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 розпізнавання у вбудованих комп’ютерних системах
(тема)

Виконав:
студент II курсу, групи СПзм-18-2
Ващенко І.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: проф. Смеляков К.С.
(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 Ващенку Івану Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Нейромережеві моделі розпізнавання у вбудованих комп’ютерних системах

затверджена наказом по університету від “ 30 ” березня 2020 р. № 43 Стз

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

3. Вхідні дані до роботи
Типи зображень і вимоги до нейромереж
Вимоги до розпізнавання зображень
Вимоги до якості результатів розпізнавання

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області
Вибір наборів даних й моделей нейромереж
Навчання нейромережі
Обчислювальний експеримент
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати – 12 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 – 12.04.20	
2	Постановка задачі дослідження	13.04.20 – 15.04.20	
3	Аналіз літератури	16.04.20 – 20.04.20	
4	Вибір наборів даних	21.04.20 – 24.04.20	
5	Вибір моделей нейромереж	25.04.20 – 28.04.20	
6	Експеримент і аналіз отриманих результатів	29.04.20 – 03.05.20	
7	Оформлення пояснювальної записки	04.05.20 – 11.05.20	
8	Подання атестаційної роботи керівникові	12.05.20 – 14.05.20	
9	Рецензування атестаційної роботи	15.05.20 – 17.05.20	
10	Подання атестаційної роботи до ЕК	18.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Смеляков К.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 78 с., 42 рис., 0 табл., 1 додаток, 24 джерел.

СХОВИЩЕ ДАНИХ, НЕЙРОННА МЕРЕЖА, ІДЕНТИФІКАЦІЯ МОДЕЛІ, МАШИННЕ НАВЧАННЯ, ФУНКЦІЯ АКТИВАЦІЇ, КАСКАДНА МОДЕЛЬ, ЛОКАЛІЗАЦІЯ ОБЛИЧЧЯ, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ОБЧИСЛЮВАЛЬНА ПОТУЖНІСТЬ.

Метою атестаційної роботи є вибір й настроювання параметрів нейронної мережі в процесі навчання, а також створення відповідної системи для знаходження обличчя людини на цифровому зображенні для умов застосування у межах вбудованої комп’ютерної системи.

У ході виконання атестаційної роботи було проаналізовано існуючі моделі та архітектури найбільш відомих нейронних мереж, алгоритми машинного навчання і відповідні технології, обрано модель знаходження обличчя для вбудованої комп’ютерної системи, а також відповідний програмний комплекс, за допомогою якого поставлено експеримент і отримано підтвердження ефективності застосування запропонованої моделі для умов реального часу для вбудованої комп’ютерної системи знаходження обличчя на цифровому зображенні.

Проведений аналіз застосування запропонованої нейромережевої технології показав її ефективність для умов застосування у межах вбудованої комп’ютерної системи.

ABSTRACT

Master’s thesis: 78 pages, 42 figures, 0 tables, 1 appendices, 24 sources.

DATA STORAGE, NEURAL NETWORK, MODEL IDENTIFICATION, MACHINE LEARNING, ACTIVATION FUNCTION, CASCADE MODEL, FACE LOCALIZATION, FACE RECOGNITION, CALCULATING POWER.

The purpose of the thesis is to select and adjust neural network parameters during training, as well as to create an appropriate system for detecting a person’s face in a digital image for conditions of use within the embedded computer system.

In the thesis the existing models and architectures of the most famous neural networks were analyzed, machine learning algorithms and corresponding technologies, the face finding model for the embedded computer system was selected, as well as the corresponding software complex, by which the experiment was performed and confirmation of the effectiveness of the proposed application was obtained. Real-time models for embedded digital face-finding.

An analysis of the application of the proposed neural network technology has shown its effectiveness for the conditions of use within the embedded computer system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АД – автоматичне диференціювання

МЛ – марковський ланцюг

МПВ – марковський процес вирішування

ШНМ – штучні нейронні мережі

ВСТУП

В теперішній час моделі, алгоритми і технології комп’ютерного зору знаходять в більшій мірі застосування в найрізноманітніших сферах професійної діяльності. В системі аналізу зразків тканин в медицині, в фармакології, в системах біометричної ідентифікації людини по радужці ока, в охоронних системах спостереження, а також у багатьох інших системах і додатках [1, 2].

В зв’язку з ростом обчислювальних потужностей сучасних систем і сервісів, а також – у зв’язку з появою великих сховищ зображень стало можливим навчання і використання глибоких нейронних мереж для вирішення задач класифікації та розпізнавання. У завдані класифікації зображень стосовно обличчя людини краще всього показали себе згорткові мережі MTCNN, R-CNN, YOLO. А також їх більш сучасні модифікації. У зв’язку з розкриттям своїх можливостей ці нейронні мережі все ширше застосовуються для вирішення задач розпізнавання, знаходження зображень облич, а деякі з них – для виявлення предметів, семантичної сегментації та інших задач [3-5].

У цьому відношенні в роботі розглядається еволюція і використання найефективніших архітектурних згорткових нейронних мереж, що використовуються в цей час для знаходження зображень обличчя. Перш за все – це мережі MTCNN, YOLO. Тому, що мережа R-CNN потребує значно більших ресурсів. Що для вбудованих систем є неприйнятним [1, 5].

Згорткова нейронна мережа – це така нейронна мережа, в якій присутній шар згортки (convolutional layer). Зазвичай в згорткових нейронних мережах також присутні шар субдіскретізації (pooling layer) і повнозв’язний шар (fully connected layer). Згорткові нейронні мережі застосовуються для розпізнавання образів [6, 7], детектування предметів, класифікації зображень, семантичної сегментації та інших завдань [8-10]. У згорткових нейронних

мережах шари згортки і субдіскретізації складаються з декількох «рівнів» нейронів, які називаються картами ознак (feature maps), або каналами (channels). Кожен нейрон такого шару з’єднаний з невеликою ділянкою попереднього шару, званим рецептивним полем. У разі зображення, карта ознак є двовимірним масивом нейронів, або просто матрицею. Інші вимірювання можуть бути використані, якщо на вхід приймається інший вид даних, наприклад, аудіо дані (одновимірний масив) або об’ємні дані (тривимірний масив) [2].

1 МОДЕЛІ ТА ТЕХНОЛОГІЇ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

1.1 Нейронні мережі

Штучні нейронні мережі (ШНМ), або конективістські системи – це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин (рисунок 1.1) [11]. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють.

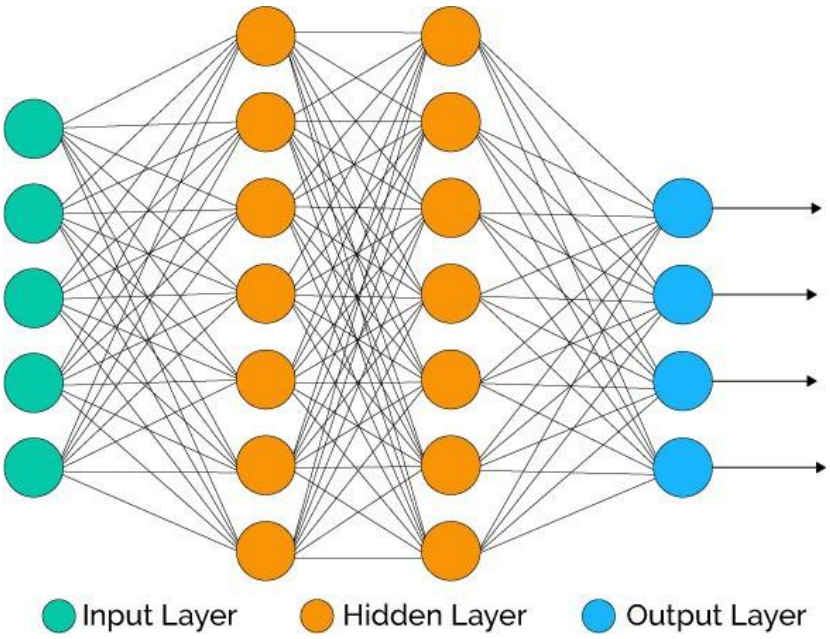


Рисунок 1.1 – Структура нейронної мережі [11]

ШНМ ґрунтується на сукупності з’єднаних вузлів, що називають штучними нейронами (аналогічно до біологічних нейронів у головному мозку тварин). Кожне з’єднання (аналогічне синапсові) між штучними нейронами може передавати сигнал від одного до іншого. Штучний нейрон, що отримує сигнал, може обробляти його, й потім сигналізувати штучним нейронам, приєднаним до нього.

В поширених реалізаціях ШНМ сигнал на з’єднанні між штучними нейронами є дійсним числом, а вихід кожного штучного нейрону обчислюється нелінійною функцією суми його входів. Штучні нейрони та з’єднання зазвичай мають вагу, яка підлаштовується в перебігу навчання. Вага збільшує або зменшує силу сигналу на з’єднанні. Штучні нейрони можуть мати такий поріг, що сигнал надсилається лише якщо сукупний сигнал перетинає цей поріг. Штучні нейрони зазвичай організовано в шари. Різні шари можуть виконувати різні види перетворень своїх входів. Сигнали проходять від першого (вхідного) до останнього (вихідного) шару, можливо, після проходження шарами декілька разів.

Первинною метою підходу ШНМ було розв’язання задач таким же способом, як це робив би людський мозок. З часом увага зосередилася на відповідності певним розумовим здібностям, ведучи до відхилень від біології. Штучні мережі використовували в ряді різноманітних задач, включно з комп’ютерним баченням, розпізнаванням мовлення, машинним перекладом, соціально-мережевим фільтруванням, грою в настільні та відеоігри, та медичним діагностуванням.

Нейронні мережі містять в собі нейрони, які складаються з:

- збудження, що залежить від дискретного параметра часу;
- можливо, порогу, що залишається незмінним, якщо його не змінить функція навчання;
- функції збудження f, яка обчислює нове збудження в заданий час;
- та функції виходу, яка обчислює вихід з активації.

Нейрон входу не має попередників, а слугує інтерфейсом входу для всієї мережі. Аналогічно, нейрон виходу не має наступників, і відтак слугує інтерфейсом виходу для всієї мережі.

Мережа складається зі з'єднань, кожне з яких передає вихід нейрону і до входу нейрону j. В цьому сенсі і є попередником j, а j є наступником і. Кожному з'єднанню призначено вагу w.

Також нейронна мережа містить в собі функцію поширення, яка обчислює вхід до нейрону j з виходів нейронів-попередників, і зазвичай має вигляд

$$p_j(\mathrm{t}) = \sum_{i=0}^{n} o_i(t) w_{ij} \tag{1.1}$$

Правило навчання – це правило або алгоритм, який змінює параметри нейронної мережі, щоби заданий вхід до мережі видавав придатний вихід. Цей процес навчання зазвичай полягає в зміні ваг та порогів змінних мережі.

Нейромережеві моделі можна розглядати як прості математичні моделі, що визначають функцію $f: X \to Y$, або розподіл над X, або над X та Y. Іноді моделі тісно пов'язують з певним правилом навчання. Поширене використання фрази «модель ШНМ» насправді є визначенням класу таких функцій (де членів цього класу отримують варіюванням параметрів, ваг з'єднань, або особливостей архітектури, таких як число нейронів або їхня зв'язність).

З математичної точки зору, нейромережеву функцію f(x) визначають як композицію інших функцій $g_i(x)$, які може бути розкладено далі на інші функції. Це може бути зручно представляти як мережеву структуру, де стрілки зображують залежність між функціями. Широко вживаним способом компонування є нелінійна зважена сума, де $f(x) = \mathrm{K}(\sum_i w_i g_i(x))$ (що часто називають функцією збудження) є визначеною наперед функцією, такою як гіперболічний тангенс, або сигмоїдна функція, або нормована експоненційна

функція, або випрямляльна функція. Важливою характеристикою функції збудження є те, що вона забезпечує плавний перехід при зміні значень входу, тобто, невелика зміна входу призводить до невеликої зміни виходу. Наведене нижче розглядає набір функцій g_i як вектор $g = (g_1, g_2, \ldots, g_n)$.

Ця схема зображує такий розклад f, із залежностями між змінними, показаними стрілками. Їх може бути інтерпретовано двома способами.

Перший погляд є функціональним: вхід x перетворювало на 3-вимірний вектор h, який відтак перетворювало на 2-вимірний вектор g, який нарешті перетворювало на f. Цей погляд найчастіше зустрічається в контексті оптимізації.

Другий погляд є ймовірнісним: випадкова змінна $F = f(G)$ залежить від випадкової змінної $G = g(H)$, яка залежить від $H = h(X)$, яка залежить від випадкової змінної X. Цей погляд найчастіше зустрічається в контексті графових моделей.

Ці два погляди є здебільшого рівнозначними. В кожному з випадків, для цієї конкретної архітектури, складові окремих шарів не залежать одна від одної (наприклад, складові g є незалежними одна від одної за заданого їхнього входу h). Це природно уможливлює якусь міру паралелізму в реалізації.

Такі мережі, як попередня, зазвичай називають мережами прямого поширення, оскільки їхній граф є спрямованим ациклічним графом. Мережі з циклами зазвичай називають рекурентними. Такі мережі зазвичай зображують у спосіб, де f показано як залежну від самої себе. Проте, не показано часову залежність, що мається на увазі.

Найбільше зацікавлення нейронними мережами викликала можливість навчання. Для заданої конкретної задачі для розв’язання та класу функцій F навчання означає використання набору спостережень для знаходження $f^* \in F$, яка розв’язує цю задачу в певному оптимальному сенсі.

Це тягне за собою визначення такої функції витрат $C: F \rightarrow R$, що, для оптимального розв'язку $f^* = C(f^*) \leq C(f) \forall f \in F$ – тобто, жоден розв'язок не має витрат, менших за витрати оптимального розв'язку.

Функція витрат C є важливим поняттям у навчанні, оскільки вона є мірою того, наскільки далеким є певний розв'язок від оптимального розв'язку задачі, яку потрібно розв'язати. Алгоритми навчання здійснюють пошук простором розв'язків, щоби знайти функцію, яка має найменші можливі витрати.

Для тих застосувань, де розв'язок залежить від даних, витрати обов'язково мусять бути функцією від спостережень, бо інакше модель не матиме зв'язку з даними. Їх часто визначають як статистику, для якої може бути зроблено лише наближення.

Коли $N \rightarrow \infty$, мусить застосовуватися якийсь різновид інтерактивного машинного навчання, в якому витрати знижуються з кожним побаченим зразком. І хоча інтерактивне машинне навчання часто застосовують за незмінного D, найкориснішим воно є у випадку, коли цей розподіл повільно змінюється з часом. В нейромережевих методах якісь різновиди інтерактивного машинного навчання часто застосовують для скінченних наборів даних.

Навіть коли можливо визначити функцію витрат спрямовану на конкретний випадок, часто використовують конкретні витрати (функцію витрат), або через те, що вони мають бажані властивості (такі як опуклість), або через те, що вони природно виникають з певного формулювання задачі (наприклад, у ймовірнісному формулюванні як обернені витрати можна використовувати апостеріорну ймовірність моделі). Кінець кінцем, функція витрат залежить від задачі.

Глибинне навчання може бути треновано розрізнювально за допомогою стандартного алгоритму зворотного поширення. Зворотне поширення – це метод обчислення градієнту функції втрат (видає витрати,

пов’язані з заданим станом) по відношенню до ваг в штучній нейронній мережі.

Основи неперервного зворотного поширення було виведено в контексті теорії керування Келлі 1960 року та Брайсоном 1961 року з використанням принципів динамічного програмування. 1962 року Дрейфус опублікував простіше виведення, засноване лише на ланцюговому правилі. Брайсон та Хо описали його як метод багатоетапної оптимізації динамічних систем 1969 року. 1970 року Ліннаінмаа остаточно опублікував загальний метод автоматичного диференціювання (АД) дискретних зв’язних мереж вкладених диференційовних функцій. Він відповідає сучасному баченню зворотного поширення, яке є ефективним навіть коли мережі є розрідженими. 1973 року Дрейфус застосував зворотне поширення для пристосування параметрів контролерів пропорційно градієнтам похибок. 1974 року Вербос зазначив можливість застосування цього принципу до ШНМ, і 1982 року він застосував метод АД Ліннаінмаа до нейронних мереж способом, який широко застосовується сьогодні. 1986 року Румельхарт, Хінтон та Вільямс зазначили, що цей метод може породжувати корисні внутрішні представлення вхідних даних в прихованих шарах нейронних мереж. 1993 року Ван став першим переможцем міжнародного змагання з розпізнавання образів за допомогою зворотного поширення.

Уточнення ваг зворотного поширення можливо здійснювати за допомогою стохастичного градієнтного спуску із застосуванням наступного рівняння

$$w_{ij}(t+1) = w_{ij}(t) + n\frac{\partial C}{\partial w_{ij}} + \varepsilon(t) \tag{1.2}$$

де n є темпом навчання, C є функцією витрат (втрат), а ε(t) – стохастичним членом. Вибір функції витрат залежить від таких чинників як тип навчання (кероване, спонтанне, з підкріпленням тощо) та функції

збудження. Наприклад, при здійсненні керованого навчання на задачі багатокласової класифікації поширеними варіантами вибору функції збудження та функції витрат є нормована експоненційна функція та функція перехресної ентропії відповідно. Нормалізовану експоненційну функцію визначають як

$$p_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}, \tag{1.3}$$

де p_j представляє ймовірність класу (вихід вузла j), а x_j та x_k представляють загальний вхідний сигнал вузлів j та k одного й того ж рівня відповідно. Перехресну ентропію визначають як

$$C = -\sum_j d_j \log p_j \tag{1.4}$$

де d_j представляє цільову ймовірність для вузла виходу j, а p_j є виходом ймовірності для j після застосування функції збудження.

Це можливо використовувати для виведення обмежувальних коробок об’єкта у вигляді двійкової маски. Їх також використовують для багатомасштабної регресії для підвищення точності визначення положення. Регресія на основі ШНМ може навчатися ознак, що схоплюють геометричну інформацію, на додачу до того, що вони слугують добрим класифікатором. Вони усувають вимогу явного моделювання частин та їхніх взаємозв’язків. Це допомагає розширити розмаїття об’єктів, яких можна навчитися. Модель складається з декількох шарів, кожен з яких має випрямляч як функцію збудження для нелінійного перетворення. Деякі шари є згортковими, тоді як деякі є повнозв’язними. Кожен згортковий шар має додаткове максимізаційне агрегування. Мережу тренують для зведення до мінімуму похибки L2 для передбачування маски, що пробігає весь тренувальний набір, що містить обмежувальні коробки, представлені як маски.

До альтернатив зворотному поширенню належать машини екстремального навчання, «безпоширні» мережі, тренування без пошуку з вертанням, «безвагові» мережі та не-конективістські нейронні мережі.

Існує три основні парадигми навчання, кожна з яких відповідає певній навчальній задачі. Ними є кероване навчання, спонтанне навчання та навчання з підкріпленням.

Кероване навчання використовує набір прикладів пар $(x,y), x \in X, y \in Y$, і має на меті пошук функції $f: X \to Y$ в дозволеному класі функцій, яка відповідає цим прикладам. Іншими словами, ми хочемо вивести відображення, на яке натякають ці дані; функцію витрат пов’язано з невідповідністю між нашим відображенням та даними, і вона неявно містить апріорне знання про предметну область.

Широко вживаними витратами є середньоквадратична похибка, яка намагається звести до мінімуму усереднену квадратичну похибку між виходом мережі, f(x), та цільовим значення у над усіма прикладами пар. Зведення до мінімуму цих витрат за допомогою градієнтного спуску для класу нейронних мереж, званого багатошаровими перцептронами (БШП), дає алгоритм зворотного поширення для тренування нейронних мереж.

Задачами, що вписуються до парадигми керованого навчання, є розпізнавання образів (відоме також як класифікація) та регресія (відома також як наближення функцій). Парадигма керованого навчання є застосовною також і до послідовнісних даних (наприклад, до розпізнавання писання вручну, мовлення та жестів). Його можна розглядати як навчання з «учителем» у вигляді функції, яка забезпечує постійний зворотний зв’язок стосовно якості отриманих досі розв’язків.

У спонтанному навчанні даються якісь дані x та функція витрат для зведення до мінімуму, якою може бути будь-яка функція від даних x та виходу мережі f.

Функція витрат залежить від задачі (предметної області моделі) та наявних апріорних припущень (неявних властивостей моделі, її параметрів, та спостережуваних змінних).

Задачі, що вписуються до парадигми спонтанного навчання, є загалом задачами оцінювання; до застосувань належать кластерування, оцінювання статистичних розподілів, стиснення та фільтрування.

У навчанні з підкріпленням дані x зазвичай не надаються, а породжуються взаємодією агента з середовищем. В кожен момент часу t агент виконує дію y(t), а середовище породжує спостереження x(t) та миттєві витрати c(t) відповідно до якоїсь (зазвичай невідомої) динаміки. Метою є визначити таку стратегію вибору дій, яка зводить до мінімуму якусь міру довготривалих витрат, наприклад, очікувані сукупні витрати. Динаміка середовища та довготривалі витрати для кожної зі стратегій є зазвичай невідомими, але їх може бути оцінено.

Формальніше, середовище моделюють як марковський процес вирішування (МПВ), тоді як стратегію визначають як умовний розподіл над діями за заданих спостережень. Взята разом, ця двійка відтак утворює марковський ланцюг (МЛ). Метою є визначити таку стратегію (тобто, МЛ), що зводить витрати до мінімуму.

ШНМ часто використовують у навчанні з підкріпленням як частину загального алгоритму. Динамічне програмування було зв’язано з штучними нейронними мережами (давши нейродинамічне програмування) Берцекасом та Цициклісом і застосовано до багатовимірних нелінійних задач, таких як присутні в маршрутизувані транспорту, природокористуванні та медицині, через здатність ШНМ пом’якшувати втрати точності навіть при зниженні щільності ґратки дискретизації для чисельного наближення розв’язків первинних задач керування.

Задачами, які вписуються до парадигми навчання з підкріпленням, є задачі керування, ігри та інші задачі послідовного ухвалювання рішень.

Алгоритм збіжного рекурсивного навчання – метод навчання, розроблений спеціально для нейронних мереж артикуляційних контролерів мозочкової моделі (АКММ). 2004 року було представлено рекурсивний алгоритм найменших квадратів для інтерактивного тренування нейронної мережі АКММ. Цей алгоритм може збігатися за один крок та уточнювати всі ваги за один крок із будь-якими новими вхідними даними. Початково він мав обчислювальну складність $O(N^3)$. На основі QR-розкладу цей рекурсивний алгоритм навчання було спрощено до O(N).

Тренування нейронної мережі по суті означає вибирання однієї моделі з множини дозволених моделей (або, в баєсовій системі, визначення розподілу над множиною дозволених моделей), що зводить витрати до мінімуму. Доступні численні алгоритми для тренування нейромережевих моделей; більшість із них можна розглядати як безпосереднє застосування теорії оптимізації та статистичного оцінювання.

Більшість використовують градієнтний спуск якогось вигляду, застосовуючи зворотне поширення для обчислення фактичних градієнтів. Це здійснюється просто взяттям похідної від функції витрат по відношенню до параметрів мережі, з наступною зміною цих параметрів у пов’язаному з градієнтом напрямку. Алгоритми тренування зворотним поширенням поділяються на три категорії:

- найшвидший спуск (зі змінним темпом навчання та імпульсом, еластичним зворотним поширенням);

- квазі-ньютонові (Бройден – Флетчер – Гольдфарб – Шанно, однокрокова січна);

- Левенберг – Марквардт та спряжені градієнти (уточнення Флетчера – Рівза, уточнення Поляка – Ріб’єра, перезапуск Павелла – Біла, масштабований спряжений градієнт).

Іншими методами для тренування нейронних мереж є еволюційні методи, генно-експресійне програмування, імітування відпалювання, очікування-максимізація, непараметричні методи та метод рою часток.

1.2 Функції активації

Функція активації сигмоїд (рисунок 1.2) належить до класу безперервних функцій і приймає на вході довільне дійсне число, а на виході дає дійсне число в інтервалі від 0 до 1. Зокрема, великі (по модулю) негативні числа перетворюються в нуль, а великі позитивні - в одиницю. Історично сигмоїда знаходила широке застосування, оскільки її вихід добре інтерпретується, як рівень активації нейрона: від відсутності активації (0) до повністю насиченою активації (1). Сигмоїда (sigmoid) виражається формулою

$$f(s) = \frac{1}{1 + e^{-s}}. \tag{1.5}$$

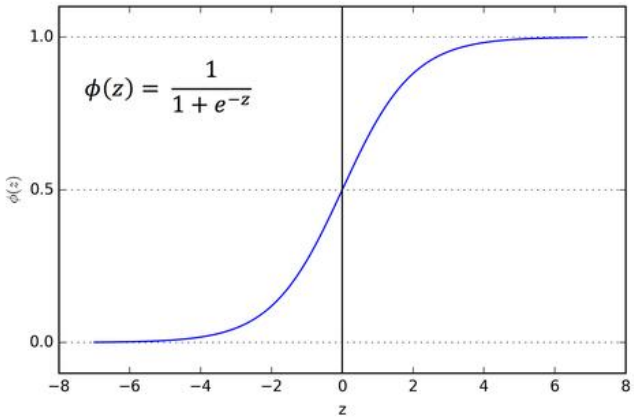


Рисунок 1.2 – Функція сигмоїди [12]

Вкрай небажана властивість сигмоїди полягає в тому, що при насиченні функції з тієї чи іншої сторони (0 або 1), градієнт на цих ділянках стає близький до нуля. В процесі зворотного поширення помилки даний (локальний) градієнт множиться на загальний градієнт. Отже, якщо локальний градієнт дуже малий, він фактично обнуляє загальний градієнт. В результаті, сигнал майже не буде проходити через нейрон до його ваг і рекурсивно до його даних. Крім того, слід бути дуже обережним при

ініціалізації ваг сігмоідних нейронів, щоб запобігти насичення. Наприклад, якщо вихідні ваги мають занадто великі значення, більшість нейронів перейде в стан насичення, в результаті чого мережа буде погано вчитися.

Сигмоїдальна функція є:

- безперервною;
- монотонно зростаючою;
- диференціюємою.

Функція активації гіперболічний тангенс:

- симетричні активаційні функції, типу гіперболічного тангенса забезпечують більш швидку збіжність, ніж стандартна логістична функція;
- функція має безперервну першу похідну;
- функція має просту похідну, яка може бути обчислена через її значення, що дає економію обчислень.

Графік функції гіперболічного тангенса наведено на рисунку 1.3.

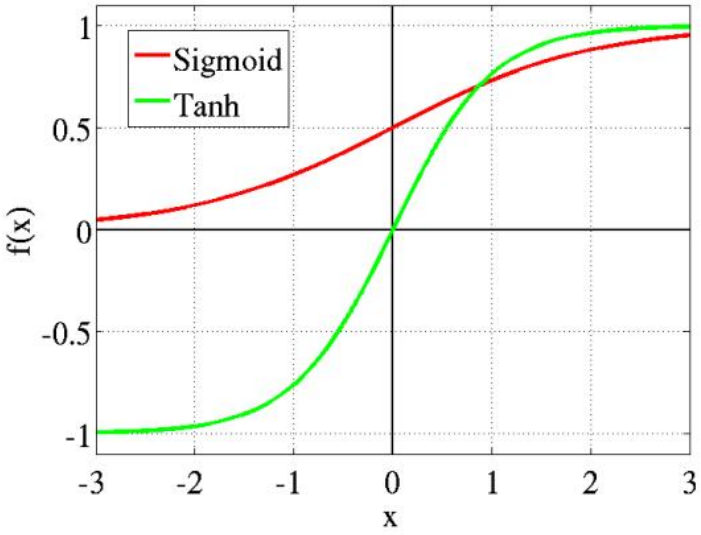


Рисунок 1.3 – Графік функції гіперболічного тангенса на фоні сігмоїди [12]

Функція активації ReLU. Відомо, що нейронні мережі здатні наблизити як завгодно складну функцію, якщо в них досить шарів і функція активації є нелінійної. Функції активації сигмоідна або тангенсальна є нелінійними, але

призводять до проблем з загасанням або збільшенням градієнтів. Однак можна використовувати і набагато простіший варіант – випрямлену лінійну функцію активації (rectified linear unit, ReLU), яка виражається формулою

$$f(s) = \max(0, s) \tag{1.6}$$

Графік функції ReLU наведено на рисунку 1.4.

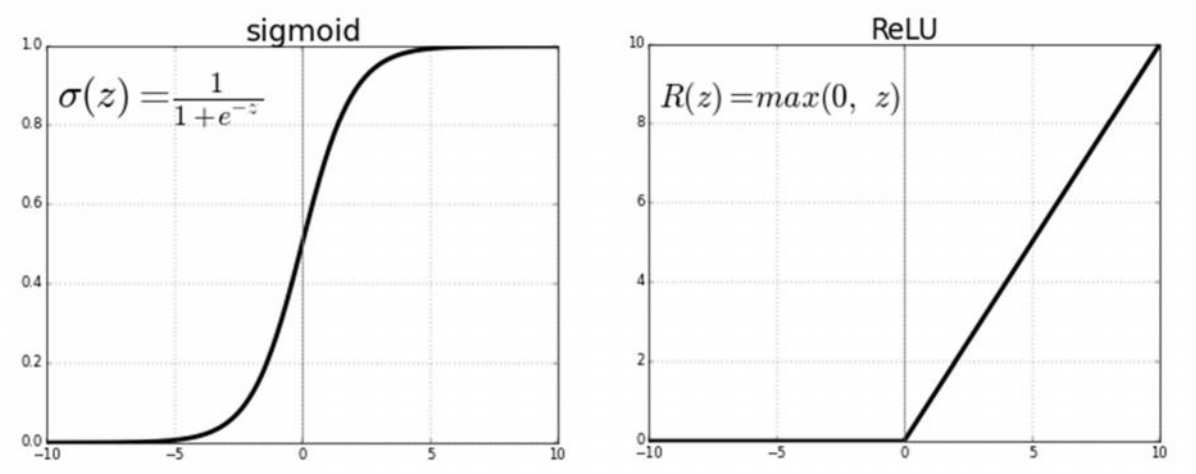


Рисунок 1.4 – Графік функції ReLU на фоні сігмоїди [12]

Переваги використання ReLU:

- її похідна дорівнює або одиниці, або нулю, і тому не може статися розростання або загасання градієнтів, тому що помноживши одиницю на дельту помилки ми отримаємо дельту помилки, якщо ж ми б використовували іншу функцію, наприклад, гіперболічний тангенс, то дельта помилки могла, або зменшитися, або зрости, або залишитися такою ж, тобто, похідна гіперболічного тангенса повертає число з різним знаком і величиною, що може сильно вплинути на загасання або розростання градієнта. Використання даної функції приводить до проріджування ваг;
- обчислення сигмоїд і гіперболічного тангенса вимагає виконання ресурсомістких операцій, в той час як ReLU може бути реалізований за допомогою простого порогового перетворення матриці активацій в нулі;

- відсікає непотрібні деталі в каналі при негативному виході.

З недоліків можна відзначити, що ReLU не завжди достатньо надійна і в процесі навчання може виходити з ладу. 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 дорівнювати нулю. Відповідно, даний нейрон буде необоротньо 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.

Розглянуті функції активації – це найбільш поширені функції. Повний перелік таких функцій, якщо потрібно більш точно адаптуватися під потреби задачі наведено на рис. 1.5.

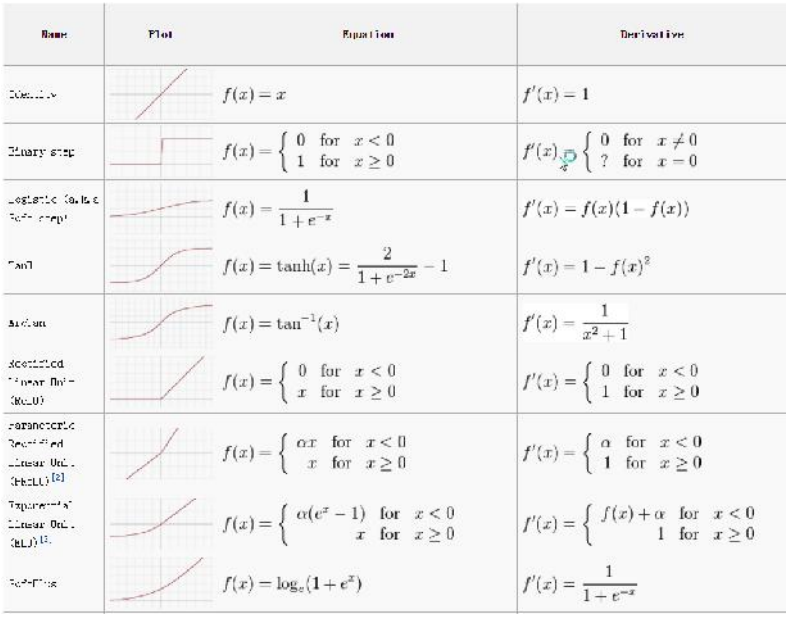


Рисунок 1.5 – Список поширених функцій активації [12]

1.3 Згорткові нейронні мережі

Згорткова нейронна мережа (Convolutional neural network) – спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном в 1988 році і націлена на ефективне розпізнавання зображень, входить до складу технологій глибокого навчання. Використовує деякі особливості зорової кори, в якій були відкриті так звані прості клітини, що реагують на прямі лінії під різними кутами, і складні клітини, реакція яких пов’язана з активацією певного набору простих клітин. Таким чином, ідея згорткових нейронних мереж полягає в чергуванні convolution layers і subsampling layers. Структура мережі – односпрямована, принципово багатошарова. Для навчання використовуються стандартні методи, найчастіше метод зворотного поширення помилки.

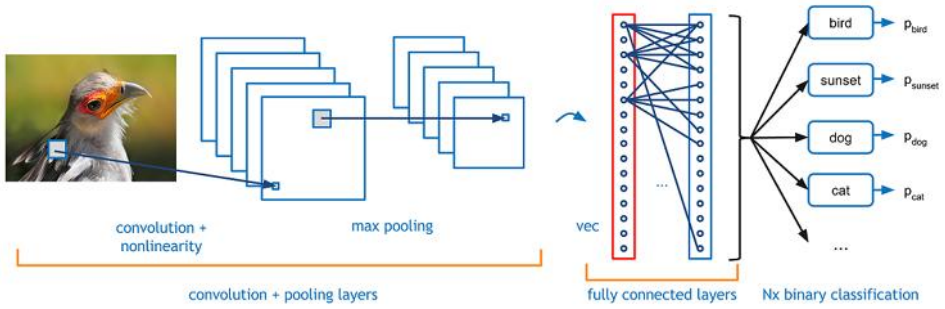


Рисунок 1.6 – Приклад ЗНН [13]

Назва архітектури мережа отримала через наявність операції згортки, суть якої в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно, а результат підсумовується і записується в аналогічну позицію вихідного зображення.

Робота згорткової нейронної мережі зазвичай інтерпретується як перехід від конкретних особливостей зображення до більш абстрактних деталей, і далі до ще більш абстрактних деталей аж до виділення понять високого рівня.

При цьому мережа самоналаштовується і виробляє сама необхідну ієрархію абстрактних ознак (послідовності карт ознак), фільтруючи незначні деталі і виділяючи істотне.

Подібна інтерпретація носить скоріше метафоричний або ілюстративний характер. Фактично «ознаки», що виробляються складною мережею, малозрозумілі і важкі для інтерпретації настільки, що в практичних системах не дуже рекомендується намагатися зрозуміти змісту цих ознак або намагатися їх «підправити», замість цього рекомендується вдосконалити саму структуру і архітектуру мережі, щоб отримати кращі результати. Так, ігнорування системою якихось істотних явищ може говорити про те, що не вистачає даних для навчання, або структура мережі має недоліки, і система не може виробити ефективних ознак для даних явищ.

У звичайному перцептроні, який представляє собою повнозв’язну нейронну мережу, кожен нейрон пов’язаний з усіма нейронами попереднього шару, причому кожний зв’язок має свій персональний ваговий коефіцієнт. У згортковій нейронній мережі в операції згортки використовується лише обмежена матриця ваг невеликого розміру, яку «рухають» по всьому оброблюваному шару (на самому початку – безпосередньо по вхідному зображенню), формуючи після кожного зсуву сигнал активації для нейрона наступного шару з аналогічною позицією. Тобто для різних нейронів вихідного шару використовуються одна і та ж матриця ваг, яку також називають ядром згортки. Її інтерпретують як графічне кодування якої-небудь ознаки, наприклад, наявність похилої лінії під певним кутом. Тоді наступний шар, що вийшов в результаті операції згортки такою матрицею ваг, показує наявність даної ознаки в оброблюваному шарі і її координати, формуючи так звану карту ознак. Природно, в згортковій нейронної мережі набір ваг не один, а ціла гама, що кодує елементи зображення (наприклад, лінії і дуги під різними кутами). При цьому такі ядра згортки не закладаються дослідником заздалегідь, а формуються самостійно шляхом навчання мережі класичним методом зворотного поширення помилки. Прохід кожним

набором ваг формує свій власний примірник карти ознак, роблячи нейронну мережу багатоканальною (багато незалежних карт ознак на одному шарі). Також слід зазначити, що при переборі шару матрицею ваг її пересувають зазвичай не на повний крок (розмір цієї матриці), а на невелику відстань. Так, наприклад, при розмірності матриці ваг 5 × 5 її зрушують на один або два нейрона (пікселя) замість п’яти, щоб не «переступити» шукану ознаку.

Операція субдискретизація, виконує зменшення розмірності сформованих карт ознак. У даній архітектурі мережі вважається, що інформація про факт наявності шуканої ознаки важливіше точного знання її координат, тому з кількох сусідніх нейронів карт ознак обирається максимальна і приймається за один нейрон ущільненої карти ознак меншої розмірності. За рахунок цієї операції, крім прискорення подальших обчислень, мережа стає більш інваріантна до масштабу вхідного зображення.

Розглянемо типову структуру згорткової нейронної мережі більш детально. Мережа складається з великої кількості шарів. Після початкового шару (вхідного зображення) сигнал проходить серію згорткових шарів, в яких чергується власне згортка і субдискретизація. Чергування шарів дозволяє складати «карти ознак» з карт ознак, на кожному наступному шарі карта зменшується в розмірі, але збільшується кількість каналів. На практиці це означає здатність розпізнавання складних ієрархій ознак. Зазвичай після проходження декількох шарів карта ознак вироджується в вектор або навіть скаляр, але таких карт ознак стають сотні. На виході згорткових шарів мережі додатково встановлюють кілька шарів повнозв’язної нейронної мережі, на вхід якої подаються кінцеві карти ознак.

Шар згортки – це основний блок згорткової нейронної мережі. Шар згортки включає в себе для кожного каналу свій фільтр, ядро згортки якого обробляє попередній шар за фрагментами (підсумовуючи результати матричного добутку для кожного фрагменту). Вагові коефіцієнти ядра згортки (невеликої матриці) невідомі і встановлюються в процесі навчання.

Особливістю згорткового шару є порівняно невелика кількість

параметрів, яка встановлюється при навчанні. Так, наприклад, якщо вихідне зображення має розмірність 100 × 100 пікселів по трьом каналам (це значить 30000 вхідних нейронів), а згортковий шар використовує фільтри с ядром 3х3 пікселя з виходом на 6 каналів, тоді в процесі навчання визначається тільки 9 ваг ядра, однак за всіма сполученнями каналів, тобто 9 × 3 × 6 = 162, в такому випадку даний шар вимагає знаходження тільки 162 параметрів, що істотно менше кількості шуканих параметрів повнозв’язної мережі (рисунок 1.7).

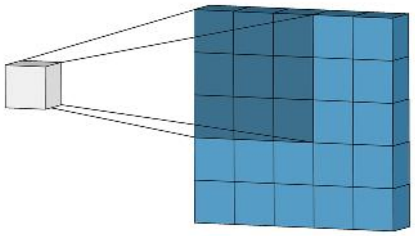


Рисунок 1.7 – Отримання двовимірної згортки (графічна інтерпритація)

Двовимірна згортка (2D convolution) – це досить проста операція: починаємо з ядра, що представляє із себе матрицю ваг (weight matrix). Це ядро «ковзає» над двовимірним зображенням, поелементно виконуючи операцію множення з тією частиною вхідних даних, над якою воно зараз знаходиться, і потім підсумовує всі отримані значення в один вихідний піксель.

Ядро повторює цю процедуру з кожною локацією, над якою воно «ковзає», перетворюючи двовимірну матрицю в іншу все ще двовимірну матрицю ознак. Ознаки на виході є зваженими сумами (де ваги є значеннями самого ядра) ознак на вході, розташованих приблизно в тому ж місці, що і вихідний піксель на вхідному шарі (рисунок 1.8).

Незалежно від того, чи потрапляє вхідна ознака в «приблизно те ж місце», він визначається в залежності від того, знаходиться він в зоні ядра, що створює вихідні дані, чи ні. Це означає, що розмір ядра ЗНН визначає

кількість ознак, які будуть об’єднані для отримання нової ознаки на виході.

У прикладі, наведеному вище, ми маємо 5 * 5 = 25 ознак на вході і 3 * 3 = 9 ознак на виході. Для стандартного шару (standard fully connected layer) ми б мали вагову матрицю 25 * 9 = 225 параметрів, а кожна вихідна ознака була би зваженою сумою всіх ознак на вході. Згортка дозволяє зробити таку операцію з усього 9-ю параметрами, адже кожна ознака на виході отримується аналізом не кожної ознаки на вході, а тільки однією вхідною, що знаходиться в «приблизно тому ж місці».

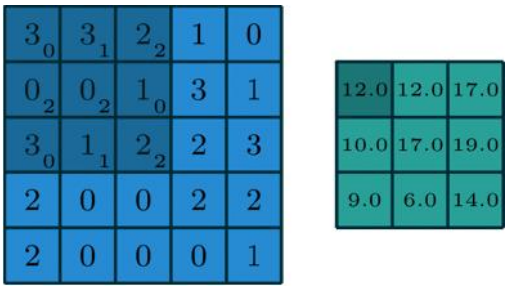


Рисунок 1.8 – Виконання операції згортки

1.4 Ключові технології обробки даних

Далі варто поглянути на дві техніки, які часто застосовуються в згорткових нейронних мережах: Padding і Striding.

Padding в процесі ковзання край по суті обрізається, перетворюючи матрицю ознак розміром 5 * 5 в матрицю 3 * 3. Крайні пікселі ніколи не виявляються в центрі ядра, тому що тоді ядру немає над чим буде ковзати за краєм. Це зовсім не ідеальний варіант, так як ми хочемо, щоб частіше розмір на виході дорівнював вхідному (рисунок 1.9). Padding додає до країв підроблені (fake) пікселі (зазвичай нульового значення, внаслідок цього до них застосовується термін «нульовий додаток» - «zero padding»). Таким чином, ядро при прослизанні дозволяє непідробним пікселям надаватися в

своєму центрі, а потім поширюється на підроблені пікселі за межами краю, створюючи вихідну матрицю того ж розміру що і вхідні.

Striding. Часто буває, що при роботі з згортковим шаром, потрібно отримати вихідні дані меншого розміру, ніж вхідні. Це зазвичай необхідно в згорткових нейронних мережах, де розмір просторових розмірів зменшуються при збільшенні кількості каналів. Один із способів досягнення цього – використання субдискретизуючих шарів (pooling layer) (наприклад, приймати середнє / максимальне значення кожної гілки розміром 2 * 2, щоб зменшити всі просторові розміри в два рази). Ще один спосіб домогтися цього – використовувати stride (крок).

Ідея stride полягає в тому, щоб пропустити деякі області, над якими ковзає ядро. Крок 1 означає, що беруться прольоти через піксель, тобто за фактом кожен проліт є стандартною згорткою. Крок 2 означає, що прольоти відбуваються через кожні два пікселя, пропускаючи всі інші прольоти в процесі і зменшуючи їх кількість приблизно в 2 рази, крок 3 означає пропуск 3-х пікселів, скорочуючи кількість в 3 рази і т.д.

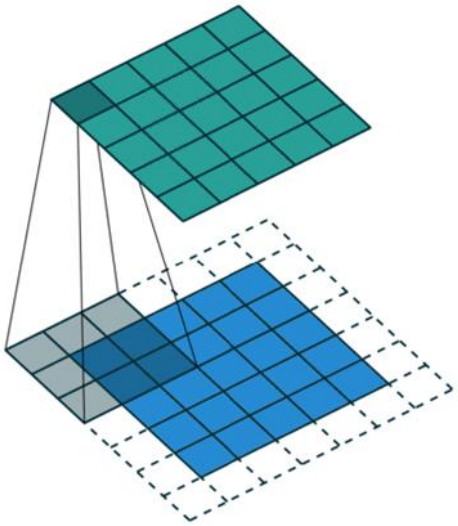


Рисунок 1.9 – Технологія Padding

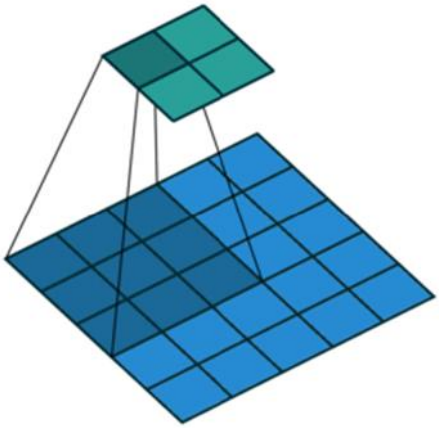


Рисунок 1.10 – Операція згортки з кроком 2

Більш сучасні мережі, такі як ResNet, повністю відмовляються від субдискретизуючих шарів у внутрішніх шарах, на користь чергуються згортки, коли необхідно зменшити розмір на виході. Підвиборочний шар також, як і згортковий має карти, але їх кількість співпадає з попереднім (згортковим) шаром, їх 6. Мета шару – зменшення розмірності карт попереднього шару. Якщо на попередній операції згортки вже були виявлені деякі ознаки, то для подальшої обробки настільки докладне зображення вже не потрібно, і воно ущільнюється до менш докладного. До того ж фільтрація вже непотрібних деталей допомагає не перенавчатися (рисунок 1.11).

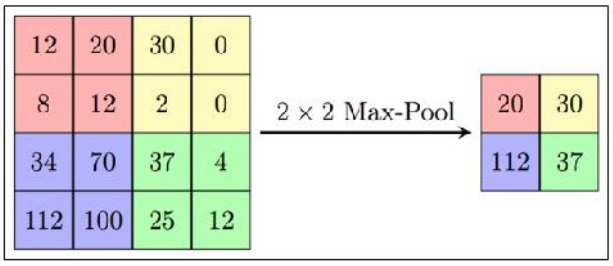


Рисунок 1.11 – Формування нової карти підвиборочного шару на основі попередньої карти згорткового шару. Операція підвибірки (Max Pooling)

У процесі сканування ядром підвиборчого шару (фільтром) карти попереднього шару, скануюче ядро не перетинається на відміну від згорткового шару. Зазвичай, кожна карта має ядро розміром 2x2, що дозволяє зменшити попередні карти згорткового шару в 2 рази. Вся карта ознак поділяється на осередки 2x2 елемента, з яких вибираються максимальні за значенням.

Зазвичай в підвиборчому шарі застосовується функція активації ReLU. Операція підвибірки (або MaxPooling) відповідно до рис. 1.11. Формально шар може бути описаний формулою

$$x^l = f(a^l * subsample(x^{l-1}) + b^l), \tag{1.7}$$

де x^l – вихід шару l; f() – функція активації; a^l, b^l – коефіцієнти зсуву шару l; subsample() – операція вибірки локальних максимальних значень.

Скалярний результат кожної згортки потрапляє на функцію активації, яка представляє собою якусь нелінійну функцію. Шар активації зазвичай логічно пов’язують з шаром згортки (вважають, що функція активації вбудована в шар згортки). Функція нелінійності може бути будь-яка з вибору дослідника, традиційно для цього використовували функції типу гіперболічного тангенса f (x) = tanh (x) або сигмоид f (x) = (1 + e-x) -1. Однак в 2000х роках була запропонована і досліджена нова функція активації – ReLU, яка дозволила суттєво прискорити процес навчання і одночасно спростити обчислення, що означає блок лінійної ректифікації, що обчислює функцію f (x) = max (0, x). Тобто по суті це операція відсікання негативної частини скалярної величини. Станом на 2017 рік ця функція і її модифікації є найбільш часто використовуваними функціями активації в глибоких нейромережах, зокрема, в згортокових.

Шар пулінг це нелінійне ущільнення карти ознак, при цьому група пікселів (зазвичай розміру 2 × 2) ущільняються до одного пікселя, проходячи

нелінійне перетворення. Найбільш споживані при цьому функція максимуму. Перетворення зачіпають непересічні прямокутники або квадрати, кожен з яких скорочується в один піксель, при цьому вибирається піксель, що має максимальне значення. Операція пулінг дозволяє істотно зменшити просторовий обсяг зображення. Пулінг інтерпретується так. Якщо на попередній операції згортки вже були виявлені деякі ознаки, то для подальшої обробки настільки докладне зображення вже не потрібно, і воно ущільнюється до менш докладного. До того ж фільтрація вже непотрібних деталей допомагає не перенавчатися. Шар пулінг, як правило, вставляється після шару згортки перед шаром наступної згортки.

З метою більш агресивного зменшення розміру одержуваних уявлень, все частіше знаходять поширення ідеї використання менших фільтрів або повна відмова від шарів пулінгу.

Після кількох проходжень згортки зображення і ущільнення за допомогою пулінгу система перебудовується від конкретної сітки пікселів з високою роздільною здатністю до більш абстрактних карт ознак, як правило на кожному наступному шарі збільшується число каналів і зменшується розмірність зображення в кожному каналі. Зрештою залишається великий набір каналів, що зберігають невелику кількість даних (навіть один параметр), які інтерпретуються як самі абстрактні поняття, виявлені з вихідного зображення.

Ці дані об’єднуються і передаються на звичайну повнозв’язну нейронну мережу, яка теж може складатися з декількох шарів. При цьому повнозв’язні шари вже втрачають просторову структуру пікселів і мають порівняно невелику розмірність (по відношенню до кількості пікселів вихідного зображення).

На практиці більшість вхідних зображень мають 3 канали, і чим глибше ви в мережі, тим більше це число. Досить просто думати о каналах, в загальному, як про «погляди» на зображення в цілому, надаючи більшого значення одним аспектам і менше іншим (рисунок 1.12).

Рисунок 1.12 – Зображення поділене на канали [14]

У більшості випадків ми маємо справу з зображеннями RGB з трьома каналами.

Ось де ключові відмінності між термінами стають потрібні: тоді як у випадку з 1 каналом, де терміни фільтр і ядро взаємозамінні, в загальному випадку вони різні.

Кожен фільтр насправді являє собою колекцію ядер, причому для кожного окремого вхідного каналу для цього шару є одне ядро, де кожне ядро унікальне.

Кожен фільтр в згортковому шарі створює тільки один вихідний канал і роблять вони це так: кожне з ядер фільтра «ковзає» по їх відповідним вхідним каналам, створюючи оброблену версію кожного з них. Деякі ядра можуть мати більшу вагу, ніж інші, для того щоб приділяти більше уваги певним вхідним каналам (наприклад, фільтр може мати червоний канал ядра з більшою вагою, ніж інші канали, і, отже, більше реагувати на відмінності в образах з червоного каналу).

Нарешті, є термін зсув. Принцип роботи зміщення полягає в тому, що кожен вихідний файл має свій термін зміщення. Зсув додається до вихідного каналу для створення кінцевого вихідного каналу.

Результат для будь-якої кількості фільтрів ідентичний: кожен фільтр обробляє вхід зі своїм відмінним від інших набором ядер і скалярним зміщенням за описаним вище процесом, створюючи один вихідний канал. Потім вони об’єднуються разом для отримання загального виходу, причому кількість вихідних каналів дорівнює числу фільтрів. При цьому зазвичай

застосовується нелінійність перед передачею входу іншого шару згортки, який потім повторює цей процес.

Потім кожна з оброблених в каналі версій підсумовується разом для формування одного каналу. Ядра кожного фільтра генерують одну версію кожного каналу, а фільтр в цілому створює один загальний вихідний канал, як це наведено на рисунку 1.14.

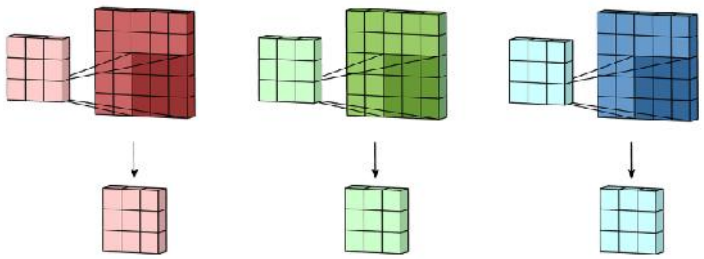


Рисунок 1.13 – Операція згортки на 3х канальному зображенні

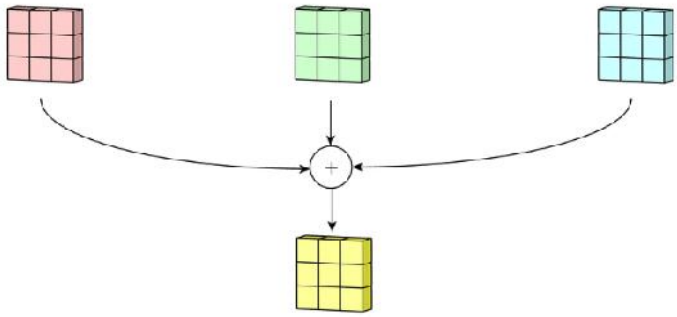


Рисунок 1.14 – Об’єднання операцій по всім каналам

Навіть з уже описаної механікою роботи згорткового шару, все ще складно пов’язати це з нейронною мережею прямого поширення (feed-forward network), і це все ще не пояснює, чому згортки масштабуються та працюють набагато краще з зображеннями.

Припустимо, що у нас є вхід 4 * 4, і ми хочемо перетворити його в сітку 2 * 2. Якби ми використовували feed-forward network, ми б переробили

вхід 4 * 4 в вектор довжиною 16 і передали його через щільно пов’язаний шар з 16 входами і 4 виходами. Можна було б візуалізувати вагову матрицю W для шару, як це наведено на рисунку 1.15. І хоча згорткові операції з ядрами можуть спочатку здатися трохи дивними, це як і раніше лінійні перетворення з еквівалентною матрицею переходу. Якби ми використовували ядро K розміру 3 на видозміненому входом розміру 4 * 4, щоб отримати вихід 2 * 2, еквівалентна матриця переходу буде виглядати як це наведено на рисунку 1.16.

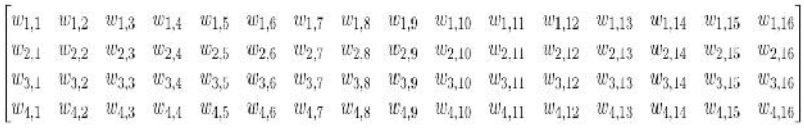


Рисунок 1.15 – Вагова матриця

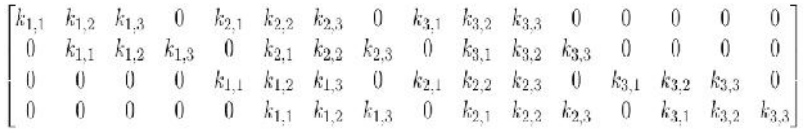


Рисунок 1.16 – Матриця переходу

В той час як наведена матриця є еквівалентною матрицею переходу, фактична операція зазвичай реалізується як зовсім інше матричне множення.

Згортка, в цілому, все ще є лінійним перетворенням, але в той же час вона також представляє собою зовсім інший вид перетворення. Для матриці з 64 елементами існує всього 9 параметрів, які повторно використовуються кілька разів. Кожен вихідний вузол отримує тільки певну кількість входів (ті, що знаходяться всередині ядра). Немає ніякої взаємодії з іншими входами, так як вага для них дорівнює 0.

Корисно представляти згорткові операції як hard prior для вагових матриць. В даному контексті, під prior мається на увазі зумовлені мережеві

параметри. Наприклад, коли використовується попередньо оброблена модель для класифікації зображень, також використовуються попередні параметри мережі як prior, як екстрактор образів для остаточного щільно пов’язаного шару.

У цьому плані, існує пряма причина того, чому обидва вони настільки ефективні (в порівнянні з їх альтернативами). Transfer learning ефективніше на порядок в порівнянні з випадковою ініціалізацією, тому що тільки потрібно оптимізувати параметри кінцевого повністю пов’язаного шару, а це означає, що можна мати фантастичну продуктивність всього лише з декількома десятками зображень в класі.

Не потрібно оптимізувати всі 64 параметра, тому більшість з них встановлено на нуль (і вони залишаться такими), а решту перетворимо в загальні параметри і в результаті отримаємо тільки 9 параметрів для оптимізації. Ця ефективність має значення, тому що, коли переходимо від 784 входів MNIST до реальних зображень 224 * 224 * 3, це більше 150 000 входів. Щільний шар, який намагається вдвічі зменшити вхід до 75 000 вхідних значень, як і раніше потребують понад 10 мільярдів параметрів. Для порівняння, в цілому ResNet-50 має близько 25 мільйонів параметрів.

Таким чином, фіксування деяких параметрів до нуля і їх зв’язування підвищує ефективність, але на відміну від випадку з transfer learning, де ми знаємо, що prior працює грамотно, тому що він працює з великим загальним набором зображень, звідки ми знаємо, що це буде працювати хоч скільки-то добре? Відповідь полягає в комбінаціях образів, що вивчаються параметрами за рахунок prior.

Таким чином, з backpropagation (метод зворотного поширення помилки), що йде у всіх напрямках від вузлів класифікації мережі, ядра мають цікаву задачу вивчення ваг для створення ознак тільки з локального набору входів.

Крім того, оскільки саме ядро застосовується по всьому зображенню, ознаки, які вивчає ядро, повинні бути досить загальними, щоб поступати з

будь-якої частини зображення.

Якщо це були якісь інші дані, наприклад, дані о встановлених додатків за категоріями, то це стало б катастрофою, тому що кількість стовпців установки додатків і типів додатків поруч один з одним не означає, що у них є «локальні загальні ознаки», спільні з датами установки додатків і часом використання.

Звичайно, у чотирьох можуть бути основні ознаки більш високого рівня (наприклад, які програми люди хочуть найбільше), які можуть бути знайдені, але це не дає нам жодних підстав вважати, що параметри для перших двох точно такі ж, як параметри для останніх двох .

Ці четверо могли бути в будь-якому (послідовному) порядку і як і раніше бути придатними.

Пікселі, проте, завжди відображаються в послідовному порядку, а сусідні пікселі впливають на піксель поруч, наприклад, якщо все сусідні пікселі червоні, досить імовірно, що піксель поруч також червоніє. Якщо є відхилення, це цікава аномалія, яка може бути перетворена в ознаку, і все це можна виявити при порівнянні пікселя зі своїми сусідами, з іншими пікселями в своїй місцевості.

Ця ідея – те, на чому були засновані більш ранні методи вилучення ознак комп'ютерним зором. Наприклад, для виявлення граней можна використовувати фільтр виявлення граней Sobel – ядро з фіксованими параметрами, чинне точно так же, як стандартна одноканальна згортка, як це наведено на рисунок 1.17.

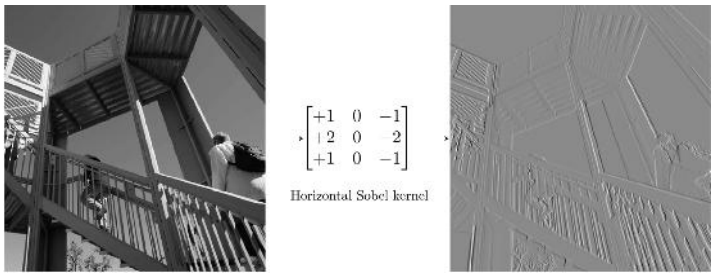


Рисунок 1.17 – Одноканальна згортка

1.5 Застосування ядра для детектування границі

Для сітки, яка не містить граней (наприклад, неба на задньому фоні), більшість пікселів мають однакове значення, тому загальний висновок ядра в цій точці дорівнює 0. Для сітки з вертикальними гранями існує різниця між пікселями зліва і праворуч від межі, і ядро обчислює цю ненулевую різницю, виявляючи ребра. Ядро за раз працює тільки з сітками 3 * 3, виявляючи аномалії в певних місцях, але застосування по всьому зображенню досить для виявлення певної ознаки в будь-якому місці на зображенні.

Але чи можуть корисні ядра бути вивчені? Для ранніх шарів, які працюють з необробленими пікселями, ми могли б очікувати детектори ознак низького рівня, такі як ребра, лінії і т.д.

Існує ціла галузь досліджень глибокого навчання, орієнтована на те, щоб зробити моделі нейронних мереж інтерпритуємими. Один з найпотужніших інструментів для цього – візуалізація ознак за допомогою оптимізації. Ідея в корені проста: оптимізуйте зображення (зазвичай Ініціалізувавши випадковим шумом), щоб активувати фільтр якомога сильніше. Такий спосіб інтуїтивно зрозумілий: якщо оптимізоване зображення повністю заповнене гранями, то це переконливий доказ того, що фільтр активований і зайнятий пошуком. Використовуючи це, ми можемо зазирнути в вивчені фільтри, як це наведено на рисунку 1.18. Хоча вони виявляють різні типи ребер, вони все ще є низькорівневими детекторами (рисунок 1.19).

Рисунок 1.18 – Візуалізація ознак для 3 каналів після першого згорткового шару

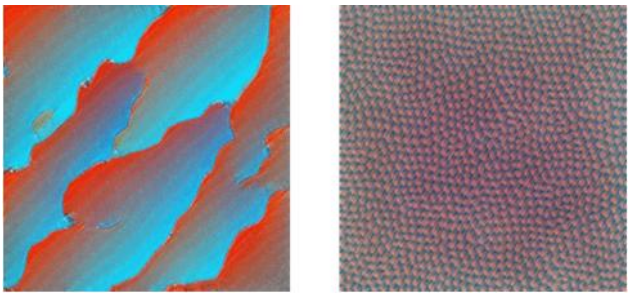


Рисунок 1.19 – Візуалізація ознак після 2-ї і 3-ї згортки

Важливо звернути увагу на те, що конвертовані зображення залишаються зображеннями. Вихід, одержуваний від невеликої сітки пікселів у верхньому лівому кутку, буде теж розташований у верхньому лівому кутку. Таким чином, можна застосовувати один шар поверх іншого для вилучення більш грубих ознак, які ми візуалізуємо.

Проте, як би глибоко не заходили наші детектори ознак, без будь-яких подальших змін вони все одно будуть працювати на дуже маленьких ділянках зображення. Незалежно від того, наскільки глибокі ваші детектори, ви не зможете знайти особи в сітці 3 * 3. І ось тут виникає ідея рецептивного поля (receptive field).

Суттєвою особливістю архітектур згорткових нейронних мереж є те, що розміри входу стають все менше і менше від початку до кінця мережі, а кількість каналів стає більше. Це, як згадувалося раніше, часто робиться за допомогою strides або pooling layers. Місцевість визначає, які вхідні дані з попереднього рівня будуть на виході наступного. Receptive field визначає, яку область вихідного входу отримує вихід.

Ідея strided convolution полягає в тому, що ми обробляємо прольоти тільки на фіксованій відстані один від одного і пропускаємо ті що посередині. З іншої точки зору, ми залишаємо тільки виходи на певній відстані один від одного, видаляючи інші. Потім ми застосовуємо нелінійність до вихідних даних, потім накладаємо ще один новий шар згортки зверху. Тут все стає цікавим. Навіть застосувавши ядро того ж

розміру (3 * 3), що має одну і ту ж локальну область, до виходу strided convolution, ядро мало б більш ефективне receptive field.

Це пов’язано з тим, що вихід strided шару як і раніше представляє з себе одне і те ж саме зображення. Це не стільки обрізка, скільки зміна розміру, тільки тепер кожен окремий піксель в виході є «представником» більшої площі (інші пікселі які були відкинуті) з того ж місця розташування вихідного введення. Тому, коли ядро наступного шару застосовується до виходу, воно працює з пікселями, зібраними з більшою області.

Таке розширення чутливого поля дозволяє верствам згортки поєднувати ознаки низького рівня (лінії, ребра) з ознаками більш високого рівня (криві, текстури).

Слідом за шаром pooling/striding мережа продовжує створювати детектори для ще більш високорівневих ознак (частин, шаблонів).

Повторне зменшення розміру зображення до 5-го блоку згорток дає розміри введення всього 7 * 7, в порівнянні з входами 224 * 224. У цей момент кожен окремий піксель являє собою величезну сітку розміром 32 * 32 пікселя. Якщо на більш ранніх шарах активація виявляла межі, то тут активація на сітці 7 * 7 потрібна для виявлення більш складних образів, наприклад, птахів.

Мережа в цілому розвивається з невеликої кількості фільтрів (64 в разі GoogLeNet), виявляючи функції низького рівня, до дуже великої кількості фільтрів (1024 в остаточній згортці), кожен з яких шукає надзвичайно специфічні ознаки високого рівня. І далі застосовується остаточний шар – pooling layer, який згортає кожну сітку 7 * 7 в один піксель, кожен канал є детектором ознак з receptive field, еквівалентним всьому зображенню.

У порівнянні з тим, що зробила б стандартна feedforward мережа, висновок тут не вражає. Стандартна feedforward мережа створила б абстрактні вектори ознак з комбінації всіх пікселів в зображенні, вимагаючи важковивчаємих обсягів даних.

Згорткова нейронна мережа з накладеними на неї priors, починає

навчання з вивчення детекторів ознак низького рівня, і коли шар за шаром її receptive field стає все більше, вчиться комбінувати ці низькорівневі ознаки в ознаки більш високого рівня; не абстрактне поєднання кожного пікселя, а сильна візуальна ієрархія.

Виявляючи низькорівневі ознаки і використовуючи їх для виявлення ознак більш високого рівня в міру поліпшення своєї візуальної ієрархії, вона в кінцевому підсумку може виявляти цілі візуальні концепції, такі як особи, птахи, дерева і т.д, саме це робить їх такими потужними і ефективними для зображень.

1.6 Навчання згорткових нейронних мереж

Для навчання згорткових нейронних мереж використовують алгоритм зворотного поширення помилки (backpropagation). Цей метод навчання багатошарової нейронної мережі називається узагальненим дельта-правилом. Метод був запропонований в 1986 р Румельхартом, Макклеланд і Вільямсом. Це ознаменувало відродження інтересу до нейронних мереж, який став згасати на початку 70-х років.

Даний алгоритм є першим і основним практично застосовним для навчання багатошарових нейронних мереж.

Для вихідного шару коригування ваг інтуїтивно зрозуміло, але для прихованих шарів довгий час не було відомо алгоритму.

Ваги прихованого нейрона повинні змінюватися прямо пропорційно помилці тих нейронів, з якими даний нейрон пов’язаний.

Ось чому зворотне поширення цих помилок через мережу дозволяє коректно налаштовувати ваги зв’язків між усіма верствами.

У цьому випадку величина функції помилки зменшується і мережа навчається.

У мережі є множина входів $x_1,...,x_n$, множина виходів Outputs і безліч внутрішніх вузлів.

Перенумеруємо всі вузли (включаючи входи і виходи) числами від 1 до N (наскрізна нумерація, незалежно від топології шарів). Позначимо через w_{ij} вагу зв'язку, що з'єднує і-й і j-й вузли, а через o_i – вихід і-го вузла. Якщо нам відомий навчальний приклад (правильні відповіді мережі t_k, k є Outputs), то функція помилки, отримана за методом найменших квадратів, виглядає так

$$E(\{w_{i,j}\}) = \frac{1}{2} \sum_{k \in Outputs} (t_k - o_k)^2. \tag{1.8}$$

В якості методу мінімізації помилки використовується метод градієнтного спуску, суть цього методу зводиться до пошуку мінімуму (або максимуму) функції за рахунок руху вздовж вектора градієнта.

Для пошуку мінімуму рух має бути здійснюватися в напрямку антіградієнта. Метод градієнтного спуску відповідно до рисунку 1.20.

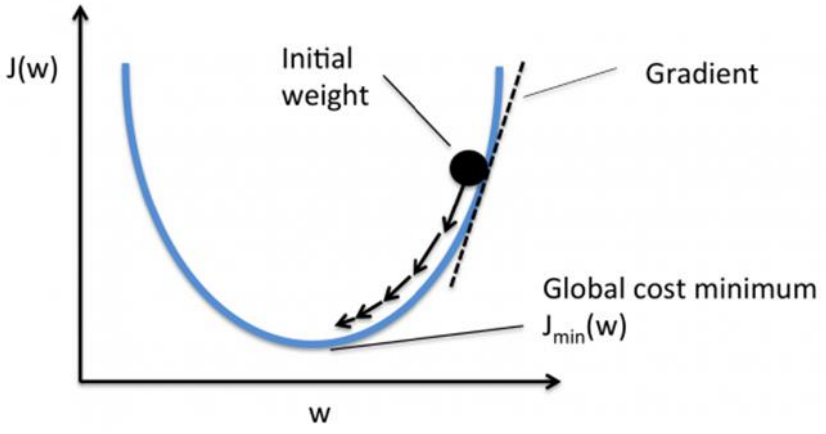


Рисунок 1.20 – Метод градінтного спуску [15]

Градієнт функції втрати вдає із себе вектор приватних похідних, обчислюється так

$$\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}}, \tag{1.9}$$

де 0 < n < 1- множник, що задає «швидкість» руху.

Алгоритм поширення помилки зводиться до наступних етапів:

- пряме поширення сигналу по мережі, обчислення стану нейронів;
- обчислення значення помилки δ для вихідного шару;
- зворотне поширення: послідовно від кінця до початку для всіх прихованих шарів обчислюємо δ;
- оновлення ваг мережі на розрахункову раніше δ помилки.

До цього моменту були розглянуті випадки поширення помилки по верствам персептрона, тобто по вихідному і прихованому, але крім них, в згорткові нейромережі є підвибірний і згортковий.

Розрахунок помилки на підвибірному шарі представляється в декількох варіантах. Перший випадок, коли підвибірний шар знаходиться перед повнозв'язним, тоді він має нейрони і зв'язку такого ж типу, як в повнозв'язну шарі, відповідно обчислення δ помилки нічим не відрізняється від обчислення δ прихованого шару. Другий випадок, коли підвибірний шар знаходиться перед згортковим, обчислення δ відбувається шляхом зворотньої згортки. Для розуміння зворотньої згортки, необхідно спершу зрозуміти звичайну згортку і те, що ковзне вікно по карті ознак (під час прямого поширення сигналу) можна інтерпретувати, як звичайний прихований шар зі зв'язками між нейронами, але головна відмінність - це те, що ці зв'язки розділяємі, тобто один зв'язок з конкретним значенням ваги може бути у кількох пар нейронів, а не тільки з одним.

Зворотня згортка – це майже той же самий спосіб обчислення дельт, що полягає в повороті ядра на 180 градусів і змінному процесі сканування згорткової карти дельт зі зміненими крайовими ефектами. Нам необхідно взяти ядро згорткової карти (наступного за підвибірним шаром) повернути його на 180 градусів і зробити звичайну згортку за обчисленими раніше дельтами згорткової карти, але так щоб вікно сканування виходило за межі карти.

Зазвичай наступний шар після згорткового це підвибірний, відповідно наше завдання обчислити дельти поточного шару (згорткового) за рахунок

знань о дельтах підвибірного шару. Насправді дельта похибка не обчислюється, а копіюється. При прямому поширенні сигналу нейрони підбивірного шару формувалися за рахунок неперекривающихся вікон сканування по згортковому шару, в процесі якого вибиралися нейрони з максимальним значенням, при зворотному поширенні, ми повертаємо дельту помилки раніше обраному максимальному нейрону, інші ж отримують нульову дельту помилки.

Згорткові нейронні мережі використовують стратегію розподілу ваги, що призводить до значного зменшення кількості параметрів, які необхідно вивчити. Наявність більших рецептивних розмірів нейронів у послідовних згорткових шарах, пов’язаних з наявністю шарів, що об’єднують, також призводять до інваріантності перекладу. Виведення прямого і зворотнього поширення буде відрізнятися в залежності від того, яким шаром ми поширюємося.

1.7 Прискорення навчання згорткових мереж

Для навчання нейронних мереж з використанням графічних прискорювачів (англ. GPU) використовують бібліотеки, що спеціалізуються на паралельному процесу роботі мережі. Наприклад, для мережі, яка розпізнає об’єкти з набором даних CIFAR-10, застосування GPU дозволило прискорити навчання більше ніж у 7 разів, порівняно з CPU. Однак межу прискоренню не встановлено, і навчання можна зробити ще швидше, якщо використовувати бібліотеку NVIDIA cuDNN. На відміну від популярного аналогу cuDNN [16] – бібліотеки NVIDIA CUDA, яка забезпечує можливість виконання на GPU будь-яких обчислень, бібліотека cuDNN спеціально розроблена для навчання глибоких нейронних мереж, особливо таких як згорткові. Вона містить оптимізовані для GPU реалізації згорткових і рекурентних мереж, різних функцій активації (напівлінійна, сигмоїдальна, гіперболічний тангенс, softmax), алгоритму зворотнього поширення помилки

та інше. Бібліотека cuDNN дозволяє навчати нейронні мережі на GPU в кілька разів швидше, ніж просто CUDA. Також варто зазначити, що бібліотека є цілком безкоштовною та її можна завантажити на сайті розробника.

2 МОДЕЛІ НЕЙРОННИХ МЕРЕЖ ЗНАХОДЖЕННЯ ОБЛИЧЧА НА ЗОБРАЖЕННІ

2.1 Модель MTCNN

В теперішній час використовується багато моделей згорткових нейронних мереж для вирішення задач локалізації, розпізнавання та класифікації зображень [16-18]. Розглядати всі ці моделі не має сенсу. В роботі зосередимося на моделях, які призначенні для знаходження обличчя на зображенні. Тому, що такі моделі йдуть в авангарді еволюції штучних нейронних мереж: їх архітектура є найбільш розвиненою і ефективною серед аналогів. В даний час існує декілька відомих моделей мереж для знаходження обличчя людини. Найбільш ефективними для вбудованих систем є моделі MTCNN і YOLO, а точніше їх найбільш сучасні модифікації. Розглянемо ці моделі та їх властивості по черзі.

Найбільш поширеною на теперішній час є модель MTCNN тому, що вона зорієнтована на знаходження облич, претренована і дуже ефективна. А також тому, що вона на виході надає не тільки координати прямокутника з обличчям (з обличчями), а також 5 особливих точок на обличчі. Це очі, кінчик носу і куточки рота. Що дозволяє адаптуватися до положення цих точок в подальшої роботі. Модель MTCNN має три згорткові мережі P-Net, R-Net та O-Net і здатна перевершити багато орієнтирів виявлення облич, зберігаючи продуктивність у режимі реального часу. Тобто ця модель – це каскад з трьох моделей, які працюють послідовно (рисунок 2.1).

Очевидно, що перше, що потрібно зробити, - це передати зображення програмі. У цій моделі ми перш за все створюємо піраміду зображень, щоб виявити обличчя різного розміру. Іншими словами, ми хочемо створити різні копії одного і того ж зображення різних розмірів, щоб шукати обличчя різного розміру всередині зображення (рисунок 2.2).

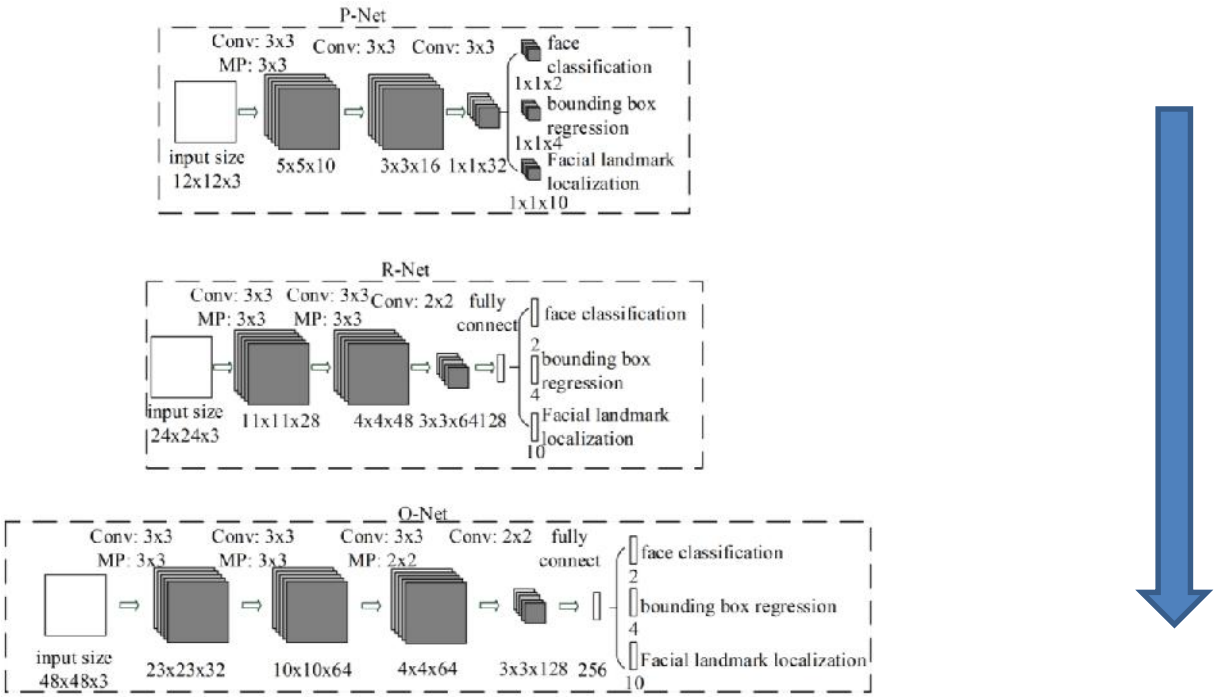


Рисунок 2.1 – Модель MTCNN [3]

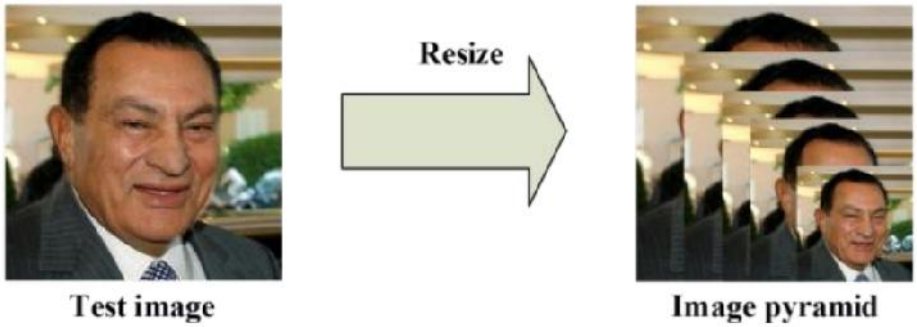


Рисунок 2.2 – Створеня піраміди зображень [3]

Для кожної масштабованої копії у нас є ядро 12 x 12 ступеня 1, яке буде проходити через кожну частину зображення, скануючи обличчя. Він починається з верхнього лівого кута, з розділу зображення від (0,0) до (12,12). Ця частина зображення передається P-Net, який повертає координати обмежувального поля, якщо він помічає обличчя. Потім він повторить цей процес з розділами (0 + 2a, 0 + 2b) до (12 + 2a, 12 + 2b), переміщуючи 12 x 12

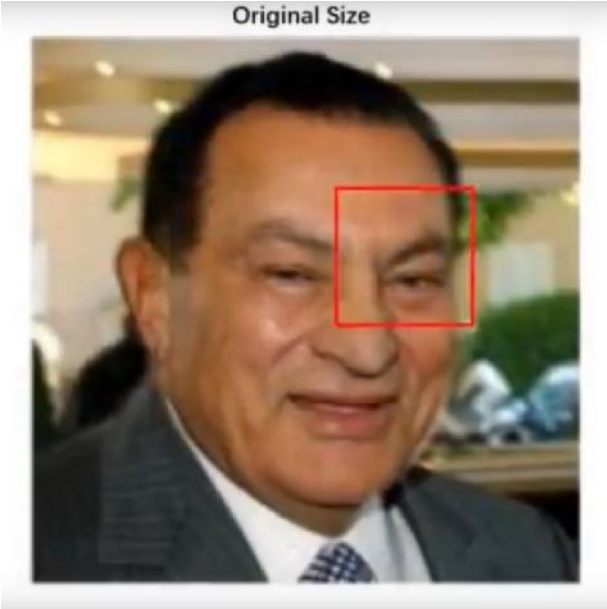
ядра 2 пікселями вправо або вниз за один раз. Зсув на 2 пікселі відомий як крок, або скільки пікселів ядро переміщується кожен раз.

Досягнення кроку в 2 (при скануванні зображення) допомагає зменшити складність обчислень без істотної шкоди для точності. Оскільки обличчя у більшості зображень значно більше двох пікселів, то малоймовірно, що ядро пропустить обличчя лише тому, що воно змістило 2 пікселі. У той же час ваш комп'ютер (або будь-яка машина, на якій працює цей код) матиме чверть обсягу операцій для обчислення, завдяки чому програма запуститься швидше і з меншою кількістю пам'яті.

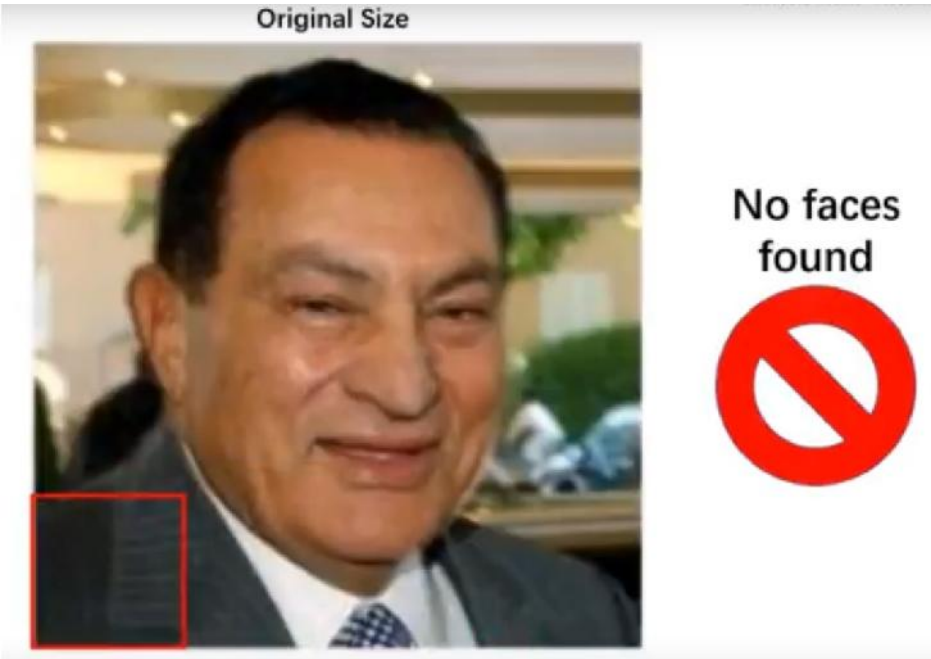
Єдиним недоліком є те, що нам доведеться перерахувати всі індекси, пов'язані з кроком. Наприклад, якщо ядро виявлено обличчя після переміщення на один крок вправо, індекс виводу повідомив би нам, що верхній лівий кут цього ядра знаходиться на рівні (1,0). Однак, оскільки крок 2, нам потрібно помножити індекс на 2, щоб отримати правильну координату: (2,0). Кожне ядро було б меншим відносно великого зображення, тому воно могло б знайти менші грані на зображенні з більшим масштабом. Так само ядро було б більшим відносно зображення меншого розміру, тому воно могло б знайти більші грані на зображенні меншого масштабу.

На наступному рисунку показано що відбудеться, якщо не масштабувати зображення. Скануюче вікно пропустить обличчя, якщо вікно суттєво менше за обличчя (рисунок 2.3). І локалізує обличчя, якщо обличчя менше за вікно (рисунок 2.4).

Після передачі зображення нам потрібно створити кілька масштабованих копій зображення і передати його в першу нейронну мережу - P-Net - і зібрати її вихід. Вибірка зразка для P-Net наведено на рисунку 2.5. Зауважимо, що фактичний вихід має 4 виміри, але для простоти я об'єднав його у двовимірний масив. Також координати для обмежувальних полів мають значення від 0 до 1: (0,0) було б лівим лівим кутом ядра, тоді як (1,1) - правою нижньою частиною ядра.



а)



б)

Рисунок 2.3 – Не знаходження обличчя, якщо вікно сканування замале [3]

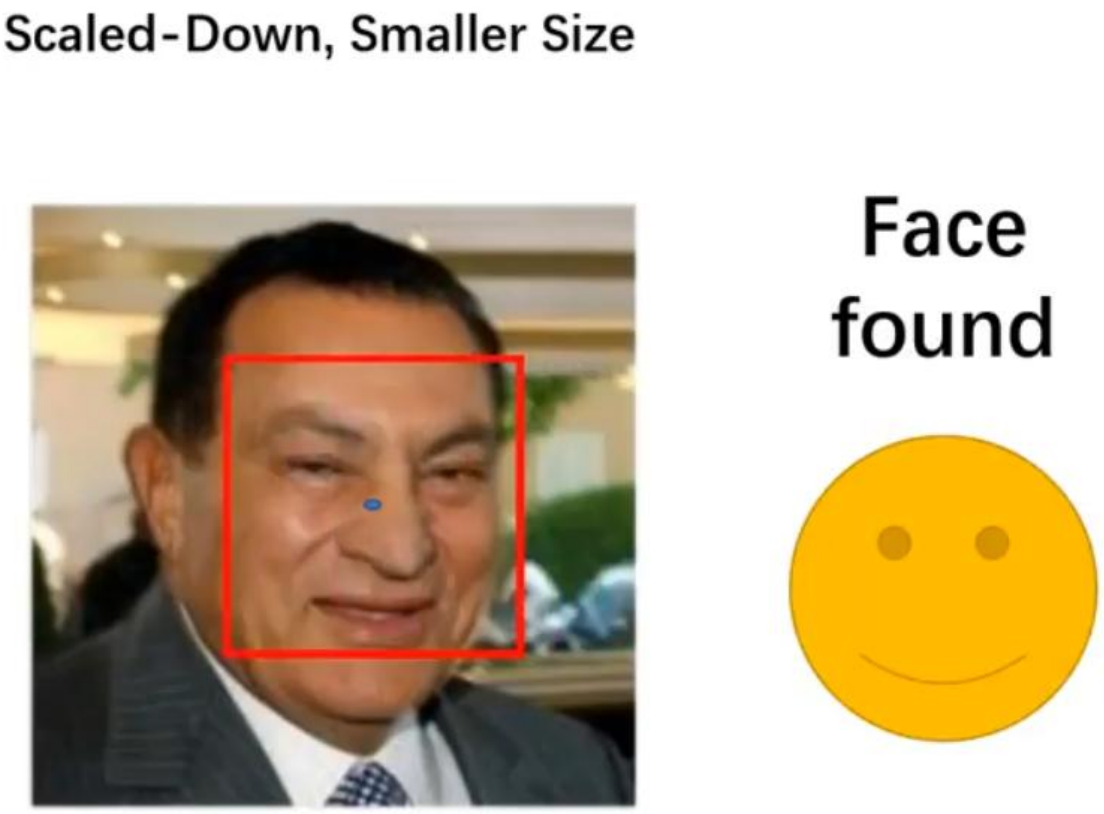


Рисунок 2.4 – Знаходження обличчя, якщо вікно сканування адекватне [3]

Kernel Coordinates	Bounding box: x1	Bounding box: y1	Bounding box: x2	Bounding box: y2	Confidence
(0,5)	0.50	0.25	0.80	0.58	0.98
(0,6)	0.45	0.17	0.75	0.49	0.96
(0,7)	0.52	0.08	0.73	0.41	0.94
(1,4)	0.42	0.34	0.67	0.66	0.92
(1,8)	0.40	0.01	0.66	0.32	0.96
(3,5)	0.32	0.22	0.49	0.59	0.88
(3,6)	0.25	0.20	0.52	0.53	0.91
(6,6)	0.01	0.18	0.25	0.50	0.89
(6,8)	0.02	0.00	0.22	0.33	0.90

Рисунок 2.5 – Вибірка для першої мережі P-Net [3]

Ваги та ухили P-Net були навчені таким чином, що він видає відносно точне обмежувальне поле для кожного ядра 12 x 12. Однак мережа впевнено стосується деяких ящиків порівняно з іншими. Таким чином, нам потрібно проаналізувати вихід P-Net, щоб отримати список рівнів довіри для кожного обмежувального вікна та видалити поля з меншою довірою (тобто поля, для яких мережа не зовсім впевнена, містять обличчя, рисунок 2.6).

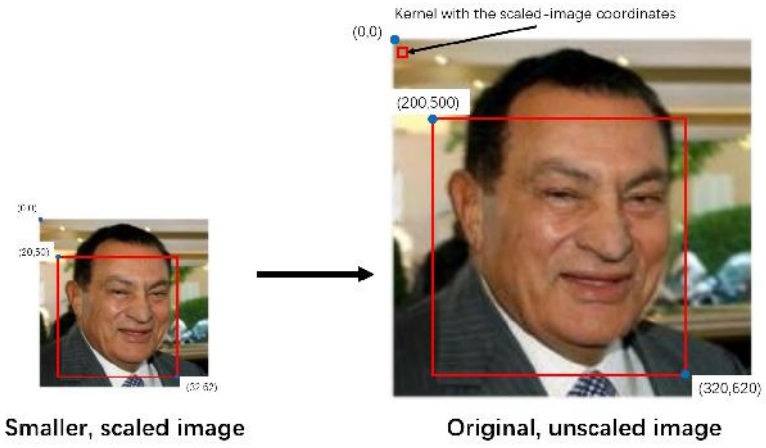


Рисунок 2.6 – Стандартизація координат ядра множенням на масштаб [3]

Після того, як ми відібрали вікна з більшою впевненістю, нам доведеться стандартизувати систему координат, перетворивши всі системи координат у фактичне, «не масштабоване» зображення. Оскільки більшість ядер знаходяться у зменшеному зображенні, їх координати будуть ґрунтуватися на меншому зображенні. Однак залишилося багато обмежувальних коробок, і багато з них перекриваються. Немаксимальне придушення або NMS – це метод, що зменшує кількість обмежувальних коробок.

У цій конкретній програмі NMS проводиться шляхом попереднього сортування обмежувальних коробок (та їх відповідних ядер 12 x 12) за їх достовірністю чи оцінкою. У деяких інших моделях NMS приймає найбільшу обмежувальну рамку замість тієї, в яку найбільше впевнена мережа. Згодом

ми обчислюємо площу кожного з ядер, а також область перекриття між кожним ядром та ядром з найвищим балом. Ядра, які сильно перекриваються ядром з високим балом, видаляються. Нарешті, NMS повертає список «вцілілих» обмежувальних коробок (рисунок 2.7).

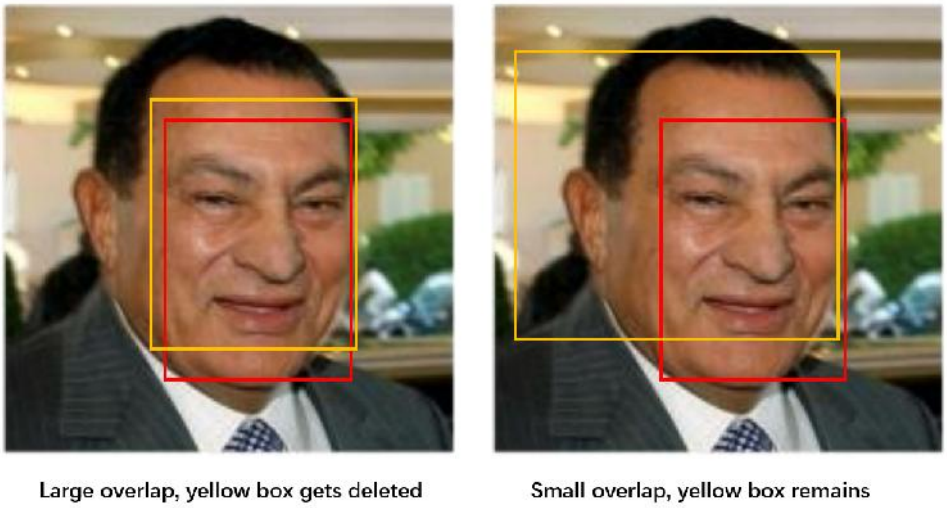


Рисунок 2.7 – Не максимальне придушення [3]

Ми проводимо NMS один раз для кожного масштабованого зображення, а потім ще раз із усіма збереженими ядрами з кожної шкали. Це позбавляється від зайвих обмежувальних коробок, що дозволяє нам звузити пошук до одного точного поля на обличчя.

Чому ми не можемо просто вибрати поле з найбільшою впевненістю та видалити все інше? На зображенні вгорі є лише одне обличчя. Однак на інших зображеннях може бути більше одного обличчя. Якщо так, ми в кінцевому підсумку видалимо всі рамки для інших граней.

Потім ми перетворюємо координатні рамки в координати фактичного зображення. Зараз координати кожного обмежувального поля є значенням між 0 та 1, причому (0,0) є лівим верхнім кутом ядра 12 x 12 та (1,1) як правому нижньому куті. Помноживши координати на фактичну ширину та

висоту зображення, ми можемо перетворити координатні рамки в стандартні координати зображення в реальному розмірі (рисунок 2.8).

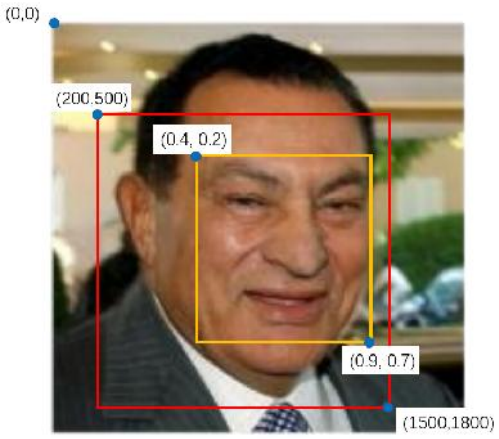


Рисунок 2.8 – Тут червоне поле – це ядро 12 x 12, тоді як жовте поле – це обмежувальне поле всередині нього [3]

На цьому зображенні червоне поле відображає ядро 24 x 24, яке змінилось розміром до початкового зображення. Ми можемо обчислити ширину та висоту ядра: 1500–200 = 300, 1800–500 = 300 (Зверніть увагу, як ширина та висота не обов’язково 12. Це тому, що ми використовуємо координати ядра в оригіналі зображення. Ширина та висота, яку ми отримуємо тут, - це ширина та висота ядра, коли повертаємо його до початкового розміру.) Потім ми множимо координатні рамки на 300: 0,4x300 = 120, 0,2x300 = 60, 0,9x300 = 270, 0,7x300 = 210. Нарешті, ми додаємо верхню ліву координату ядра, щоб отримати координати граничного поля: (200 + 120, 500 + 60) і (200 + 270, 500 + 210) або (320,560) та (470,710).

Оскільки обмежувальні коробки можуть бути не квадратними, ми переставляємо обмежувальні коробки на квадрат, подовжуючи більш короткі сторони (якщо ширина менша за висоту, ми розширюємо її набік; якщо висота менша за ширину, ми розширюємо її вертикально).

Нарешті, ми зберігаємо координати обмежувальних коробок і передаємо їх на 2 етап (до наступної нейронної мережі).

Іноді зображення може містити лише частину обличчя, що визирає з боку кадру. У такому випадку мережа може повернути обмежувальне поле, яке частково знаходиться поза рамкою, як обличчя Пола Маккартні на фотографії нижче (рисунок 2.9).

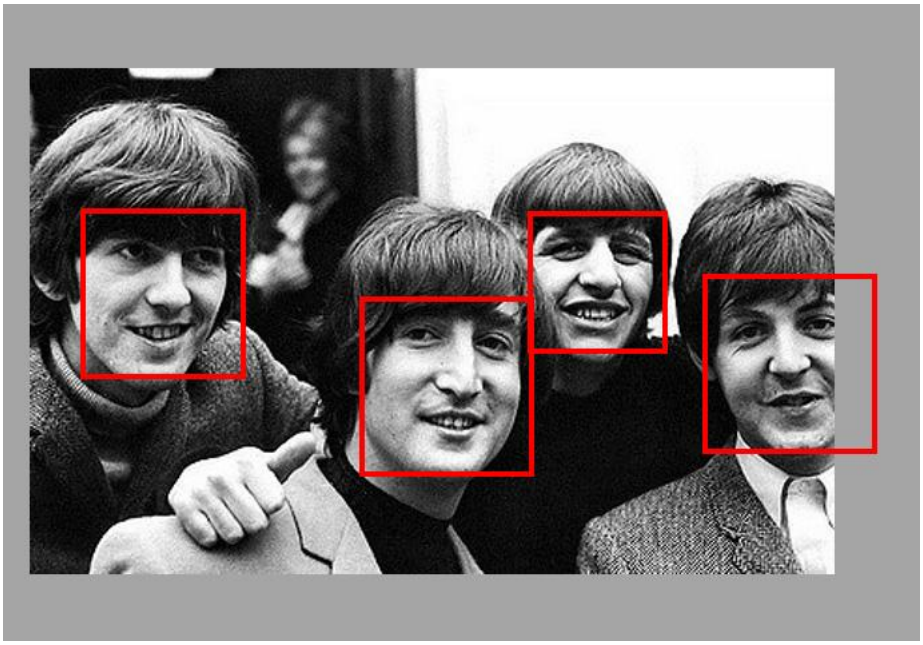


Рисунок 2.9 – «Бітлз» та їх обмежувальні рамки. Рамка Пола Маккартні виходить за межі і вимагає підкладки [3]

Для кожного обмежувального вікна ми створюємо масив однакового розміру та копіюємо значення пікселів (зображення у обмежувальному полі) у новий масив. Якщо обмежувальне поле не виходить за межі, ми лише копіюємо частину зображення в обмежувальному полі до нового масиву і заповнюємо все інше 0. На зображенні вище новий масив для обличчя Маккартні матиме значення пікселів у ліва сторона поля та кілька стовпців розміром 0s біля правого краю. Цей процес заповнення масивів з 0s називається padding.

Після того як ми прокладемо масиви обмежувальної коробки, ми змінимо їх розмір до 24 x 24 пікселів і нормалізуємо їх до значень від -1 до 1. В даний час значення пікселів становлять від 0 до 255 (значення RGB).

Віднімаючи кожне значення пікселя на половину 255 (127,5) і ділимо його на 127,5, ми можемо зберегти їх значення між -1 і 1.

Тепер, коли ми маємо численні масиви зображень 24 x 24 (стільки ж, скільки обмежувальних коробок, які пережили Етап 1, оскільки кожне з цих обмежувальних коробок було змінено і нормалізовано в ці ядра), ми можемо подавати їх у R-Net та збирати його вихід.

Вихід R-Net схожий з P-Net: Він включає координати нових, більш точних обмежувальних коробок, а також рівень довіри кожного з цих обмежувальних коробок.

Ще раз ми позбавляємось ящиків із меншою впевненістю та виконуємо NMS на кожній коробці для подальшого усунення зайвих коробок. Оскільки координати цих нових обмежувальних полів базуються на обмежувальних полях P-Net, нам потрібно перетворити їх у стандартні координати.

Після стандартизації координат ми переставляємо обмежувальні поля на квадрат, який слід передати O-Net.

Перш ніж ми зможемо перейти в обмежувальні коробки від R-Net, ми повинні спочатку прокладати будь-які ящики, які перебувають поза межами. Потім, після зміни розміру коробки до 48 x 48 пікселів, ми можемо перейти в обмежувальні поля в O-Net.

Виходи O-Net трохи відрізняються від результатів P-Net та R-Net. O-Net забезпечує 3 виходи: координати обмежувального поля (out [0]), координати 5 орієнтирів обличчя (out [1]) та рівень довіри кожного поля (out [2]). Ще раз ми позбавляємося від ящиків із нижчим рівнем довіри та стандартизуємо як координатні рамки, так і координати орієнтиру обличчя. Нарешті, ми запускаємо їх через останню NMS. У цей момент на зображенні повинно бути лише одне обмежувальне поле.

Останній крок – упакувати всю інформацію у словник із трьома полями: «поле», «впевненість» та «ключові точки». "Поле" містить координати обмежувального поля, «впевненість» містить рівень довіри

мережі для кожного вікна, а «ключові точки» містить координати кожного орієнтира обличчя (очі, ніс і кінцеві точки рота).

Коли ми хочемо реалізувати цю модель у власному коді, все, що нам потрібно зробити, – це calltect_faces, і нам буде надано цей словник з усіма необхідними координатами, щоб намалювати обмежувальні поля та позначити риси обличчя [3].

2.2 Модель YOLOv3

Загальна модель нейронної мережі YOLOv3 наведена на рисунок 2.10, а її використання – на рисунок 2.11. «Ви дивитесь лише один раз» - алгоритм, який використовує згорткові нейронні мережі для виявлення об’єктів. Ви дивитесь лише один раз; YOLO – це один із швидших алгоритмів виявлення об’єктів. Хоча це не найточніший алгоритм виявлення об’єктів, але це дуже хороший вибір, коли нам потрібно виявлення в режимі реального часу.

Порівняно з алгоритмами розпізнавання, алгоритм виявлення не тільки прогнозує мітки класу, але і виявляє розташування об’єктів. Отже, він не тільки класифікує зображення до категорії, але також може виявляти кілька об’єктів у межах зображення. Цей алгоритм застосовує єдину нейронну мережу до повного зображення. Це означає, що ця мережа ділить зображення на регіони та прогнозує обмежувальні поля та ймовірності для кожного регіону. Ці граничні поля зважуються за прогнозованими ймовірностями.

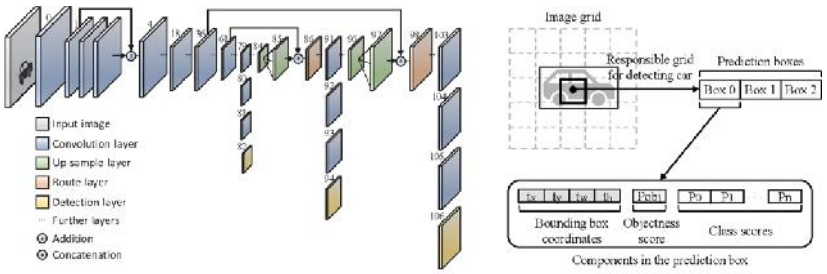


Рисунок 2.10 – Архітектура YOLOv3 [19]

Рисунок 2.11 – Робота YOLOv3, де: Коло – OpenCV Haar Cascade Face Detector, прямокутник – Deepsight YOLO Face Detector [20]

YOLO використовує лише згоркові шари, що робить його повністю згортковою мережею (FCN). У статті YOLO v3 автори представляють нову, глибшу архітектуру екстрактора функцій під назвою Darknet-53. Як видно з назви, він містить 53 згорткових шари, за кожним із них слід пройти рівень нормалізації партії та активацію Leaky ReLU. Жодна форма об’єднання не використовується, а згортковий шар з кроком 2 використовується для зменшення вибірки карт функцій. Це допомагає запобігти втраті функцій низького рівня, які часто відносять до об’єднання (рисунок 2.12).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рисунок 2.12 – Модель Darknet-53 [21]

YOLO інваріантний розміру вхідного зображення. Однак на практиці ми можемо захотіти дотримуватися постійного розміру вводу через різні проблеми, які показують голову лише тоді, коли ми реалізуємо алгоритм.

Великою проблемою серед цих проблем є те, що якщо ми хочемо обробляти свої зображення партіями (зображення в партіях можуть оброблятися паралельно GPU, що призводить до збільшення швидкості), нам потрібно мати всі зображення фіксованої висоти та ширини. Це потрібно для об’єднання декількох зображень у велику партію.

Мережа зменшує вибір зображення за допомогою фактора, який називається кроком мережі. Наприклад, якщо крок мережі становить 32, то вхідне зображення розміром 416 x 416 дасть вихід з розміром 13 x 13. Як правило, крок будь-якого шару в мережі дорівнює коефіцієнту, на який виходить шар менший, ніж вхідне зображення в мережу.

Перейдемо до інтерпретації результату. Перші речі, які потрібно знати: вхід – це група зображень форми (m, 416, 416, 3), а вихід – це список обмежувальних полів разом з розпізнаними класами. Кожне обмежувальне поле представлено 6 числами (pc, bx, by, bh, bw, c). Якщо розкласти с на 80-мірний вектор, кожне обмежувальне поле подається 85 числами.

Так само, як і у всіх об’єктових детекторах, ознаки, засвоєні згортковими шарами, передаються класифікатору / регресору, який робить передбачення виявлення (координати обмежувальних коробок, мітки класу тощо).

У YOLO прогнозування здійснюється за допомогою згорткового шару, який використовує 1 x 1 згортки. Отже, перше, що слід помітити, це наш вихід - карта функцій. Оскільки ми використовували 1x1 згортки, розмір карти прогнозування є саме розміром карти об’єктів перед нею. У YOLO v3 спосіб інтерпретації цієї карти передбачення полягає в тому, що кожна комірка може передбачити фіксовану кількість обмежувальних полів.

Наприклад, якщо на карті об’єктів є записи (B x (5 + C)). B представляє кількість обмежувальних коробок, які кожна клітина може передбачити. Згідно з документом, кожне з цих обмежувальних коробок Б може спеціалізуватися на виявленні певного виду об’єктів. У кожному з обмежувальних полів є атрибути 5 + C, які описують центральні координати, розміри, показник об’єктності та конфіденційність класу C для кожного обмежувального вікна. YOLO v3 прогнозує 3 обмежуючих поля для кожної комірки.

Ми очікуємо, що кожна клітинка карти об’єктів спрогнозує об’єкт через одне з обмежувальних вікон, якщо центр об’єкта потрапить у

сприйнятливе поле цієї комірки. Це пов’язано з тим, як навчається YOLO, де лише одне обмежувальне вікно відповідає за виявлення будь-якого заданого об’єкта. По-перше, ми повинні встановити, до якої з комірок належить цей обмежувальний ящик. Для цього ми ділимо вхідне зображення на сітку розмірів, рівних розміру кінцевої карти функцій. Подивіться на приклад нижче, де вхідне зображення - 416 x 416, а кроковий рух мережі - 32. Як було зазначено раніше, розміри карти функцій становитимуть 13 x 13. Потім розділимо вхідне зображення на 13 x 13 клітини (рисунок 2.13).

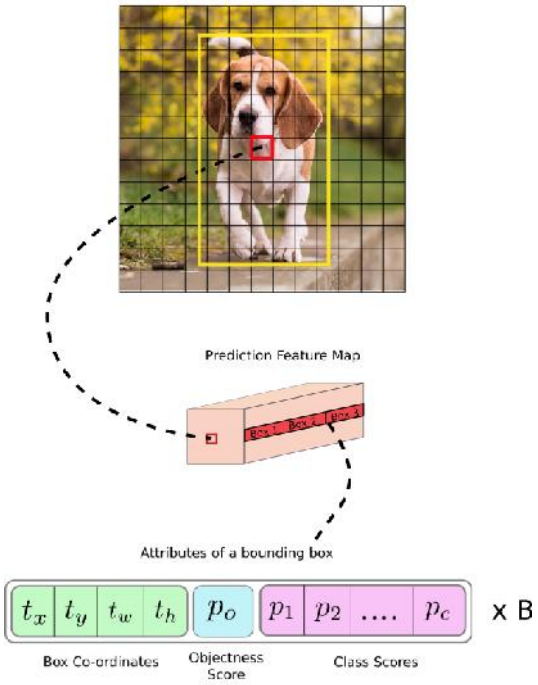


Рисунок 2.13 – Модель прогнозування [21]

Потім клітинку (на вхідному зображенні), що містить центр основного поля істини об’єкта, вибирають таким, що відповідає за прогнозування об’єкта. На зображенні це клітина, позначена червоним кольором, що містить центр основного поля правди (позначений жовтим кольором).

Тепер червона клітинка - це 7-а клітинка в 7-му ряду в сітці. Тепер ми призначаємо 7-ю клітинку в 7-му ряду на карті об’єктів (відповідну клітинку на карті об’єктів) як ту, що відповідає за виявлення собаки.

Тепер ця клітина може передбачити три обмежуючих поля. Хто з них буде присвоєний етикетці правдивої собаки? Для того, щоб зрозуміти це, ми повинні обернути голову навколо концепції якорів. (Зауважте, що комірка, про яку ми говоримо тут, - це клітина на карті функції передбачення. Ми ділимо вхідне зображення на сітку лише для того, щоб визначити, яка комірка карти функції передбачення відповідає за прогнозування). Для кращого розуміння ми можемо проаналізувати ще один приклад із більшою кількістю зображень.

Зараз ми будемо використовувати 5 анкерних ящиків. Тож ми можемо думати про архітектуру YOLO як наступне: IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85). Хтось може запитати, звідки взялося це 19 число? Відповідь: так само, як і раніше 608/32 = 19608/32 = 19. Давайте докладніше розглянемо, що представляє це кодування (рисунок 2.14).

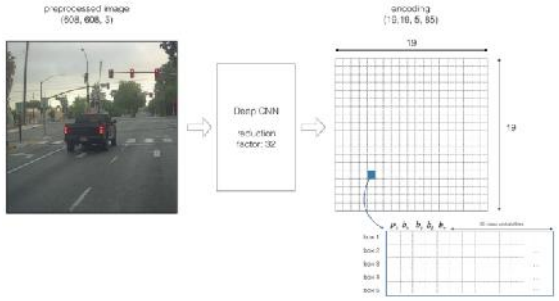


Рисунок 2.14 – Модель прогнозування [21]

Якщо центр/середина об’єкта потрапляє в комірку сітки, ця комірка відповідає за виявлення цього об’єкта. Оскільки ми використовуємо 5 ядерних ящиків, кожна з комірок 19x19 таким чином кодує інформацію про 5 ящиків. Якорні ящики визначаються лише їх шириною та висотою. Для простоти ми вирівняємо останні два розміри кодування форми (19, 19, 5, 85). Отже, вихід Deep CNN дорівнює (19, 19, 425), як це показано на рисунку 2.14.

Тепер для кожного поля (кожної комірки) ми обчислимо наступний елемент нового продукту та витягнемо ймовірність того, що поле містить певний клас (рисунок 2.15).

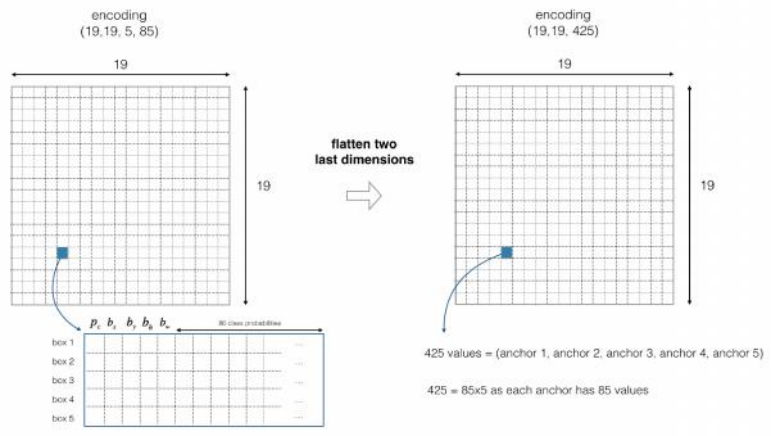


Рисунок 2.15 – Модель вихіда мережі [21]

Ось один із способів візуалізації того, що YOLO прогнозує на зображенні: для кожної з комірок сітки 19x19 знайдіть максимум балів ймовірності (взяття максимуму як через 5 ящикових ящиків, так і для різних класів). Колір цієї комірки сітки залежно від того, який об’єкт вважає комірку сітки найбільш ймовірним (рисунок 2.16).

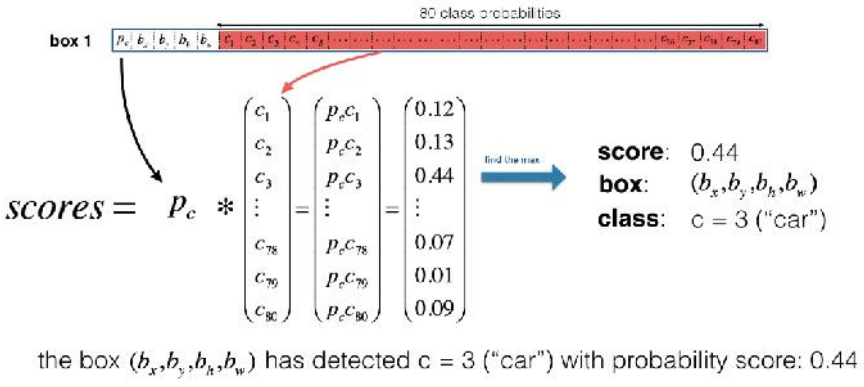


Рисунок 2.16 – Модель прогнозування об’єкту [21]

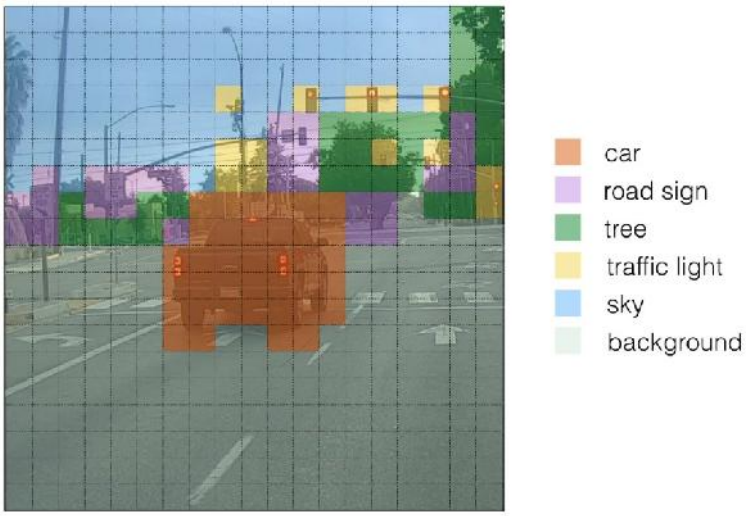


Рисунок 2.17 – Результат прогнозування положення об’єкту [21]

Наведена вище візуалізація не є основною частиною самого алгоритму YOLO для прогнозування; це просто хороший спосіб візуалізації проміжного результату алгоритму. Інший спосіб візуалізації результатів YOLO – побудувати графіки обмежувальних полів, які він виводить. Це призводить до такої візуалізації (рисунок 2.18).

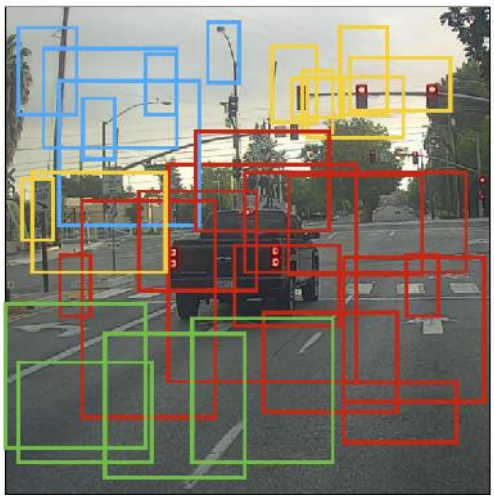


Рисунок 2.18 – Інший формат виводу результатів прогнозування [21]

На рисунку вище ми побудували лише рамки, яким модель призначала велику ймовірність, але це все ще занадто багато коробок. Ми хотіли б

відфільтрувати вихід алгоритму до значно меншої кількості виявлених об’єктів. Для цього ми будемо використовувати придушення без максимальних значень.

Може мати сенс передбачити ширину та висоту обмежувальної коробки, але на практиці це призводить до нестабільних градієнтів під час тренувань. Натомість більшість сучасних детекторів об’єктів прогнозують перетворення логічного простору або просто зміщують заздалегідь задані обмежувальні поля, що називаються якорями. Потім ці перетворення застосовуються до анкерних коробок для отримання прогнозу. YOLO v3 має три якоря, в результаті чого передбачати три обмежувальні поля на комірку.

Якори – це різновиди обмежувачів вікна обмеження, які були обчислені на наборі даних COCO, використовуючи кластеризацію k-середніх. Ми будемо прогнозувати ширину та висоту коробки як компенсацію від центроїдів кластера. Центральні координати поля щодо місця застосування фільтра прогнозуються за допомогою сигмоїдної функції.

Розміри обмежувального вікна прогнозуються, застосувавши перетворення журнального простору до виводу, а потім помноживши на якір.

Тут прогнози bw і bh нормалізуються по висоті та ширині зображення. (Навчальні ярлики вибираються таким чином). Отже, якщо передбачення bx та by для поля, що містить собаку, дорівнюють (0,3, 0,8), то фактична ширина та висота на карті об’єктів 13 x 13 дорівнює (13 * 0,3, 13 * 0,8) (рисунок 2.19).

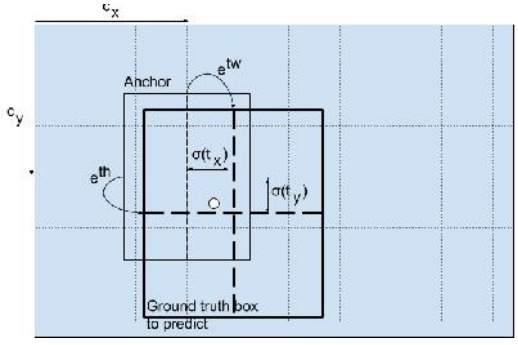


Рисунок 2.19 – Прогнозування положення і розмірів рамок [21]

Величина всіх оцінок об’єкта представляє ймовірність того, що об’єкт міститься в обмежувальному полі. Це має бути майже 1 для червоної та сусідньої сітки, тоді як майже 0 для, скажімо, сітки в кутах.

Оцінка об’єктивності також передається через сигмоїду, оскільки її слід трактувати як ймовірність.

Якщо говорити про конфіденційність класу, вони представляють вірогідність виявленого об’єкта, що належить до певного класу (Собака, кішка, людина, машина, велосипед тощо). У старих версіях YOLO він використовувався для програмування оцінок класу.

В YOLO автори вирішили використовувати натомість сигмоїд. Причина полягає в тому, що оцінки класів Softmaxing припускають, що класи взаємно виключають. Простими словами, якщо об’єкт належить до одного класу, то він не може належати до іншого класу. Це стосується бази даних COCO, яку ми реалізуємо спочатку.

YOLOv3 прогнозує для трьох різних масштабів. Шар виявлення використовується для виявлення на картах характеристик трьох різних розмірів, що мають кроки 32, 16, 8 відповідно. Це означає, що з введенням 416 x 416 ми робимо виявлення за масштабами 13 x 13, 26 x 26 та 52 x 52.

Мережа зменшує вибірку вхідного зображення до першого шару виявлення, де виявлення проводиться за допомогою функціональних карт шару з кроком 32. Далі шари збільшуються зразком на коефіцієнт 2 і з’єднуються з картами функцій попередніх шарів, що мають ідентичну карту характеристик розміри. Інше виявлення тепер проводиться на шарі з кроком 16. Таку ж процедуру перебору вгорі повторюють, і остаточно виявляють на шарі кроку 8. У кожній шкалі кожна комірка прогнозує 3 обмежувальні коробки за допомогою 3 якорів, що робить загальну кількість використовуваних якорів 9. Це допомагає покращити виявлення дрібних предметів, часті скарги на більш ранні версії YOLO. Підвищення розміщення може допомогти мережі вивчити дрібнодисперсні функції, які є важливими для виявлення дрібних об’єктів.

Стосовно Обробки результатів фільтрування. Для зображення розміром 416 x 416 YOLO прогнозує ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 обмежувальних коробок. Однак у випадку із нашим зображенням є лише один об’єкт - собака. Отже, ви можете запитати, як ми зменшуємо виявлення з 10647 до 1? Спочатку ми фільтруємо коробки на основі їх оцінки об’єктивності. Як правило, поля, що мають бали нижче порогового рівня (наприклад, нижче 0,5), ігноруються. Далі, Немаксимальне придушення (NMS) має намір усунути проблему декількох виявлень одного зображення. Наприклад, усі 3 обмежувальні поля червоної комірки сітки можуть виявляти коробку або сусідні комірки можуть виявляти один і той же об’єкт, тому NMS використовується для видалення декількох виявлень.

Навіть після фільтрування за пороговою шкалою за балами класів ми все-таки закінчуємо безліч перекриваючих скриньок. Другий фільтр для вибору потрібних вікон називається NMS. NMS використовує дуже важливу функцію, яку називають «Перехрестя над Союзом» або IoU (рисунок 2.20).

Розглянуті нейронні мережі MTCNN і YOLOv3 на теперішній час є самими ефективними для знаходження обличчя людини на фотографії.

Мережа MTCNN дає більшу точність і 5 особливих точок на обличчі (2 центра ока, кінчик носу і кути рота). Але працює повільніше чим YOLOv3. У наступному розділі ці нейронні мережі досліджуються практично на основі чого робиться висновок стосовно ефективності їх практичної реалізації.

Інші мережі, наприклад, відома мережа R-CNN, э дуже неефективною як с точки зору навчання, так і з точки зору функціонування. Тим більше у вбудованих системах. Тому далі розглядаємо лише мережі MTCNN і YOLOv3.

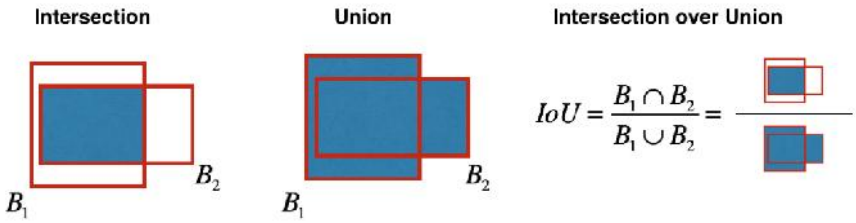


Рисунок 2.20 – Фільтр IoU [21]

3 ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ У ВБУДОВАНИХ СИСТЕМАХ МАШИННОГО ЗОРУ

3.1 Імплементація нейронної мережі MTCNN

Для вирішення питання стосовно використання типу нейронної мережі у середині вбудованої системи технічного зору було проведено експеримент.

А саме було проведено порівняльній аналіз двох моделей – YOLOv3 та MTCNN. Для цього були використані дані з набору Wider Face [22]. Це набір даних базового рівня виявлення обличчя, який складається з 32 203 обраних зображень із загальнодоступних наборів даних WIDER. Зображення в наборі дуже різноманітні і спеціально призначені для адекватного машинного навчання. У наборі даних є портретні зображення, групові зображення, зображення зі спортивних змагань тощо. Різноманітність зображень допомагає нам порівнювати продуктивність алгоритмів за неоптимальних умов. Після проведення експерименту було встановлено, що мережа MTCNN надає майже на 10% більшу точність, ніж мережа YOLO. Крім того мережа MTCNN прогнозує 5 особливих точок на обличчі, причому, також за умов, коли ми не бачимо обличчя цілком (рисунок 3.1). Базуючись на цьому ми можемо вирішувати низку інших задач. Наприклад, розділяти обличчя на зони для розпізнавання емоційного стану людини.

3.2 Імплементація нейронної мережі YOLOv3

Проте мережа YOLOv3 має свої переваги. Вона майже у три рази швидша за MTCNN. Крім того, вона спроектована таким чином, щоб настроюватися на розпізнавання об’єктів різних класів. Тому вона має переваги перед MTCNN у відношенні якості розпізнавання об’єктів багатьох класів. Її слід застосовувати за основу для вбудованих систем як найшвидшу.

Рисунок 3.1 – Результат імплементації нейронної мережі MTCNN [23]

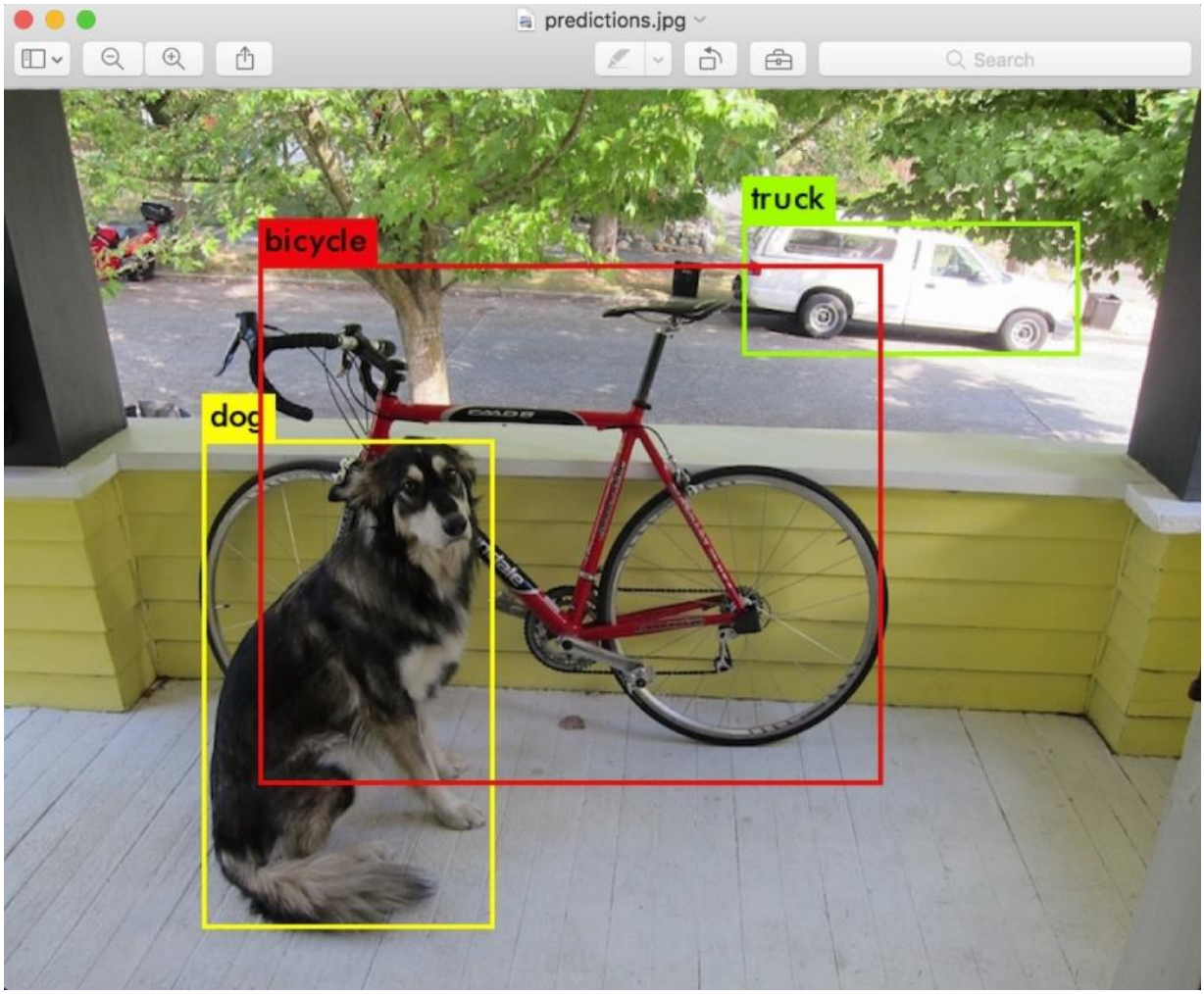


Рисунок 3.2 – Результат імплементації нейронної мережі YOLOv3 [24]

ВИСНОВКИ

В результаті виконання роботи можна зробити наступні висновки.

Згідно завданню проаналізовані основні моделі сучасних нейронних мереж та особливості їх практичного застосування.

Показано, що мережа YOLOv3 є найшвидшою і за рахунок своєї архітектури, у середньому, найбільш адекватно класифікує об’єкти різних класів, якщо вони присутні на зображенні. Тому її можна рекомендувати і застосовувати за основу для вбудованих систем як найшвидшу.

Проте якщо ми плануємо розпізнавати об’єкти якогось конкретного класу, то ліпше використовувати таку мережу, яка спеціально для цього призначена.

Так після проведення експерименту було встановлено, що мережа MTCNN надає майже на 10% більшу точність, ніж мережа YOLO, при знаходженні обличчя на зображенні. Крім того мережа MTCNN прогнозує 5 особливих точок на обличчі, причому, також за умов, коли ми не бачимо обличчя цілком. Базуючись на цьому ми можемо вирішувати низку інших задач. Наприклад, розділяти обличчя на зони для розпізнавання емоційного стану людини. Таким чином, для роботи з обличчями людини слід застосовувати нейронну мережу MTCNN.

Сформульовані висновки цілком підтверджуються результатами проведення експериментів на реальних даних. Обидві архітектури MTCNN і YOLOv3 за своїми характеристиками є прийнятними для вбудованих систем за критеріями часу і точності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ч. Аггарвал Нейронные сети и глубокое обучение. – М: Диалектика-Вильямс, 2020. – 752 с.

2. Саймон Хайкин Нейронные сети. – М: Диалектика, 2018. – 1104 с.

3. MTCNN Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff.

4. YOLO Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006.

5. R-CNN Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

6. Шапиро Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман. – М. : БИНОМ, 2006. – 752 с.

7. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М. : Техносфера, 2005. – 1072 с.

8. Sonka M. Image processing, analysis and machine vision / M. Sonka, V. Hlavak, R. Boyle. – California (USA) : Cole Publishing Company, 1999. – 770 p.

9. Форсайт Д. Компьютерное зрение. Современный подход / Д. Форсайт, Ж. Понс. – М. : Вильямс, 2004. – 928 с.

10. Яне Б. Цифровая обработка изображений / Б. Яне. – М. : Техносфера, 2007. – 584 с.

11. Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef.

12. Activation Function [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

13. Convolutional Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

14. RGB Image [Електронний ресурс] – Режим доступу до ресурсу: http://www.adsell.com/scanning101.html.

15. Gradient [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/How-does-stochastic-gradient-descent-work.

16. Методы компьютерной обработки изображений / под ред. В. А. Сойфера. – М. : Физматлит, 2003. – 784 с.

17. Шлезингер М. И. Десять лекций по статистическому и структурному распознаванию / М. И. Шлезингер, В. Главач. – К. : Наукова думка, 2004. – 545с.

18. Машталир В. П. Точечно-множественные методы обработки информации / В. П. Машталир. – Х. : Бизнес Информ, 2001. – 199 с.

19. YOLO Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.semanticscholar.org/paper/Gaussian-YOLOv3%3A-An-Accurate-and-Fast-Object-Using-Choi-Chun/e263db 19d 071 480387 9727801f1 a700d928f9233.

20. YOLO Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.baseapp.com/deepsight/opencv-vs-yolo-face-detector/.

21. YOLOv3 Neural Network [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/yolo-v3-theory-explained-33100f6d193.

22. Wider Face Dataset [Електронний ресурс] – Режим доступу до ресурсу: http://shuoyang1213.me/WIDERFACE/.

23. MTSN [Електронний ресурс] – Режим доступу до ресурсу: https://kpzhang93.github.io/MTCNN_face_detection_alignment/.

24. YOLOv3. YOLOv3 [Електронний ресурс] / YOLOv3 – Режим доступу до ресурсу: https://pjreddie.com/darknet/yolo/.