Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Оптимізація дорожнього руху
з використанням штучного інтелекту
(тема)

Виконав:
студент 2 курсу, групи КІТм-20-1
Костров Д.Р.
(прізвище, ініціали)

Спеціальність 123 Комп'ютерна інженерія
Тип програми освітньо-професійна
Освітня програма Комп'ютерні інтелектуальні технології
Керівник проф. Руденко О.Г.
(посада, прізвище, ініціали)

Допускається до захисту _____
(підпис)

Зав. кафедри КІТС _____ проф. Руденко О.Г.
(підпис)

2021 р.

Харківський національний університет радіоелектроніки

Факультет	Комп’ютерної інженерії та управління
Кафедра	Комп’ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп’ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп’ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кострову Д.Р.
(прізвище, ініціали)

1. Тема роботи (проекту) Оптимізація дорожнього руху з використанням з використанням штучного інтелекту

затверджена наказом по університету від “ 08 ” листопада 2021 р. № 1666Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 грудня 2021 р.

3. Вхідні дані до роботи Актуальність інтелектуальної системи. Теоретичні відомості про способи обробки зображення з камер та аналіз часових рядів. Вибір кращого алгоритму для розпізнавання зображення та прогнозування часових рядів. Результати тестування системи.

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області
Постановка задачі
Знаходження рішення оптимізації дорожнього руху
Вибір архітектури нейромережі для визначення об’єктів
Вибір архітектури нейромережі для прогнозування часових рядів
Реалізація програмного забезпечення
Тестування системи
Аналіз результатів роботи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням кафедри)

8 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному 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	Видача та узгодження теми проекту	08.11.2021	
2	Аналіз проблемної галузі, постановка задачі,	09.11.2021 – 13.11.2021	
	вибір інструментальних засобів		
3	Розробка моделі штучного інтелекту	14.11.2021 – 18.11.2021	
4	Збір даних для часових рядів	19.11.2021 – 20.11.2021	
5	Розробка алгоритму прогнозування	21.11.2021 – 27.11.2021	
6	Проведення тестування системи	28.11.2021 – 30.11.2021	
7	Оформлення пояснювальної записки	01.12.2021 – 09.12.2021	
8	Перевірка виконаного проекту керівником	10.12.2021	
9	Захист проекту	16.12.2021 – 17.12.2021	

Дата видачі завдання ____ _____ 2021 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Руденко О.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 81 с., 30 рис., 1 дод., 16 джерел

ШТУЧНИЙ ІНТЕЛЕКТ, ОБРОБКА ЗОБРАЖЕНЬ, YOLO, OPENCV, TENSORFLOW, CNN, RNN, LSTM, ОПТИМІЗАЦІЯ ДОРОЖНЬОГО РУХУ

Метою кваліфікаційної роботи є створення системи яка за допомогою штучного інтелекту допоможе оптимізувати дорожній рух. Вибір найкращої архітектури нейронної мережі визначення об’єктів та для прогнозування часових рядів.

У результаті виконання кваліфікаційної роботи потрібно виявити сучасні проблеми дорожнього руху, та за допомогою технології комп’ютерного зору та прогнозування вирішити їх.

ABSTRACT

Qualification project: 81 pages, 30 figures, 1 appendice, 16 sources.

ARTIFICIAL INTELLIGENCE, IMAGE PROCESSING, YOLO, OPENCV, TENSORFLOW, CNN, RNN, LSTM, TRAFFIC OPTIMIZATION

The purpose of the qualification work is to create a system that with the help of artificial intelligence will help optimize traffic. Choosing the best neural network architecture to identify objects and predict time series.

As a result of the qualification work, it is necessary to identify current traffic problems and solve them with the help of computer vision and forecasting technology

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

АНОТАЦІЯ

КВАЛІФІКАЦІЙНОЇ РОБОТИ

рівень вищої освіти другий (магістерський)

Оптимізація дорожнього руху
з використанням штучного інтелекту
(тема)

Виконав:
студент 2 курсу, групи КІТм-20-1
Костров Д.Р.
(прізвище, ініціали)

Спеціальність 123 Комп'ютерна інженерія
Тип програми освітньо-професійна
Освітня програма Комп'ютерні інтелектуальні технології
Керівник проф. Руденко О.Г.
(посада, прізвище, ініціали)

2021 р.

ВСТУП

Штучний інтелект, нейронні мережі, інтелектуальні системи та пристрої, ці словосполучення вже стали невід'ємними частинами нашого життя. Зараз важко уявити себе без інтелектуальних пристроїв, кожна людина у кишені має смартфон приєднаний до мережі інтернет, більшість носить на руках фітнес браслети та смарт годинники, усі вони зчитують з нас данні та за допомогою штучного інтелекту намагаються поліпшити наше життя та своєчасно попередити про небезпеку. Тому важко уявити сферу у якій не застосовується штучний інтелект або тільки планується впровадження, бо за останні роки він показав свою ефективність. Штучний інтелект у порівняні з людиною показує кращі та точніші результати у процесі прогнозування та прийняття рішення.

Сьогодні у великих містах на дорогах можна спостерігати багато незадовільнених людей через те, що доводиться деякий час чекати у заторах. За даними щорічного дослідження Traffic Index у 2020 році Харків займає тринадцяте місце серед міст світу із самими великими заторами на дорогах, до цього списку входить більше чотирьох ста міст. Так через затори у пікові години поїздка у середньому збільшується на двадцять хвилин.

Багато процесів дорожнього руху можуть бути значно поліпшені. Кожен водій, якому доводиться чекати на світлофорі кілька хвилин поспіль, навіть якщо для цього немає видимої причини – за винятком того, що система світлофора працює за фіксованою схемою, яка повністю не залежить від поточної дорожньої ситуації, може мати до цього відношення. Використання штучного інтелекту для підтримки руху трафіку відповідно до поточної ситуації має багато переваг:

- безперервний рух, без пробок, це корисно для навколишнього середовища;

- дозволяє оптимізувати багато бізнес-процесів, такі як поставки, що приносить велику користь економіці.

ОСНОВНА ЧАСТИНА

Штучний інтелект (ШІ) відноситься до моделювання людського інтелекту в машинах, які запрограмовані на те, щоб мислити як люди і імітувати їх дії. Цей термін може також застосовуватися до будь-якої машини, яка виявляє риси, пов’язані з людським розумом, такі як навчання та вирішення проблем.

ШІ є найважливішою галуззю інформатики в епоху великих даних. ШІ народився п’ятдесят років тому і пройшов довгий шлях, досягнувши обнадійливого прогресу, особливо в області машинного навчання, інтелектуального аналізу даних, комп’ютерного зору, експертних систем, обробки природної мови, робототехніки.

Ідеальною характеристикою штучного інтелекту є його здатність раціоналізувати та вживати дії, які мають найбільші шанси досягти конкретної мети. Підмножиною штучного інтелекту є машинне навчання, яке стосується концепції, згідно з якою комп’ютерні програми можуть автоматично вчитися на нових даних і пристосовуватися до них без допомоги людей. Машинне навчання – найпопулярніша галузь штучного інтелекту. Інші класи ШІ включають імовірнісні моделі, системи штучних нейронних мереж, глибоке навчання та ін. Методи глибокого навчання дозволяють автоматично навчатися шляхом поглинання величезної кількості неструктурованих даних.

Метою кваліфікаційної роботи є створення інтелектуальної системи, яка у реальному часі отримує зображення з камер відеоспостереження та переводить їх у зрозумілі для комп’ютера данні. А потім за допомогою штучного інтелекту проводиться аналіз на основі якого приймається рішення для регулювання світлофора, щоб збільшити пропускну здатність трафіку. На сьогоднішній день майже у всіх випадках ми маємо звичайні світлофори, які працюють у певному статичному режимі. Системі потрібно слідкувати за усім, що відбувається на дорозі, чи стоять люди на світлофорі, чи переходять дорогу, де більша кількість машин утворює затор. Аналізуючи усі дані слід визначити, який саме колір

потрібно запалити на тому чи іншому світлофорі.

Затори на дорогах дають прямий і непрямий вплив на економіку країни і здоров'я її жителів. Якщо взяти світ в цілому, то можна сказати, що через довге очікування на перехресті, витрачається багато палива у порожню. Затори на дорогах впливають і на індивідуальному рівні. Втрата часу, особливо в години пік, психічний стрес і додаткове забруднення навколишнього середовища в результаті глобального потепління також є деякими важливими факторами, викликаними заторами на дорогах.

Згорткові нейронні мережі (ЗНМ) – це досить широкий клас архітектур, основна ідея яких полягає в тому, щоб повторно використати одні й ті ж частини нейронної мережі для роботи з різними маленькими, локальними ділянками входів. Як і багато інших нейронні архітектури, ЗНМ відомі досить давно, і в наші дні у них вже знайшлося багато найрізноманітніших застосувань, але основним додатком, заради якого люди колись придумали згорткові мережі, залишається обробка зображень.

Згорткові нейронні мережі забезпечують часткову стійкість до змін масштабу, зсувів, поворотів, зміни ракурсу та інших спотворень. Згорткові нейронні мережі об'єднують три архітектурних ідеї, для забезпечення інваріантності до зміни масштабу, повороту зсуву і просторовим спотворенням:

- локальні рецепторні поля (забезпечують локальну двовимірну зв'язність нейронів);
- загальні синаптичні коефіцієнти (забезпечують детектування деяких рис в будь-якому місці зображення і зменшують загальне число вагових коефіцієнтів);
- ієрархічна організація з просторовими підвибірками.

ЗНМ складається з різних видів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling, підвибірка) шари і шари «звичайної» нейронної мережі – персептрона.

Рекурентні нейронні мережі (RNN) – це сучасний алгоритм послідовної обробки даних, який використовується Siri від Apple і голосовим пошуком

Google. Це перший алгоритм, який запам’ятовує свої вхідні дані завдяки внутрішній пам’яті, що робить його ідеально відповідним для завдань машинного навчання, пов’язаних з послідовними даними. Це один з алгоритмів, що стоять за сценами дивовижних досягнень в області глибокого навчання за останні кілька років.

RNN – це потужний і надійний тип нейронної мережі, і він відноситься до найбільш перспективних використовуваних алгоритмів, тому що він єдиний, у якого є внутрішня пам’ять.

Як і багато інших алгоритмів глибокого навчання, рекурентні нейронні мережі відносно старі. Спочатку вони були створені в 1980-х роках, але тільки в останні роки можна побачити їх справжній потенціал. Збільшення обчислювальної потужності поряд з величезними обсягами даних, з якими тепер доводиться працювати, і винаходом довготривалої короткочасної пам’яті (LSTM) в 1990-х роках дійсно вивело RNN на передній план.

LSTM допомагають зберегти помилку, яку можна поширити назад через час і шари. Підтримуючи більш постійну похибку, вони дозволяють рекурентним мережам продовжувати навчатися протягом багатьох часових кроків (понад 1000), тим самим відкриваючи канал для дистанційного зв’язування причин і наслідків. Це одна з головних проблем машинного навчання та штучного інтелекту, оскільки алгоритми часто стикаються з середовищами, де сигнали винагороди рідкісні та відкладені.

LSTM містять інформацію за межами звичайного потоку рекурентної мережі в закритій комірці. Інформацію можна зберігати в комірці, записувати або зчитувати з неї, як дані в пам’яті комп’ютера. Комірка приймає рішення про те, що зберігати і коли дозволити читання, запис і стирання, через ворота, які відкриваються та закриваються. На відміну від цифрового сховища на комп’ютерах, однак, ці вентилі є аналоговими, реалізованими з по елементним множенням на сигмоїди, які всі знаходяться в діапазоні 0-1. Аналоговий має перевагу перед цифровим у тому, що він диференційований, і тому підходить для зворотного поширення.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було виявлено проблеми пов’язанні з заторами та вирішення їх за допомогою штучного інтелекту. Розглянуто як штучний інтелект розуміє зображення за допомогою згорткових мереж. Також виявлено, що для роботи інтелектуальної системи потрібні високопродуктивні сервери, бо обробка зображення вимагає великих затрат на обробку інформації, а також прогнозування результатів. Було розглянуто найпоширеніші архітектури нейронної мережі для визначення образів – дворівневі та одноріневі.

Одноріневі мережі використовують так звані регіони на зображенні, щоб визначити, чи знаходиться в цьому регіоні певний об’єкт. Але в цих нейронних мережах є дві ключові проблеми: вони не дивляться на картинку «повністю», а тільки на окремі регіони, і вони відносно повільні. YOLO краще тим, що ця архітектура не має двох проблем, і показує свою ефективність.

Отже підводячи підсумок можна сказати, що розглянута проблема в кваліфікаційній роботі є дуже актуальною на сьогоднішній день, люди по всьому світу намагаються вирішити її, але до цього часу це було складно зробити через не дуже розвинений штучний інтелект та невеликі обчислювальні потужності систем. Тому така інтелектуальна система у подальшому матиме великий попит.

Ключові слова: штучна нейронна мережа, машинне навчання, глибоке навчання, згорткова нейронна мережа, рекурентна нейронна мережа, Довга короткострокова пам’ять.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Костров Д.Р. Оптимізація дорожнього руху з використанням штучного інтелекту. *Радіоелектроніка та молодь у XXI столітті*: матеріали 25-го міжнар. мол. форуму. Харків, 2011. С. 185-186.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BPTT – Backpropagation through time

CNN – Convolutional Neural Networks

COCO – Common Objects in Context

DNN – Deep Neural Network

GPU – Graphics processing unit

LSTM – Long short-term memory

RNN – Recurrent Neural Network

YOLO – You Only Look Once

ГН – Глибоке навчання

ЗНМ – Згорткова нейронна мережа

ШІ – Штучний інтелект

ВСТУП

Штучний інтелект, нейронні мережі, інтелектуальні системи та пристрої, ці словосполучення вже стали невід'ємними частинами нашого життя. Зараз важко уявити себе без інтелектуальних пристроїв, кожна людина у кишені має смартфон приєднаний до мережі інтернет, більшість носить на руках фітнес браслети та смарт годинники, усі вони зчитують з нас данні та за допомогою штучного інтелекту намагаються поліпшити наше життя та своєчасно попередити про небезпеку. Тому важко уявити сферу у якій не застосовується штучний інтелект або тільки планується впровадження, бо за останні роки він показав свою ефективність. Штучний інтелект у порівняні з людиною показує кращі та точніші результати у процесі прогнозування та прийняття рішення [1].

Сьогодні у великих містах на дорогах можна спостерігати багато незадовільнених людей через те, що доводиться деякий час чекати у заторах. За даними щорічного дослідження Traffic Index у 2020 році Харків займає тринадцяте місце серед міст світу із самими великими заторами на дорогах, до цього списку входить більше чотирьох ста міст. Так через затори у пікові години поїздка у середньому збільшується на двадцять хвилин [2].

Транспортна мережа явно досягає своїх меж. Це було предметом громадського обговорення протягом десятиліть. Але як можна уникнути пробок, оптимізувати транспортний потік і знизити число дорожньо-транспортних пригод зі смертельними наслідками? Загальновідомо, що зміна правил дорожнього руху сама по собі не вирішить проблему.

Штучний інтелект може використовуватися як вибірково, так і комплексно для дорожнього руху і особливо для водіння. Деякі з функцій, в яких успішно використовується ШІ – це, наприклад, автоматичне розпізнавання відстані або паркування. Ці системи стають все більш складними і точними завдяки великим обсягам навчальних даних.

Штучний інтелект володіє величезним потенціалом в області дорожнього

руху. Це може зробити водіння не тільки більш комфортним і безпечним, але також більш екологічним і інтелектуальним. Передові алгоритми, засновані на принципах штучного інтелекту, машинного навчання і великих даних, враховують обсяг трафіку в режимі реального часу для оптимізації транспортного потоку і підвищення індивідуальної мобільності. В результаті користувач досягає місця призначення максимально швидко, безпечно та з комфортом.

Багато процесів дорожнього руху можуть бути значно поліпшені. Кожен водій, якому доводиться чекати на світлофорі кілька хвилин поспіль, навіть якщо для цього немає видимої причини – за винятком того, що система світлофора працює за фіксованою схемою, яка повністю не залежить від поточної дорожньої ситуації, може мати до цього відношення. Використання штучного інтелекту для підтримки руху трафіку відповідно до поточної ситуації має багато переваг:

- безперервний рух, без пробок, це корисно для навколишнього середовища;

- дозволяє оптимізувати багато бізнес-процесів, такі як поставки, що приносить велику користь економіці.

Всі ці фактори сприяють оптимізації загальної транспортної системи. Це вигідно кожному учаснику дорожнього руху, навіть тим, хто раніше міг брати участь у дорожньому русі лише в обмеженій мірі без допомоги цифрових інструментів.

На якість програмного забезпечення, призначеного для використання в дорожньому русі, впливає, з одного боку, програмування алгоритмів, але також в значній мірі кількість і якість навчальних даних. Чим надійніше і реалістичніше навчальні дані для машинного навчання, тим більше можливостей для безпечного проектування дорожнього руху.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття та визначення

Штучний інтелект (ШІ) відноситься до моделювання людського інтелекту в машинах, які запрограмовані на те, щоб мислити як люди і імітувати їх дії. Цей термін може також застосовуватися до будь-якої машини, яка виявляє риси, пов’язані з людським розумом, такі як навчання та вирішення проблем.

ШІ є найважливішою галуззю інформатики в епоху великих даних. ШІ народився п’ятдесят років тому і пройшов довгий шлях, досягнувши обнадійливого прогресу, особливо в області машинного навчання, інтелектуального аналізу даних, комп’ютерного зору, експертних систем, обробки природної мови, робототехніки.

Ідеальною характеристикою штучного інтелекту є його здатність раціоналізувати та вживати дії, які мають найбільші шанси досягти конкретної мети. Підмножиною штучного інтелекту є машинне навчання, яке стосується концепції, згідно з якою комп’ютерні програми можуть автоматично вчитися на нових даних і пристосовуватися до них без допомоги людей. Машинне навчання – найпопулярніша галузь штучного інтелекту. Інші класи ШІ включають імовірнісні моделі, системи штучних нейронних мереж, глибоке навчання та ін. Методи глибокого навчання дозволяють автоматично навчатися шляхом поглинання величезної кількості неструктурованих даних [3].

Глибоке навчання (ГН) є формою машинного навчання, що моделює моделі в даних як складні, багатошарові мережі. Оскільки глибоке навчання є найбільш загальним способом моделювання проблеми, воно має потенціал для вирішення складних завдань, таких як комп’ютерне бачення та обробка природної мови, які перевершують як звичайне програмування, так і інші методи машинного навчання [4].

Глибоке навчання не тільки може принести корисні результати там, де інші

методи виходять з ладу, але також можуть створювати більш точні моделі, ніж інші методи, і може скоротити час, необхідний для створення корисної моделі. Однак навчання моделей глибокого навчання вимагає великої обчислювальної потужності. Іншим недоліком глибокого навчання є складність інтерпретації глибоких моделей навчання.

Визначальною ознакою глибокого навчання є те, що навчальна модель має більше одного прихованого шару між входом і виходом. У більшості випадків глибоке навчання означає використання глибоких нейронних мереж. Є, однак, кілька алгоритмів, які впроваджують глибоке навчання, використовуючи інші види прихованих шарів, крім нейронних мереж.

Комп'ютерний зір – це міждисциплінарна наукова область, яка займається питанням того, як комп'ютери можуть отримати розуміння на високому рівні за допомогою цифрових зображень чи відео. З точки зору інженерії, він прагне зрозуміти та автоматизувати завдання, які може виконувати зорова система людини [5].

Завдання комп'ютерного зору включають методи отримання, обробки, аналізу та розуміння цифрових зображень, а також вилучення даних із великих розмірів реального світу з метою отримання числової або символічної інформації, наприклад у формах рішень. Розуміння в цьому контексті означає перетворення зорових зображень в описі світу, які мають сенс для процесів мислення і можуть викликати відповідні дії. Це розуміння можна розглядати як розплутування символічної інформації із даних зображень за допомогою моделей, побудованих за допомогою геометрії, фізики, статистики та теорії навчання.

Наукова дисципліна комп'ютерного зору стосується теорії штучних систем, що витягують інформацію із зображень. Дані зображення можуть приймати різні форми, такі як відео послідовності, види з декількох камер, багатовимірні дані із 3D-сканера.

Згорткові нейронні мережі (ЗНМ) – це досить широкий клас архітектур, основна ідея яких полягає в тому, щоб повторно використати одні й ті ж частини

нейронної мережі для роботи з різними маленькими, локальними ділянками входів. Як і багато інших нейронні архітектури, ЗНМ відомі досить давно, і в наші дні у них вже знайшлося багато найрізноманітніших застосувань, але основним додатком, заради якого люди колись придумали згорткові мережі, залишається обробка зображень.

1.2 Мета та задачі кваліфікаційної роботи

Метою кваліфікаційної роботи є створення інтелектуальної системи, яка у реальному часі отримує зображення з камер відеоспостереження та переводить їх у зрозумілі для комп'ютера данні. А потім за допомогою штучного інтелекту проводиться аналіз на основі якого приймається рішення для регулювання світлофора, щоб збільшити пропускну здатність трафіку. На сьогоднішній день майже у всіх випадках ми маємо звичайні світлофори, які працюють у певному статичному режимі. Системі потрібно слідкувати за усім, що відбувається на дорозі, чи стоять люди на світлофорі, чи переходять дорогу, де більша кількість машин утворює затор. Аналізуючи усі дані слід визначити, який саме колір потрібно запалити на тому чи іншому світлофорі.

Затори на дорогах дають прямий і непрямий вплив на економіку країни і здоров'я її жителів. Якщо взяти світ в цілому, то можна сказати, що через довге очікування на перехресті, витрачається багато палива у порожню. Затори на дорогах впливають і на індивідуальному рівні. Втрата часу, особливо в години пік, психічний стрес і додаткове забруднення навколишнього середовища в результаті глобального потепління також є деякими важливими факторами, викликаними заторами на дорогах.

Забезпечення економічного зростання і комфорту учасників дорожнього руху – це дві вимоги до розвитку країни, яке неможливе без безперебійного транспортного потоку. З розвитком транспортного сектора за рахунок збору інформації про дорожній рух, влада приділяє більше уваги моніторингу заторів на дорогах. Прогнозування заторів на дорогах надає владі необхідний час для

планування розподілу ресурсів, щоб зробити подорож більш комфортним для мандрівників.

Прогнозування заторів має два основних етапи збору даних та розробку моделі прогнозування. Кожен крок методології важливий і може вплинути на результати, якщо його не виконати правильно. Після збору даних, обробка даних відіграє життєво важливу функцію для підготовки наборів для навчання та тестування.

Система передбачає наявність камер відеоспостереження на світлофорі, щонайменше по одній у кожному напрямку дороги, приклад системи на рисунку 1.1. Для швидкого реагування системи потрібні значні обчислювальні потужності сервера або командного центру. Це зумовлено тим, що камери в реальному часі будуть передавати великі об’єми інформації, тому системі прийдеться за дуже короткий час проаналізувати зображення та на основі цього робити певні висновки.

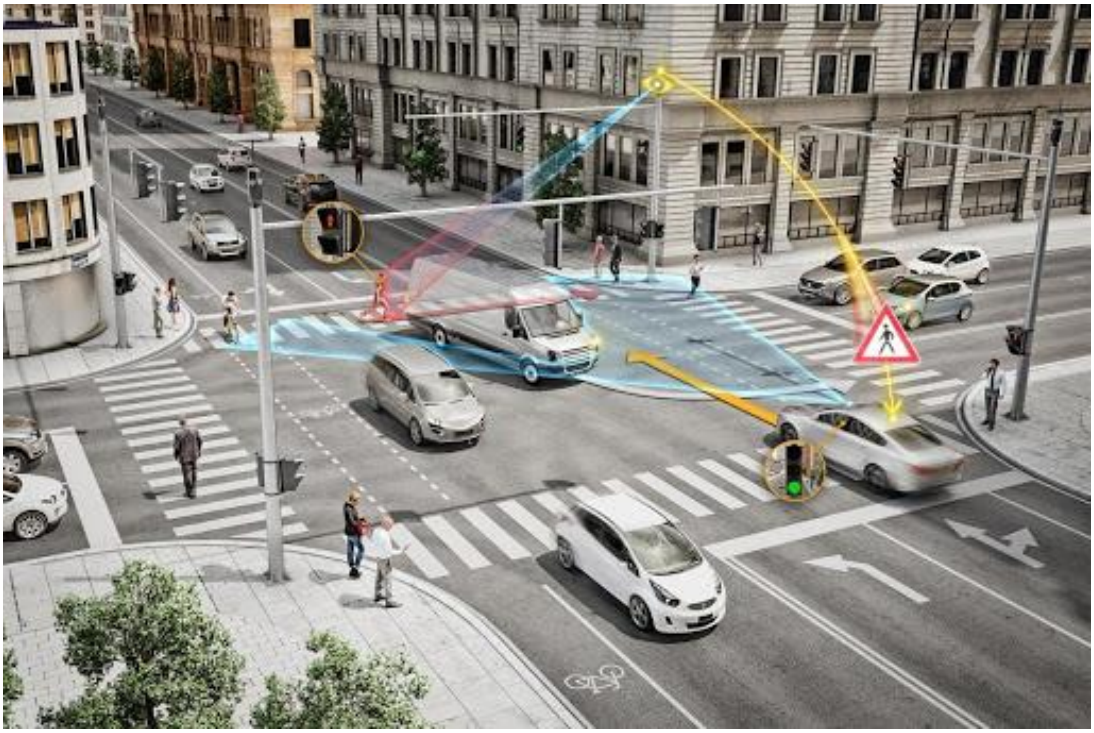

Рисунок 1.1 – Приклад системи для оптимізації дорожнього трафіку

2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Середовище програмування

PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django. PyCharm розроблена міжнародною компанією JetBrains на основі IntelliJ IDEA. PyCharm працює під операційними системами Windows, Mac OS X і Linux [6].

PyCharm забезпечує інтелектуальне завершення коду, інспекції коду, виділення помилок на льоту і швидкі виправлення, а також автоматичні рефакторингом коду і широкі можливості навігації.

Величезний набір готових інструментів PyCharm включає інтегрований відладчик і засіб запуску тестів, профайлер Python; вбудований термінал, інтеграція з основними VCS і вбудованими інструментами баз даних, можливість віддаленої розробки за допомогою віддалених перекладачів, вбудований ssh-термінал і інтеграція з Docker і Vagrant.

PyCharm інтегрується з IPython Notebook, має інтерактивну консоль Python і підтримує Anaconda, а також кілька наукових пакетів, включаючи Matplotlib і NumPy.

PyCharm Professional Edition має кілька варіантів ліцензій, які відрізняються функціональністю, вартістю та умовами використання. PyCharm Professional Edition безкоштовна для освітніх установ і проектів з відкритим кодом. Існує також вільна версія Community Edition з усіченим набором можливостей, яка поширюється під ліцензією Apache 2.

PyCharm надає широкі можливості реорганізації коду за допомогою рефакторингів Rename та Delete, Extract Method, Introduce Variable, Inline Variable та багатьох інших. Рефакторинги враховують особливості конкретної мови або фреймворку, допомагаючи вносити зміни по всьому проекту.

2.2 Мова програмування

Python – інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв’язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об’єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [2].

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносність програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв’язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;
- зручний для розв’язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);
- відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ [7].

Інтерпретатор мови Python і багата стандартна бібліотека (як вихідні тексти, так і бінарні дистрибутиви для всіх основних операційних систем) можуть бути отримані з сайту Python, і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C＋＋ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

2.2.1 Бібліотека NumPy

NumPy – це фундаментальний пакет для наукових обчислень на Python. Це бібліотека Python, яка надає об’єкт багатовимірного масиву, різні похідні об’єкти (такі як масиви і матриці в масках) і набір процедур для швидких операцій з масивами, включаючи математичні, логічні, маніпуляції з формами, сортування, вибір, введення/виведення, дискретні перетворення Фур’є, базову лінійну алгебру, базові статистичні операції, випадкове моделювання та багато іншого.

В основі пакета Numpy лежить об’єкт ndarray. Він інкапсулює N-мірні масиви однорідних типів даних, при цьому багато операцій виконуються в скомпільованому коді для підвищення продуктивності. Існує кілька важливих відмінностей між масивами NumPy і стандартними списками Python:

Масиви NumPy мають фіксований розмір при створенні, на відміну від списків Python (які можуть динамічно зростати). Зміна розміру ndarray призведе до створення нового масиву і видалення вихідного.

Всі елементи в масиві NumPy повинні мати один і той же тип даних і, отже, будуть мати однаковий розмір в пам’яті. Виняток: можна мати масиви об’єктів (Python, включаючи NumPy), що дозволяє створювати масиви елементів різного розміру. Масиви NumPy полегшують складні математичні та інші типи операцій з великою кількістю даних. Як правило, такі операції виконуються більш

ефективно і з меншою кількістю коду, ніж це можливо при використанні вбудованих списків.

Зростаюча кількість наукових і математичних пакетів на основі Python використовують масиви Numpy; хоча вони зазвичай підтримують введення списку Python, вони перетворюють такі введення в масиви Numpy перед обробкою і часто виводять масиви Numpy. Іншими словами, для ефективного використання багатьох (можливо, навіть більшості) сучасних наукових/математичних програм на основі Python недостатньо просто знати, як використовувати вбудовані типи списків – також необхідно знати, як використовувати масиви NumPy.

Питання, що стосуються розміру послідовності і швидкості, особливо важливі в наукових обчисленнях. Як простий приклад розглянемо випадок множення кожного елемента в одномірного списку на відповідний елемент в іншому списку тієї ж довжини. Якщо дані зберігаються в двох списках Python, a і b.

Приклад по елементного множення на мові Python:

```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```

Це дає правильний результат, але якщо a і b містять мільйони чисел, буде заплачено велику ціну за неефективність циклу в Python. Можна було б виконати ту ж задачу набагато швидше на мові C, написавши (для ясності нехтуючи оголошеннями змінних і ініціалізаціями, виділенням пам’яті і т. д.)

Приклад по елементного множення на мові C:

```
for (i = 0; i < rows; i++): {
  c[i] = a[i]*b[i];
}
```

Це заощаджує всі накладні витрати, пов’язані з інтерпретацією коду Python і маніпулюванням об’єктами Python, але за рахунок переваг, отриманих від кодування на Python. Крім того, необхідна робота з кодування збільшується зі збільшенням розмірності даних.

У випадку двомірного масиву приклад коду на C (скорочений, як і раніше) розширюється до:

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j]; }}
```

Numpy дає найкраще з обох світів: операції по елементах є «режимом за замовчуванням», коли задіяний ndarray, але операція по елементах швидко виконується по передньо скомпільованим кодом на мові C.

Приклад по елементного множення за допомогою NumPy:

```
c = a * b
```

Він робить те ж саме, що і в попередніх прикладах, на швидкості, близькій до C, але з простотою коду, яку ми очікуємо від чогось, заснованого на Python. Дійсно, ідіома NumPy ще простіше. Цей останній приклад ілюструє дві функції Numpy, які є основою більшої частини його можливостей: векторизація та трансляція.

Векторизація описує відсутність будь-яких явних циклів, індексування і т.д. В коді – ці речі, звичайно, відбуваються просто «за лаштунками» в оптимізованому, попередньо скомпільованому коді на мові C. Векторизований код має безліч переваг, серед яких:

- векторизований код більш лаконічний і зручний для читання;
- менша кількість рядків коду зазвичай означає менше помилок;
- код більше схожий на стандартну математичну нотацію (що, як правило,

полегшує правильне кодування математичних конструкцій);

- векторизація призводить до більш «пітонічного» коду, без векторизації код був би завалений неефективними і важкими для читання циклами.

Трансляція – це термін, який використовується для опису неявної по елементної поведінки операцій; взагалі кажучи, в NumPy всі операції, не тільки арифметичні, але і логічні, бітові, функціональні і т.д., поводяться таким неявним по елементним чином, тобто вони транслюються. Більш того, в наведеному вище прикладі a і b можуть бути багатовимірними масивами однакової форми, або скаляром і масивом, або навіть двома масивами різної форми, за умови, що менший масив «розширюється» до форми більшого таким чином, щоб результуюча трансляція була однозначною [8].

Numpy повністю підтримує об’єктно-орієнтований підхід, починаючи, знову ж таки, з ndarray. Наприклад, ndarray – це клас, що володіє численними методами і атрибутами. Багато його методів відображають функції в самому зовнішньому просторі імен Numpy, дозволяючи програмісту програмувати в будь-якій парадигмі, якій вони віддають перевагу. Ця гнучкість дозволила діалекту масиву Numpy і класу Numpy ndarray стати фактичною мовою багатовимірного обміну даними, використовуваним в Python.

2.2.2 Бібліотека TensorFlow

TensorFlow – це платформа з відкритим кодом, розроблена дослідниками Google для запуску машинного навчання, глибокого навчання та інших статистичних та прогнозних аналітичних завдань. Як і аналогічні платформи, він призначений для оптимізації процесу розробки та виконання передових аналітичних додатків для користувачів, таких як фахівці з обробки даних, статистики та розробники прогнозних моделей.

Програмне забезпечення TensorFlow обробляє набори даних, які розташовані у вигляді обчислювальних вузлів у вигляді графа. Ребра, що з'єднують вузли в графі, можуть представляти багатовимірні вектори або

матриці, створюючи так звані тензори. Оскільки програми TensorFlow використовують архітектуру потоку даних, яка працює з узагальненими проміжними результатами обчислень, вони особливо відкриті для дуже великомасштабних додатків паралельної обробки, загальним прикладом яких є нейронні мережі.

Фреймворк включає в себе набори як високорівневі, так і низькорівневі API. Google рекомендує використовувати високорівневі, коли це можливо, для спрощення розробки конвеєрів даних і програмування додатків. Проте, знаючи, як використовувати низькорівневі API-інтерфейси, що називаються ядром TensorFlow, може бути корисним для експериментів та налагодження додатків, говорить компанія; це також дає користувачам «розумну модель» внутрішньої роботи технології машинного навчання [9].

Додатки TensorFlow можуть працювати як на звичайних процесорах, так і на високопродуктивних графічних процесорів (GPU), а також на власних тензорних процесорах Google (TPU), які є одними пристроями, спеціально розробленими для прискорення завдань TensorFlow. Перші TPU Google, опубліковані публічно в 2016 році, були використані всередині компанії в поєднанні з TensorFlow для живлення деяких додатків і онлайн-сервісів компанії, включаючи алгоритм пошуку RankBrain і технологію відображення вулиць.

2.2.3 Бібліотека OpenCV

OpenCV – це бібліотека програмного забезпечення для комп’ютерного зору та машинного навчання з відкритим вихідним кодом. OpenCV був створений для забезпечення загальної інфраструктури для додатків комп’ютерного зору і прискорення використання машинного сприйняття в комерційних продуктах. Будучи продуктом з ліцензією BSD, OpenCV спрощує для підприємств використання та зміну коду.

Бібліотека налічує понад 2500 оптимізованих алгоритмів, які включають в себе повний набір як класичних, так і сучасних алгоритмів комп’ютерного зору

і машинного навчання. Ці алгоритми можуть використовуватися для виявлення та розпізнавання обличчя, ідентифікації об’єктів, класифікації дій людини у відео, відстеження рухів камери, відстеження рухомих об’єктів, вилучення 3D-моделей об’єктів, створення 3D-хмар точок зі стерео камер, зшивання зображень для отримання зображення з високою роздільною здатністю всієї сцени, пошуку схожих зображень з бази даних зображень, видалення червоних очей з зображень, зроблених за допомогою спалаху, відстеження рухів очей, розпізнавання пейзажів і установки маркерів для накладення на нього доповненої реальності і т. д. Спільнота користувачів OpenCV налічує понад 47 тисяч осіб, а передбачувана кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними органами [10].

Поряд з добре зарекомендованими себе компаніями, такими як Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, які використовують бібліотеку, існує безліч стартапів, таких як Applied Minds, VideoSurf і Zeitera, які широко використовують OpenCV. Розгорнуті програми OpenCV охоплюють широкий діапазон: від зшивання зображень з видом на вулицю, виявлення вторгнень у відео спостереження в Ізраїлі, допомоги роботам в навігації і підборі предметів у гаражі Willow, виявлення нещасних випадків з утопленням у басейні в Європі, запуску інтерактивного мистецтва в Іспанії і Нью-Йорку, перевірки злітно-посадкових смуг на предмет сміття в Туреччині, перевірки етикеток на продуктах на фабриках по всьому світу до швидкого розпізнавання осіб в Японії.

Він має інтерфейси C++, Python, Java і MATLAB і підтримує Windows, Linux, Android і Mac OS. OpenCV в основному орієнтується на додатки для бачення в реальному часі і використовує інструкції MMX і SSE, коли вони доступні. В даний час активно розробляються повнофункціональні інтерфейси CUDA і OpenCL. Існує понад 500 алгоритмів і приблизно в 10 разів більше функцій, які складають або підтримують ці алгоритми. OpenCV спочатку написаний на C++ і має шаблонний інтерфейс, який легко працює з контейнерами STL.

2.3 Згорткова нейронна мережа

Згорткові нейронні мережі забезпечують часткову стійкість до змін масштабу, зсувів, поворотів, зміни ракурсу та інших спотворень. Згорткові нейронні мережі об’єднують три архітектурних ідеї, для забезпечення інваріантності до зміни масштабу, повороту зсуву і просторовим спотворенням:

- локальні рецепторні поля (забезпечують локальну двовимірну зв’язність нейронів);
- загальні синаптичні коефіцієнти (забезпечують детектування деяких рис в будь-якому місці зображення і зменшують загальне число вагових коефіцієнтів);
- ієрархічна організація з просторовими підвибірками.

ЗНМ складається з різних видів шарів: згорткові (convolutional) шари, субдискретизуючі (subsampling, підвибірка) шари і шари «звичайної» нейронної мережі – персептрона, відповідно до рисунка 2.1.

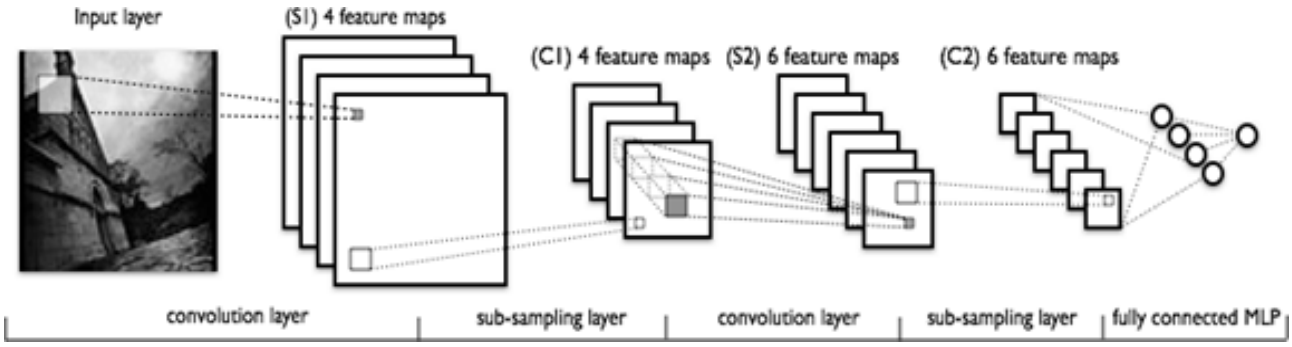

Рисунок 2.1 – Топологія згорткової нейронної мережі

Згорткові мережі є вдалою серединою між біологічно правдоподібними мережами і звичайним багатошаровим персептроном, приклад на рисунку 2.2. На сьогоднішній день кращі результати в розпізнаванні зображень отримують з їх допомогою. В середньому точність розпізнавання таких мереж перевершує звичайні ІШНМ на 10-15%. ЗНМ – це ключова технологія Deep Learning [11].

Основною причиною успіху ЗНМ стала концепція загальних ваг. Незважаючи на великий розмір, ці мережі мають невелику кількість параметрів, що настроюються в порівнянні з їх предком – неокогнітроном. Є варіанти ЗНМ, схожі на неокогнітрон, в таких мережах відбувається, часткова відмова від пов’язаних ваг, але алгоритм навчання залишається тим же і грунтується на зворотному поширені помилки.

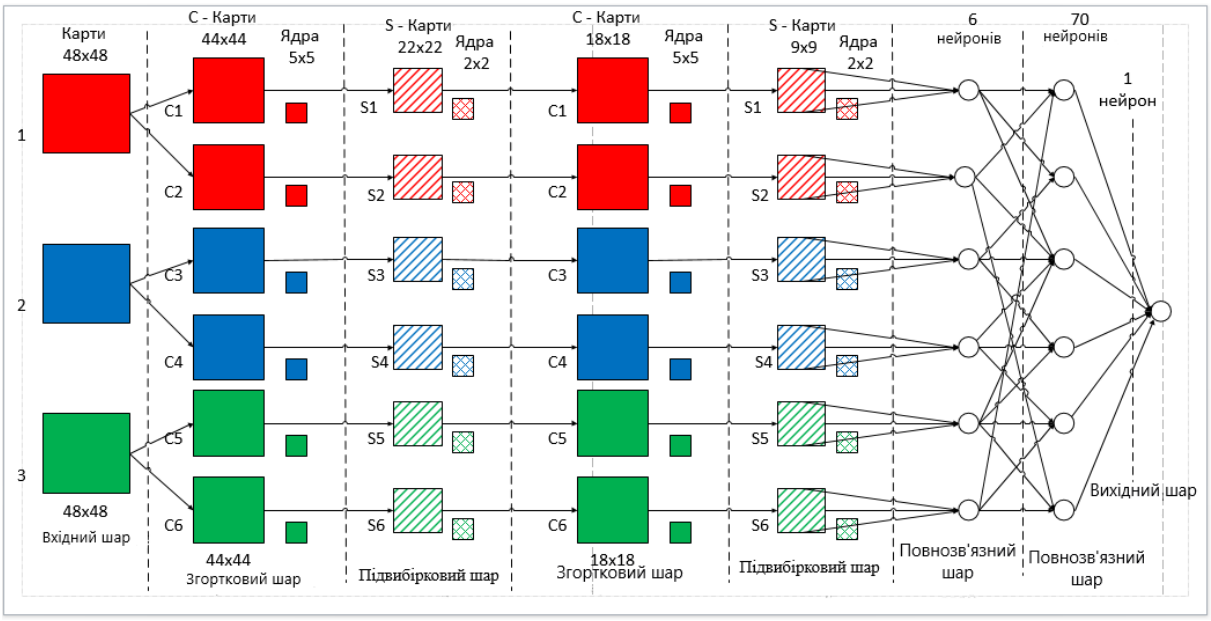

Рисунок 2.2 – Топологія згорткової нейронної мережі

ЗНМ можуть швидко працювати на послідовній машині і швидко навчатися за рахунок чистого розпаралелювання процесу згортки по кожній карті, а також зворотної згортки при поширенні помилки по мережі.

Можна виділити наступні етапи, що впливають на вибір топології:

- визначити розв’язувану задачу нейромережею (класифікація, прогнозування, модифікація);

- визначити обмеження в розв’язуваної задачі (швидкість, точність відповіді);

- визначити вхідні (тип: зображення, звук, розмір: 100x100, 30x30, формат: RGB, в градаціях сірого) і вихідні дані (кількість класів) [11].

2.3.1 Вхідний шар

Вхідні дані являють собою кольорові зображення. Якщо розмір буде занадто великий, то обчислювальна складність підвищиться, відповідно обмеження на швидкість відповіді будуть порушені, визначення розміру в даній задачі вирішується методом підбору. Якщо вибрати розмір занадто маленький, то мережа не зможе виявити ключові ознаки. Кожне зображення розбивається на 3 канали: червоний, синій, зелений. Таким чином виходить 3 зображення розміру 48x48 пікселів.

Вхідний шар враховує двовимірну топологію зображень і складається з декількох карт (матриць), карта може бути одна, в тому випадку, якщо зображення представлено у відтінках сірого, інакше їх 3, де кожна карта відповідає зображенню з конкретним каналом (червоним, синім і зеленим).

Вхідні дані кожного конкретного значення пікселя нормалізуються в діапазон від 0 до 1, за формулою (2.1):

$$f(p, min, \max) = \frac{p - min}{\max - min}, \tag{2.1}$$

де f – функція нормалізації;

p – значення конкретного кольору пікселя від 0 до 255;

min – мінімальне значення пікселя – 0;

max – максимальне значення пікселя – 255

2.3.2 Згортковий шар

Згортковий шар представляє з себе набір карт (інша назва – карти ознак, в побуті це звичайні матриці), у кожної карти є синаптичне ядро (в різних джерелах його називають по-різному: скануюче ядро або фільтр).

Кількість карт визначається вимогами до задачі, якщо взяти велику

кількість карт, то підвищиться якість розпізнавання, але збільшиться обчислювальна складність. Виходячи з аналізу наукових статей, в більшості випадків пропонується брати співвідношення один до двох, тобто кожна карта попереднього шару (наприклад, у першого згорткового шару, попереднім є вхідний) пов'язана з двома картами згорткового шару, відповідно до топології на рисунку 2.3, кількість карт – 6.

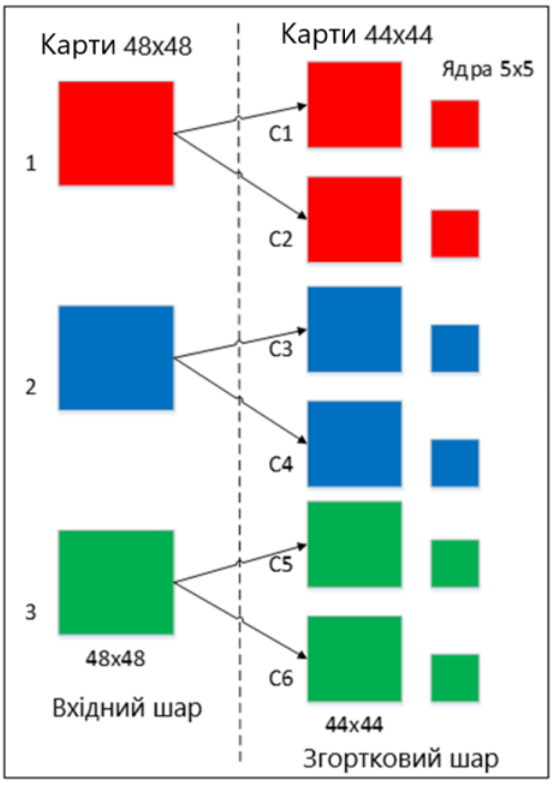

Рисунок 2.3 – Організація зв'язків між картами згорткового шару і попереднього

Розміри карт згорткового шару – однакові і обчислюються за формулою (2.2):

$$(w, h) = (mW - kW + 1, mH - kH + 1), \quad (2.2)$$

де (w, h) – обчислювальний розмір згорткової карти;

mW – ширина попереньої карти;

mH – висота попередньої карти;

kW – ширина ядра; *kH* – висота ядра.

Ядро являє собою фільтр або вікно, яке ковзає по всій області попередньої карти і знаходить певні ознаки об’єктів. Наприклад, якщо мережу навчали на безлічі осіб, то одне з ядер могло б в процесі навчання видавати найбільший сигнал в області ока, рота, брови або носа, інше ядро могло б виявляти інші ознаки. Розмір ядра зазвичай беруть в межах від 3x3 до 7x7, як на рисунку 2.4. Якщо розмір ядра маленький, то воно не зможе виділити будь-які ознаки, якщо занадто велике, то збільшується кількість зв’язків між нейронами. Також розмір ядра вибирається таким, щоб розмір карт згорткового шару був парним, це дозволяє не втрачати інформацію при зменшенні розмірності в підвибірковому шарі [12].

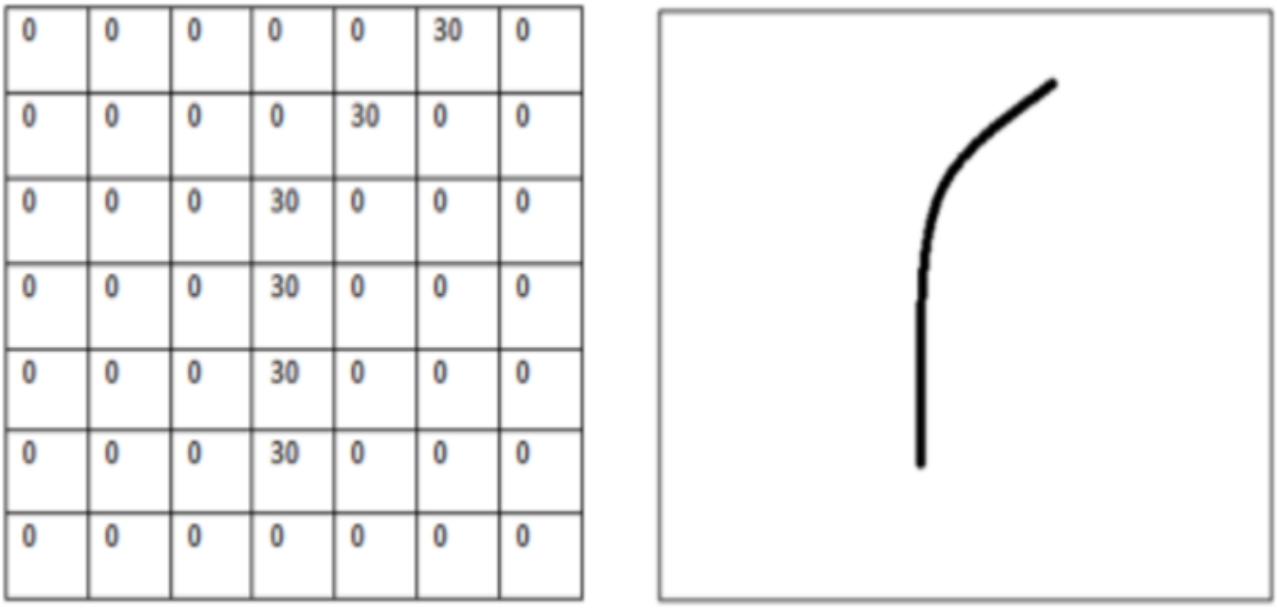

Рисунок 2.4 – Приклад ядра з навченою ознакою

Ядро являє собою систему поділюваних ваг або синапсів, це одна з головних особливостей згорткової нейромережі. У звичайній багатошаровій мережі дуже багато зв’язків між нейронами, тобто синапсів, що вельми уповільнює процес детектування. У згорткової мережі – навпаки, загальні ваги

дозволяють скоротити число зв’язків і дозволити знаходити одну і ту ж ознаку по всій області зображення.

В цьому місці, де жовте вікно, буде великий відгук (сигнал), що говорить про наявність цієї ознаки на зображенні як на рисунку 2.5.

Спочатку значення кожної карти згорткового шару рівні 0. Значення ваг ядер задаються випадковим чином в області від -0.5 до 0.5. Ядро ковзає по попередній карті і виробляє операцію згортка, яка часто використовується для обробки зображень, формула (2.3):

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l]\,, \qquad (2.3)$$

де f – початкова матриця зображення;

g – ядро згортки.

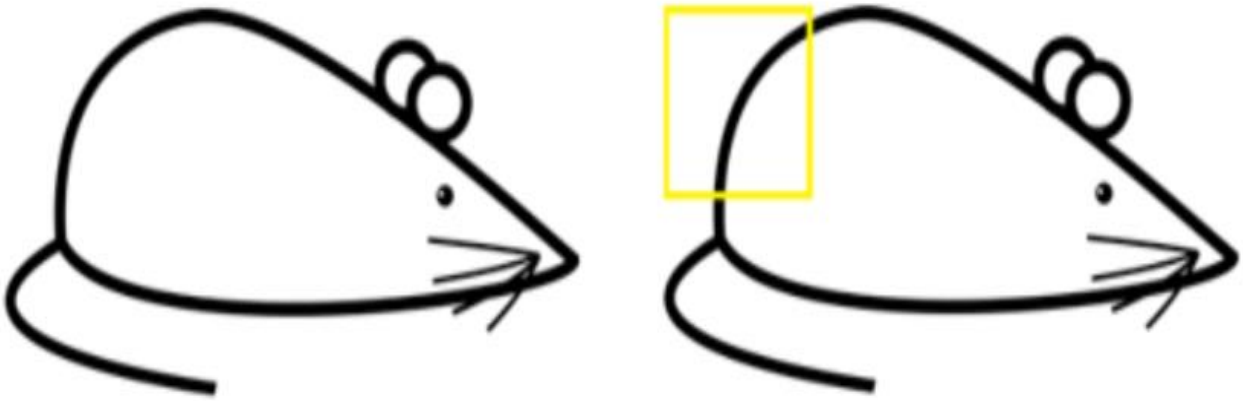

Рисунок 2.5 – Вхідне зображення

Неформально цю операцію можна описати наступним чином – вікном розміру ядра g проходимо із заданим кроком (зазвичай 1) усе зображення f, на кожному кроці поелементно множимо вміст вікна на ядро g, результат підсумовується і записується в матрицю результату, як на рисунку 2.6.

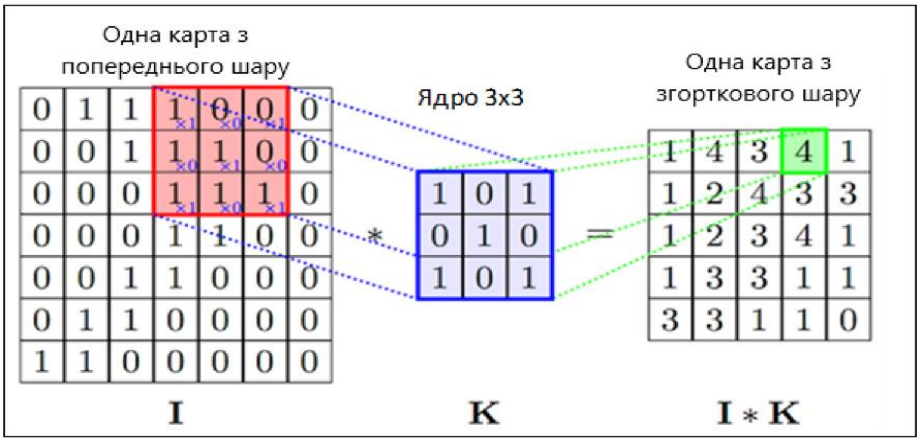

Рисунок 2.6 – Операція згортки і отримання значень згорткової карти (valid)

При цьому в залежності від методу обробки країв вихідної матриці результат може бути менше вихідного зображення (valid), такого ж розміру (same) або більшого розміру (full), відповідно до рисунка 2.7.

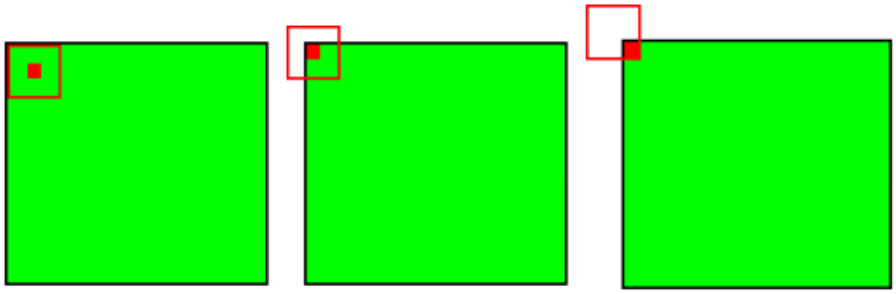

Рисунок 2.7 – Три види згортки вихідної матриці

У спрощеному вигляді цей шар можна описати формулою (2.4):

$$x^l = f(x^{l-1} * k^l + b^l), \tag{2.4}$$

де x^l – вихід шару l;

f() – функція активації;

b^l – коефіцієнт зсуву шару *l*;

* – операція згортки входу *x* з ядром *k*.

При цьому за рахунок крайових ефектів розмір вихідних матриць зменшується, формула (2.5):

$$x_j^l = f\left(\sum_i x_i^{l-1} * k_j^l + b_j^l\right) \quad , \tag{2.5}$$

де x_j^l – карта ознак *j* (вихід шару *l*);

f() – функція активації;

b^l – коефіцієнт зсуву шару *l* для карти ознак j;

k_j^l – ядро згортки *j* карти, шару *l*;

* – операція згортки входу *x* з ядром *k*.

2.3.3 Підвибірковий шар

Підвибірковий шар також, як і згортковий має карти, але їх кількість збігається 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 [12].

У процесі сканування ядром підвибіркового 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 розділяється на осередки 2x2 елемента, з яких вибираються максимальні за значенням.

Зазвичай в підвибірковому шарі застосовується функція активації ReLU. Операція підвибірки (або MaxPooling – вибір максимального) відповідно до рисунку 2.8.

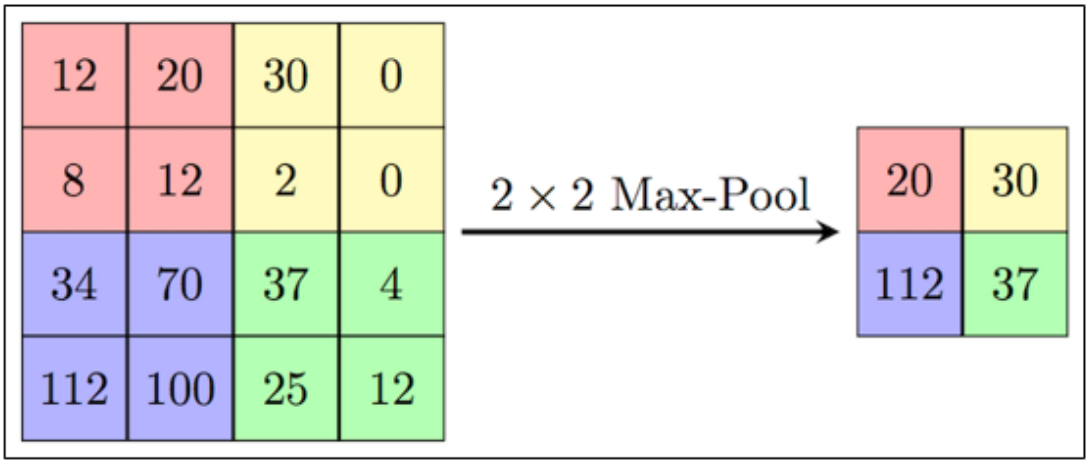

Рисунок 2.8 – Формування нової карти підвибіркового шару на основі попередньої карти згорткового шару

Формально шар може бути описаний формулою (2.6):

$$x^l = f(a^l * subsample(x^{l-1}) + b^l), \quad (2.6)$$

де x^l – вхідний шар l;

$f()$ – функція активації;

a^l, b^l – коефіцієнт зсуву шару l;

subsample() – операція вибірки локальних максимальних значень.

2.3.4 Повнозв’язний шар

Останній з типів шарів – це шар звичайного багатошарового персептрона як на рисунку 2.9. Мета шару – класифікація, моделює складну нелінійну функцію, оптимізуючи яку, поліпшується якість розпізнавання.

Нейрони кожної карти попереднього підвибіркового шару пов’язані з одним нейроном прихованого шару. Таким чином число нейронів прихованого шару дорівнює числу карт підвибіркового шару, але зв’язки можуть бути не обов’язково такими.

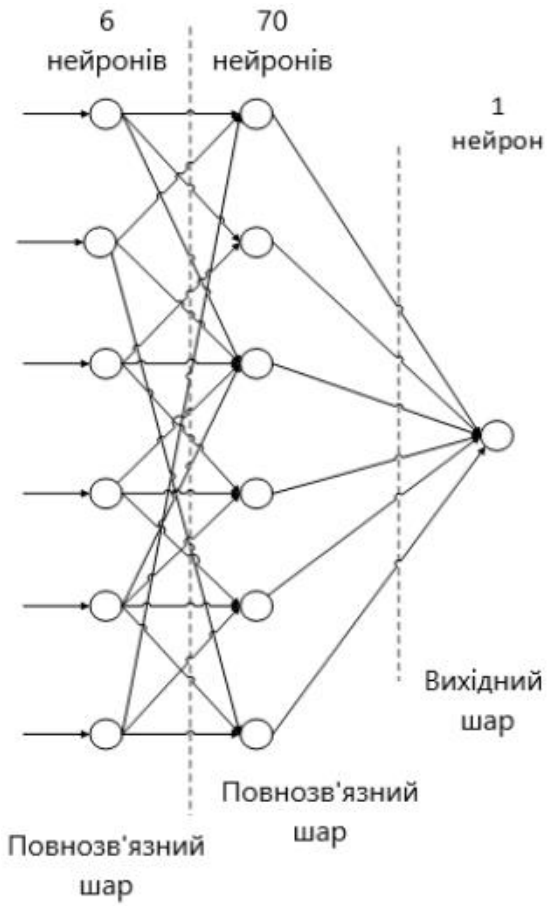

Рисунок 2.9 – Зв'язки між картами повнозв’язного шару та вихідного шару

Наприклад, тільки частина нейронів будь-якої з карт підвибіркового шару може бути пов’язана з першим нейроном прихованого шару, а частина, що залишилася з другим, або всі нейрони першої карти пов’язані з нейронами 1 і 2 прихованого шару. Обчислення значень нейрона можна описати формулою (2.7):

$$x_j^l = f\left(\sum_i x_i^{l-1} * w_{i,j}^{l-1} + b_j^{l-1}\right) \quad , \tag{2.7}$$

де x_j^l – карта ознак j (вихід шару l);

$f()$ – функція активації;

b^l – коефіцієнт зсуву шару;

$w_{i,j}^l$ – матриця вагових коефіцієнтів шару l.

2.3.5 Вихідний шар

Вихідний шар пов’язаний з усіма нейронами попереднього шару. Кількість нейронів відповідає кількості розпізнаваних класів, тобто 2 – автомобіль і не автомобіль. Але для зменшення кількості зв’язків і обчислень для бінарного випадку можна використовувати один нейрон і при використанні в якості функції активації гіперболічний тангенс, вихід нейрона зі значенням -1 означає приналежність до класу не автомобілі, навпаки вихід нейрона зі значенням 1 – означає приналежність до класу автомобілі.

Одним з етапів розробки нейронної мережі є вибір функції активації нейронів. Вид функції активації багато в чому визначає функціональні можливості нейронної мережі і метод навчання цієї мережі. Класичний алгоритм зворотного поширення помилки добре працює на двошарових і тришарових нейронних мережах, але при подальшому збільшенні глибини починає відчувати проблеми. Одна з причин – так зване загасання градієнтів. У міру поширення помилки від вихідного шару до вхідного на кожному шарі відбувається множення поточного результату на похідну функції активації. Похідна у традиційної сигмоїдної функції активації менше одиниці на всій області визначення, тому після декількох шарів помилка стане близькою до нуля. Якщо ж, навпаки, функція активації має необмежену похідну (як, наприклад, гіперболічний тангенс), то може статися вибухове збільшення помилки в міру поширення, що призведе до нестійкості процедури навчання [13].

2.4 Рекурентна нейронна мережа

Рекурентні нейронні мережі (RNN) – це сучасний алгоритм послідовної обробки даних, який використовується Siri від Apple і голосовим пошуком Google. Це перший алгоритм, який запам’ятовує свої вхідні дані завдяки внутрішній пам’яті, що робить його ідеально відповідним для завдань машинного навчання, пов’язаних з послідовними даними. Це один з алгоритмів,

що стоять за сценами дивовижних досягнень в області глибокого навчання за останні кілька років.

RNN – це потужний і надійний тип нейронної мережі, і він відноситься до найбільш перспективних використовуваних алгоритмів, тому що він єдиний, у якого є внутрішня пам’ять [14].

Як і багато інших алгоритмів глибокого навчання, рекурентні нейронні мережі відносно старі. Спочатку вони були створені в 1980-х роках, але тільки в останні роки можна побачити їх справжній потенціал. Збільшення обчислювальної потужності поряд з величезними обсягами даних, з якими тепер доводиться працювати, і винаходом довготривалої короткочасної пам’яті (LSTM) в 1990-х роках дійсно вивело RNN на передній план.

Завдяки своїй внутрішній пам’яті RNN можуть запам’ятовувати важливі речі про отримані ними вхідних даних, що дозволяє їм бути дуже точними в прогнозуванні того, що буде далі. Ось чому вони є кращим алгоритмом для послідовних даних, таких як часові ряди, мова, текст, фінансові дані, аудіо, відео, погода і багато іншого. Рекурентні нейронні мережі можуть сформувати набагато більш глибоке розуміння послідовності та її контексту в порівнянні з іншими алгоритмами.

Для правильного розуміння RNN, знадобляться робочі знання «звичайних» нейронних мереж з прямим зв’язком і послідовних даних.

Послідовні дані – це в основному просто упорядковані дані, в яких пов’язані речі слідують один за одним. Прикладами можуть служити фінансові дані або послідовність ДНК. Найбільш популярним типом послідовних даних, можливо, є дані часових рядів, які представляють собою просто серію точок даних, перерахованих в тимчасовому порядку.

Нейронні мережі RNN і мережі з прямим зв’язком отримують свої назви за способом передачі інформації.

У нейронній мережі з прямим зв’язком інформація рухається тільки в одному напрямку, від вхідного шару, через приховані шари, до вихідного шару. Інформація рухається прямо по мережі і ніколи не торкається вузла двічі.

Нейронні мережі з прямим зв’язком не запам’ятовують інформацію, яку вони отримують, і погано передбачають, що буде далі. Оскільки мережа прямого зв’язку враховує тільки поточне введення, вона не має поняття про порядок в часі. Він просто не може згадати нічого про те, що відбувалося в минулому, крім його навчання.

У RNN інформація циклічно проходить через цикл. Коли вона приймає рішення, вона враховує поточні вхідні дані, а також те, що вона дізналася з вхідних даних, отриманих раніше [14].

Рисунок 2.10 ілюструє різницю в потоці інформації між RNN і нейронною мережею з прямим зв’язком.

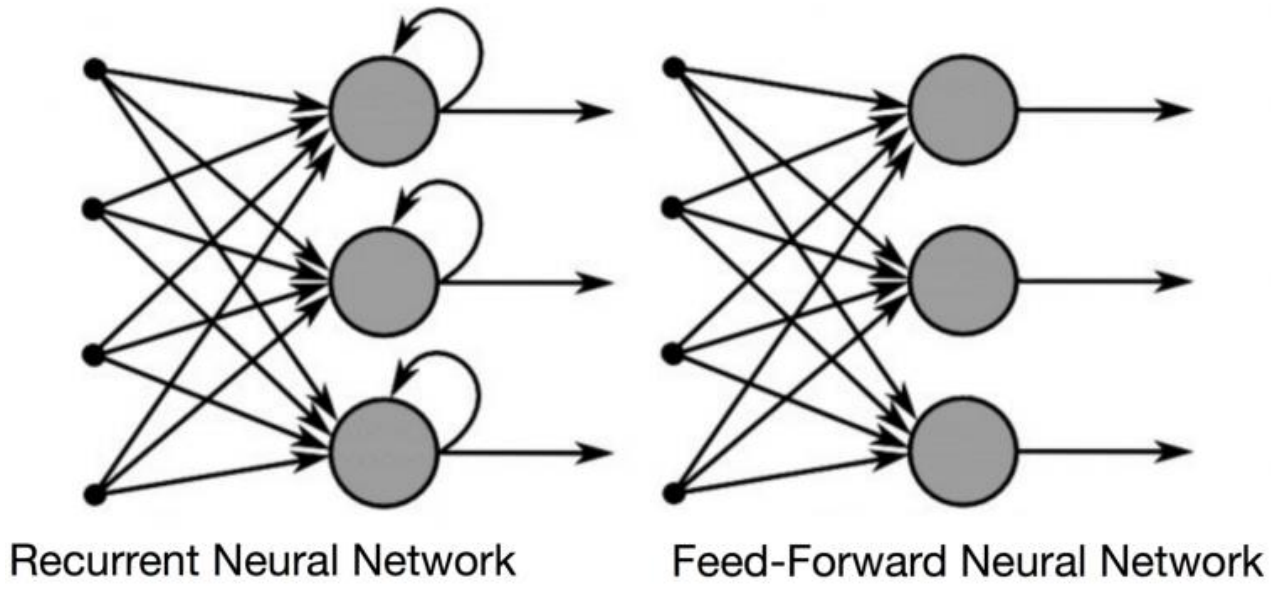

Рисунок 2.10 – Різниця рекурентної мережі від мережі з прямим зв’язком

Звичайна RNN має короткочасну пам’ять. У поєднанні з LSTM вона також має довготривалу пам’ять.

Ще один хороший спосіб проілюструвати концепцію пам’яті рекурентної нейронної мережі – пояснити її на прикладі: уявимо, що у нас є звичайна нейронна мережа з прямим зв’язком, і дамо їй слово «neuron» в якості вхідних даних, і вона обробляє слово символ за символом. До того часу, коли вона

досягає символу «r», мережа вже забуває про «n», «e» і «u», що робить практично неможливим для цього типу нейронної мережі передбачити, який символ буде наступним.

Однак рекурентна нейронна мережа здатна запам’ятовувати ці символи завдяки своїй внутрішній пам’яті. Вона робить висновок, копіює цей висновок і зациклює його назад в мережу. Простіше кажучи – рекурентні нейронні мережі додають безпосереднє минуле до поточного.

Таким чином, у RNN є два входи: поточне і недавнє минуле. Це важливо, тому що послідовність даних містить важливу інформацію про те, що буде далі, ось чому RNN може робити те, що інші алгоритми не можуть.

Нейронна мережа з прямим зв’язком привласнює, як і всі інші алгоритми глибокого навчання, матрицю ваг своїм вхідним даним, а потім видає вихідні дані. Зверніть увагу, що RNN застосовують ваги до поточного, а також до попереднього вхідного сигналу. Крім того, рекурентна нейронна мережа також змінить ваги як для градієнтного спуску, так і для зворотного поширення в часі.

Зворотне поширення (скорочено BP або backprop) – відоме як алгоритм робочої конячки в машинному навчанні. Зворотне поширення використовується для обчислення градієнта функції помилки по відношенню до ваг нейронної мережі. Алгоритм проходить зворотний шлях через різні шари градієнтів, щоб знайти часткову похідну помилок по відношенню до ваг. Потім backprop використовує ці ваги для зменшення допустимих помилок при навчанні.

У нейронних мережах в основному виконується пряме поширення, щоб отримати результат моделі і перевірити, чи є цей результат правильним або неправильним, щоб отримати помилку. Зворотне поширення – це не що інше, як рух назад по нейронній мережі, щоб знайти часткові похідні помилки по відношенню до ваг, що дозволяє відняти це значення з ваг.

Ці похідні потім використовуються методом градієнтного спуску, алгоритмом, який може ітеративно мінімізувати задану функцію. Потім він регулює ваги вгору або вниз, залежно від того, що зменшує помилку. Саме так нейронна мережа навчається в процесі навчання. Отже, за допомогою зворотного

поширення в основному намагаються налаштувати вагу моделі під час тренування. Рисунок 2.11 ілюструє концепцію прямого поширення і зворотного поширення в нейронній мережі з прямим зв’язком [14].

BPTT – це в основному просто модне слово для зворотного поширення на розгорнутому RNN. Розгортання – це візуалізація та концептуальний інструмент, який допомагає зрозуміти, що відбувається в мережі. У більшості випадків при реалізації рекурентної нейронної мережі в загальних рамках програмування автоматично виконується зворотне поширення, але необхідно зрозуміти, як це працює, щоб усунути проблеми, які можуть виникнути в процесі розробки.

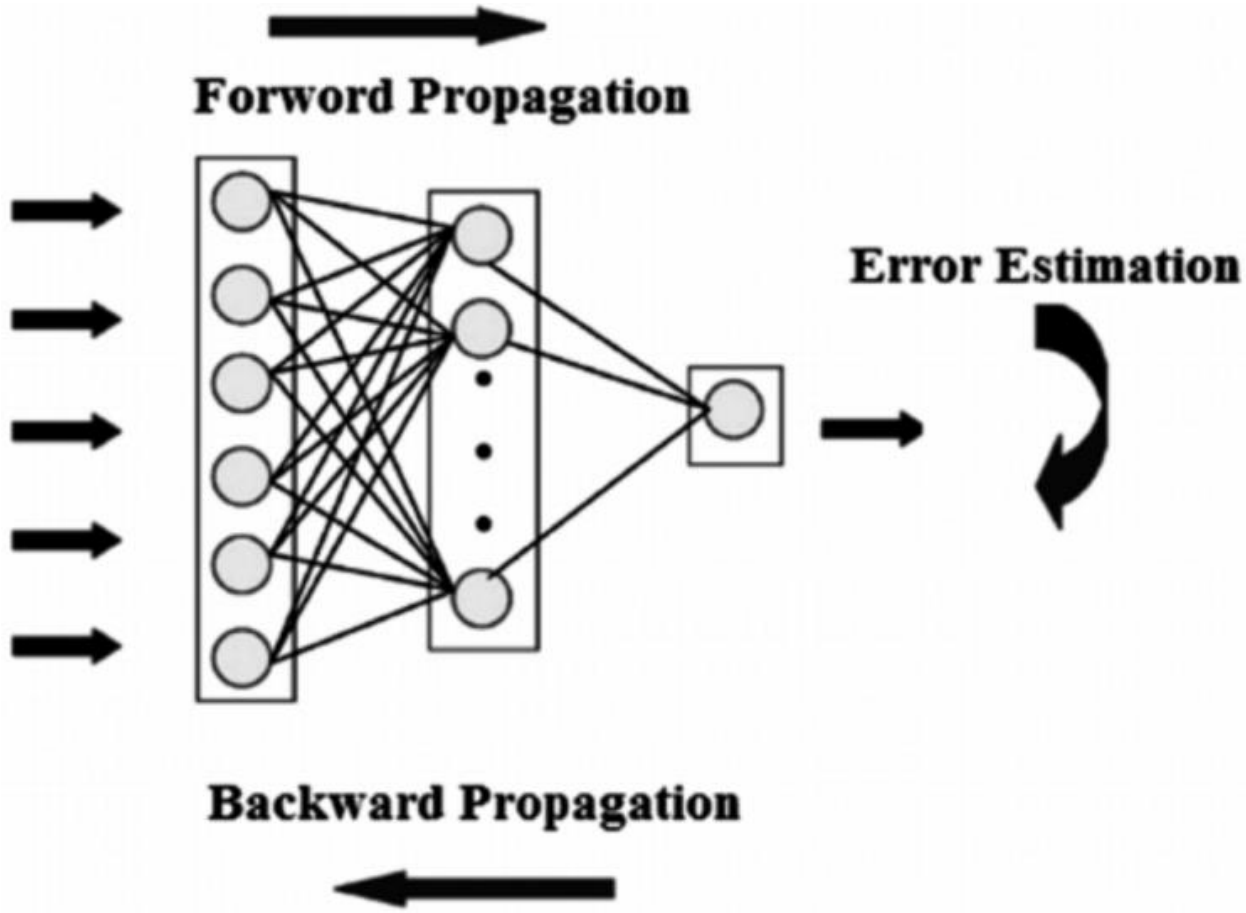

Рисунок 2.11 – Пряме та зворотне поширення в нейронній мережі з прямим зв’язком.

Можна розглядати рекурентну нейрону мережу як послідовність нейронних мереж, які тренуються одна за одною зі зворотним поширенням.

Розглянемо рисунок 2.12 на якому зліва RNN розгорнуто після знаку рівності. Треба звернути увагу, що після знаку рівності немає циклу, так як візуалізуються різні тимчасові кроки та інформація передається від одного тимчасового кроку до наступного. Цей рисунок також демонструє, чому RNN можна розглядати як послідовність нейронних мереж.

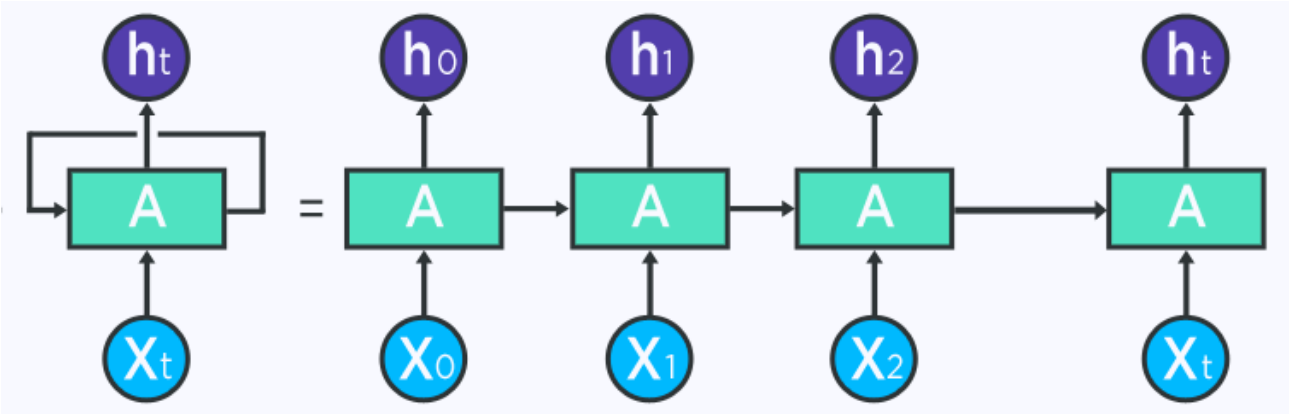

Рисунок 2.12 – Розгорнута версія RNN

Якщо робити BPTT потрібна концептуалізація розгортання, оскільки помилка даного тимчасового кроку залежить від попереднього тимчасового кроку. В обох випадках помилка передається назад від останнього до першого тимчасового кроку при розгортанні всіх тимчасових кроків. Це дозволяє обчислювати помилку для кожного кроку за часом, що дозволяє оновлювати ваги. Треба звернути увагу, що BPTT може бути дорогим з обчислювальної точки зору, якщо є велика кількість тимчасових кроків.

Є дві основні перешкоди, з якими довелося зіткнутися RNN, але щоб зрозуміти їх, вам спочатку потрібно знати, що таке градієнт. Градієнт є частковою похідною по відношенню до його вхідних даних. Простими словами градієнт вимірює, наскільки зміниться результат функції, якщо трохи змінити вхідні дані. Також градієнт можна увити як нахил функції. Чим вище ухил, тим крутіше нахил і тим швидше модель може навчатися. Але якщо нахил дорівнює нулю, модель припиняє навчання. Градієнт просто вимірює зміну всіх ваг з

урахуванням зміни помилки [14].

Розривні градієнти – це коли алгоритм без особливих причин надає високу важливість ваг. На щастя, цю проблему можна легко вирішити, обрізаючи або здавлюючи градієнти.

Зникаючі градієнти виникають, коли значення градієнта занадто малі, і в результаті модель перестає навчатися або займає занадто багато часу. Це була серйозна проблема в 1990-х роках, і це було набагато важче вирішити, ніж вибухові градієнти. На щастя, це було вирішено за допомогою концепції LSTM Зеппа Хохрайтера та Юргена Шмідхубера.

2.5 Мережа LSTM

LSTM допомагають зберегти помилку, яку можна поширити назад через час і шари. Підтримуючи більш постійну похибку, вони дозволяють рекурентним мережам продовжувати навчатися протягом багатьох часових кроків (понад 1000), тим самим відкриваючи канал для дистанційного зв’язування причин і наслідків. Це одна з головних проблем машинного навчання та штучного інтелекту, оскільки алгоритми часто стикаються з середовищами, де сигнали винагороди рідкісні та відкладені.

LSTM містять інформацію за межами звичайного потоку рекурентної мережі в закритій комірці. Інформацію можна зберігати в комірці, записувати або зчитувати з неї, як дані в пам’яті комп’ютера. Комірка приймає рішення про те, що зберігати і коли дозволити читання, запис і стирання, через вентилі, які відкриваються та закриваються. На відміну від цифрового сховища на комп’ютерах, однак, ці вентилі є аналоговими, реалізованими з по елементним множенням на сигмоїди, які всі знаходяться в діапазоні 0-1. Аналоговий має перевагу перед цифровим у тому, що він диференційований, і тому підходить для зворотного поширення [15].

Ці ворота діють на сигнали, які вони отримують, і, подібно до вузлів нейронної мережі, вони блокують або передають інформацію на основі її

потужності та імпорту, яку вони фільтрують за допомогою власних наборів ваг. Ці коефіцієнти, як і коефіцієнти, що модулюють вхідні та приховані стани, коригуються за допомогою процесу навчання періодичних мереж. Тобто комірки дізнаються, коли дозволити даним вводити, залишати або видалятися за допомогою ітераційного процесу припущень, зворотного поширення помилки та коригування ваг за допомогою градієнтного спуску. На рисунку 2.13 показано, як дані протікають через комірку пам'яті та керуються її воротами.

Починаючи знизу, потрійні стрілки показують, де інформація надходить у комірку в кількох точках. Ця комбінація поточного вхідного та минулого стану комірки подається не тільки в саму комірку, а й до кожного з її трьох воріт, які вирішують, як буде оброблятися вхід.

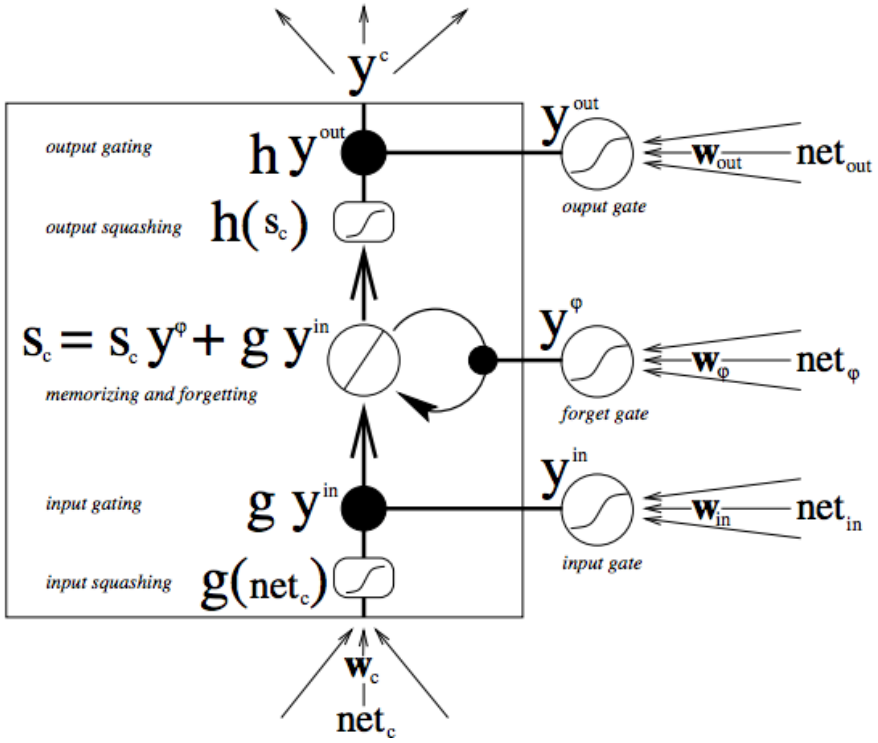

Рисунок 2.13 – Мережа LSTM

Чорні точки – це самі вентилі, які відповідно визначають, чи дозволити новий вхід, стерти поточний стан комірки та/або дозволити цьому стану впливати на вихід мережі на поточному етапі часу. S_c – поточний стан комірки

пам’яті, а gy^{in} – поточний вхід до неї. Треба запам’ятати, що кожне із воріт може бути відкритим або закритим, і на кожному кроці вони повторно комбінують свої відкриті та закриті стани. Комірка може забути свій стан чи ні, бути записаною чи ні, і зчитуватися чи ні, на кожному кроці часу.

Важливо зазначити, що комірки пам’яті LSTM надають різну роль додаванню та множенню в перетворенні вхідних даних. Центральний знак плюс на рисунку ?, по суті, є секретом LSTM. Як би нерозумно це здавалося, ця базова зміна допомагає їм зберегти постійну помилку, коли її потрібно поширювати назад на глибину. Замість того, щоб визначати наступний стан комірки шляхом множення її поточного стану на нові вхідні дані, вони додають два, і це буквально робить різницю [15].

Різні набори ваг фільтрують вхід для введення, виведення та забуття. Вентиль забуття представлений у вигляді лінійної ідентифікаційної функції, тому що якщо вентиль відкритий, поточний стан комірки пам’яті просто множиться на одиницю, щоб поширюватися вперед ще на один часовий крок.

На рисунку 2.14 – ви можна побачити, як працюють ворота, причому прямі лінії представляють закриті ворота, а порожні кола — відкриті. Лінії та кола, що йдуть горизонтально вниз по прихованому шару, є воротами забуття.

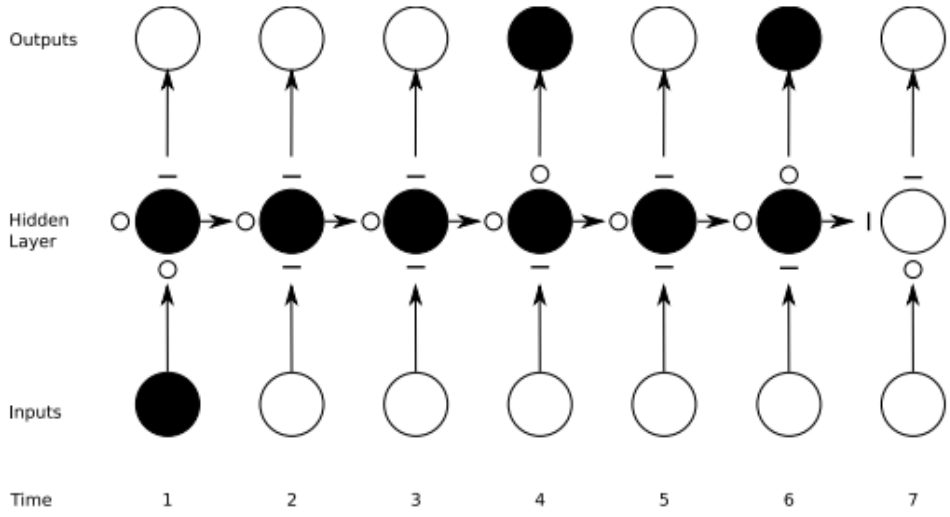

Рисунок 2.14 – Демонстрація роботи вентилів

2.6 Часові ряди та їх аналіз

Часовий ряд – це ряд точок даних, проіндексованих (перелічених, або відкладених на графіку) в хронологічному порядку. Найчастіше часовий ряд є послідовністю, взятою на рівновіддалених точках у часі, які йдуть одна за одною. Таким чином, він є послідовністю даних дискретного часу. Прикладами часових рядів є висоти океанських припливів, кількості сонячних плям та ін.

Часові ряди дуже часто представляють за допомогою лінійних діаграм. Часові ряди використовуються в статистиці, обробці сигналів, розпізнаванні образів, економетриці, фінансовій математиці, прогнозуванні погоди, розумному транспорті та передбаченні траєкторій, передбаченні землетрусів, електроенцефалографії, автоматичному керуванні, астрономії, технологіях зв’язку, а також значною мірою в будь-якій області прикладної науки та інженерії, яка включає часові вимірювання.

Аналіз часових рядів – це специфічний спосіб аналізу послідовності точок даних, зібраних за певний проміжок часу. При аналізі часових рядів аналітики записують точки даних через послідовні інтервали протягом певного періоду часу, а не просто записують точки даних періодично або випадково. Однак цей тип аналізу – це не просто збір даних за певний час. Що відрізняє дані часового ряду від інших даних, так це те, що аналіз може показати, як змінні змінюються з часом. Іншими словами, час є важливою змінною, оскільки він показує, як дані коригуються в ході точок даних, а також кінцеві результати. Він забезпечує додаткове джерело інформації та встановлений порядок залежностей між даними. Аналіз часових рядів зазвичай вимагає великої кількості точок даних для забезпечення узгодженості та надійності. Великий набір даних гарантує, що у вас є репрезентативний розмір вибірки, а аналіз може прорізати зашумлені дані. Це також гарантує, що будь-які виявлені тенденції або закономірності не є викидами та можуть враховувати сезонні відхилення. Крім того, дані часових рядів можна використовувати для прогнозування майбутніх даних на основі історичних даних.

Аналіз часових рядів допомагає організаціям зрозуміти основні причини тенденцій або системних закономірностей з часом. Використовуючи візуалізацію даних, бізнес-користувачі можуть побачити сезонні тенденції та глибше дізнатися, чому ці тенденції виникають. За допомогою сучасних аналітичних платформ ці візуалізації можуть вийти далеко за межі лінійних графіків. Коли організації аналізують дані через послідовні інтервали, вони також можуть використовувати прогнозування часових рядів, щоб передбачити ймовірність майбутніх подій. Прогнозування часових рядів є частиною прогнозної аналітики. Він може показувати ймовірні зміни в даних, як-от сезонність або циклічну поведінку, що забезпечує краще розуміння змінних даних і допомагає краще прогнозувати. Сучасні технології дозволяють нам щодня збирати величезні обсяги даних, і легше, ніж будь-коли, зібрати достатньо послідовних даних для всебічного аналізу.

3 ОПИС АРХІТЕКТУРИ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИЗНАЧЕННЯ ОБ’ЄКТІВ

Існує кілька архітектур нейронних мереж, створених для визначення об’єктів. Вони в основному поділяються на «дворівневі», такі як R-CNN, fast R-CNN і faster R-CNN, і «однорівневі», такі як YOLO.

«Дворівневі» нейронні мережі, перераховані вище, використовують так звані регіони на зображенні, щоб визначити, чи знаходиться в цьому регіоні певний об’єкт.

Зазвичай це виглядає так (для faster R-CNN, яка є найшвидшою з перерахованих дворівневих систем):

а) подається картинка/кадр на вхід;

б) кадр пускається через CNN для формування feature maps;

в) окремою нейронною мережею визначаються регіони з високою ймовірністю знаходження в них об’єктів;

г) далі ці регіони за допомогою RoI pooling стискаються і подаються в нейронну мережу, що визначає клас об’єкта в регіонах;

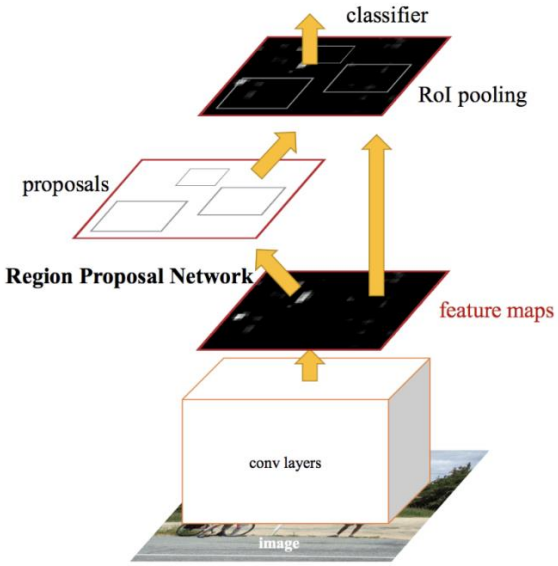

Рисунок 3.1 – Архітектура R-CNN

Але в цих нейронних мережах є дві ключові проблеми: вони не дивляться на картинку «повністю», а тільки на окремі регіони, і вони відносно повільні. YOLO краще тим, що ця архітектура не має двох проблем, і вона довела неодноразово свою ефективність.

Взагалі Архітектура YOLO в перших блоках не сильно відрізняється за «логікою блоків» від інших детекторів, тобто на вхід подається картинка, далі створюються feature maps за допомогою CNN (правда в YOLO використовується своя CNN під назвою Darknet-53), потім ці feature maps певним чином аналізуються, видаючи на виході позиції і розміри bounding boxes і класи, яким вони належать [16].

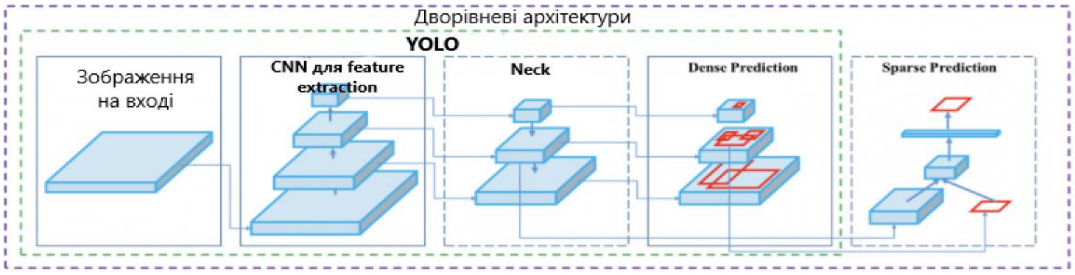

Рисунок 3.2 – Відмінність YOLO від дворівневих архітектур

З Sparse Prediction ми розібралися трохи раніше – це просто повторення того, як дворівневі алгоритми працюють: визначають окремо регіони і потім класифікують ці регіони.

Neck (або «шия») – це окремий блок, який створений для того, щоб агрегувати інформацію від окремих шарів з попередніх блоків (як показано на рисунку 3.2) для збільшення точності передбачення.

І, нарешті, те, що відрізняє YOLO від всіх інших архітектур – блок під назвою Dense Prediction. Це дуже цікаве рішення, яке якраз дозволило YOLO вирватися в лідери по ефективності визначення об'єктів. YOLO (You Only Look Once) несе в собі філософію дивитися на картинку один раз, і за цей один перегляд (тобто один прогін картинки через одну нейронну мережу) робити всі необхідні визначення об’єктів, як це продемонстровано на рисунку 3.3 [16].

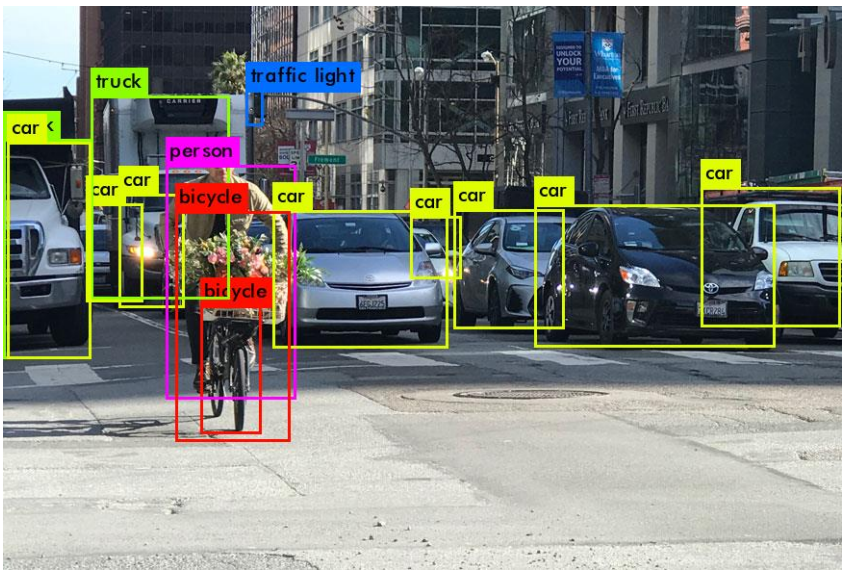

Рисунок 3.3 – Результат знаходження об’єктів за допомогою архітектури YOLO

Що робить YOLO коли навчається на даних:

а) крок перший зазвичай зображення переформовують під розмір 416x416 перед початком навчання нейронної мережі, щоб можна було їх подавати батчами (для прискорення навчання).

б) крок другий діле зображення на клітини розміром **a**x**a**. В YOLOv3-4 прийнято ділити на клітини розміром 13x13 (рису 3.4).

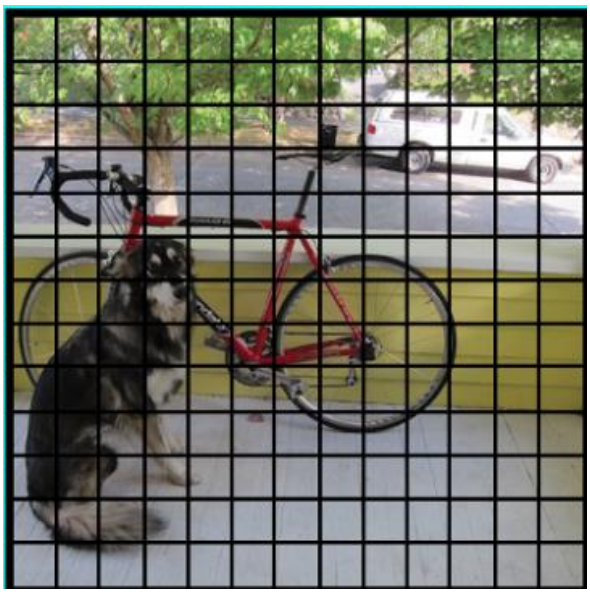

Рисунок 3.4 – Розбиття зображення на клітини

Сфокусуємося на клітинках, на які ми розділили картинку/кадр. Такі клітини, які називаються grid cells, лежать в основі ідеї YOLO (рис. 3.5). Кожна клітина є «якорем», до якого прикріплюються bounding boxes. Тобто навколо клітини малюються кілька прямокутників для визначення об’єкта (оскільки незрозуміло, якої форми прямокутник буде найбільш підходящим, їх малюють відразу кілька і різних форм), і їх позиції, ширина і висота обчислюються щодо центру цієї клітини.

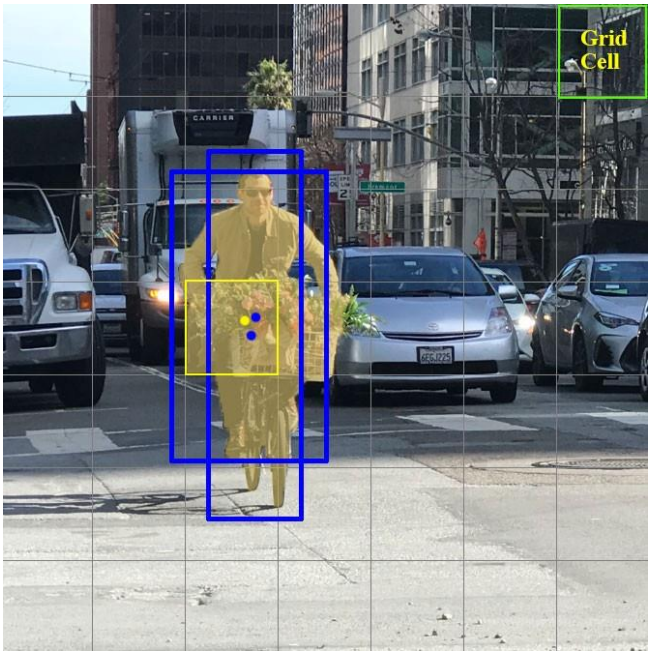

Рисунок 3.5 – Основна ідея YOLO (grid cells)

Розглянемо як малюються ці прямокутники (bounding boxes) навколо клітини. І подивимося як визначається їх розмір і позиція. Тут в боротьбу вступає техніка anchor boxes (в перекладі – якірні коробки, або «якірні прямокутники»). Вони задаються на самому початку або самим користувачем, або їх розміри визначаються виходячи з розмірів bounding boxes, які є в датасеті, на якому буде тренуватися YOLO (використовується K-means clustering і IoU для визначення найбільш підходящих розмірів). Зазвичай задають близько 3 різних anchor boxes, які будуть намальовані навколо (або всередині) однієї клітини, як це зображено на рисунку 3.6.

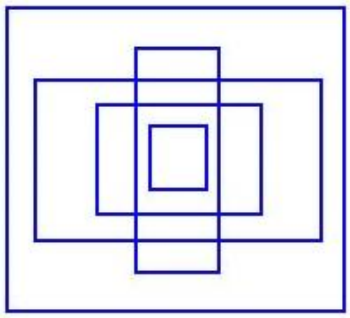

Рисунок 3.6 – Відображення anchor boxes

Крок 3. Зображення із датасету пускається через нашу нейронну мережу (зауважимо, що крім картинки в тренувальному датасеті у нас повинні бути визначені позиції і розміри справжніх bounding boxes для об’єктів, які є на ній. Це називається «анотація» і робиться це в основному вручну).

Для кожної клітини, потрібно зрозуміти дві принципові речі:

а) який з anchor boxes, з 3 намальованих навколо клітини, нам підходить найбільше і як його можна трохи підправити для того, щоб він добре вписував в себе об’єкт;

б) який об’єкт знаходиться всередині цього anchor box і чи є він взагалі.

Тоді output у YOLO повинен бути таким:

а) на виході для кожної клітини ми хочемо отримати (рис. 3.7):

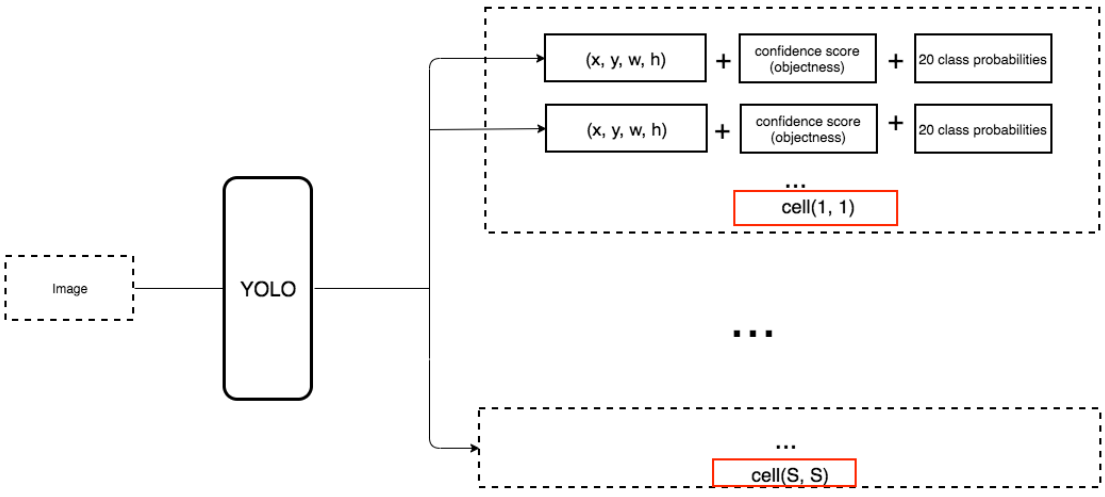

Рисунок 3.7 – Результат який ми хочемо отримати

б) output повинен включати в себе ось такі параметри (рис 3.8):

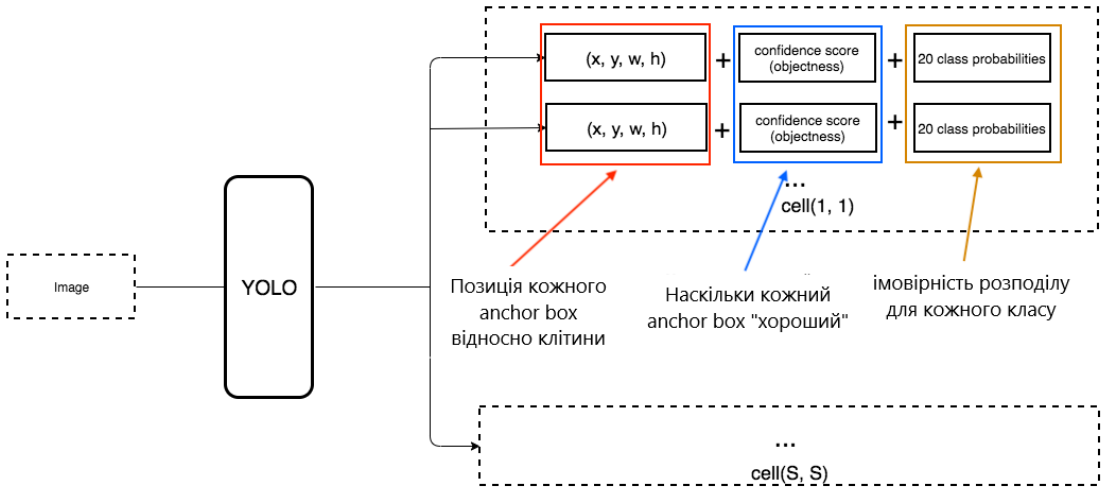

Рисунок 3.8 – Параметри, що входять до output

Розглянемо як визначається параметр objectness. Насправді цей параметр визначається за допомогою метрики IoU під час навчання. Метрика IoU зображена на рисунку 3.9

На початку ми можемо виставити поріг для цієї метрики, і якщо наш передбачений bounding box буде вище цього порога, то у нього буде objectness рівній одиниці, а всі інші bounding boxes, у яких objectness нижче, будуть виключені. Ця величина objectness знадобиться нам, коли ми будемо вважати загальний confidence score (наскільки ми впевнені, що це саме потрібний нам об’єкт розташований всередині передбаченого прямокутника) у кожного певного об’єкта [16].

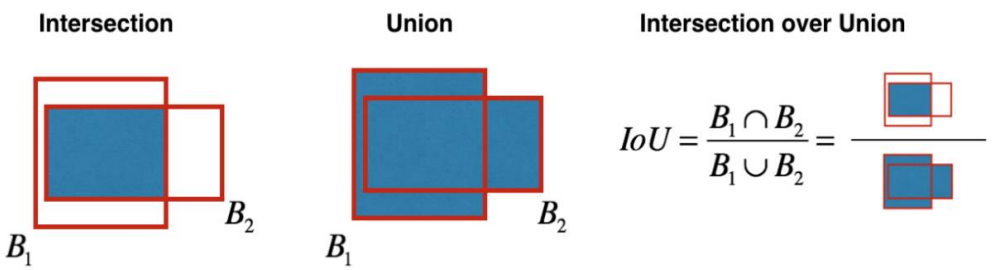

Рисунок 3.9 – Метрика IoU

А тепер починається найцікавіша частина. Уявімо, що ми творці YOLO і нам потрібно натренувати її на те, щоб розпізнавати машини на кадрі/картинці. Ми подаємо зображення із датасету в YOLO, там відбувається feature extraction на початку, а в кінці у нас виходить CNN шар, який розповідає нам про всі клітини, на які ми «розділили» наше зображення. І якщо цей шар розповідає нам «неправду» про клітини на зображенні, то у нас повинен бути великий Loss, щоб потім його зменшувати при подачі в нейронну мережу наступних картинок. На рисунку 3.10 зображено схему того, як YOLO створює останній шар.

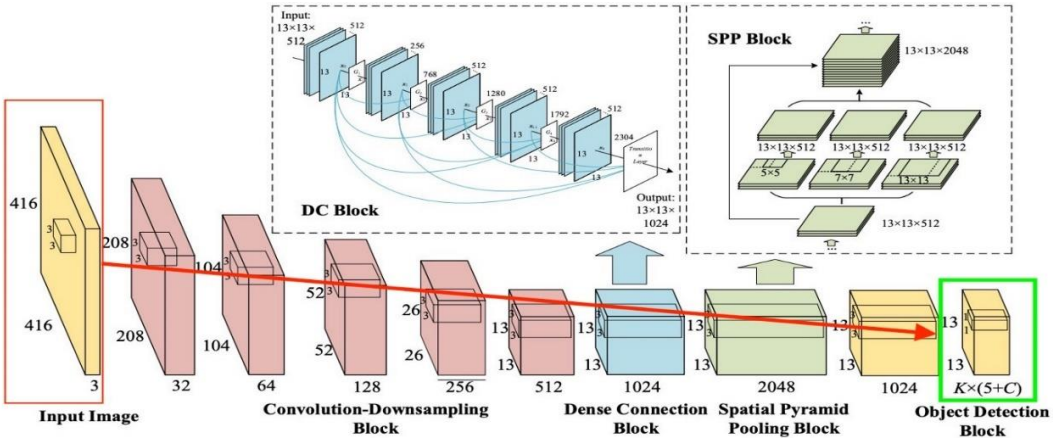

Рисунок 3.10 Створення останнього шару в YOLO

Як ми бачимо з рисунку 3.10, цей шар, розміром 13x13 (для картинок початкового розміру 416x416) для того, щоб розповідати про «кожну клітинку» на зображенні. З цього останнього шару і дістається інформація, яку ми хочемо.

YOLO передбачає 5 параметрів (для кожного anchor box для певної клітини):

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x, \\
b_y &= \sigma(t_y) + c_y, \\
b_w &= p_w e^{t_w}, \\
b_h &= p_h e^{t_h}, \\
\Pr(object) * IoU(b, object) &= \sigma(t_o),
\end{aligned}
\tag{3.1}
$$

де tx, ty, tw, th, – те, що передбачила YOLO,

cx, cy – координата верхньої лівої точки потрібної grid cell,

pw, ph – ширина та висота певного anchor box,

bx, by, bw, bh – параметри для с передбаченого bounding box,

σ(to) – confidence score.

Для наочності є хороша візуалізація на рисунку 3.11.

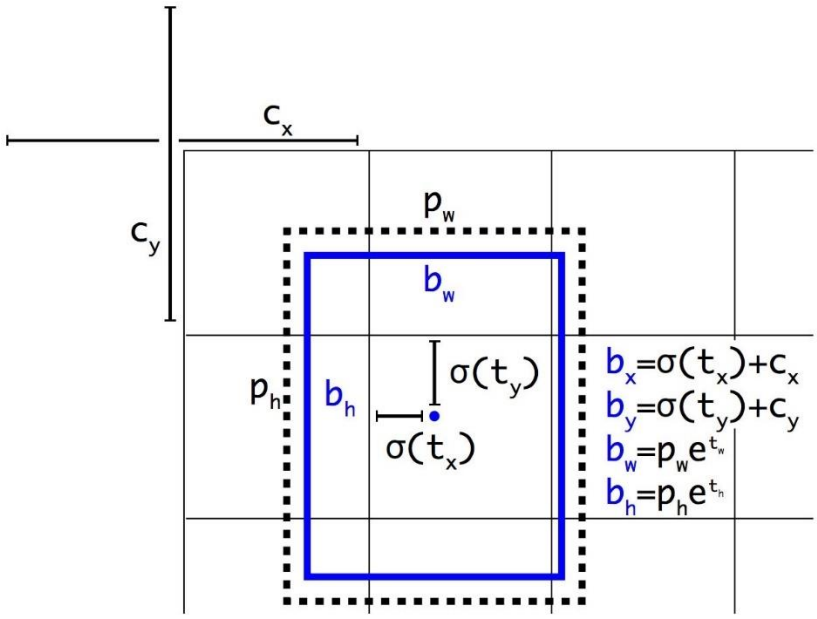

Рисунок 3.11 – Візуалізація параметрів у YOLO

Як можна зрозуміти із рисунку 3.11, завдання YOLO – максимально точно передбачити ці параметри, щоб максимально точно визначати об’єкт на зображенні. А confidence score, який визначається для кожного передбаченого bounding box, є певним фільтром для того, щоб відсіяти зовсім неточні передбачення. Для кожного передбаченого bounding box ми множимо його IoU на ймовірність того, що це певний об’єкт (імовірнісний розподіл розраховується під час навчання нейронної мережі), беремо кращу ймовірність з усіх можливих, і якщо число після множення перевищує певний поріг, то ми можемо залишити цей передбачений bounding box на картинці.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Розпізнавання об’єктів

У роботі використовується модуль DNN OpenCV для прямої роботи з YOLO. DNN означає глибоку нейронну мережу. OpenCV має вбудовану функцію для виконання алгоритмів DNN.

У цьому проекті буде виявлено і класифіковано автомобілі на дорозі і підрахована кількість транспортних засобів, що проїжджають по дорозі. Дані будуть зберігатися для аналізу різних транспортних засобів, які проїжджають по дорозі.

Буде створено дві програми для виконання цього проекту. Першим буде трекер для виявлення транспортних засобів з використанням OpenCV, який відстежує кожен виявлений транспортний засіб на дорозі, а другий буде основною програмою виявлення.

Трекер в основному використовує концепцію дистанції Евкліда, що на рисунку 4.1, для відстежування об’єкта. Він обчислює різницю між двома центральними точками об’єкта в поточному кадрі в порівнянні з попереднім, і якщо відстань менше порогової відстані, то він підтверджує, що об’єкт є тим самим з попереднього кадру.

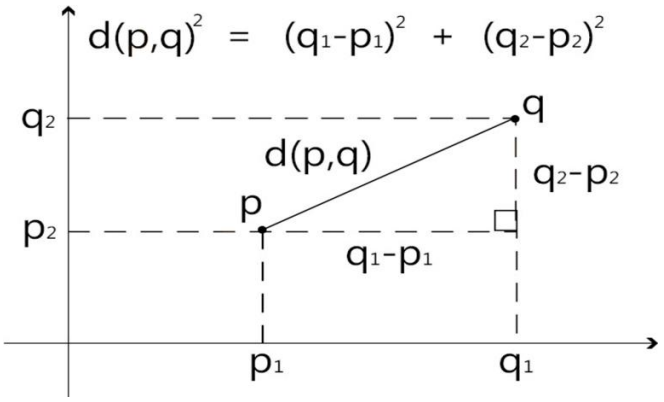

Рисунок 4.1 – Концепцію дистанції Евкліда для відстеження об'єкта

Приклад програмної реалізації обчислення різниці між двома точками об’єкта у поточному кадрі в порівнянні з попереднім:

```
#Get center point of new object
for rect in objects_rect:
    x, y, w, h, index = rect
    cx = (x + x + w) // 2
    cy = (y + y + h) // 2
    #Find out if that object was detected already
    same_object_detected = False
    for id, pt in self.center_points.items():
        dist = math.hypot(cx - pt[0], cy - pt[1])
        if dist < 25:
            self.center_points[id] = (cx, cy)
            # print(self.center_points)
            objects_bbs_ids.append([x, y, w, h, id, index])
            same_object_detected = True
            break
```

У першому циклі буде отримана центральна точка нового об’єкта та ставлено флаг який вказує на те що об’єкта ще не було у кадрі, далі метод math.hypot() повертає евклідову відстань, щоб переконатися що об’єкт є тим самим. Потім флаг перемикається на вказівку присутності існуючого об’єкту.

Кроки для виявлення та класифікації транспортних засобів за допомогою OpenCV:

а) крок 1 – імпортування необхідних пакетів та ініціалізування мережі;

б) крок 2 – зчитування кадрів з відеофайлу;

в) крок 3 – попередня обробка кадрів і запуск виявлення;

г) крок 4 – подальша обробка вихідних даних;

д) крок 5 – відстежування і підрахунок всіх транспортних засобів на дорозі;

е) крок 6 – збереження остаточних даних в CSV-файл.

Крок 1 – імпортування необхідних пакетів та ініціалізування мережі.

По-перше, треба імпортувати всі необхідні пакети, необхідні для проекту. Потім ініціалізувати об’єкт EuclideanDistTracker() з програми відстеження, яку було створено раніше, і встановити об’єкт як «tracker».

Приклад імпортуючих бібліотек у проекті:

```
import cv2
import csv
import collections
import numpy as np
from tracker import *
tracker = EuclideanDistTracker()
confThreshold = 0.1
nmsThreshold = 0.2
```

Змінні confThreshold та nmsThreshold є мінімальним порогом оцінки достовірності для виявлення і порогом придушення, відмінним від максимального, відповідно.

Приклад створення ліній на зображені:

```
middle_line_position = 225
up_line_position = middle_line_position - 15
down_line_position = middle_line_position + 15
```

Ці позиції ліній є позиціями перетину ліній, які будуть використовуватися для підрахунку транспортних засобів.

Наведемо програмну реалізацію для збереження coco.names у список:

```
classesFile = "coco.names"
classNames = open(classesFile).read().strip().split('\n')
```

YOLO навчається на наборі даних COCO, тому треба читати файл, що містить всі імена класів, і зберегти імена в списку. Набір даних COCO містить 80 різних класів. Для цього проекту потрібно виявити лише легкові автомобілі, мотоцикли, автобуси та вантажівки, тому required_class_index містить індекс цих класів з набору даних COCO.

Приклад зчитування індексів класів з набору даних COCO:

```
modelConfiguration = 'yolov3-320.cfg'
modelWeights = 'yolov3-320.weights'
net = cv2.dnn.readNetFromDarknet(modelConfiguration,
modelWeights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
np.random.seed(42)
colors=np.random.randint(0,255,size=(len(classNames),3),
dtype='uint8')
```

Мережа налаштовується за допомогою функції cv2.dnn.readNetFromDarknet(). Для використання графічного процесоро, треба встановити net.setPreferableBackend як DNN_BACKEND_CUDA, а net.setPreferableTarget як DNN_TARGET_CUDA. Якщо відсутні бібліотеки для CUDA, або потрібно використати процесор можна закометувати ці строки.

Використовуючи функцію np.random.randint(), генерується випадковий колір для кожного класу в наборі даних. Ці кольори будуть використовуватися, щоб намалювати прямокутники навколо об’єктів. Функція random.seed() зберігає стан випадкової функції, щоб вона могла генерувати деяке випадкове число при кожному виконанні, навіть якщо вона буде генерувати ті ж випадкові числа і на інших машинах.

Крок 2 – зчитування кадрів з відеофайлу.

Приклад завантаження та отримання кадрів з відеофайлу:

```
cap = cv2.VideoCapture('video.mp4')
def realTime():
    while True:
    success, img = cap.read()
    img = cv2.resize(img,(0,0),None,0.5,0.5)
    ih, iw, channels = img.shape
    cv2.line(img,(0,middle_line_position),(iw,
    middle_line_position),(255, 0, 255), 1)
```

Відеофайл зчитується за допомогою об’єкта VideoCapture і об’єкт встановлюється як cap.read(), що зчитує кожен кадр з об'єкта захоплення. Використовуючи cv2.reshape(), кадр стискається на 50 відсотків. Потім за допомогою функції cv2.line() малюються пересічні лінії в рамці. І, нарешті, використовуємо функцію cv2.imshow() для відображення вихідного зображення.

Крок 3 – попередня обробка кадру і запуск виявлення.

Приклад обробки зображення:

```
input_size = 320
blob = cv2.dnn.blobFromImage(img, 1/255, (input_size, input_size), [0, 0, 0], 1, crop=False)
net.setInput(blob)
layersNames = net.getLayerNames()
outputNames = [(layersNames[i[0]-1]) for i in net.getUnconnectedOutLayers()]
```

Використана версія YOLO приймає в якості вхідних даних об’єкти зображення 320x320. Вхід в мережу являє собою об’єкт blob. Функція dnn.blobFromImage() приймає зображення в якості вхідних даних і повертає змінений розмір і нормалізований об’єкт blob. Функція net.forward() використовується для передачі зображення в мережу та повертає результат повертає результат. І, нарешті, викликається користувацька функція postProcess() для подальшої обробки вихідних даних.

Крок 4 – подальша обробка вихідних даних:

Прямий мережевий вихід містить 3 виходи. Кожен вихідний об’єкт являє собою вектор довжиною 85, а саме:

- 4 обмежувальні границі (по центру осі абсцис, по центру осі ординат, по ширині, по висоті);
- 1 впевненість у коробці;
- 80 впевненостей у класі;

По-перше, визначається порожній список global detected_classNames, в якому будуть зберігатися всі виявлені класи у фреймі.

Отже, визначимо функцію пост-обробки на прикладі :

```
detected_classNames = []
def postProcess(outputs,img):
   global detected_classNames
   height, width = img.shape[:2]
    boxes = []
    classIds = []
    confidence_scores = []
    detection = []
    for output in outputs:
        for det in output:
            scores = det[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if classId in required_class_index:
                if confidence > confThreshold:
                    w,h      =      int(det[2]*width)      ,
                    int(det[3]*height)
                    x,y     =     int((det[0]*width)-w/2)     ,
                    int((det[1]*height)-h/2)
                    boxes.append([x,y,w,h])
                    classIds.append(classId)
                confidence_scores.append(float(confidence))
```

По-перше, визначається порожній список global detected_classNames, в якому будуть зберігатися всі виявлені класи у фреймі. Використовуючи два цикли for, перебирається кожен вектор кожного виводу і збирається оцінка достовірності і класичний індекс.

Після цього йде перевірка, чи перевищує показник достовірності класу встановлений поріг на початку. Потім збирається інформація про клас і зберігаються точки координат поля, ідентифікатор класу і показник достовірності в трьох окремих списках. YOLO іноді дає кілька обмежувальних рамок для одного об'єкта, тому доводиться зменшувати кількість рамок виявлення і вибирати кращу рамку виявлення для кожного класу.

Використовуючи метод NMSBoxes(), зменшується кількість рамок і беруться тільки кращі рамки виявлення для класу. Функція cv2.putText() малює текст в рамці, а cv2.rectangle() малює обмежуючу рамку навколо виявленого об’єкта.

Наведемо програмну реалізацію вибору найкращої рамки:

```
indices = cv2.dnn.NMSBoxes(boxes, confidence_scores,
confThreshold, nmsThreshold)
for i in indices.flatten():
  x,y,w,h = boxes[i][0], boxes[i][1], boxes[i][2], boxes[i][3]
  color = [int(c) for c in colors[classIds[i]]]
  name = classNames[classIds[i]]
  detected_classNames.append(name)
  cv2.putText(img,f'{name.upper()}{int(confidence_scores[i]*
  100)}%',(x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
cv2.rectangle(img, (x, y), (x + w, y + h), color, 1)
detection.append([x,y,w,h,
required_class_index.index(classIds[i])])
```

Використовуючи метод NMSBoxes(), зменшується кількість рамок і беруться тільки кращі рамки виявлення для класу. Функція cv2.putText() малює текст в рамці, а cv2.rectangle() малює обмежуючу рамку навколо виявленого об’єкта.

Крок 5 – відстеження та підрахунок всіх транспортних засобів на дорозі.

Приклад трекінгу транспортних засобів

```
boxes_ids = tracker.update(detection)
for box_id in boxes_ids:
     count_vehicle(box_id)
```

Після отримання всіх виявлених даних відстежуються всі об’єкти за допомогою функції відстеження. Функція tracker.update() відстежує кожен виявлений об’єкт і оновлює положення про нього.

Count_vehicle – це користувацька функція, яка підраховує кількість транспортних засобів, які перетнули дорогу.

Також створюється два тимчасових порожніх списку для зберігання ідентифікаторів транспортних засобів, які входять в лінію перетину в’їзду. Списки up_list і down_list призначені для підрахунку цих чотирьох класів транспортних засобів на маршруті вгору і вниз.

Приклад реалізації функції count_vehicle():

```
#List for store vehicle count information
temp_up_list = []
temp_down_list = []
up_list = [0, 0, 0, 0]
down_list = [0, 0, 0, 0]
#Function for count vehicle
def count_vehicle(box_id):
    x, y, w, h, id, index = box_id
    # Find the center of the rectangle for detection
    center = find_center(x, y, w, h)
    ix, iy = center
```

До функції find_center() передаються наступні параметри позиції повертає центральну точку прямокутника.

Далі потрібно відстежувати положення кожного транспортного засобу та їх відповідні ідентифікатори.

Приклад відстежування транспортних засобів:

```
# Find the current position of the vehicle
    if (iy > up_line_position) and (iy < middle_line_position):
        if id not in temp_up_list:
            temp_up_list.append(id)
    elif iy < down_line_position and iy > middle_line_position:
        if id not in temp_down_list:
            temp_down_list.append(id)
    elif iy < up_line_position:
        if id in temp_down_list:
            temp_down_list.remove(id)
            up_list[index] = up_list[index]+1
    elif iy > down_line_position:
        if id in temp_up_list:
            temp_up_list.remove(id)
            down_list[index] = down_list[index] + 1
```

Спочатку перевіряється, чи знаходиться об’єкт між лінією перетину вгору і середньою лінією перетину, потім ідентифікатор об’єкта – це тимчасова змінна, збережена в up_list для підрахунку транспортних засобів на маршруті. І ми робимо зворотне і для транспортних засобів, що прямують за маршрутом.

А потім перевіряється, чи перетнув об’єкт нижню лінію чи ні. Якщо об’єкт перетнув лінію вниз, то ідентифікатор об’єкта зараховується як транспортний засіб, що йде по маршруту вгору, і ми додаємо 1 з певним типом лічильника класів.

Потрібні тільки координат у, тому що ми рахуємо транспортні засоби по осі Y.

Приклад реалізації додавання кола у рамку:

```
cv2.circle(img, center, 2, (0, 0, 255), -1
```

Cv2.circle () малює коло в рамці. У даному випадку малюється центральна точка автомобіля.

Приклад реалізації виведення підрахунків:

```
cv2.putText(img,"Up",(110,20),cv2.FONT_HERSHEY_SIMPLEX,
font_size,
font_color, font_thickness)
cv2.putText(img,"Down",(160,20),cv2.FONT_HERSHEY_SIMPLEX,
font_size, font_color, font_thickness)
cv2.putText(img,"Car:"+str(up_list[0])+""+
str(down_list[0]), (20, 40),
cv2.FONT_HERSHEY_SIMPLEX,font_size,font_color, font_thickness)
cv2.putText(img,"Motorbike:"+str(up_list[1])+""+
str(down_list[1]), (20, 60),
cv2.FONT_HERSHEY_SIMPLEX,font_size,font_color, font_thickness)
cv2.putText(img,"Bus:"+str(up_list[2])+""+
str(down_list[2]), (20, 80),
cv2.FONT_HERSHEY_SIMPLEX,font_size,font_color, font_thickness)
cv2.putText(img,"Truck:"+str(up_list[3])+""+
str(down_list[3]), (20, 100),
cv2.FONT_HERSHEY_SIMPLEX,font_size,font_color, font_thickness)
```

Нарешті, малюються підрахунки, щоб показати, як автомобіль розраховує на кадрі в режимі реального часу.

Крок 6 – збереження остаточних даних у файл CSV.

Приклад реалізації збереження даних

```
    with open("data.csv", 'w') as f1:
      cwriter = csv.writer(f1)

cwriter.writerow(['Direction','car','motorbike','bus','truck'])
      up_list.insert(0, "Up")
      down_list.insert(0, "Down")
      cwriter.writerow(up_list)
      cwriter.writerow(down_list)
    f1.close()
```

За допомогою функції open() відкривається новий файл data.csv тільки з дозволом на запис. Потім записуються 3 рядки, перший рядок містить назви класів і напрямки, а 2-я і 3-я рядки містять кількість маршрутів вгору і вниз відповідно. Функція writerow() записує рядок у файл.

Також є реалізація для пошуку об’єктів на статичному зображені для розширення функціоналу програми.

Неможливо підрахувати транспортні засоби, які рухаються по дорозі в певному напрямку на статичному зображенні. Тому що це безперервний процес. Але може класифікувати і підраховувати кількість транспортних засобів, які присутні на дорозі або зображенні. А пізніше ми зможемо проаналізувати отримані дані.

Приклад знаходження об’єктів на статичному зображені:

```
    image_file = 'vehicle classification-image02.png'
    def from_static_image(image):
        img = cv2.imread(image)
        blob=cv2.dnn.blobFromImage(img,1/255,(input_size,
input_size),
        [0, 0, 0], 1, crop=False)
        net.setInput(blob)
        layersNames = net.getLayerNames()
        outputNames = [(layersNames[i[0] - 1]) for i in
        net.getUnconnectedOutLayers()]
        outputs = net.forward(outputNames)
        postProcess(outputs,img)
        frequency = collections.Counter(detected_classNames)
        print(frequency)
```

Спочатку створюється функція, яка приймає файл зображення в якості вхідних даних. Використовуючи функцію cv2.imread (), зчитується зображення.

Після цього той же самий процес повторюється, що і на попередньому кроці для виявлення об’єктів.

Раніше всі виявлені об’єкти зберігалися в списку «etected_classNames». Отже, використовуючи collections.Counter(detected_classNames) можна обчислити частоту елементів у списку. Він повертає словник, що містить елемент в якості ключа словника і частоту елемента в якості значення цього конкретного ключа.

Приклад виведення інформації на статичному зображені

```
     cv2.putText(img, "Car: "+str(frequency['car']), (20, 40),
     cv2.FONT_HERSHEY_SIMPLEX,        font_size,        font_color,
font_thickness)
     cv2.putText(img,"Motorbike:"+str(frequency['motorbike']),(20,
60),
     cv2.FONT_HERSHEY_SIMPLEX,        font_size,        font_color,
font_thickness)
     cv2.putText(img, "Bus:"+str(frequency['bus']), (20, 80),
     cv2.FONT_HERSHEY_SIMPLEX,        font_size,        font_color,
font_thickness)
     cv2.putText(img, "Truck:"+str(frequency['truck']),(20, 100),
     cv2.FONT_HERSHEY_SIMPLEX,        font_size,        font_color,
font_thickness)
     cv2.imshow("image", img)
     cv2.waitKey(0)
```

Після цього у рамці малюються підрахункові тексти. Cv2.imshow () показує вихідне зображення в новому вікні opencv. Cv2.waitKey(0) утримує вікно відкритим до тих пір, поки не буде натиснута будь-яка клавіша. І, нарешті, дані записуються до CSV-файл.

На рисунку 4.2 можна побачити як програма знаходить та виділяє об’єкти у кадрі, для статичного режиму буде таке ж зображення але без ліній та підрахунку за напрямком.

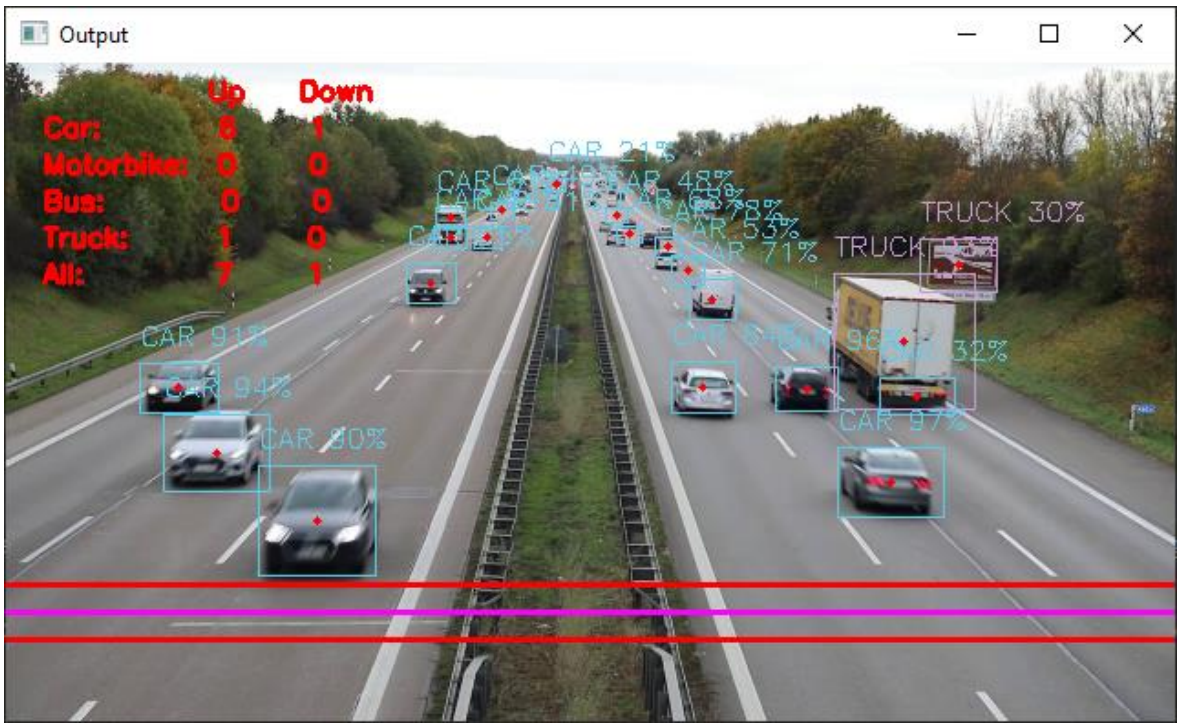

Рисунок 4.2 – Вивід роботи програми

4.2 Прогнозування часового ряду

В даному випадку задачу можна сформувати як проблему регресії. Тобто, враховувати кількість машин проїжджаючих перехрестя спочатку за попередній день, потім на теперішній, а з накопиченням даних – за неділю, місяць, рік і т. д.

Для початку подивимося як дані виглядають при початковому отримані результатів за один день на рисунку 4.3. Цих даних дуже мало для того, щоб почати прогнозувати кількість транспортного засобу на дорогах, тому система з кожним днем збирає все більше інформації про трафік та намагається спрогнозувати поведінку у майбутньому.

Програма містить просто функцію для перетворення стовпця даних у набір даних із двох стовпців: перший стовбець, що містить кількість машин за часом на вчорашній день, другий кількість машин на сьогодні, для прогнозування.

Для початку треба імпортувати усі функцію та класи, які будуть використовуватися. А саме бібліотеку для глибокого навчання Keras, яка містить у бібліотеці Tensorflow, та Pandas.

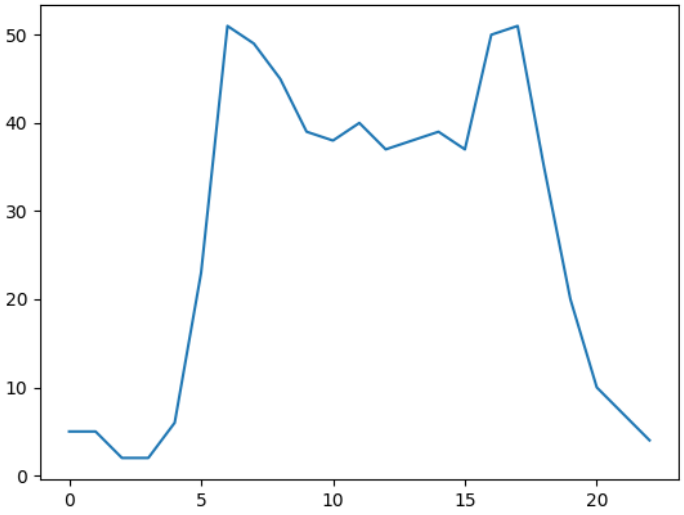

Рисунок 4.3 – Відображення даних за один день

Перед початком, рекомендується виправити початкове число випадкових чисел, щоб забезпечити відтворюваність отриманих результатів.

LTSM чутливий до масштабу вхідних даних, особливо коли використовуються сигмоїдні (за замовчуванням) або tanh функції активації. Доброю практикою може стати масштаб даних у діапазоні від 0 до 1, що так же називається нормалізацією. Можна легко нормалізувати набір даних за допомогою класу передньої обробки MinMaxScaler з бібліотеки scikit-learn.

Приклад нормалізації даних:

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

Після того, треба скомпілювати дані та оцінити на навички моделі на навчальному наборі даних, слід отримати уявлення о навичках моделі на нових невидимих даних. Для звичайної задачі класифікацію або регресії це робилося за допомогою перехресної перевірки.

Для часових рядів важлива послідовність значень, Простий метод, який можна використовувати – розділити упорядкований набір даних на навчальні та тестові набори даних. Приведений нижче приклад обчислює індексі точки поділу

и ділить дані на набори начальних даних з 67% спостережень, котрі можна використовувати для навчання моделі, залишаючи решту 33% для тестування моделі.

Приклад обчислення індексів точок:

```
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:],
dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

Далі визначається функція для створення нового набору даних. Функція приймає два аргументи: dataset, який являє собою масив NumPy, котрий потрібно перетворити у набір даних, та look_back, який являє собою кількість попередніх часових кроків, котрі будуть використовуватися у якості вхідних змінних для прогнозування наступного періоду часу – у даному випадку береться за замовчуванням 1.

Створюється набір даних, де X – кількість машин у минулому часі, а Y – кількість машин на поточний момент.

Приклад реалізації коду, для підготовки набору даних для тренування та тестування моделі:

```
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

Мережа LSTM очікує, що вхідні дані будуть представлені з певною структурою масиву в формі: зразки, часові кроки, особливості.

На даний момент дані представлені у вигляді: зразки, особливості, далі треба сформувати проблему як один часовий крок для кожної вибірки. Можна перетворити підготовлені тренувальні та тестові вхідні дані в очікувану структуру за допомогою numpy.reshape().

Тепер можна приступити до проектування нашої мережі LSTM. Вона має видимий шар з 1 входом, прихований шар з 4 блоками або нейронами та вихідний шар, котрий робить прогнозування одного значення. Для блоків LSTM використовується функція активації сигмоїда за замовчуванням. Мережа навчена для 100 епох та використовує batch_size який дорівнює 1.

Приклад створення LSTM мережі:

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

Через те, як був підготовлений набір даних, треба здвинути прогнозування так, щоб вони вирівнялися по осі X з вихідним набором даних. Після підготовки дані відображаються на графіку, на якому вхідні дані відображаються синім кольором, прогнози для тренувальних даних – зеленим, а прогнозування для невидимого набору тестових даних – помаранчевим.

Приклад ілюструє виведення інформації на монітор:

```
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back,:]=
trainPredict
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset
)-1, :] = testPredict
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Даллі на рисунку 4.4 можна побачити що модель добре підійшла як для тренувальних, так і для тестових наборів даних.

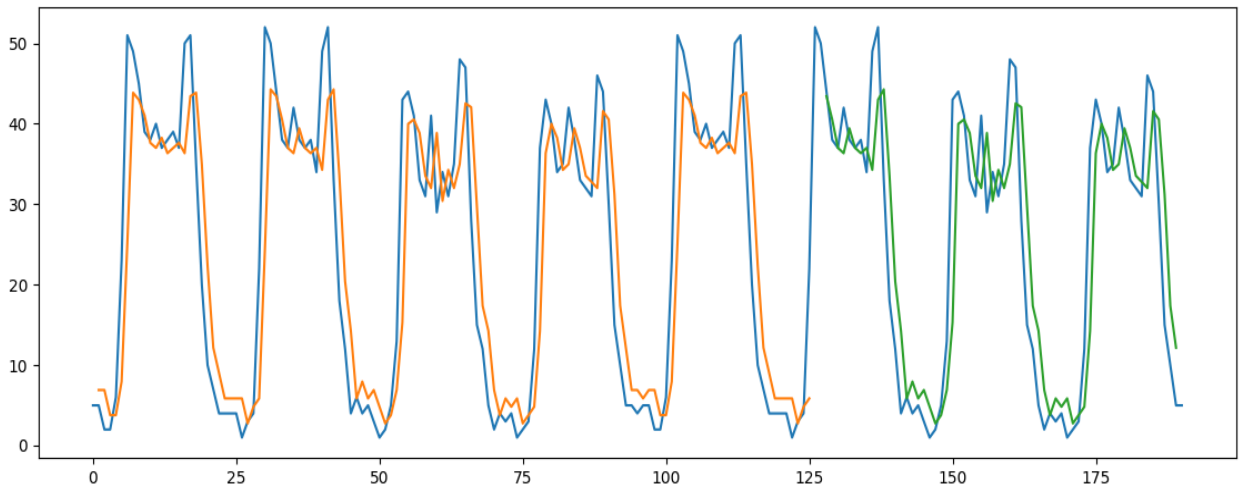

Рисунок 4.4 – Результат навченої мережі LSTM

Фінальною частиною є регулювання світлофору в залежності від кількості транспортних засобів у певному часовому проміжку, якщо передбачається що буде багато машин, слід збільшити час для проїзду даного перехрестя. Класифікація транспортних засобів у подальшому дозволить розширити алгоритми оптимізації, наприклад давання певному класу більший пріоритет для зменшення затору.

ВИСНОВКИ

Прогнозування заторів привертає більше уваги за останні кілька десятиліть. З розвитком інфраструктури кожна країна стикається з проблемою заторів. Тому прогнозування заторів може дозволити органам влади спланувати та вжити необхідних заходів, щоб уникнути його. Розвиток штучного інтелекту та доступність великих даних спонукали дослідників застосовувати різні моделі в цій галузі.

Машинне навчання, особливо глибоке навчання, має переваги в цьому випадку. Тому алгоритми глибокого навчання з часом стали все більш популярними, оскільки вони можуть оцінювати великий набір даних. Однак широкий спектр алгоритмів машинного навчання ще не застосований. Таким чином, широкі можливості для досліджень у сфері прогнозування заторів все ще переважають.

Штучний інтелект у прогнозуванні дорожнього руху може допомогти вчасно виявити ризики та зробити керування транспортним засобом комфортнішим та легшим. ШІ також робить продуктивний внесок у регулювання транспортних потоків і планування цілих дорожніх систем. Аналізуючи схеми руху, обсяги руху та інші дані, проект дорожніх мереж можна адаптувати до переважних умов – для оптимального транспортного потоку, який окупається не лише з економічної, але й з екологічної точки зору.

При виконанні кваліфікаційної роботи було виявлено проблеми пов’язанні з заторами та вирішення їх за допомогою штучного інтелекту. Розглянуто як штучний інтелект розуміє зображення за допомогою згорткових мереж. Також виявлено, що для роботи інтелектуальної системи потрібні високопродуктивні сервери або локальні центри, бо обробка зображення з камер вимагає великих затрат на обробку інформації, а також прогнозування результатів. Було розглянуто найпоширеніші архітектури нейронної мережі для визначення образів – дворівневі та однорівневі. До однорівневних відносять такі мережі як

R-CNN, fast R-CNN і faster R-CNN. Ці мережі використовують так звані регіони на зображенні, щоб визначити, чи знаходиться в цьому регіоні певний об’єкт. Але в цих нейронних мережах є дві ключові проблеми: вони не дивляться на картинку «повністю», а тільки на окремі регіони, і вони відносно повільні. YOLO краще тим, що ця архітектура не має двох проблем, і показує свою ефективність.

Також розглянуто, що таке часові ряди та спосіб їх аналізу для вирішення поставленої задачі. Для вирішення цієї проблеми була задіяна рекурентна нейронна мережа а саме LSTM. Бо вона допомагає зберегти помилку, яку можна поширити назад через час і шари. Підтримуючи більш постійну похибку, вона дозволяє рекурентним мережам продовжувати навчатися протягом багатьох часових кроків (понад 1000), тим самим відкриваючи канал для дистанційного зв’язування причин і наслідків. Це одна з головних проблем машинного навчання та штучного інтелекту, оскільки алгоритми часто стикаються з середовищами, де сигнали винагороди рідкісні та відкладені.

Отже підводячи підсумок можна сказати, що розглянута проблема в кваліфікаційній роботі є дуже актуальною на сьогоднішній день, люди по всьому світу намагаються вирішити її, але до цього часу це було складно зробити через не дуже розвинений штучний інтелект та невеликі обчислювальні потужності систем. Тому така інтелектуальна система у подальшому матиме великий попит.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Костров Д.Р. Оптимізація дорожнього руху з використанням штучного інтелекту. *Радіоелектроніка та молодь у XXI столітті*: матеріали 25-го міжнар. мол. форуму. Харків, 2011. С. 185-186.

2. Kharkiv traffic URL: https://www.tomtom.com/en_gb/traffic-index/kharkiv-traffic (дата звернення: 09.11.2021)

3. Müller A., Guido S. Introduction to Machine Learning with Python: A Guide for Data Scientists. USA: O'Reilly Media, 2016. 400 p.

4. Glassner A. Deep Learning: A Visual Approach. San Francisco: No Starch Press, 2021. 776 p.

5. Lakshmanan V. Practical Machine Learning for Computer Vision: End-to-End Machine Learning for Images. Boston: O'Reilly Media, 2021. 482 p.

6. PyCharm URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата звернення 12.11.2021).

7. What is Python? Executive Summary URL: https://www.python.org/doc/essays/blurb/ (дата звернення 11.11.2021).

8. What is NumPy? URL: https://numpy.org/doc/stable/user/whatisnumpy.html (дата звернення 14.11.2021).

9. Why TensorFlow URL: https://www.tensorflow.org/about (дата звернення 13.11.2021).

10. About – OpenCV URL: https://opencv.org/about/ (дата звернення 15.11.2021)

11. Aggarwal C. Neural Networks and Deep Learning: A Textbook. NY: Springer, 2018. 520 p.

12. Mohut S., Md. Rezaul K., Pradeep P. Practical Convolutional Neural Networks: Implement advanced deep learning models using Python. Birmingham: Packt Publishing, 2018. 218 p.

13. Zafar I., Tzanidou G., Burton R., Patel N., Araujo L.Hands-On

Convolutional Neural Networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python. Birmingham. Packt Publishing, 2018. 274 p.

14. Kostadinov S. Recurrent Neural Networks with Python Quick Start Guide: Sequential learning and language modeling with TensorFlow. Birmingham: Packt Publishing, 2018. 122 p.

15. Babcock J., Bali R. Babcock J. Generative AI with Python and TensorFlow 2: Create images, text, and music with VAEs, GANs, LSTMs, Transformer models. Birmingham: Packt Publishing, 2021. 488 p.

16. YOLO: Real-Time Object Detection URL: https://pjreddie.com/darknet/ yolo/ (дата звернення 18.11.2021).