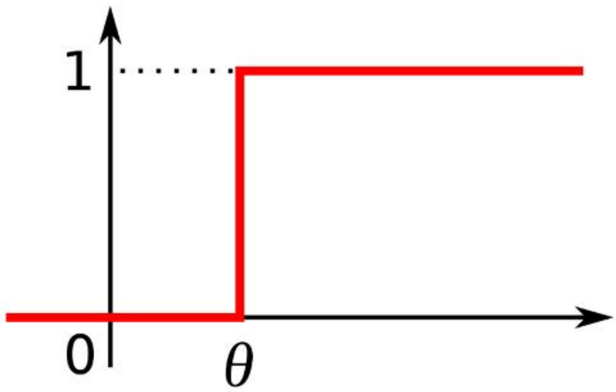

Рисунок 1.2 – Функція одиничного стрибка

2. Частково-лінійна функція.

Графік функції (рисунок 1.3) – пряме відображення наступного опису:

$$F(s) = \begin{cases} 0, \; s \leq -\frac{1}{2}, \\ |s|, \; -1 < \; s < 1, \\ 1, \; s \geq \frac{1}{2} \end{cases} \tag{1.4}$$

Де коефіцієнт підсилення в лінійній області оператора передбачається дорівнювати одиниці. Ця функція активації може розглядатися як апроксимація нелінійного підсилення [1]. Якщо лінійна область оператора не досягає порога насичення, то він стає лінійним суматором. Якщо коефіцієнт підсилення лінійної області вважати нескінченно великим, то функція вироджується в порогову.

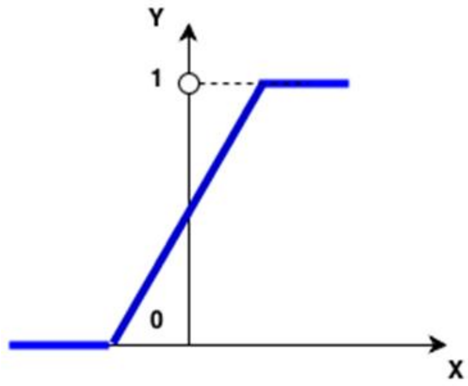

Рисунок 1.3 – Частково-лінійна функція

3. Сигмоїдальна функція.

Це стрімко зростаюча функція, яка тримає баланс проміж лінійною та нелінійною поведінкою. Прикладом сигмоїдальної функції може буті логістична функція, яка задається наступним виразом:

$$F(s) = \frac{1}{1+ e^{-a}}. \qquad (1.5)$$

Де а – параметр нахилу сигмоїдальної функції. Змінюючи цей параметр мі маємо змогу будувати функції різної крутизни (рисунок 1.4). Сигмоїдна функція історично часто використовується через подібність до рівня активації нейрона: від зовсім неактивного (0), до повністю активованого (1). На практиці, сигмоїдна нелінійність не так давно вийшла з моди і досить рідко використовується. Вона має два суттєві недоліки. Сигмоїда подавляє та вбиває градієнти. Зовсім непотрібна особливість сигмоїдного нейрону полягає в тому, що коли активізація близька до обох «кінців» біля 1 або 0, градієнт у цих областях близький до 0. Під час зворотного розповсюдження похибки локальний градієнт буде помножений на вихідний. Таким чином, якщо локальний градієнт дуже малий, то він зробить і глобальний дуже малим, і розповсюдження помилки також буде малим, рекурсивно поширюючи цю проблему. Окрім того, слід слідкувати за ініціалізацією вагових коефіцієнтів. Вихід сигмоїди не централізований відносно нуля. Це є проблемою, оскільки нейрони у глибших шарах нейронної мережі отриматимуть також не центровані відносно нуля значення. Це матиме вплив на динаміку під час градієнтного спуску, тому що дані на вході нейрону будуть завжди позитивними, тоді градієнт вагових коефіцієнтів буде або цілком позитивним, або цілком негативним.

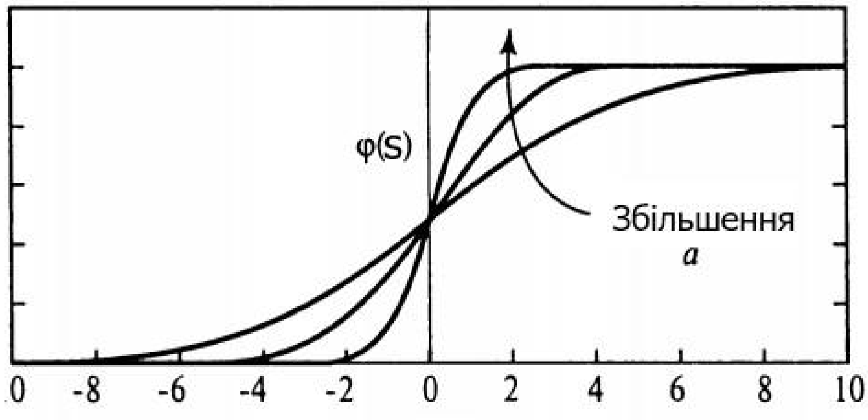

Рисунок 1.4 – Сигмоїдальна функція з змінним параметром *a*

4. Гіперболічний тангенс.

Гіперболічний тангенс (hyperbolic tangent, tanh) приймає на вході довільне дійсне число, а на виході дає дійсне число в інтервалі від -1 до 1.

$$F(s) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{1.6}$$

Подібно сигмоїду, гіперболічний тангенс може насичуватися. Однак, на відміну від сигмоида, вихід даної функції відцентрований щодо нуля. Отже, на практиці завжди краще використовувати гіперболічний тангенс, а не сигмоїд.

5. ReLU

В останні роки велику популярність придбала функція активації під назвою «випрямляч» (rectifier, за аналогією з однополуперіодним випрямлячем в електротехніці) [5]. Нейрони з цією функцією активації називаються ReLU (rectified linear unit). ReLU має наступну формулу:

$$F(s) = \max(0, s) \tag{1.7}$$

Обчислення сигмоїд і гіперболічного тангенса вимагає виконання ресурсномістких операцій, таких як спорудження до рівня, в той час як ReLU може бути реалізований за допомогою простого порогового перетворення матриці активацій в нулі. Крім того, ReLU не схильний до насичення. Застосування ReLU істотно підвищує швидкість збіжності стохастичного градієнтного спуску (в деяких випадках до 6 разів) в порівнянні з сигмоїдом і гіперболічним тангенсом[5]. Вважається, що це обумовлено лінійним характером і відсутністю насичення даної функції. На жаль, ReLU не завжди достатньо надійні і в процесі навчання можуть виходити з ладу («вмирати»). Наприклад, великий градієнт, що проходить через ReLU, може привести до такого оновлення ваг, що даний нейрон ніколи більше не активується. Якщо це станеться, то, починаючи з даного моменту, градієнт, що проходить через цей нейрон, завжди буде дорівнює нулю. Відповідно, даний нейрон буде необоротно виведений з ладу. Наприклад, при дуже великій швидкості навчання (learning rate), може виявитися, що до 40% ReLU «мертві» (тобто, ніколи не активуються). Ця проблема вирішується за допомогою вибору належної швидкості навчання.

1.2 Типи нейронних мереж

Визначившись у загальних рисах, що є нейронна мережа, можна виділити основні типи їх класифікації. Перш ніж приступити до класифікації необхідно ввести одно уточнення. Кожна мережа має перший шар нейронів, який називається вхідним. Він не виконує ніяких обчислень і перетворень, його завдання полягає тільки в одному: прийняти і розподілити по інших нейронах вхідні сигнали. Це єдиний шар, який є загальним для усіх типів нейронних мереж, подальша їх структура і є критерієм для основного ділення.

Одношарова нейронна мережа. Це структура взаємодії нейронів, при якій після попадання вхідних даних в перший вхідний шар відразу передається в шар виходу кінцевого результату. При цьому перший вхідний шар не вважається, оскільки він не виконує ніяких дій, окрім прийому і розподілу, про це вже було сказано вище. А другий шар робить усі потрібні обчислення і обробки і відразу видає кінцевий результат. Вхідні нейрони об’єднані з основним шаром синапсами, що мають різний ваговий коефіцієнт, що забезпечує якість зв’язків.

Багатошарова нейронна мережа. Як зрозуміло з визначення, цей вид нейронних мереж окрім вхідного і вихідного шарів має ще і проміжні шари. Їх кількість залежить від міри складності самої мережі. Вона більшою мірою нагадує структуру біологічної нейронної мережі. Такі види мереж були розроблені зовсім нещодавно, до цього усі процеси були реалізовані за допомогою одношарових мереж. Відповідно подібне рішення має набагато більше можливостей, ніж її предок. В процесі обробки інформації кожен проміжний шар є проміжним етапом обробки і розподілу інформації.

З точок на площині і з’єднань між ними можна побудувати безліч графічних фігур, що називаються графами. Якщо кожну точку уявити собі як один нейрон, а з’єднання між точками – як дендрит і синапси, то ми отримаємо нейронну мережу. Але не всяке з’єднання нейронів буде працездатне або взагалі доцільно. Тому на сьогодні існує тільки декілька працюючої і реалізованої програмно архітектури нейромереж. Я тільки коротко опишу їх пристрій і класи вирішуваних ними завдань. По архітектурі зв’язків нейромережі можуть бути згруповані в два класи: мережі прямого розповсюдження (рисунок 1.5), в яких зв’язки не мають петель, і мережі рекурентного типу, в яких можливі зворотні зв’язки (рисунок 1.6)

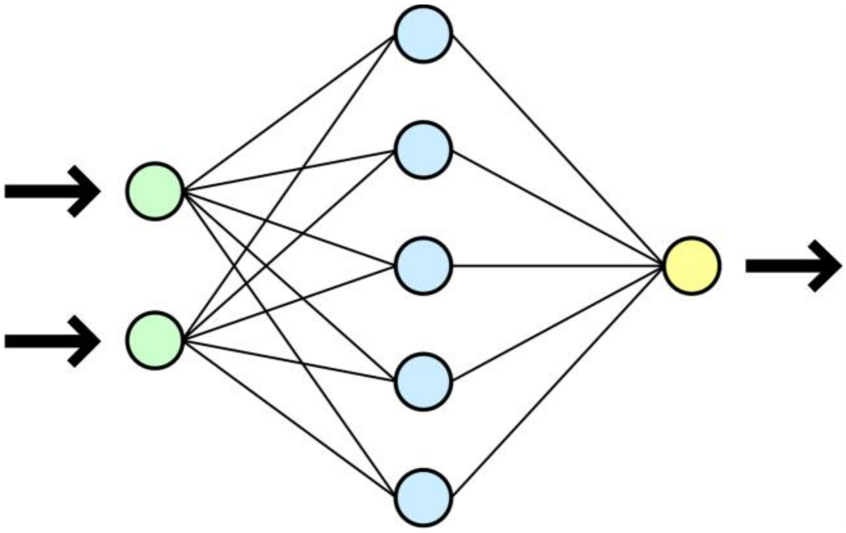

Рисунок 1.5 – Мережа прямого розповсюдження

Мережі прямого поширення підрозділяються на одношарові перцептрони (мережі) і багатошарові перцептрони (мережі). Назву перцептрона для нейромереж придумав американський нейрофізіолог Ф. Розенблатт, що придумав в 1957 році перший нейропроцесорний елемент (НПЭ), тобто нейромережа. Він же довів збіжність області рішень для персептрона при його навчанні. Відразу після цього почалося бурхливе дослідження в цій області і був створений найперший нейрокомп’ютер Mark I. Багатошарові мережі відрізняються тим, що між вхідними і вихідними даними розташовуються декілька так званих прихованих шарів нейронів, що додають більше нелінійних зв’язків в модель. Розглянемо облаштування простої багатошарової нейромережі.

Будь – яка нейронна мережа складається з вхідного шару і вихідного шару. Відповідно подаються незалежні і залежні змінні. Вхідні дані перетворяться нейронами мережі і порівнюються з виходом. Якщо відхилення більше за задане, то спеціальним чином змінюються ваги зв’язків нейронів між собою і порогові значення, нейронів. Знову відбувається процес обчислень вихідного значення і його порівняння з еталоном. Якщо відхилення менше заданої погрішності, то процес навчання припиняється. Окрім вхідного і вихідного шарів у багатошаровій мережі існують так звані приховані шари. Вони є нейронами, які не мають безпосередніх входів початкових даних, а пов’язані тільки з виходами вхідного шару і з входом вихідного шару. Таким чином, приховані шари додатково перетворять інформацію і додають нелінійності в моделі.

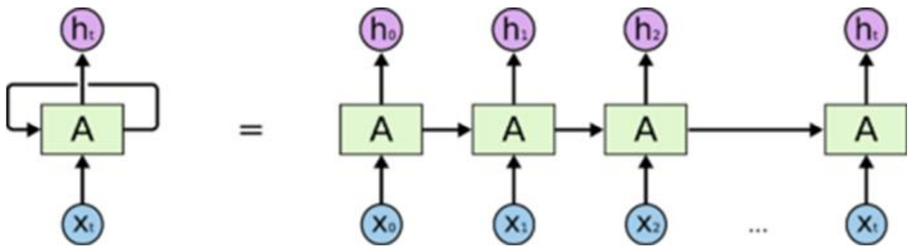

Рисунок 1.6 – Рекуррентна мережа

Якщо одношарова нейромережа дуже добре справляється із завданнями класифікації, оскільки вихідний шар нейронів порівнює отримані від попереднього шару значення з порогом і видає значення або нуль, тобто менше порогового значення, або одиницю – більше за пороговий (для випадку порогової внутрішньої функції нейрона), і не здатний вирішувати більшість практичних завдань (що було доведене Мінським і Пейпертом), то багатошаровий перцептрон з сигмоидными вирішальними функціями здатний апроксимує будь – який функціональний залежність (це було доведено у вигляді теореми). Але при цьому не відоме ні потрібне число шарів, ні потрібна кількість прихованих нейронів, ні необхідний для навчання мережі час. Ці проблеми досі стоять перед дослідниками і розробниками нейромереж. Особисто мені здається, що увесь ентузіазм в застосуванні нейромереж будується саме на доведенні цієї теореми. Розглянемо, як нейрони можуть моделювати різні класи функцій.

Клас рекурентних нейромереж набагато більший, та і самі мережі складніші по своїй будові (рисунок 1.5). Поведінка рекурентних мереж описується диференціальними або різницевими рівняннями, як правило, першого порядку. Це набагато розширює сфери застосування нейромереж і способи їх навчання. Мережа організована так, що кожен нейрон отримує вхідну інформацію від інших нейронів, можливо, і від самого себе, і від довкілля. Цей тип мереж має важливе значення, оскільки з їх допомогою можна моделювати нелінійні динамічні системи. Серед рекурентних мереж можна виділити мережі Хопфилда і мережі Кохонена. За допомогою мереж Хопфилда можна обробляти неврегульовані (рукописні букви), впорядковані в часі (тимчасові ряди) або просторі (графіки) зразки. Рекурентна нейромережа простого виду була введена Хопфилдом і побудована вона з N нейронів, пов’язаних кожен з кожним окрім самого себе, причому усі нейрони є вихідними. Нейромережу Хопфилда можна використати в якості ассоціативної пам’яті.

Мережу Кохонена ще називають «картою ознак, що самоорганізуються». Мережа такого типу розрахована на самостійне навчання. Під час навчання повідомляти їй правильні відповіді необов’язково. В процесі навчання на вхід мережі подаються різні зразки. Мережа уловлює особливості їх структури і розділяє зразки на кластери, а вже навчена мережа відносить кожен приклад, що знову поступає, до одного з кластерів, керуючись деяким критерієм «близькості». Мережа складається з одного вхідного і одного вихідного шару. Кількість елементів у вихідному шарі безпосередньо визначає, скільки різних кластерів мережа зможе розпізнати. Кожен з вихідних елементів отримує на вхід увесь вхідний вектор. Як і у всякій нейронній мережі, кожному зв’язку приписана деяка синоптична вага. У більшості випадків кожен вихідний елемент сполучений також зі своїми сусідами. Ці внутрісловні зв’язки відіграють важливу роль в процесі навчання, оскільки коригування вагів відбувається тільки в околиці того елементу, який якнайкраще відгукується на черговий вхід. Вихідні елементи змагаються між собою за право вступити в дію і «отримати урок». Виграє той з них, чий вектор вагів виявиться ближчий за усіх до вхідного вектору.

1.3 Глибинні нейронні мережі

Можна зробити систему більш гнучкою та потужною за допомогою додаткових прихованих шарів. Штучні нейронні мережі з безліччю прихованих шарів між вхідним і вихідним шарами називаються глибинними нейронними мережами (DNN), і вони можуть моделювати складні нелінійні зв’язки [3].

1.3.1 Багатошаровий перцептрон (MLP)

Багатошаровий перцептрон (MLP) має три або більше шарів (рисунок 1.7).

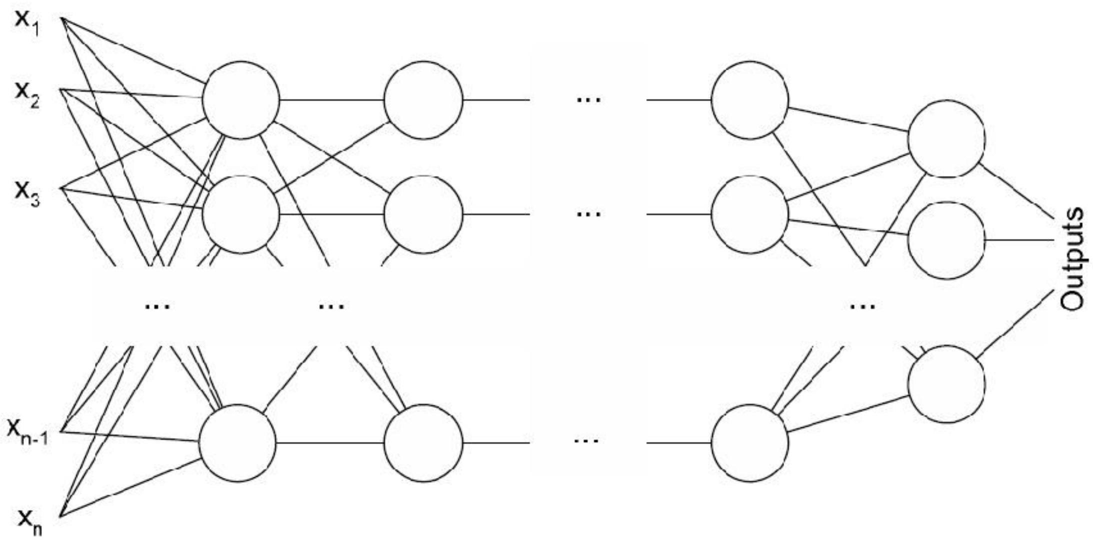

Рисунок 1.7 – Багатошаровий перцептрон

Він використовує нелінійну функцію активації (головним чином гіперболічну, дотичну або логістичну функцію), яка дозволяє класифікувати дані, які не є лінійно відокремлюваними. Кожен вузол у шарі з’єднується з кожним вузлом у наступному шарі, роблячи мережу повністю підключеною. Наприклад, багатошарові програми перцептронної обробки мови (NLP) – це розпізнавання мовлення та машинний переклад.

1.3.2 Згортальна нейронна мережа (CNN)

Згортальна нейронна мережа (CNN) містить один або декілька згорткових шарів (рисунок 1.8), об’єднаних або повністю пов’язаних, і використовує варіації багатошарових перцептронів, обговорених вище. Згортальні шари використовують операцію згортання на вхід, передаючи результат наступному шару. Ця операція дозволяє зробити мережу глибшою із значно меншими параметрами. Згортальні нейронні мережі демонструють неабиякі результати при застосуванні їх до зображень та мовлення.

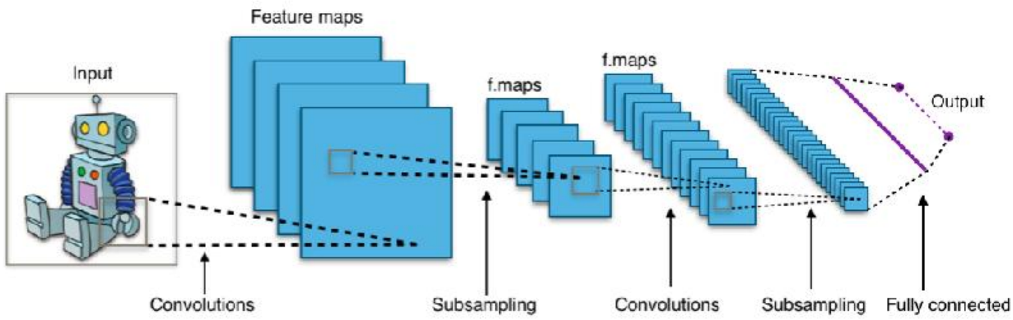

Рисунок 1.8 – Згортальна нейронна мережа

1.3.3 Рекурсивна нейронна мережа (Recursive neural network)

Рекурсивна нейронна мережа – це тип глибокої нейронної мережі, сформованої шляхом застосування одного і того ж набору ваг рекурсивно (рисунок 1.9) над структурою для створення структурованого прогнозування вхідних структур змінного розміру або скалярного прогнозування на ньому шляхом проходження заданої структури в топологічному порядку. У найпростішій архітектурі нелінійність, наприклад, tanh і вагова матриця, що ділиться по всій мережі, використовуються для об’єднання вузлів в батьківські.

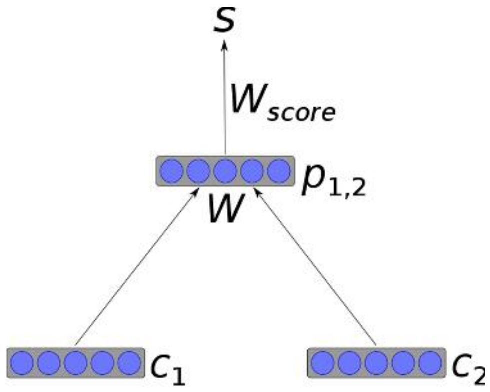

Рисунок 1.9 – Рекурсивна нейронна мережа

1.3.4 Рекурентна нейронна мережа (Recurrent neural network)

Рекурентна нейронна мережа (RNN), на відміну від нейронної мережі, що подається, є варіантом рекурсивної штучної нейронної мережі, в якій зв’язки між нейронами здійснюють спрямований цикл.

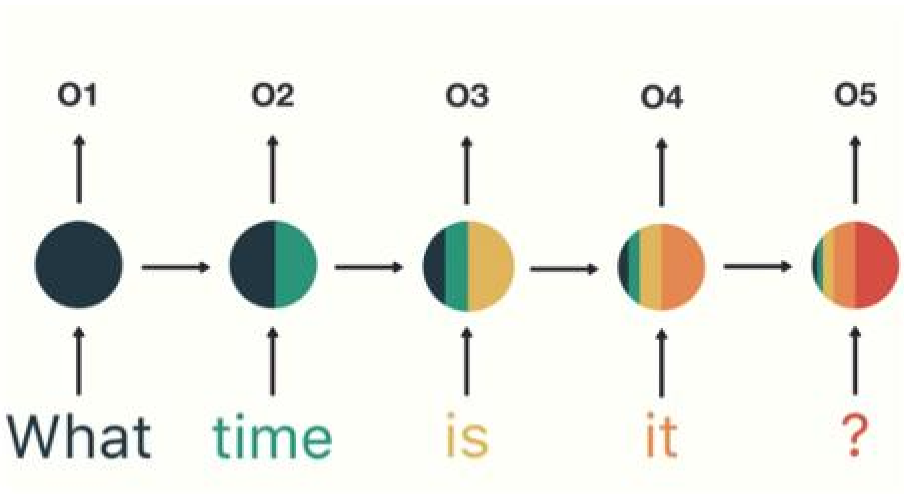

Рисунок 1.10 – Рекурентна нейронна мережа

Ця пам’ять дозволяє користувачам вирішувати проблеми NLP, такі як підключення розпізнавання рукописного вводу або розпізнавання мови.

У цій нейромережі вихід попереднього кроку подається як вхід до поточного кроку. Це означає, що вихід залежить не тільки від теперішніх входів, але і від стану нейронів попереднього кроку. Традиційні нейронні мережі, що мали місце до розвитку рекурентних нейронних мереж, означають, що всі входи та виходи незалежні один від одного, але у випадках, коли потрібно передбачити наступне слово речення, потрібні попередні слова, і таким чином важливо запам’ятати попередні слова. Отже, після цього з’явився лише RNN, який вирішив цю проблему за допомогою прихованого шару. Ключовою і найбільш важливою особливістю RNN є прихований стан, який запам’ятовує деякі деталі в послідовності.

RNN має «пам’ять», в якій згадуються всі знання про те, що було виміряно. Для отримання результату він використовує однакові параметри для кожного, оскільки він виконує однакову функцію на всіх входах або прихованих шарах. На відміну від інших нейронних мереж, це зменшує складність параметрів.

Перевагою рекурентної нейронної мережі (RNN) є те, що вона фіксує послідовну інформацію, знайдену у вхідних даних, тобто зв’язок між словами в тексті при прогнозуванні наступного (рисунок 1.10).

1.3.5 Довга короткострокова пам’ять (Long short-term memory)

Довга короткострокова пам’ять (LSTM) – це специфічна архітектура рекурентної нейронної мережі (RNN), яка була розроблена для більш точного моделювання тимчасових послідовностей та їх далеких залежностей (рисунок 1.11).

LSTM не використовує функцію активації у своїх періодичних компонентах, збережені значення не змінюються, а градієнт не має тенденції до зникнення під час тренування.

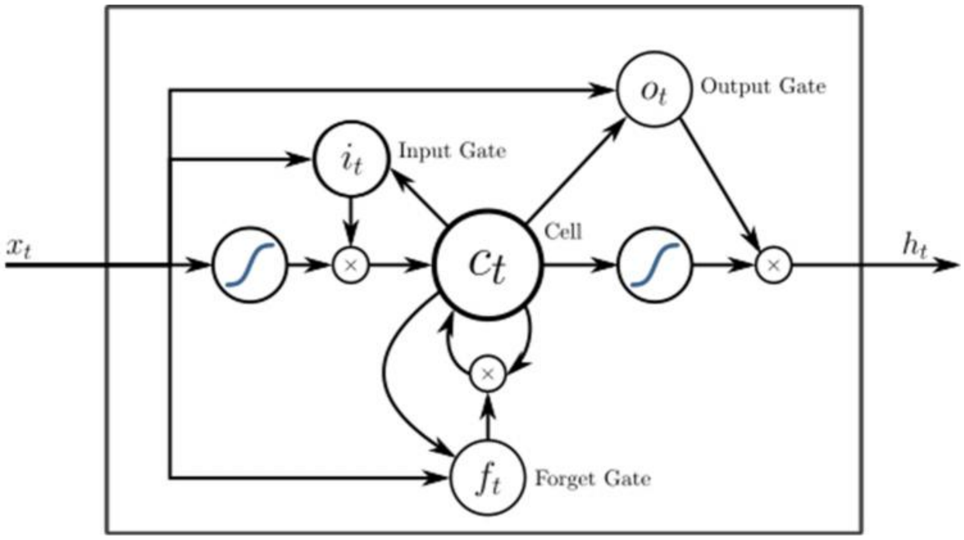

Рисунок 1.11 – Довга короткострокова пам’ять

Зазвичай одиниці LSTM реалізуються в «блоках» з кількома одиницями. Ці блоки мають три або чотири «gate» (наприклад, вхідні ворота, забуті ворота, вихідні ворота), які керують нанесенням інформаційного потоку на логістичну функцію.

2 ПІДГОТОВКА ДАНИХ

2.1 Попередня обробка зображень

Згортки цифрового зображення широко застосовуються в сучасних фото редакторах для представлення зображення і оптимізації масштабу [10-12]. Для тих же цілей їх можна застосовувати і в системах розпізнавання образів на етапі попередньої обробки [7-9]. При роботі зі згортальними нейронними мережами (ЗНМ) [13-15] такі алгоритми широко використовуються для підготовки вхідних даних. А також для оптимізації часу передачі даних по каналах зв’язку в різних умовах функціонування мережі [16-17]. Провівши аналіз можна сказати, що в даний час застосовується велика кількість різноманітних алгоритмів згортки зображення [10]. Кожен алгоритм за рахунок своїх особливостей дає різні результати для різних типів зображень.

Виходячи з вищесказаного випливає те, що актуальними є завдання аналізу ефективності представлення даних і застосування алгоритмів згортки зображень, що містять лінійні і малорозмірні об’єкти на межі дозволу[10].

Добре відомо, що на напівтоновому зображенні різні дефекти помітні набагато краще, ніж на кольоровому. Тому перед використанням зображення перетворюється в півтоновумодель. Існують різні способи перетворення кольорового зображення в півтонове [10, 12].

У колірних просторах YUV і YIQ використовувані в PAL і NTSC яскравість I обчислюється так

$$I = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{2.1}$$

При переході між колірними просторами (наприклад: RGB → HSI) використовують формулу середнього.

$$I = \frac{1}{3} \times (R + G + B) \qquad (2.2)$$

Оскільки яскравість зображення у цифровому форматі приймає цілочисельні невід’ємні значення, тут і далі необхідно округлити результат до цілого.

При цьому для обліку особливостей сприйняття зображення людським оком в моделі HDTV використовують коефіцієнти відмінні від використовуваних в колірних просторах YUV і YIQ.

$$I = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B \qquad (2.3)$$

Вибір правила отримання напівтонового зображення для цілей роботи не є значимим. За замовчуванням для перетворення зображення в напівтоновий формат використовується правило (1).

Також необхідно обрати коефіцієнт зменшення масштабу зображення. Проаналізувавши сучасне стану питання виходить [10, 13], що найчастіше використовуються коефіцієнти 2 і 3. При використанні СНС, як правило, використовується фактор 2 для плавного ітераційного зменшення зображення в 2 рази, що дає мінімальну ступінь згладжування дрібних деталей. У класичних системах, найчастіше, використовується фактор 3, що дозволяє адекватно враховувати положення пікселів об’єктів і ліній в околиці щодо її центру.

У цій роботі розглядаються коефіцієнти 2 і 3. Крім того, розглянуто використання коефіцієнта 4 для аналізу ступеня згладжування / зникнення дрібних деталей на знімку. І підтвердження того факту, що дворазове зменшення знімка в 2 рази краще одноразового зменшення в 4 рази.

Далі вибираємо тип згортки. Було реалізовано шість алгоритмів згортки. За різними джерелами, кожна з них в більшій чи меншій мірі підходить для певного типу зображень. Для цілей роботи розглянемо наскільки вони підходять для роботи з лініями і дрібними деталями.

В роботі розглядаються такі типи згорток:

1. яскравість центрального пікселя околиці:

$$
\begin{gathered}
f` = f(\mathcal{E}, \eta), \\
\mathcal{E} = x_m + 0.5 \cdot [x_m - x_m], \\
\eta = y_m + 0.5 \cdot [y_m - y_m],
\end{gathered}
\tag{2.4}
$$

де x_m, x_m, y_m, y_m – граничні координати пікселів розглянутої околиці.

```
def get_center_value(arr):
    if not len(arr): return 0
    position = floor(len(arr)/2)
    return arr[position]return report;
}
```

Приклад 2.1 – Яскравість центрального пікселя околиці (файл convolution.py)

2. середнє арифметичне:

$$
f` = a = \frac{1}{n} \cdot \sum_{i=1}^{n} f_i, \tag{2.5}
$$

де f_i – яскравості пікселів розглянутої околиці загальним числом.

```
def get_avg_value(arr):
    if not len(arr): return 0
    sum = reduce(lambda x, y: x+y, arr)
    return sum/len(arr)
```

Приклад 2.2 – Середнє арифметичне (файл Entity.as)

3. яскравість найближча до середнього в околиці:

$$
f` = \arg(\min |f_i - a|), \tag{2.6}
$$

де f_i – яскравості пікселів розглянутої околиці загальним числом.

```
def get_closest_to_avg_value(arr):
    avg = get_avg_value(arr)
    currDist = 255
    closest = avg
    for item in arr:
        tempDist = floor(fabs(avg - item))
        if(tempDist < currDist):
            currDist = tempDist
            closest = item
    return closest
```

Приклад 2.3 – Формування звіту про характеристики (файл convolution.py)

4. усічене середнє:

$$f` = \frac{1}{n-2k} \cdot \sum_{i=1+k}^{n-k} f_i \ , \tag{2.7}$$

для значень k=1, k=0.25n, f_i, – яскравості пікселів розглянутої околиці загальним числом.

```
def get_cutted_avg_value(arr, type):
    srtdArr = sorted(arr)
    length = len(srtdArr)
    arrToCalcAvg = []
    if type == 1 and length > 2:
        arrToCalcAvg = srtdArr[1:length-1]
    elif type == 0.5 and length > 2:
        cutter = ceil(length/4)
        arrToCalcAvg = srtdArr[cutter:length-cutter]
    else:
        return get_center_value(srtdArr)
    return get_avg_value(arrToCalcAvg)
```

Приклад 2.4 – Усічене середнє (файл convolution.py)

5. медіана ряду яркостей в околиці:

$$f` = m \ (f_i), \tag{2.8}$$

де f_i – яскравості пікселів розглянутої околиці загальним числом.

6. середнє арифметичне на безлічі з яркостей - найближчих до яскравості центрального пікселя в розглянутій околиці:

$$f` = a_{\ m} = \frac{1}{m} \cdot \sum_{i=1}^{m} f_i, \tag{2.9}$$

f_i - найближчі до яскравості центру міста яскравості пікселів околиці загальним числом.

```
def get_adaptive_value(arr, m=3):
    if not len(arr): return 0
    centerValue = get_center_value(arr)
    srtdArr = sorted(arr)
    closestCount = floor(m/2) if isEven(m) else floor((m-1)/2)
    centerIndex = srtdArr.index(centerValue)
    leftIndex = max([centerIndex - closestCount, 0])
    rightIndex = min([centerIndex+closestCount+1, len(arr)])
    slicedArr = arr[leftIndex : rightIndex]
    return get_avg_value(slicedArr)
```

Приклад 2.5 – Середнє арифметичне на безлічі з яркостей (файл convolution.py)

Деякі з розглянутих згорток є неадаптивними, оскільки вони завжди замінюють розподіл яскравості в околиці визначеним значенням яскравості. У той же самий час інші згортки дозволяють отримувати нове значення яскравості з урахуванням розподілу яскравості в околиці.

Якщо поглянути на табличні результати [6], то можна помітити, що всі три коефіцієнта показали збереження рівня однорідності, а при алгоритмах усіченого середнього та середнього арифметичного, для коефіцієнтів 3 і 4 показник рівня однорідності збільшився. З боку показника контрасту можна помітити стабільно високе значення для алгоритму яскравості середнього пікселя, цей алгоритм добре себе показав на всіх коефіцієнтах. Слідом за ним стабільні показники показує алгоритм середнього арифметичного. Найгірше себе показав алгоритм середнього арифметичного на безлічі яркостей найближчих до яскравості центрального пікселя. У той же час на коефіцієнтах 3 і 4 високими значеннями відзначилися алгоритми медіани і яскравості найближчій до середнього. В цілому показники коефіцієнтів 3 і 4 є вище ніж для згортки з коефіцієнтом 2.

Таким чином, найкращі результати згортки зображення досягаються при використанні усереднення згорток і зменшенні зображення з використанням коефіцієнтів 3 або 4.

2.2 Вибір та підготовка набору даних

Вибір набору данних є важливим кроком при роботі з нейромережевими моделями класифікації зображень.

Таблиця 2.1 – Доступні набори даних

Dataset	Training Set Size	Testing Set Size	Number of Classes
Flowers	2500	2500	5
Cifar10	60k	10k	10
MNIST	60k	10k	10
Tiny Imagenet	100k	10k	200
ImageNet	1.2M	50k	1000

Представлені відкриті набори даних (таблиця 2.1), які розповсюджені серед громади, що працює у сфері класифікації зображень. Перелічені набори даних ретельно підбирались і редагувались роками, деякі набори розроблялись під спеціальні змагання. Наприклад з 2010 року ведеться проект ILSVRC (англ. ImageNet Large Scale Visual Recognition Challenge – Кампанія по широкомасштабному розпізнаванню образів в ImageNet), в рамках якого різні програмні продукти щорічно змагаються в класифікації і розпізнаванні об’єктів і сцен в базі даних ImageNet. Так як ImageNet налічує 1.2M навчальних прикладів, що призводить до того, що завантаження може зайняти кілька годин і може використовуватися до 500 Гб простору на диску. Навчати модель з нуля за допомогою такого набору даних може зайняти кілька днів, залежно від налаштувань обладнання і це не зважаючи на те що є способи навчання моделі на декількох графічних процесорах та / або декількох процесорах, синхронно або асинхронно. Ці фактори призвели до того, що з’явились такі набори даних, як Tiny ImageNet. Він в свою чергу менший, але схожий за принципом, набір даних, що був розроблений у Стенфордському університеті, для того щоб вирішення проблеми класифікації зображень було доступно більшому колу людей. Він налічує 100 тисяч навчальних прикладів, які потім дозволяють розпізнавати до 200 типів зображень. База даних рукописних цифр MNIST, має навчальний набір з 60000 прикладів і тестовий набір 10 000 прикладів. Це підмножина більшого набору, доступного від NIST. Цифри були нормалізовані за розміром і центровані на зображені фіксованого розміру. Проте розпізнавання тексту на зображенні є специфічним завданням, тому вибір падає на набір даних CIFAR-10. Він складається з 60000 кольорових зображень 32x32 в 10 класах, з 6000 зображень на клас. Набір даних розділений на п’ять навчальних наборів і один тестовий набір, кожен з яких має 10000 зображень. Тестовий набір містить рівно 1000 випадково вибраних зображень з кожного класу. Навчальні набори містять решту зображень у довільному порядку, але деякі навчальні набори можуть містити більше зображень з одного класу, ніж іншого.

Класи зображень взаємовиключні. Зображення автомобілів та вантажівок немає пересікаються. «Автомобіль» включає седани, позашляховики, речі такого роду. «Вантажівка» включає лише великі вантажні автомобілі.

Для обраного набору даних нам потрібно завантажити необроблені приклади та перетворити їх у вихідний формат TFRecord TensorFlow, приклад 2.6. Кожен TFRecord містить буфер протоколу TF-Example.

```
def run(dataset_dir):
  if not tf.gfile.Exists(dataset_dir):
    tf.gfile.MakeDirs(dataset_dir)
  training_filename = _get_output_filename(dataset_dir, 'train')
  testing_filename = _get_output_filename(dataset_dir, 'test')

  if tf.gfile.Exists(training_filename) and
tf.gfile.Exists(testing_filename):
    return
  dataset_utils.download_and_uncompress_tarball(_DATA_URL, dataset_dir)
  with tf.python_io.TFRecordWriter(training_filename) as
tfrecord_writer:
    offset = 0
    for i in range(_NUM_TRAIN_FILES):
      filename = os.path.join(dataset_dir,
                              'cifar-10-batches-py',
                              'data_batch_%d' % (i + 1))  # 1-indexed.
      offset = _add_to_tfrecord(filename, tfrecord_writer, offset)
  with tf.python_io.TFRecordWriter(testing_filename) as tfrecord_writer:
    filename = os.path.join(dataset_dir,
                            'cifar-10-batches-py',
                            'test_batch')
    _add_to_tfrecord(filename, tfrecord_writer)
  labels_to_class_names = dict(zip(range(len(_CLASS_NAMES)),
_CLASS_NAMES))
  dataset_utils.write_label_file(labels_to_class_names, dataset_dir)
  _clean_up_temporary_files(dataset_dir))
```

Приклад 2.6 – tfrecord файли для обраного набору даних(файл convolution.py)

Коли сценарій закінчиться, буде створено кілька нових файлів TFRecord, приклад 2.7.

```
cifar10_test.tfrecord...
cifar10_train.tfrecord
labels.txt
```

Приклад 2.7 – tfrecord файли для обраного набору даних(файл convolution.py)

Файли TFRecord представляють дані для навчання та валідації. Також буде створений файл labels.txt, який містить відображення від цілих міток до назв класів.

```
def get_split(split_name, dataset_dir, file_pattern=None, reader=None):
  if split_name not in SPLITS_TO_SIZES:
    raise ValueError('split name %s was not recognized.' % split_name)
  if not file_pattern:
    file_pattern = _FILE_PATTERN
  file_pattern = os.path.join(dataset_dir, file_pattern % split_name)

  if not reader:
    reader = tf.TFRecordReader

  keys_to_features = {
      'image/encoded':         tf.FixedLenFeature((),         tf.string,
default_value=''),
      'image/format':         tf.FixedLenFeature((),         tf.string,
default_value='png'),
      'image/class/label': tf.FixedLenFeature(
          [], tf.int64, default_value=tf.zeros([], dtype=tf.int64)),
  }

  items_to_handlers = {
      'image': slim.tfexample_decoder.Image(shape=[32, 32, 3]),
      'label': slim.tfexample_decoder.Tensor('image/class/label'),
  }

  decoder = slim.tfexample_decoder.TFExampleDecoder(
      keys_to_features, items_to_handlers)

  labels_to_names = None
  if dataset_utils.has_labels(dataset_dir):
    labels_to_names = dataset_utils.read_label_file(dataset_dir)

  return slim.dataset.Dataset(
      data_sources=file_pattern,
      reader=reader,
      decoder=decoder,
      num_samples=SPLITS_TO_SIZES[split_name],
      items_to_descriptions=_ITEMS_TO_DESCRIPTIONS,
      num_classes=_NUM_CLASSES,
      labels_to_names=labels_to_names)
```

Приклад 2.8 – Визначення набору даних Slim (файл cifar10.py)

Після створення файлів TFRecord є можливість легко визначити набір даних Slim, приклад 2.8, який зберігає вказівники на файл даних, а також різні інші фрагменти метаданих, такі як мітки класу, розділення навчання / тесту та розбір TFExample protos.

3 ОПИС АЛГОРИТМІВ

CNN - це мережі, що передаються вперед, тому інформаційний потік відбувається лише в одному напрямку, від їх входів до їх результатів. Так само, як штучні нейронні мережі (ANN) біологічно натхненні, так і CNN. Зорова кора мозку, що складається з чергуються шарів простих і складних клітин (Hubel & Wiesel, 1959, 1962), мотивує їх архітектуру. CNN архітектури бувають декількох варіантів; однак, загалом, вони складаються з згортання та об’єднання (або підсистеми) шарів, які згруповані в модулі. Дотримуйтесь цих модулів або один, або більше повністю пов’язаних шарів, як у стандартній нейромережі подачі. Модулі часто укладаються один на одного, щоб сформувати глибоку модель і це результує в типову архітектуру CNN для завдання класифікації зображень іграшки (рисунок 3.1).

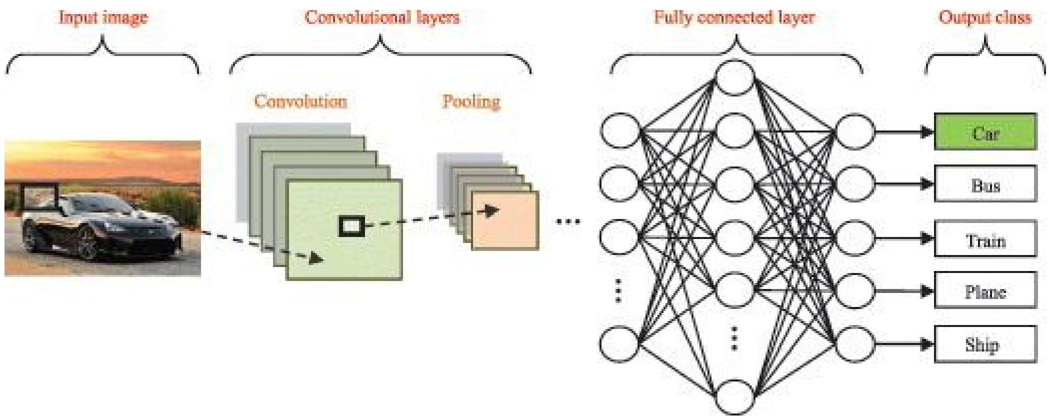

Рисунок 3.1 – Типова архітектура CNN

Зображення вводиться безпосередньо в мережу, після чого слідує кілька етапів згортання та об’єднання. Після цього представлення цих операцій подають один або більше повністю пов’язаних шарів. Нарешті, останній повністю пов’язаний шар виводить мітку класу. Незважаючи на те, що це найпопулярніша базова архітектура, знайдена в літературі, за останні роки було запропоновано кілька змін архітектури з метою підвищення точності класифікації зображень або зниження витрат на обчислення. Ці мережі заглибилися настільки глибоко, що уявити всю їх модель стало надзвичайно важко. Ми припиняємо слідкувати і ставимось до них як до чорного ящика. Ілюстрації моделей в цьому розділі забезпечують більш компактний вигляд кожної з моделей, не потребуючи прокручування вниз пару разів, щоб побачити шар softmax.

Додамо, що безліч архітектур CNN, що можна побачити, є результатом багатьох речей – вдосконаленого комп'ютерного обладнання, конкуренції ImageNet, вирішення конкретних завдань, нових ідей тощо.

3.1 Архітектура Inception_v1

Google, будучи Google, був зацікавлений у застосуванні глибоких нейронних мережних методів для своїх цілей, але спочатку їм довелося придумати спосіб зменшити обчислювальний тягар глибоких нейронних мереж при отриманні найсучаснішої продуктивності.

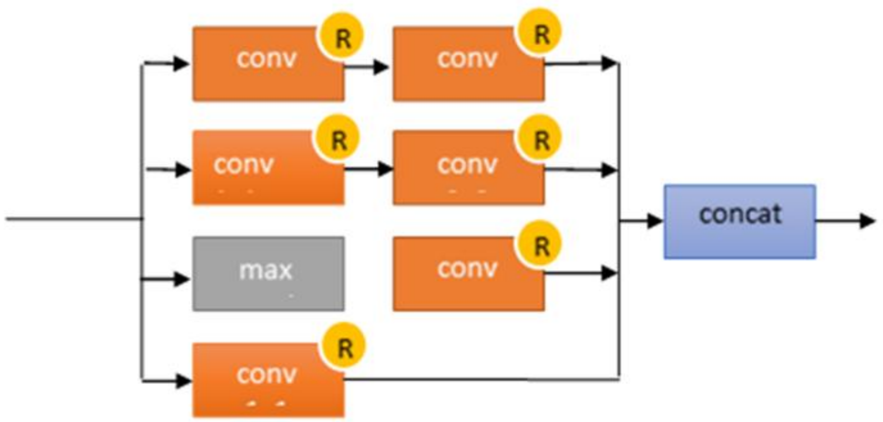

Рисунок 3.2 – Inception module зі зменшенням розміра

Ми відразу маємо виділити, що це зовсім інша структура порівняно з попередником:

- використовує 1x1 згортку;
- середнє глобальне об’єднання в кінці замість шарів FC;
- впровадження початкового модуля;

В цій мережі обмежують кількість вхідних каналів, додаючи додаткову згортку 1x1 до згорток 3x3 та 5x5. Також цікаво відзначити згортку 1x1 після модуля максимального об’єднання (max pool).

Суть полягає в тому, що 1x1 згортки використовуються як модуль зменшення розмірів для зменшення вузьких місць, що призводить до збільшення глибини та ширини.

```
end_point = 'Mixed_3b'
with tf.variable_scope(end_point):
  with tf.variable_scope('Branch_0'):
    branch_0 = slim.conv2d(net, 64, [1, 1], scope='Conv2d_0a_1x1')
  with tf.variable_scope('Branch_1'):
    branch_1 = slim.conv2d(net, 96, [1, 1], scope='Conv2d_0a_1x1')
    branch_1 = slim.conv2d(branch_1, 128, [5, 5], scope='Conv2d_0b_5x5')
  with tf.variable_scope('Branch_2'):
    branch_2 = slim.conv2d(net, 16, [1, 1], scope='Conv2d_0a_1x1')
    branch_2 = slim.conv2d(branch_2, 32, [3, 3], scope='Conv2d_0b_3x3')
  with tf.variable_scope('Branch_3'):
    branch_3 = slim.max_pool2d(net, [3, 3], scope='MaxPool_0a_3x3')
    branch_3 = slim.conv2d(branch_3, 32, [1, 1], scope='Conv2d_0b_1x1')
  net = tf.concat(
      axis=3, values=[branch_0, branch_1, branch_2, branch_3])
end_points[end_point] = net
if final_endpoint == end_point: return net, end_points
```

Приклад 3.1 – Inception module (файл inception_v1.py)

Вона побудована на основі «Inception модулів» (рисунок 3.2). Дизайн архітектури модуля Inception є результатом дослідження наближених розріджених структур. У кожному модулі представлено 3 ідеї:

- маючи паралельні вежі згортки з різними фільтрами з подальшою
- конкатенацією, фіксує різні характеристики на 1×1, 3×3 та 5×5, тим самим «кластеризуючи» їх;
- для зменшення розмірності використовуються 1×1 згортки для видалення вузьких вузьких місць;
- завдяки функції активації від згортки 1×1, її додавання також; додає нелінійності. ця ідея заснована на роботі «мережа в мережі».

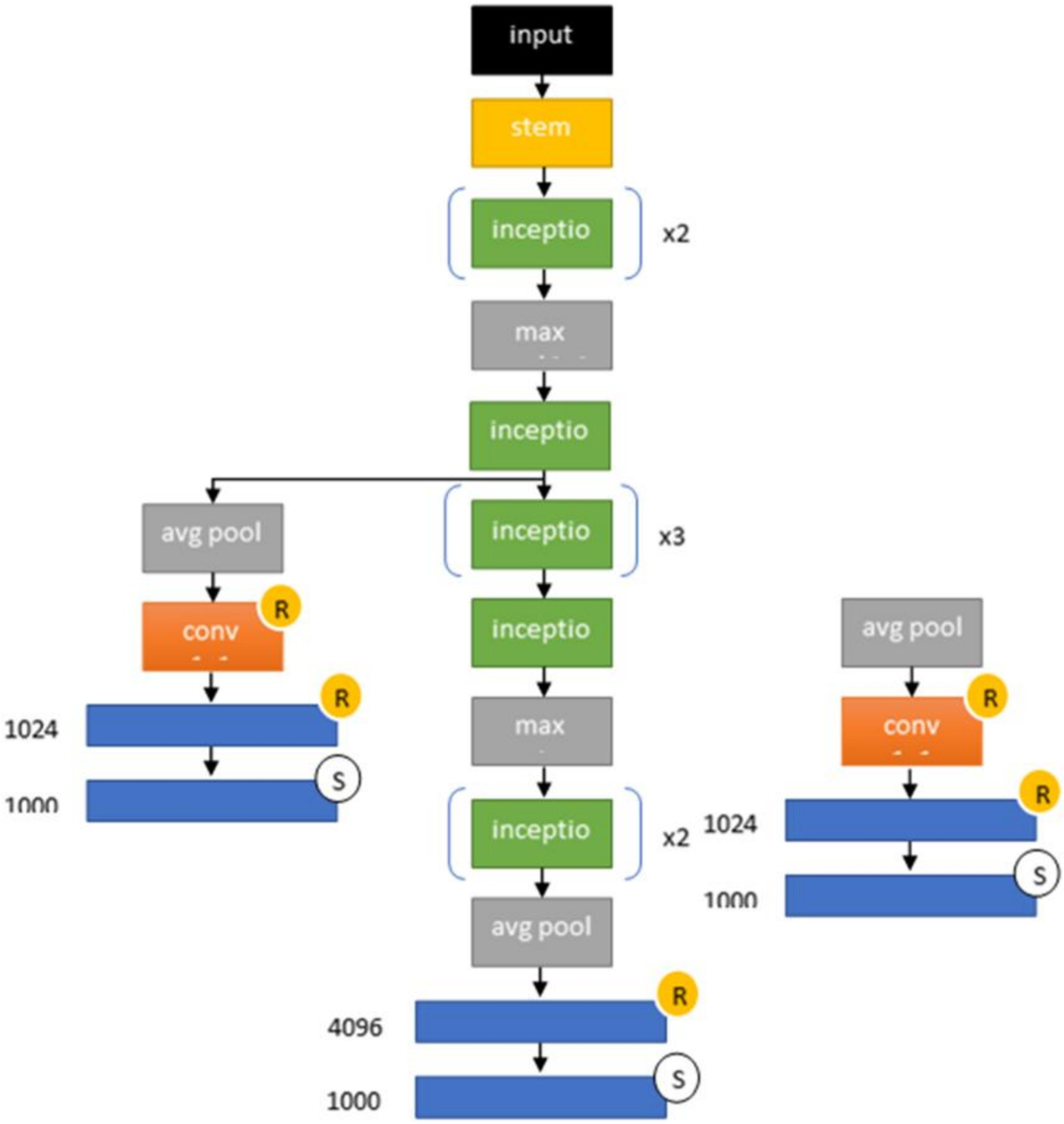

Рисунок 3.3 – Архітектура Inception_v1

Ця 22-шарова архітектура (рисунок 3.3) з 5M параметрів називається Inception-v1.

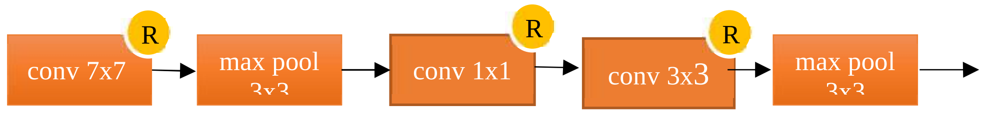

Рисунок 3.4 – Stem модуль

Автори також ввели два допоміжні класифікатори для заохочення дискримінації на нижчих стадіях класифікатора, для збільшення градієнтного сигналу, який поширюється назад, та для додаткової регуляризації. Допоміжні мережі (гілки, підключені до допоміжного класифікатора) відкидаються під час виведення результату.

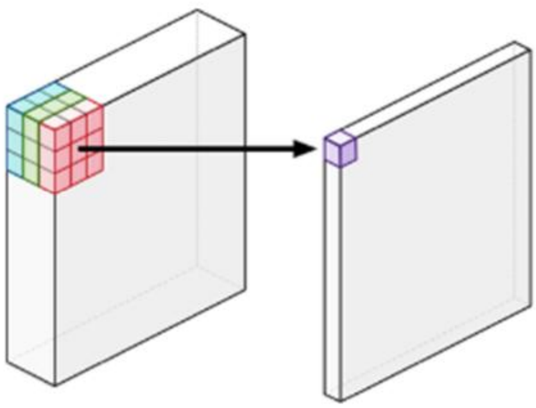

Рисунок 3.5 – Звичайна згортка

Для боротьби з градієнтами, що зникають, автори запровадили допоміжні класифікатори (Auxiliary Classifiers), які по суті застосували softmax до виходів двох Inception модулів, а також обчислили допоміжні втрати над тими ж мітками. Потім ці допоміжні втрати додаються до загальної втрати з вагою 0,3. Важливо усвідомити, що допоміжні класифікатори застосовуються лише під час тренувань.

Новою концепцією, що була додана до мережі є блок «Global Average Pooling» (рисунок 3.6) майже в кінці моделі. Раніше використовувалися шари FC, що означає, що на цьому шарі всі входи були просто підключені до кожного вихідного вузла.

Ми можемо бачити GAP як екстремальний тип зменшення розмірів, де тензор з розмірами h × w × d зменшується в розмірах, щоб мати розміри 1 × 1 × d.

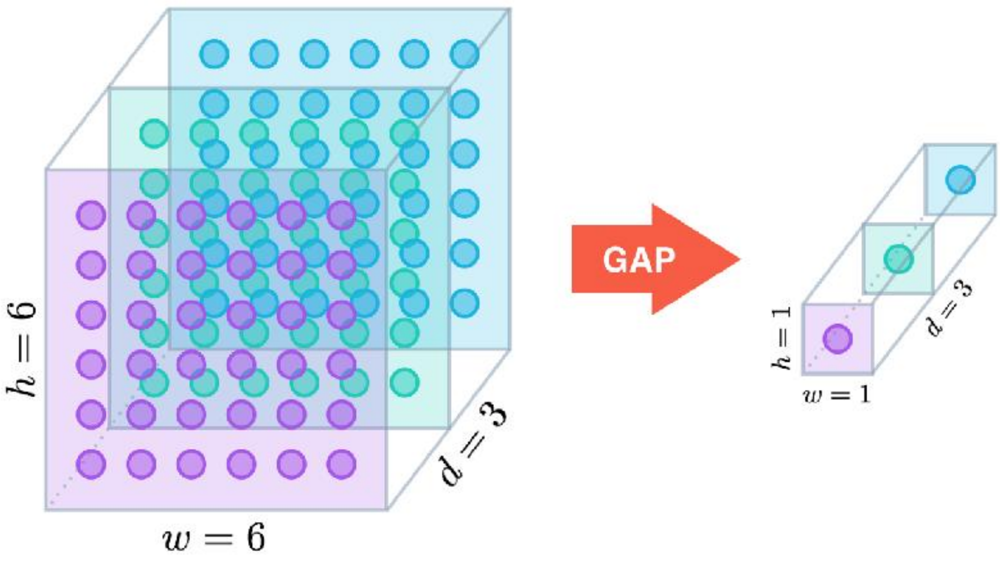

Рисунок 3.6 – Global Average Pooling блок

Це відбувається простим усередненням кожної h × w карти функції. Р 3 переваги:

- GAP порівняно з FC є більш притаманним структурі згортки;
- застосовуючи відповідність між картами зображень та категоріями. Таким чином, карти карт можна легко інтерпретувати як категорії довіри категорій;
- для оптимізації GAP непотрібен жоден параметр, і, таким чином уникається перенавчання, де, як відомо, шари FC схильні до перенавчання;

- підсумовує просторову інформацію, таким чином, вона є більш надійною для просторових перекладів вхідних даних.

Варто відзначити, що Головною ознакою цієї архітектури є покращене використання обчислювальних ресурсів всередині мережі.

3.2 Архітектура ResNet

З останніх кількох CNN ми не бачили нічого, крім збільшення кількості шарів у дизайні та досягнення кращих характеристик. Але «зі збільшенням глибини мережі точність стає насиченою, а потім швидко знижується». Люди з Microsoft Research вирішили цю проблему з ResNet - використовуючи пропускні з’єднання (рисунок 3.7) (підключення до швидкого доступу, залишки), будуючи більш глибокі моделі. Ідея полягає в тому, що замість того, щоб просто укладати додаткові шари в мережу, ми додаємо їх як залишкові блоки (мають відображення ідентичності).

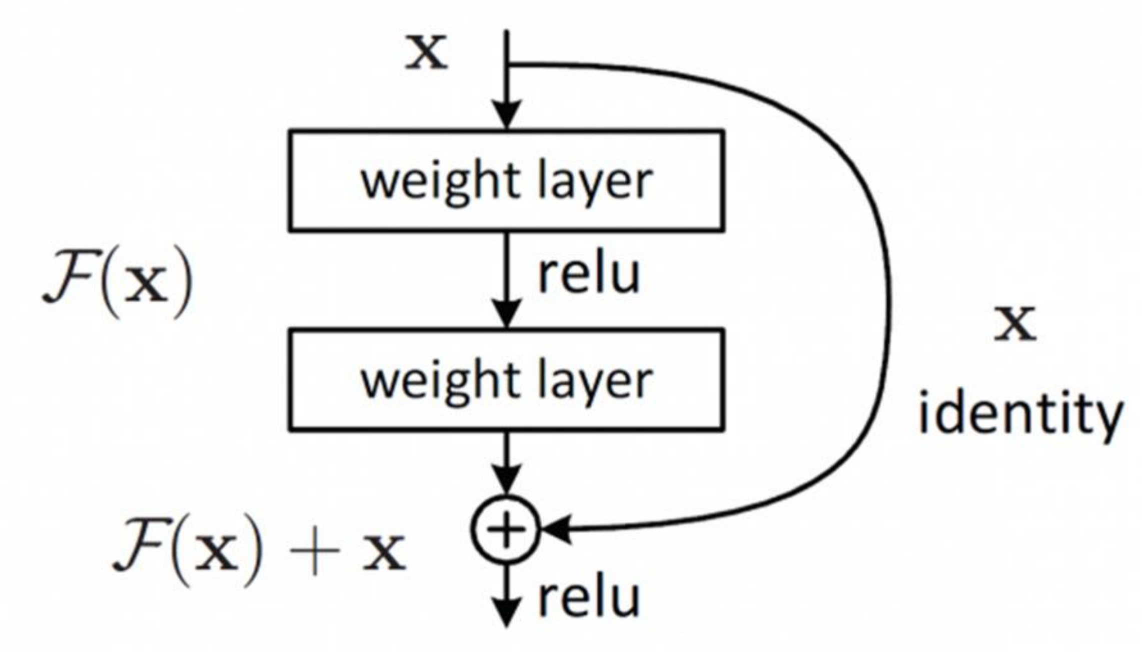

Рисунок 3.7 – Пропускні з’єднання

Щоб досягти результатів у цій схемі, автори роботи переробили основне відображення та змусили нелінійні шари вивчити відображення (формула 3.1):

$$f(x) = H(x) - x \tag{3.1}$$

замість звичайного відображення H(x) і вихідне відображення переробляють на представлене (формула 3.2):

$$f(x) + x \tag{3.2}$$

З’єднання швидкого доступу (shortcut connections) пропускають один або кілька шарів і виконують зіставлення ідентифікаторів. Їх виходи додаються до виходів stacked layers.

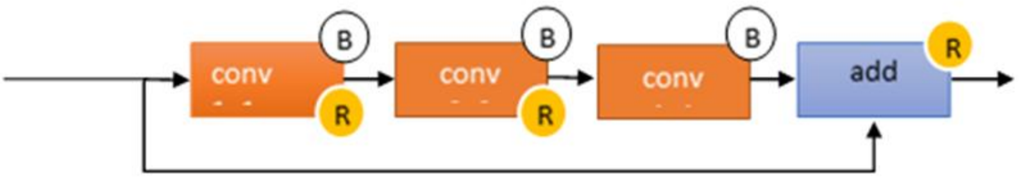

Рисунок 3.8 – Identity block

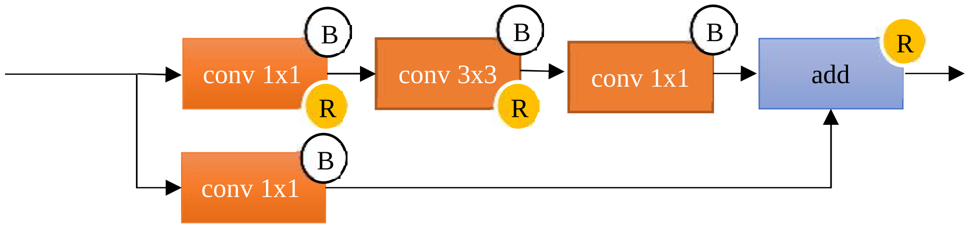

Рисунок 3.9 – Conv block

Основним будівельним блоком для ResNets є блоки conv (рисунок 3.9) та identity (рисунок 3.8). Оскільки вони схожі, алгоритм ResNet-50 спрощується (рисунок 3.10).

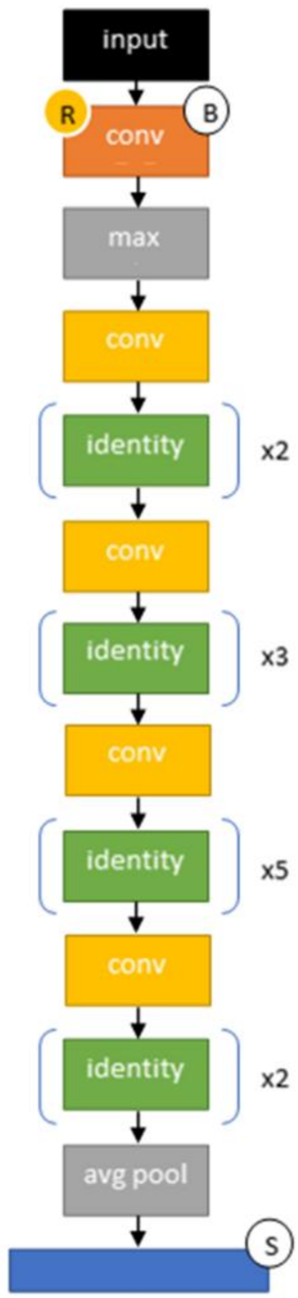

Рисунок 3.10 – ResNet-50

Ще одна перевага введення цих з’єднань ярликів полягає в тому, що вони не додають додаткових параметрів і не збільшують складність обчислень, оскільки вони використовуються як відображення ідентичності. Однак вони гарантують, що більш глибокі мережі працюють так само добре, як і на більш дрібних аналогах.

Використовуючи ResNet, можна вирішити безліч проблем, таких як:

- ResNet відносно легко оптимізувати: «прості» мережі (які просто складають шари) показують велику помилку навчання, коли глибина збільшується.

- ResNet дозволяє відносно легко збільшити точність завдяки збільшенню глибини, чого з іншими мережами домогтися складніше.

ResNet є одним з перших, хто приймає пакетну нормалізацію (документ про пакетні норми, автор автор Ioffe та Szegedy, був поданий до ICML у 2015 році).

3.3 Архітектура MobileNet_v1

MobileNets – це невеликі моделі з низькою затримкою та малою потужністю (рисунок 3.11), параметризовані для задоволення ресурсних обмежень у різних випадках використання. Вони можуть будуватися для класифікації, виявлення, вбудовування та сегментації, аналогічно тому, як використовуються інші популярні великомасштабні моделі, такі як Inception.

Рисунок 3.11 – Архітектура MobileNet_v1

Легка глибока нейронна мережа побудована за допомогою впровадженої концепції глибоко відокремлених згортків. Якщо звичайна згортка поєднує вхідні канали в один піксель, то глибока згортка виконує свою роботу на кожному каналі окремо, що означає, що кожен канал отримує відповідний набір ваг. Фільтруючи канали, ми можемо мати фільтри, такі як кольорові фільтри, але також крайові детектори та інші детектори функцій.

Можемо оцінити прямолінійність та простоту архітектури (рисунок 3.12).

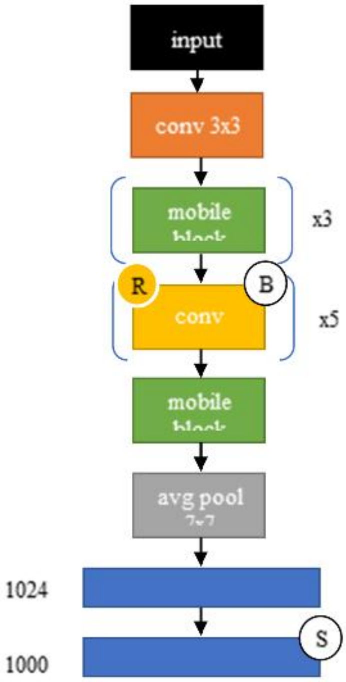

Рисунок 3.12 – Архітектура MobileNet_v1

Після згорткового шару з кроком 2 ми знаходимо 3 мобільних блоки (рисунок 3.13), які містять такі шари:

- глибинної згортки;
- точкової згортки;
- глибинної згортки з кроком 2;
- точкової згортки.

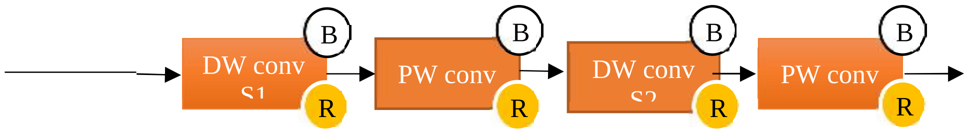

Рисунок 3.13 – Mobile block

Потім 5 згорткових блоків (рисунок 3.14) містять глибинний шар (рисунок 3.16), за яким слідує точковий шар (рисунок 3.15).

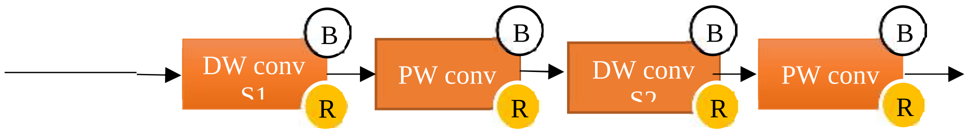

Рисунок 3.14 – Conv block

Закінчується все ще одним мобільним блоком перед об’єднанням та класифікаційним шаром.

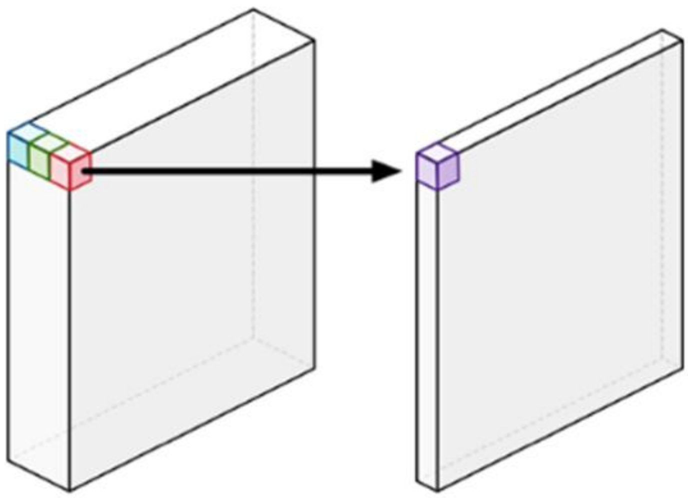

Рисунок 3.15 – Точкова згортка

Точкова згортка насправді є згорткою 1 × 1 для зміни розмірності. Якщо звичайна згортка поєднує вхідні канали в один піксель, то глибока згортка виконує свою роботу на кожному каналі окремо, що означає, що кожен канал отримує відповідний набір ваг.

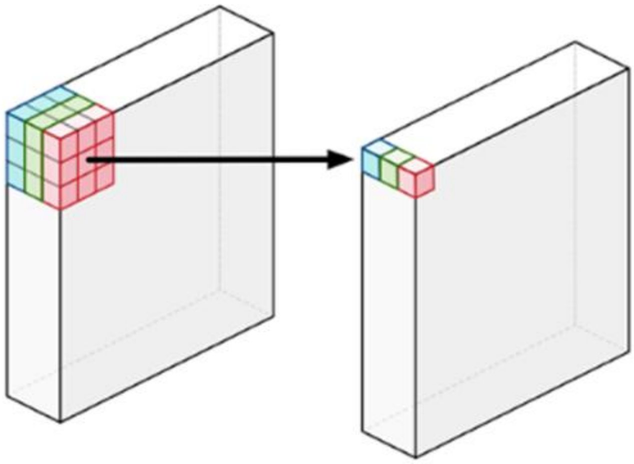

Рисунок 3.16 – Глибинна згортка

Глибинна згортка є просторовою згорткою x × x, орієнтованою на канал. Наприклад, у нас є 3 канали, тоді ми матимемо 3 x × x просторових згортки.

```
DepthSepConv(kernel=[3, 3], stride=1, depth=32),
DepthSepConv(kernel=[1, 1], stride=1, depth=32),
DepthSepConv(kernel=[3, 3], stride=2, depth=64),
DepthSepConv(kernel=[1, 1], stride=1, depth=128),
```

Приклад 3.2 – Mobile block (файл mobilenet_v1.py)

Щодо продуктивності, MobileNet (4 мільйони парам) приблизно в 3 рази швидший за Inception (24 мільйони парам) і в 10 × швидше, ніж VGGNet-16 (138 мільйонів парам). На сучасному смартфоні MobileNet може працювати на 20 FPS з точністю ImageNet 70,6%.

І завдяки цьому Google створив першу архітектуру, яка могла працювати на мобільних пристроях з розумною швидкістю та низькою обчислювальною вартістю, що також вигідно з точки зору споживання акумулятора.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

4.1 Вибір технологій для реалізації програми

4.1.1 Вибір мови програмування

Класифікація зображень складний процес, який складається з різних етапів, починаючи з обробки набору даних і закінчуючи виведенням результатів. Кожен етап супроводжується великою кількістю інформації яку потрібно обробити, тому вибір технології для реалізації програми є дуже важливим. Неправильно обрана технологія може призвести до втрати часу на кожному етапі та навіть може призвести до провалу всього проекту.

Жан Франсуа Пудже, з відділу машинного навчання IBM, висловив свою думку, що Python є найпопулярнішою мовою для AI та ML та базував її на трендових результатах пошуку на indeed.com.

Розглянемо сильні сторони Python і з’ясуємо, чому Python, це вдалий вибір, при реалізації проектів AI та ML:

1) чудова екосистема бібліотек та пакетів – бібліотека – це модуль або група модулів, опублікованих з різних джерел, таких як PyPi, які містять заздалегідь написаний фрагмент коду, який дозволяє користувачам досягати певних функцій або виконувати різні дії. Бібліотеки Python надають елементи базового рівня, тому розробникам не потрібно розробляти їх з самого початку;

2) низький бар’єр для входу – мова програмування Python нагадує повсякденну англійську мову, що полегшує процес навчання. Його простий синтаксис дозволяє комфортно працювати зі складними системами, забезпечуючи чіткі відносини між елементами системи. На додаток до цього, існує багато документації, і спільнота Python завжди готова, щоб допомогти та дати пораду;

3) гнучкість – він пропонує можливість вибрати або використовувати

об’єктно-орієнтований підхід або сценарії. Також не потрібно перекомпілювати вихідний код, розробники можуть впроваджувати будь-які зміни та швидко бачити результати. Програмісти можуть комбінувати Python та інші мови для досягнення своїх цілей;

4) незалежність від платформи – Python для розвитку машинного навчання може працювати на будь-якій платформі, включаючи Windows, MacOS, Linux, Unix та двадцять один. Щоб перенести процес з однієї платформи на іншу, розробникам потрібно здійснити кілька невеликих змін та змінити деякі рядки коду для створення виконуваної форми коду для обраної платформи. Розробники можуть використовувати такі пакети, як PyInstaller, щоб підготувати свій код для роботи на різних платформах. Це економить час та гроші на тести на різних платформах та робить загальний процес більш простим та зручним.

4.1.2 Вибір бібліотеки для машинного навчання

Розглянемо деякі ключові подібності та відмінності між двома популярними бібліотеками глибокого навчання: PyTorch та TensorFlow.

TensorFlow розроблений Google Brain і активно використовується в Google як для наукових досліджень, так і для виробничих потреб.

PyTorch – близький родич бібліотеки Torch на основі lua, який був розроблений та використаний у Facebook. Однак PyTorch - це не простий набір обгортків для підтримки популярної мови програмування, він був перероблений та адаптований, щоб бути швидким та відчувався нативно.

В даний час TensorFlow розглядається, як інструмент, дослідниками так і галузевими професіоналами. Бібліотека добре задокументована, і якщо документації не вистачить, в Інтернеті є багато надзвичайно добре написаних навчальних посібників. Можливо знайти сотні реалізованих та навчених моделей на github.

PyTorch новий у порівнянні зі своїм конкурентом, але швидко набирає обертів. Документація та офіційні підручники також приємні. PyTorch також включає декілька реалізацій популярних архітектур комп'ютерного зору, які надзвичайно прості у використанні.

Обидві бібліотеки працюють на тензорах і розглядають будь-яку модель як спрямований ациклічний графік (DAG), але вони різко відрізняються від того, як їх можна визначити.

TensorFlow відповідає ідіомі «дані як код, а код – це дані». У TensorFlow ви визначаєте графік статично перед тим, як модель може працювати. Вся комунікація із зовнішнім світом здійснюється через об'єкт tf.Session та tf.Placeholder, які є тензорами, які під час виконання будуть заміщені зовнішніми даними. У PyTorch все набагато імперативніше та динамічніше: ви можете визначати, змінювати та виконувати вузли під час проходження, без спеціальних інтерфейсів сеансу чи заповнювачів. Загалом, рамки більш тісно інтегровані з мовою Python і більшу частину часу відчувають себеріднішими. Коли ви пишете в TensorFlow, іноді ви відчуваєте, що ваша модель знаходиться за цегляною стіною з кількома крихітними отворами для спілкування.

Однак ці підходи відрізняються не лише з точки зору інженерії програмного забезпечення: існує кілька динамічних архітектур нейронних мереж, які можуть отримати користь від динамічного підходу. Нагадаємо RNN: зі статичними графіками довжина послідовності вводу залишатиметься незмінною. Це означає, що якщо ви розробляєте модель аналізу настроїв для англійських речень, ви повинні зафіксувати довжину речення до деякого максимального значення та обклеїти всі менші послідовності нулями. Не надто зручно, так. І ви отримаєте більше проблем у домені рекурсивних RNN та дерев-RNN. В даний час Tensorflow має обмежену підтримку динамічних входів через Tensorflow Fold. У PyTorch це за замовчуванням.

Оскільки графік обчислень у PyTorch визначений під час виконання, ви можете використовувати наші улюблені інструменти налагодження Python, такі як pdb, ipdb, налагоджувач PyCharm або старі надійні заяви друку. Це не так з TensorFlow. У вас є можливість використовувати спеціальний інструмент під назвою tfdbg, який дозволяє оцінювати вирази tensorflow під час виконання та переглядати всі тензори та операції в межах сеансу. Звичайно, ви не зможете налагодити з ним жодний код python, тому потрібно буде використовувати pdb окремо.

Tensorboard є приголомшливою, коли мова йде про візуалізацію. Цей інструмент поставляється з TensorFlow, і він дуже корисний для налагодження та порівняння різних тренувань. Наприклад, розгляньте, що ви тренували модель, потім налаштували кілька гіперпараметрів і знову її навчили. Обидва запуски можуть відображатися на Tensorboard одночасно, щоб вказати на можливі відмінності. Тензорна дошка може:

- відобразити графік моделі;
- діаграма скалярних змінних;
- візуалізуйте розподіли та гістограми;
- візуалізуйте зображення;
- візуалізуйте вставки;
- відтворити аудіо.

Тензорна дошка може відображати різні підсумки, які можна зібрати за допомогою модуля tf.summary. Ми визначимо підсумкові операції для прикладу нашого показника іграшок та використаємо tf.summary.FileWriter для збереження їх на диску. Цей інструмент дуже зручно використовувати у хмарних екземплярах, оскільки це веб-версія. Конкурентом із tansorboard з боку PyTorch є visdom. Він не такий повнофункціональний, але трохи зручніший у використанні. Також є інтеграції з Tensorboard. Крім того, є можливість безкоштовно використовувати стандартні інструменти для побудови графіків – matplotlib та seaborn.

Якщо ми почнемо говорити про розгортання, TensorFlow зараз є явним переможцем: чи є TensorFlow Serving, який є основою для розгортання ваших моделей на спеціалізованому сервері gRPC. Мобільний також підтримується. При повененні назад до PyTorch, є можливість використовувати Flask або іншу альтернативу для кодування REST API поверх моделі. Це може бути зроблено і з моделями TensorFlow, якщо gRPC не підходить для вашого використання. Однак TensorFlow Serving може бути кращим варіантом, якщо продуктивність викликає занепокоєння. Tensorflow також підтримує розподілене навчання, якого PyTorch наразі не вистачає.

Однією з найбільших особливостей, що відрізняє PyTorch від TensorFlow, є декларативний паралелізм даних: є можливість використовувати torch.nn.DataParallel для обгортання будь-якого модуля, і він буде (майже магічно) паралелізований над пакетним розміром. Таким чином, є можливість використовувати декілька графічних процесорів майже без зусиль.

З іншого боку, TensorFlow дозволяє точно налаштувати кожну операцію, яку потрібно виконати на певному пристрої. Тим не менш, визначення паралелізму є набагато ручнішим і вимагає ретельного продумування. Зважаючи на це, використовуючи TensorFlow, є можливість досягти всього, що є можливість зробити в PyTorch, але з більшими зусиллями (у вас є більший контроль як бонус). Також варто зазначити, що обидві бібліотеки підтримують розподілене виконання та надають інтерфейси високого рівня для визначення кластерів.

Розглянемо порівняння Фреймворк або бібліотека, де програма Фреймворк дає нам корисні абстракції у певній галузі та зручний спосіб їх використання для вирішення конкретних проблем. У цьому полягає суть, яка відокремлює фреймворк від бібліотеки. Таким чином, і TensorFlow, і PyTorch забезпечують корисні абстракції для зменшення кількості кодового коду та прискорення розробки моделі. Основна відмінність між ними полягає в тому, що PyTorch може відчувати себе більш «пітонічним» і має об’єктно-орієнтований підхід, тоді як TensorFlow має кілька варіантів, з яких є можливість обрати.

TensorFlow – це дуже потужна та зріла бібліотека для глибокого навчання з сильними можливостями візуалізації та кількома варіантами, які можна використовувати для розробки моделі високого рівня. Він має готові до виробництва варіанти розгортання та підтримку мобільних платформ. Якщо поєднати ці фактори та врахувати те, що ця бібліотека має гарну підтримки громади, вичерпну документацію та багаті навчальні ресурси в різних формах (TensorFlow має цілий MOOC), то стає зрозуміло, що ця бібліотека найбільше підходить для реалізації поточних завдань.

4.2 Аналіз результатів експериментів

Нейронна мережа з глибоким навчанням тренується на навчальному наборі, як правильно відображати набір вхідних множин на набір множини результату.

Таблиця 4.1 Результати оцінки мережей

Model	File	Top-1 Accuracy	Training time
Inception V1	Inception_v1.py	86.6	4
ResNet_50	ResNet_v1.py	80.4	7
MobileNet V1	MobileNet_v1.py	73.6	5

Основною одиницею виміру ефективності нейронної мережі прийнято вважати точність. Точність – це частка передбачень, яку наша модель зробила правильно. Формально точність має таке визначення:

$$A = \frac{N \quad o \; c \quad p \quad ct}{T \quad n \quad o \; p} \qquad (4.1)$$

Проблема з використанням точності як основної метрики ефективності полягає в тому, що вона не справляється добре, коли є сильний дисбаланс класу. Хороша альтернатива, яку слід враховувати, - це використовувати похибку для оцінки та діагностики того, наскільки добре навчається модель.

Сюди входять усі міркування процесу оптимізації, такі як перенавчання, недонавчання та конвергенція. Проблема навчання подається як проблема пошуку або оптимізації, і алгоритм використовується для розподілу можливих наборів ваг, які модель може використовувати для того, щоб робити досить хороші або хороші прогнози. Не можливо розрахувати ідеальні ваги для нейронної мережі; занадто багато невідомих.

Зазвичай модель нейронної мережі навчається за допомогою алгоритму оптимізації стохастичного градієнта спуску, а ваги оновлюються за допомогою алгоритму зворотного розповсюдження помилок.

«Градієнт» у градієнтному спуску відноситься до градієнта помилки. Модель із заданим набором ваг використовується для прогнозування і обчислюється похибка для цих прогнозів.

Алгоритм спуску градієнта прагне змінити ваги, щоб наступна оцінка зменшила помилку, тобто алгоритм оптимізації рухався вниз по градієнту (або нахилу) помилки. Чим менша похибка, тим краща модель (якщо тільки модель не надто навчена на даних тренувань). Похибка обчислюється при навчанні та валідації, і його інтерпретація полягає в тому, наскільки добре працює модель для цих двох наборів.

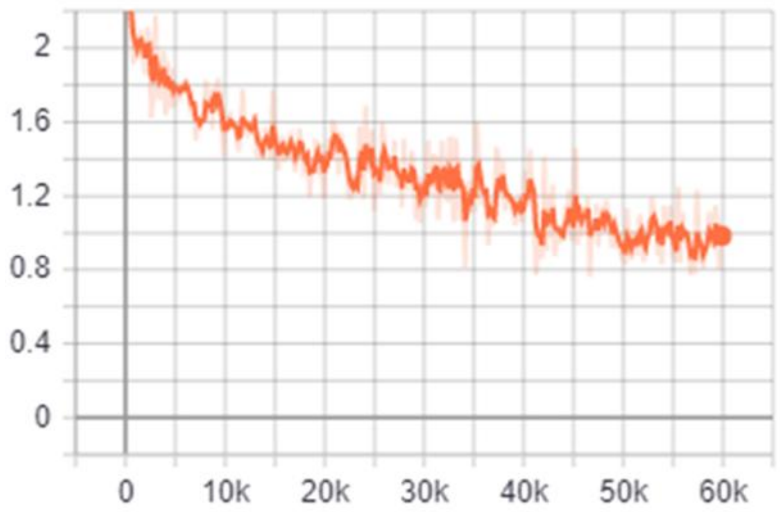

Рисунок 4.1 – Підсумок помилок mobilenet_v1

На відміну від точності, похибка не є відсотком. Це підсумок помилок, зроблених для кожного прикладу в навчальних наборах чи валідаціях.

Як можна побачити (рисунок 4.1) нейронна мережа mobilenet_v1, яка була тренована на наборі даних cifar10, відображає повільне зниження похибки протягом всього навчання, проте починаючи приблизно за шагу 40k, похибка коливається в рамках від 0,9 до 1,1.

```
python train_image_classifier.py
      --train_dir="/tmp/mobilenet_v1_cifar10"
      --dataset_name=cifar10
      --dataset_split_name=train
      --dataset_dir=/tmp/data/cifar10
      --model_name=mobilenet_v1
      --preprocessing_name=mobilenet_v1
      --max_number_of_steps=60000
      --batch_size=50
      --learning_rate=0.01
      --save_interval_secs=60
      --save_summaries_secs=60
      --log_every_n_steps=100
      --optimizer=sgd
      --learning_rate_decay_type=fixed
      --weight_decay=0
python eval_image_classifier.py
      --checkpoint_path=/tmp/mobilenet_v1_cifar10
      --eval_dir=/tmp/mobilenet_v1_cifar10_eval
      --dataset_name=cifar10
      --dataset_split_name=test
      --dataset_dir=/tmp/data/cifar10
      --model_name=mobilenet_v1
```

Приклад 4.1 – Скрипт навчання та валідації mobilenet_v1

В найкращому сценарії значення похибки повинне знижуватися на протязі всього навчання, тому така поведінка нейронної мережі дає розуміння, що модель вже достатньо натренована на навчальних даних і подальше навчання може призвести до того, що модель буде перенавчена для конкретного набору даних. Це в свою чергу матиме наслідки при використанні навченої моделі не на даних з конкретного набору, а на якихось сторонніх даних. З боку точності ця нейронна мережа показала результат у 73.6 відсотків, що є дуже високим результатом звачаючи на те, що ця мобільна мережа є меньшою по кількості згорток серед усіх перелічених нейронних мереж та орієнтована на мобільні телефони, які, навідміну від стаціонарних комп'ютерів, не мають таких великих потужностей.

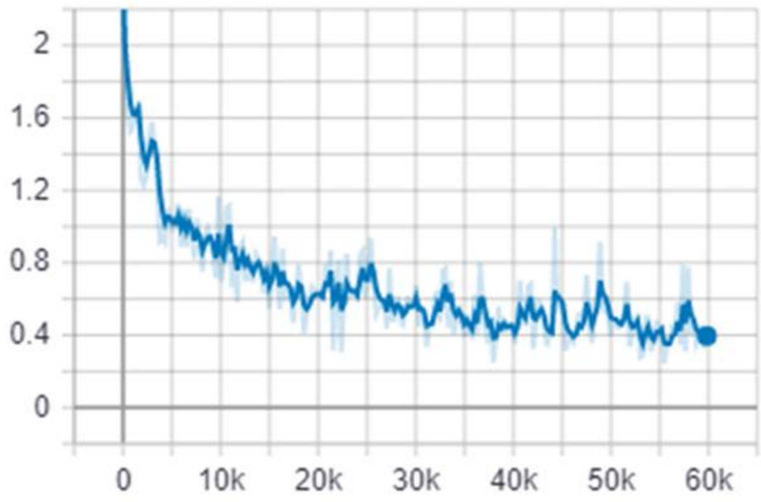

Рисунок 4.2 – Підсумок помилок inception_v1

```
python train_image_classifier.py
      --train_dir="/tmp/inception_v1_cifar10"
      --dataset_name=cifar10
      --dataset_split_name=train
      --dataset_dir=/tmp/data/cifar10
      --model_name=inception_v1
      --preprocessing_name=inception_v1
      --max_number_of_steps=60000
      --batch_size=50
      --learning_rate=0.01
      --save_interval_secs=60
      --save_summaries_secs=60
      --log_every_n_steps=100
      --optimizer=sgd
      --learning_rate_decay_type=fixed
      --weight_decay=0
python eval_image_classifier.py
      --checkpoint_path=/tmp/inception_v1_cifar10
      --eval_dir=/tmp/inception_v1_cifar10_eval
      --dataset_name=cifar10
      --dataset_split_name=test
      --dataset_dir=/tmp/data/cifar10
      --model_name=inception_v1
```

Приклад 4.2 – Скрипт навчання та валідації inception_v1

Для нейронної мережі inception_v1 за перші 10к тренувальних даних бачимо стрімке падіння помілки з 2 до 0.9 і згодом повільне зменшення на до самого кінця навчання. Графік (рисунок 4.2) демонструє розмірене випрямлення показників помилки, що свідчить про те що мережа на протязі всього навчання корегувала свої ваги так, що фінальна величина помилки склала 0,39. Це також дуже добре відображено показником точності, яке проміж трьох мереж є найбільшим і складає 86.6.

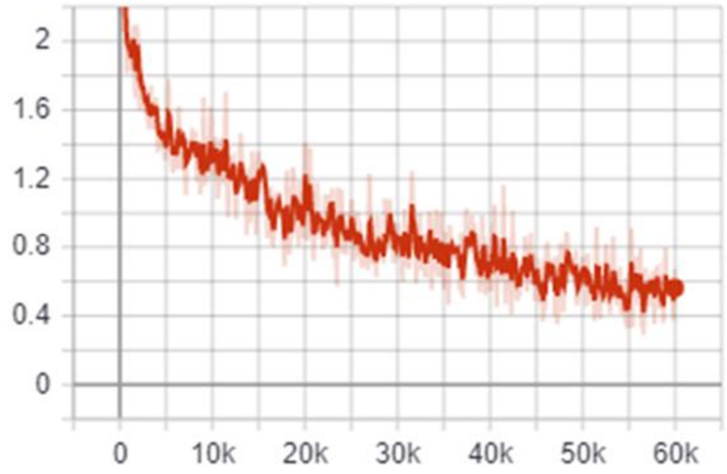

Рисунок 4.3 – Підсумок помилок resnet_v1_50

```
python eval_image_classifier.py
      --checkpoint_path=/tmp/resnet_v1_50_cifar10
      --eval_dir=/tmp/resnet_v1_50_cifar10_eval
      --dataset_name=cifar10
      --dataset_split_name=test
      --dataset_dir=/tmp/data/cifar10
      --model_name=resnet_v1_50
```

Приклад 4.3 – Скрипт валідації mobilenet_v1

Нейронна мережа resnet_v1_50 в свою чергу за продовженістю навчання посіла впевнене перше місце з великим відривом від inception_v1, проте це через те, що вона має більше згорот у своїй струтурі аніж інші мережі і це дало свій результат. Як можна побачити на графіку (рисунок 4.3) мережа демострує стабільне зменьшення похибки на протязі всього навчання без різких скачків на початку та з досить непоганою величиною помилки у кінці, яка становить 0,56.

```
python train_image_classifier.py
--train_dir="/tmp/resnet_v1_50_cifar10"
--dataset_name=cifar10
--dataset_split_name=train
--dataset_dir=/tmp/data/cifar10
--model_name=resnet_v1_50
--preprocessing_name=resnet_v1_50
--max_number_of_steps=60000
--batch_size=50
--learning_rate=0.01
--save_interval_secs=60
--save_summaries_secs=60
--log_every_n_steps=100
--optimizer=sgd
--learning_rate_decay_type=fixed
--weight_decay=0
```

Приклад 4.4 – Скрипт тренування mobilenet_v1

З такими даними після навчання ця мережа на тестовому наборі даних змогла показати 80.4% точності, що хоч і не є найбільшим серед згаданих мереж, проте є дуже високим.

ВИСНОВКИ

Під час роботи над атестаційною роботою, був проведений огляд і аналіз літератури по темі магістерської атестаційної роботи. Було розглянуто поняття нейрона, його нелінійну модель, яка включає набір синапсів, суматор та функцію активації. Так як існує велика кількість функцій активації, то були розглянуті ті, які використовуються на практиці. Виявилося, що сигмоїда, яка часто використовується через подібність до рівня активації нейрона, має недоліки пов’язані з подавленням градієнта. Натомість краще використовувати гіперболічний тангенс, бо його значення на виході буде завжди відцентровано відносно нуля. Проте обчислення сигмоїд і гіперболічного тангенса – це ресурсномісткі операції і на порятунок приходить функція активації ReLu, але треба пам’ятати, шо ця функція теж має недоліки, які пов’язані з втратою ReLU під час навчання нейронної мережі.

В ході проведення експерименту було виявлено, що Inception_v1. продемонстрував найкращий результат серед розглянутих мереж. В свою чергу MobileNet показав дуже хороший результат, не зважаючи на те, що це меньша за розміром мережа і вона розрахована в першу чергу на мобільні пристрої. Також було виявлено, що більша кількість слоїв в мережі суттєво впливає на час тренування мережі і попередня обробка зображень, впливає на швидкість роботи моделей та зменшує обчислювальну складність.

Якщо продовжити роботу над цими моделями, то результати можливо покращити, і це можливо зробити за рахунок визначення коефіцієнтів масштабування для оптимізації розміру нейромережі, а також за рахунок додавання пропускних з’єднань і використання пакетної нормалізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Хайкин С. Нейронные сети: полный курс Neural Networks: A Comprehensive Foundation. 2-е изд. – М.: Вильямс, 2006. – 1104 с. – ISBN 0-13-273350-1.
2. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия - Телеком, 2001. – 382 с. – ISBN 5-93517-031-0.
3. 7 types of Artificial Neural Networks for Natural Language Processing. [Електронний ресурс]. – Режим доступу: www/ URL: https://medium.com /@datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2
4. Глава 3. Основы ИНС. [Електронний ресурс]. – Режим доступу: www/ URL:https://neuralnet.info/chapter/%d0%be%d1%81%d0%bd%d0%be%d0%b2%d1%8b-%d0%b8%d0%bd%d1%81/
5. Ключевые рекомендации по глубокому обучению (Часть 2). [Електронний ресурс]. – Режим доступу: www/ URL: http://datareview.info /article/eto-nuzhno-znat-klyuchevyie-rekomendatsii-po-glubokomu-obucheniyu-chast-2/
6. K. Smelyakov, M. Shupyliuk, V. Martovytskyi, D. Tovchyrechko and O. Ponomarenko, «Efficiency of image convolution», 2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL), Sozopol, Bulgaria, 2019, P. 578-583, doi: 10.1109/CAOL46282.2019.9019450.
7. Milan Sonka, Vaclav Hlavac, Roger Boyle, Image Processing, Analysis, and Machine Vision (4th ed.). – Cengage Learning, 2014. – P. 896
8. Smelyakov K., Sandrkin D., Ruban I., Martovytskyi V., Romanenkov Y. Search by Image. New Search Engine Service Model / International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), 9-12 Oct. 2018 (Kharkiv, Ukraine). – P. 181-186.
9. Igor Ruban, Kirill Smelyakov, Martovytskyi Vitalii, Pribylnov Dmitry,

Nataliia Bolohova Method of neural network recognition of ground-based air objects // IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), 24-27 May. – 2018. – P. 589-592.

10. Rafael C. Gonzalez, Richard E. Woods Digital Image Processing, 4th edition Pearson/Prentice Hall, 2018. – P. 1168

11. David A. Forsyth, Jean Ponce Computer Vision: A Modern Approach (2nd ed.). – Pearson Education Limited, 2015. – P. 792

12. Arsenov A., Ruban I., Smelyakov K., Chupryna A. Evolution of Convolutional Neural Network Architecture in Image Classification Problems // Selected Papers of the XVIII International Scientific and Practical Conference on IT and Security (ITS 2018). – CEUR Workshop Processing. – Kyiv, Ukraine, November 27, 2018. – P. 35-45.

13. Kirill Smelyakov, Dmytro Yeremenko, Anton Sakhon, Vitalii Polezhai, Anastasiya Chupryna Braille character recognition based on neural networks // IEEE Second International Conference on Data Stream Mining & Processing (DSMP), August 21-25. – 2018. – P. 509-513.

14. Lemeshko O., Yeremenko O. Enhanced method of fast re-routing with load balancing in software-defined networks // Journal of ELECTRICAL ENGINEERING. – 2017. – Vol. 68, Iss. 6. – P. 444-454.

15. Tkachov V.M., Savanevych V. Method for transfer of data with intermediate storage // IEEE First International Scientific-Practical Conference «Problems of Infocommunications. Science and Technology» (PICS&T-2014). – October 14-17, 2014. – P. 105-106.

16. V. Mukhin, Yu. Romanenkov, Ju. Bilokin, A. Rohovyi, A. Kharazii, V. Kosenko, N. Kosenko, Ju. Su The Method of Variant Synthesis of Information and Communication Network Structures on the Basis of the Graph and Set-Theoretical Models // International Journal of Intelligent Systems and Applications (IJISA), Vol.9, No.11, P. 42-51, 2017.