МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
Секція інформаційно-комунікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«Інтелектуальна система оцінки функціонального стану стічних труб»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Коробов А.Г.**

Студент групи ІН-71 **Коплик А.В.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності "Комп'ютерні науки та інформаційні технології" денної форми навчання Коплика Артема Віталійовича.

Тема: "Інтелектуальна система оцінки функціонального стану стічних труб"

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) інформаційна технологія аналізу відеоданих інспекції стічних труб; 3) створення набору даних для навчання моделі; 4) навчання та оцінка згорткової нейронної мережі; 5) реалізація алгоритму інформаційної системи розпізнавання функціонального стану стічних труб;

Дата видачі завдання "_____" _____2020 р.

Керівник випускної роботи _____ Коробов А.Г.

Завдання прийняв до виконання _____ Коплик А.В.

РЕФЕРАТ

Записка: 62 стор., 28 рис., 2 табл., 2 додатка, 21 джерело.

Об’єкт дослідження — застосування згорткових нейронних мереж для інспекції стічних труб.

Мета роботи — розробка алгоритму обробки відеоданих інспекції стічних труб, з використанням глибоких нейронних мереж, для детектування місць з’єднання труб та формування звіту з картою їх розташування.

Методи дослідження — аналіз відеоданих інспекції, синтез інтелектуальної системи

Результати — розроблено інтелектуальну систему, що визначає положення стиків труб, відносно даних лічильника пройденого камерою шляху на відеоданих інспекції. Для цього використовується об’єднання прогнозів моделей класифікації орієнтації камери та визначення показників одометра. В результаті створюються звіти в форматі відео і таблиці з мітками відстані та часу.

ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ, СТІЧНІ ТРУБИ,
ДЕТЕКТУВАННЯ СТИКІВ ТРУБ, ВІДЕОІНСПЕКЦІЯ,
ІНТЕЛЕКТУАЛЬНА СИСТЕМА

ЗМІСТ

ВСТУП

Каналізаційні мережі, призначені для транспортування стічних вод, зіграли одну із найважливіших ролей в розвитку цивілізації, адже вони пов’язані зі збільшенням тривалості життя людей, за рахунок поліпшення санітарних умов густонаселених районів, де гігієна є ключовим чинником для запобігання епідемій [1]. По мірі старіння каналізаційної системи її труби поступово руйнуються зі швидкістю, що залежить від умов експлуатації та інших зовнішніх факторів, які можуть вплинути на структурну цілісність труб.

Інспекція за допомогою систем відеоспостереження (CCTV) отримала широке визнання як ефективна технологія інспекції для підземної інфраструктури [2]. Каналізаційні труби можуть проходити під землею на тисячі кілометрів, тому збирається величезна кількість відеоматеріалів CCTV, оцінка яких займає багато часу і може потребувати великої команди технологів.

Особлива увага приділяється контролю дефектів, пов’язаних зі стиками стічних труб, адже вони можуть не тільки викликати повну втрату гідравлічної пропускної здатності, а й ускладнити оцінку проведеної інспекції [3]. Тому локалізація цих елементів каналізаційної системи є важливою задачею для прогнозування і попередження серйозних пошкоджень.

Головне завдання цієї бакалаврської роботи полягає в розробці інтелектуальної системи для визначення місць з’єднання труб, з допомогою навчання згорткових нейронних мереж, що в поєднанні з моделями детектування дефектів може забезпечити покращення якості звіту інспекції та оцінки функціонального стану стічних труб.

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасні технології інспектування підземних стічних труб

Сучасні підземні каналізаційні системи побудовані в вигляді складної мережі трубопроводів. Через складність виявлення і діагностики дефектів всередині системи, їх обслуговування є непростим завданням для муніципалітетів по всьому світу. Оскільки каналізаційні труби прокладені під дорогами і вулицями, про їх наявність легко забути, поки вони не вийдуть з ладу. Заміна всієї каналізаційної труби коштує дорого і може потребувати масштабних земляних робіт, що тягне за собою порушення дорожнього руху. Більш економічно вигідним варіантом є ремонт труб до їх поломки, але для цього необхідно проводити регулярні інспекції [4]. Однак каналізаційні труби важко обстежити, так як більшість труб недоступні для інспекторів через їх малий діаметр. Для труб великого діаметру загальний вміст стічних вод і наявність токсичних газів робить інспекцію ризиком для здоров'я працівників.

Найбільш поширеним методом оцінки стану каналізаційної труби є використання CCTV. Його широке використання для проведення інспекцій пояснюється тим, що це набагато безпечніша процедура, порівняно зі звичайним оглядом людиною. Крім того, в порівнянні з іншими видами інспекції, наприклад, лазерним скануванням чи ультразвуковими датчиками, відеозапис CCTV може надати візуальні результати, які легко зрозуміти, інтерпретувати і класифікувати [1].

Інспекція CCTV зазвичай включає в себе два основних процеси: збір відео на місці і оцінка відео за межами об'єкта. Процес збору даних зазвичай виконується технологами інспекції з дотриманням певного стандарту інспекційної операції, наприклад, PACP, що поширений на Північно Американському континенті [3]. Згідно з цим стандартом дефекти стічних труб каналізаційної системи можна поділити на структурні, конструкційні, а також дефекти обслуговування і експлуатації (рис. 1.1).

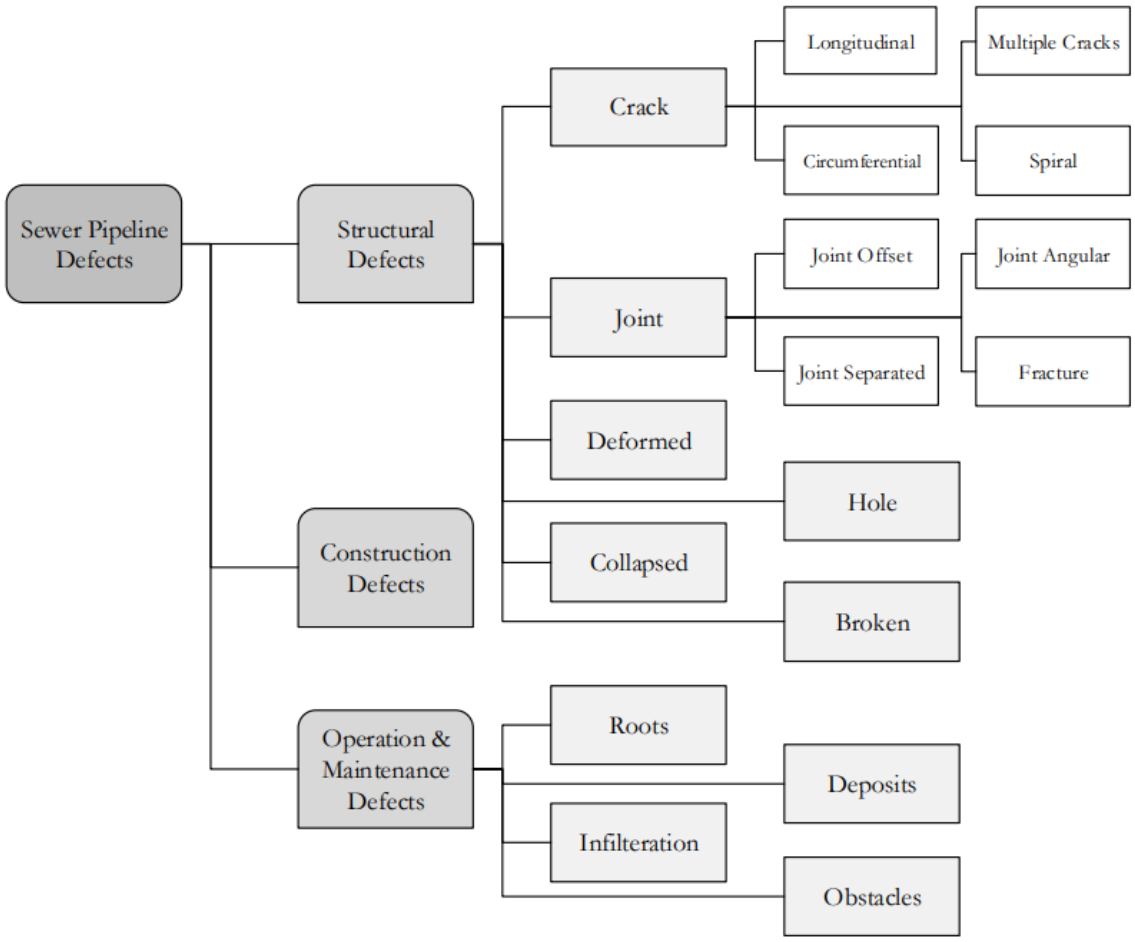


Рисунок 1.1 – Класи дефектів каналізаційних трубопроводів відповідно стандарту PACP [3, 5].

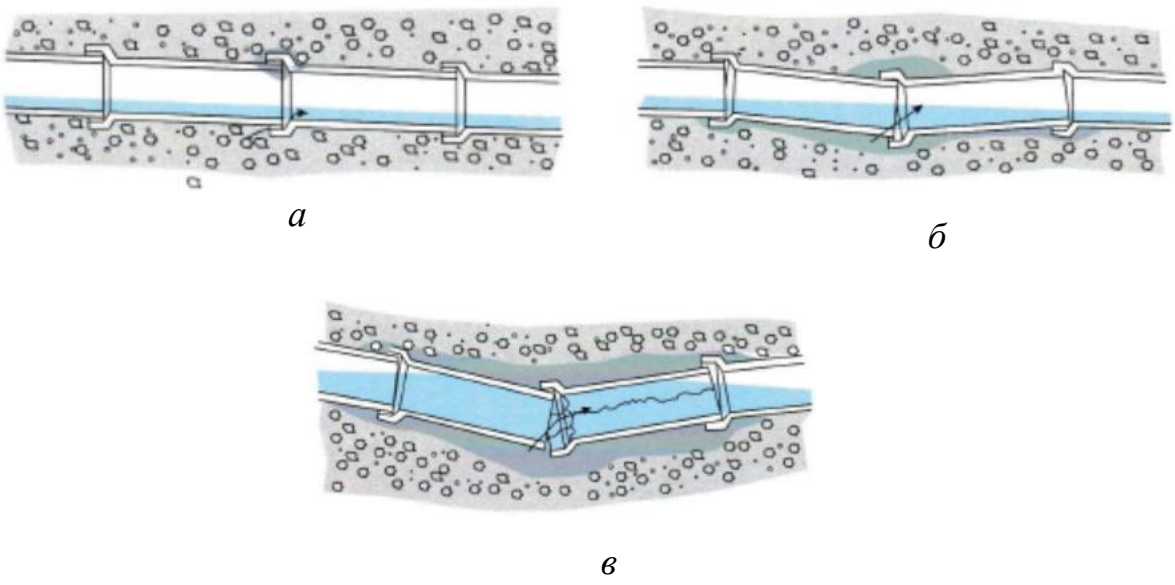
Відповідно до стандарту PACP [3], класи дефектів можна поділити на точкові (Point Defects) та протяжні (Continuous Defects). Точкові дефекти кодуються індивідуально, в міру їх виявлення, в той час як протяжні включають пошкодження, що розташовані по всій довжині труби, як-то коріння. Протяжні дефекти додатково поділяються на дійсно безперервні (Truly Continuous Defect), які тривають без перерви більше одного метра, наприклад, подовжня тріщина; та повторювані (Repeated Continuous Defect), що виявляються принаймні в 3 з 4 з'єднань (75%) вздовж труби. Протяжні дефекти, які виникають на більшій частині стиків труб, вважаються повторюваними.

Місця з'єднання труб (joint) – це одні з найслабших місць в каналізаційній системі, де дефекти часто зустрічаються і можуть призвести до втрати трубою

здатності виконувати свої основні функції. Більше того, будь-яке пошкодження, що з'являється в радіусі 8 дюймів або 200 мм від місця з'єднання безперервних сегментів труб, відзначається в інспекційному звіті [3].

Втрата гідравлічної пропускної здатності може бути спричинена просіданням труби (рис. 1.2), що відбувається за три етапи:

- створення розриву через потенційні дефекти на місцях з'єднання труб;
- інфільтрація ґрунтових вод, що призводить до утворення порожнин навколо стиків та, як результат, втрата опори ґрунту, що дозволяє трубі рухатись і просідати;
- нерівномірне навантаження на трубу через зсув стиків і відсутність опори від навколишнього ґрунту може призвести до появи тріщин або зруйнування труби.



а *б* *в*

Рисунок 1.2 – Ілюстрація етапів просідання труби [3]:

а – утворення розриву; *б* – інфільтрація; *в* – руйнування труби

По мірі прискорення процесу, тріщини або розломи можуть також ламатися і деформуватися. Камера може опинитись під водою через підйом водного рівня.

Часто зустрічається проблема нездатності виконати всебічний огляд через обмеження обладнання або наявність перешкод, таких як зруйновані частини труби. Рис. 1.3 ілюструє, як зсув місця з'єднання труб може ускладнити процес обстеження. У відрізку між двома перешкодами є тріщини, розриви та уламки, але оскільки цю частину не вдалося перевірити, ці дефекти не будуть відповідно зафіксовані [3].

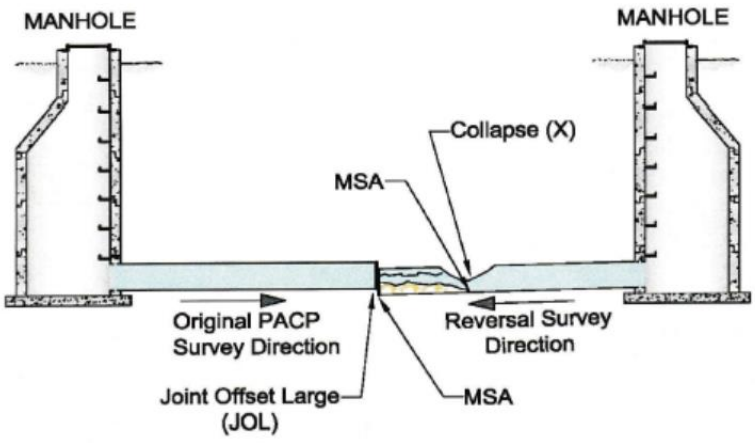


Рисунок 1.3 – Ілюстрація показує, як зсув місця з'єднання стічних труб впливає на складність оцінювання проведеного огляду [3]

Незважаючи на такі недоліки CCTV-інспекцій, як залежність від вмінь оператора, якості відео та ефекту освітлення, ця технологія залишається найпоширенішим способом оцінки функціонального стану каналізаційних трубопроводів. Замовники таких інспекцій хочуть отримати звіти, що включають точні карти дефектів та їх і візуальні підтвердження для проведення оцінки стану системи і ухвалення рішень.

Останні досягнення технологій обробки візуальних даних привели до різкого зниження вартості використання камер для інспектування. Поліпшення можливостей комп'ютерного обладнання робить можливим застосування алгоритмів методів комп'ютерного зору та машинного навчання.

В працях [1, 5] дослідники провели огляд публікацій на тему використання комп’ютерного зору для інспектування стічних труб. Методи, запропоновані дослідниками, в більшості, включають виявлення дефектів за допомогою методів

обробки зображень, таких як морфологія, виділення ознак, виявлення об'єктів і використання алгоритму машинного навчання для класифікації. Останні дослідження показують ефективність використання штучних нейронних мереж і, головним чином, алгоритмів глибокого навчання. Ці методи забезпечують точність результатів і здатність до узагальнення ознак.

В алгоритмах глибокого навчання мережа навчається виділяти ознаки автоматично. Відкидається трудомісткий етап створення ознак, що є критичним в звичайних методах машинного навчання. На рис. 1.4 показана відмінність екстракції ознак в класичних підходах машинного навчання і в підході глибокого навчання [5].

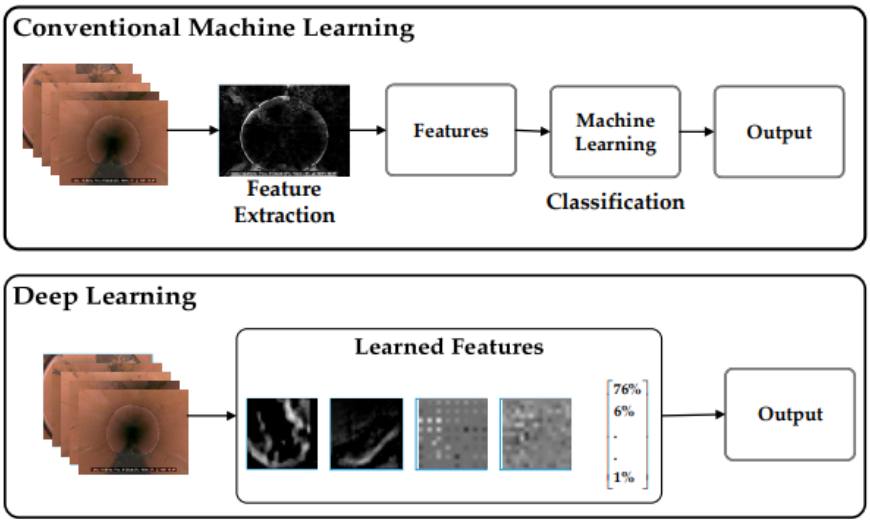


Рисунок 1.4 – Порівняння підходів машинного та глибокого навчання [5].

Проте використання даної технології вимагає пошуку оптимальної архітектури багатошарової нейронної мережі, для вирішення конкретного завдання. Адже вона впливає на ефективність узагальнення ознак та кількість, необхідних для навчання моделі, розмічених даних.

Таким чином, інспекція стічних труб – актуальна та складна задача, що вимагає великих витрат часу та коштів. Тому гостро стоїть проблема автоматизації цього процесу. Ефективним рішенням може бути розбиття

інспекції на окремі задачі та використання технології глибоких нейромереж для їх виконання.

1.2 Аналіз моделей та методів штучного інтелекту для візуального аналізу даних

Для інформаційних інтелектуальних систем, образ є сукупністю даних про абстрактний чи реальний об'єкт, що дозволяє групувати його з іншими об'єктами відповідно до вимог задачі. Умови конкретного завдання дозволяють уникнути надмірності в описі образу, тобто спростити його аналіз при розпізнаванні. Це особливо важливо в тих випадках, коли обробляються великі об'єми даних. Ознаками називають характеристики об'єкту, необхідні для вирішення завдання розпізнавання. В таких завданнях часто доводиться виділяти групи об'єктів певного типу. Для цього потрібно використовувати ознаки, що дозволяють відрізнити одну групу від інших. Такі групи називають класами. Завдання може полягати у виділенні тільки одного класу, а все, що не належить йому, розглядати як клас «інше». Тому розпізнавання будь-якої кількості образів розглядається як завдання класифікації множини об’єктів.

Системи візуального аналізу даних можуть бути простими і складними, залежно від особливостей класів об'єктів і використовуваних для цього ознак. Прості системи виконують розпізнавання одним алгоритмом на основі набору ознак одного типу. Тоді як складні – використовують ознаки різних типів. Такі системи часто є багаторівневими, тобто результати, отримані одним алгоритмом розпізнавання стають вхідними даними для іншого алгоритму. Для побудови комплексної системи розпізнавання, спочатку необхідно провести аналіз максимальної кількості доступної інформації про об'єкти інтересу.

За останні роки дослідники галузі комп'ютерного зору досягли значних успіхів. Деякі із завдань комп'ютерного зору – це класифікація зображень, виявлення і сегментація об'єктів. Класифікація зображень – присвоєння зображенню мітки класу з обмеженого набору, до якого належить зображення.

Іноді окрім класифікації потрібно локалізувати об’єкт інтересу на зображенні. Виявлення об'єктів є дещо складнішою задачею. На вхід також подається зображення, але класифікувати потрібно усі шукані класи і створити навколо них обмежуючі рамки. Найскладніше завдання – це сегментація, коли недостатньо обмежувальної рамки, а треба точно виділити маскою знайдений на зображенні об’єкт.

На ефективність аналізу складних зображень випливає спосіб представлення ознак об’єкту розпізнавання. Тому велика частина зусиль по впровадженню алгоритмів машинного навчання спрямована на розробку конвеєрів попередньої обробки і перетворення даних, в результаті яких створюється представлення даних, здатне підтримувати ефективне машинне навчання. Така розробка ознак важлива, але трудомістка і підкреслює слабкість сучасних алгоритмів навчання. Галузь, що досліджує навчання представленням даних, які полегшують отримання корисної інформації при побудові класифікаторів, називається навчанням ознак або навчанням представлень. У праці [6] дослідники даної галузі описують універсальні принципи ефективного представлення даних:

– гладкість: передбачає, що для функції f, яка описує навчену модель, виконується умова $f(x) \approx f(y)$ при $x \approx y$;
– множинні пояснюючі чинники: кожен новий чинник, при навчанні моделі, узагальнюється до множини конфігурацій інший чинників;
– ієрархія пояснювальних чинників: поняття, корисні для опису навколишнього світу, можуть бути визначені в термінах інших понять, в ієрархії, причому більш абстрактні поняття, що знаходяться вище в ієрархії, визначаються в термінах менш абстрактних.
– напів-контрольоване навчання (semi-supervised learning): при вхідних даних X та цілі Y для прогнозування, підмножина чинників, що пояснюють розподіл X, пояснює велику частину Y, враховуючи X. Представлення, корисні для $P(X)$, зазвичай корисні при вивченні $P(Y|X)$, що дозволяє

розділити статистичну силу між завданнями навчання керованого і без учителя.

- спільні чинники між задачами: різні задачі моделей, що мають спільні пояснюючі чинники, використовують свої статистичні зв’язки;
- многовиди (manifolds): густина ймовірності концентрується поблизу областей, які мають розмірність меншу ніж початковий простір, в я кому знаходяться дані;
- природна кластеризація: різні значення категоріальних змінних, таких як класи об’єктів, асоціюються з окремими многовидами;
- часова та просторова когеренція: послідовні в часі або просторово близькі спостереження, як правило, пов'язані з одним і тим же значенням відповідних категоріальних понять, або призводять до невеликого переміщення на поверхні високощільного многовиду.
- розрідженість (sparsity): для всіх спостережень X тільки невелика частина можливих чинників є значимою. Це може бути представлено ознаками, які часто дорівнюють нулю, тобто тим що більшість виявлених ознак не чутлива до невеликих змін X;
- простота залежностей чинників: в хороших представленнях даних високого рівня, чинники зв’язані між собою простими, як правило, лінійними залежностями. Також аргументи цього положення беруться до уваги, коли на вихід вже навченого представлення даних підключається лінійний класифікатор;

Для задачі комп’ютерного зору, не можна покладатися тільки на прості параметричні моделі, такі як лінійні моделі, оскільки вони не можуть охопити достатню частину складності цієї задачі, якщо не надати відповідний простір ознак. Як детально описано в роботі [7] більшість цих алгоритмів використовують тільки принцип локального узагальнення, тобто припущення, що цільова функція досить гладка, тому вони покладаються на приклади для явного визначення зморшок цільової функції. Узагальнення в основному

досягається шляхом локальної інтерполяції між сусідніми навчальними прикладами. Хоча гладкість може бути корисним припущенням, її недостатньо для вирішення проблеми «прокляття розмірності», оскільки кількість складок (підйомів і спадів цільової функції) може рости експоненційно з кількістю відповідних взаємодіючих чинників, коли дані представлені в необробленому вхідному просторі. Для уникнення цієї проблеми слід позбавлятися від надлишкових ознак.

Для боротьби з проблемою «прокляття розмірності» було розроблено алгоритми, що ґрунтуються на ядрах (kernel machines). В основу цих алгоритмів покладені властивість гладкості цільової функції та лінійна залежність факторів. Однак якщо на вхід алгоритму надходять «сирі» дані, то замість гладкості функції ми отримаємо множину піків і впадин, що може зростати експоненційно з кількістю впливаючих факторів. Тому ядерні методи придатні лише для попередньо підготованого ознакового простору [6].

Хороші представлення даних – виразні, це означає, що навчене представлення помірного розміру може відбивати велику кількість можливих вхідних конфігурацій. Виразність моделі можна оцінити за числом потрібних їй параметрів в порівнянні з кількістю вхідних областей (чи конфігурацій), які вона може розрізнити. Для навчання унітарним представленням, таким як традиційні алгоритми кластеризації, суміші Гауса, алгоритми найближчих сусідів, дерева рішень або Гаусові SVM, потрібно $O(N)$ параметрів (і/або $O(N)$ прикладів) для розрізнення $O(N)$ вхідних областей. ОМБ (обмежена машина Больцмана), розріджене кодування, автокодувальники або багатошарові нейронні мережі, використовуючи тільки $O(N)$ параметрів, можуть представляти до $O(2^k)$ вхідних областей, де k – кількість ненульових елементів в розрідженому представленні, і $k = N$ в нерозрідженому ОМБ та інших щільних представленнях. Ці представлення даних є розподіленими, або розрідженими. Їх експоненціальна потужність з'являється тому, що кожен параметр може повторно використовуватися для різних прикладів даних, які навіть не є один одному

близькими сусідами, тоді як в алгоритмах з локальним узагальненням різні області вхідного простору асоціюються тільки зі своїм набором параметрів. Під розподіленими представленнями розуміються ті, в яких k з N елементів або значень ознак можуть змінюватися незалежно один від одного, наприклад, якщо вони не є взаємовиключними. Кожне поняття представляється за допомогою включення або активності k ознак, при цьому кожна ознака бере участь в представленні багатьох понять. Розріджені представлення – це розподілені представлення, в яких тільки декілька елементів можуть бути змінені одночасно, тобто $k < N$ [6].

Ключовим аспектом в стратегії представлення даних є глибина. Глибока архітектура часто показує себе ефективною і має дві значні переваги: вона припускає повторне використання ознак та потенційно може навчатися значно більшим абстракціям ознак на верхніх, найбільш віддалених від вхідних даних, шарах. Абстрактніші поняття найчастіше можуть визначатися в термінах менш абстрактних. Композиції низькорівневих ознак формують високорівневі ознаки, створюючи ієрархію. Абстрактніші поняття зазвичай інваріантні до більшості локальних змін на вході, що робить представлення даних, які використовують ці принципи, значно більш нелінійними функціями від сирих вхідних даних. Мета побудови інваріантних ознак полягає у виключенні чутливості представлення в напрямах зміни даних, неінформативних для поставленого завдання. Таким чином, часто метою витягання ознак є розділення безлічі різних інформативних чинників даних, наприклад, у відео з людьми це визначення особи, здійснюваної дії, положення відносно камери, тощо. Навчання таких інваріантних ознак є метою в задачі розпізнавання образів. Оскільки інформативність може бути невідомою, важливим є завдання розподілення пояснюючих чинників – формування якомога більшого числа незалежних ознак різних рівнів абстрактності. Різниця між інваріантними ознаками та навчанням розділенню пояснюючих чинників в збереженні або видаленні «зайвої» інформації.

У загальнішому випадку різні чинники змінюються в різних тимчасових і просторових масштабах, а багато категоріальних понять, що представляють інтерес, змінюються повільно. При спробі охопити такі категоріальні змінні, цей пріоритет можна забезпечити, зробивши відповідні представлення такими, що повільно змінюються, тобто штрафуючи зміни значень в часі або просторі. Цей пріоритет був введений в роботі [8]. Принцип ідентифікації факторів, що повільно змінюються, в даних, які змінюються в часі, вивчався в роботах [8, 9] як принцип для пошуку зручних представлень даних. Часова когеренція була успішно використана в глибокій архітектурі для моделювання відео [10]. Знання про часову когеренцію можуть бути виражені різними шляхами, найпростіший – це квадрат різниці між значеннями ознак в моменти часу t і $t + 1$. З іншого боку, замість нормування квадрата зміни, можна нормувати абсолютне значення, щоб затвердити положення, що більшість змін в часі мають бути повністю нульовими, як це інтуїтивно здається для чинників реального світу. Також можна припустити, що деякі чинники можуть бути дійсно краще представлені групою чисел, таких як x, y, z – координат положення в просторі [11], ніж просто скаляром, і ці групи чисел мають тенденцію змінюватися разом. Структурована розріджена [12], може бути використана для цієї мети .

Напів-контрольоване навчання (Semi-Supervised Learning, SSL) є чимось середнім між навчанням без учителя і контрольованим, метою якого є класифікація великої кількості нерозмічених навчальних даних з використанням лише невеликого числа розмічених. Існуючі методи SSL можна умовно розділити на дві категорії: гіпотези про кластеризацію і многовиди. Кластерне припущення стверджує, що увесь набір даних має тенденцію формувати декілька дискретних кластерів, і точки даних в одному кластері, швидше за все, мають одну і ту ж мітку. Це означає, що приклади в різних класах добре розділені, і межа ухвалення рішення потрапляє в область з низькою щільністю між

кластерами. Гіпотеза многовидів припускає, що розподіл даних відповідає многовиду, розмірність якого менша, ніж у звичайного простору. Іншими словами, мітки прикладів повинні плавно змінюватися уздовж многовиду [13].

Інший спосіб підвищити інформативність опису ознак є передавальне навчання (Transfer learning) – це здатність алгоритму навчання використовувати загальні риси між різними завданнями для обміну статистичною силою і перенесення знань між задачами. Такі алгоритми вивчають представлення, що відбивають основні чинники, підмножина яких може бути релевантною для кожного конкретного завдання (рис. 1.5).

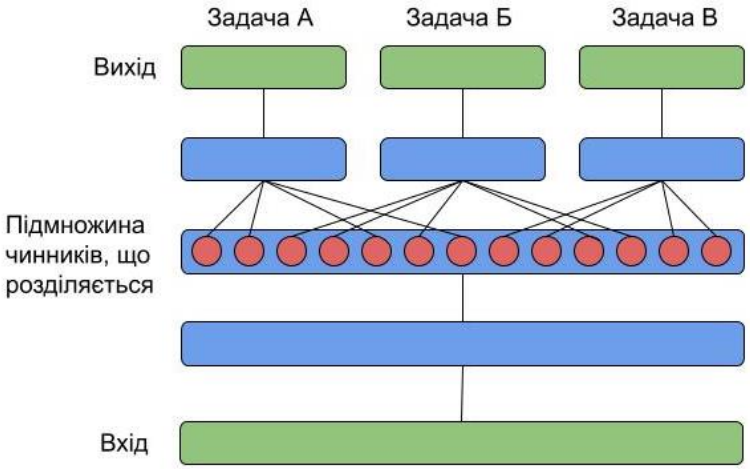


Рисунок 1.5 – Ілюстрація навчання представлень, що виявляє пояснюючі чинники (середній прихований шар), що лежать в основі видимих вхідних даних, і обчислюють відповідні цілі для кожної із трьох задач. Оскільки ці підмножини перетинаються, спільне використання статистичної сили допомагає узагальненню в навчанні [6].

Для мультизадачного навчання (Multi-task learning) підхід на базі представлень даних також має переваги [14] оскільки узагальнює пояснюючі чинники між завданнями.

Аналіз візуальної інформації – це область з найбільшою кількістю застосувань для сіамської нейронної мережі, що показано в огляді даної теми [15].

Сіамська нейронна мережа може стати кращим вибором для мінімізації розмірності простору ознак і створення незмінюваного представлення. Вона складається з двох однакових штучних нейронних мереж, кожна з яких здатна навчати приховане представлення вхідного вектору. нейронні мережі є перцептронами з прямолінійним рухом і використовують зворотне поширення помилки під час навчанню. Вони працюють паралельно в тандемі і порівнюють свої виходи у кінці, зазвичай через косинусну відстань, що є мірою схожості між вхідною парою екземплярів даних (рис. 1.6). Вихідні дані, згенеровані сіамською нейронною мережею, можна розглядати як семантичну схожість між спроектованим представленням двох вхідних векторів. [15]

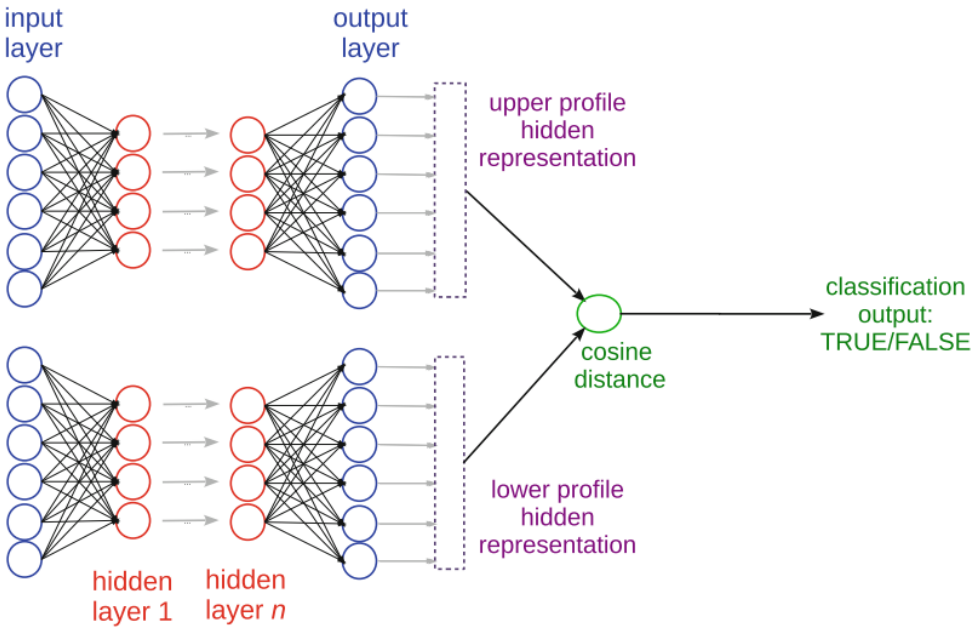


Рисунок 1.6 – Структура моделі сіамської нейронної мережі [15].

Під час навчання нейронна мережа порівнює отримані значення, з відповідним значенням «базової істини» і обчислює статистичну помилку (зазвичай середньоквадратичну помилку або помилку перехресної ентропії). Після цього нейронна мережа відправляє помилку назад на попередні шари і відповідним чином оновлює ваги нейронів, використовуючи метод, що називається зворотним поширенням помилки (error back-propagation). Навчання припиняється, коли нейронна мережа досягає спочатку заданого максимального

числа ітерацій. Після навчання моделі її можна застосувати до тестового набору: нейронна мережа оброблятиме кожен екземпляр тестових даних (тільки один раз, від першого шару ліворуч до останнього шару справа) і генеруватиме передбачення значень. Після того, як нейронна мережа згенерує передбачуване значення для кожного екземпляру тестового набору, можна використати його для обчислення матриці невідповідностей (confusion matrix)[15].

Глибока згорткова нейронна мережа – це особливий тип нейронних мереж, який показує найбільші успіхи в завданнях, пов'язаних з комп'ютерним зором і обробкою зображень. Деякі з цікавих сфер застосування згорткових нейронних мереж включають класифікацію і сегментацію зображень, виявлення об'єктів, обробку відео. Ефективність їх навчання значною мірою обумовлена використанням декількох етапів виділення ознак, які можуть автоматично вивчати представлення з даних.

Типова архітектура згорткових нейронних мереж зазвичай складається з шарів згортки (convolution) і об'єднання (poolin), що чергуються, за якими йде один або декілька повноз’єднаних шарів (Fully connected layers) у кінці [16].

На рис. 1.7 показана загальна архітектура CNN для класифікації зображень. Існує два етапи навчання мережі: прямий етап і зворотний етап. По-перше, основна мета прямого етапу - представити вхідне зображення з поточними параметрами(вагами і зміщенням) в кожному шарі. Потім вихід пророцтва використовується для обчислення вартості втрат з урахуванням істинних міток. По-друге, на основі вартості втрат зворотний етап обчислює градієнти кожного параметра за допомогою ланцюгових правил. Усі параметри оновлюються на основі градієнтів і готуються для наступного прямого обчислення. Після достатньої кількості ітерацій прямого і зворотного етапів навчання мережі може бути зупинене.

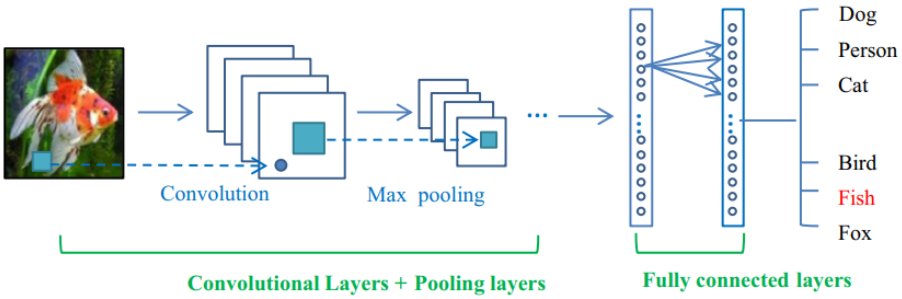


Рисунок 1.7 – Базова схема згорткової нейронної мережі для класифікації зображень [16].

Згортковий шар складається з набору ядер, де кожен нейрон виступає в ролі фільтру, який працює шляхом розділення зображення на невеликі фрагменти, що називають рецептивними полями (рис. 1.8). Розділення зображення на невеликі блоки допомагає у витяганні характерних ознак. Вхідний розмір нейронів в згорткових шарах – це розмір рецептивного поля, яке ковзає по площі вхідного зображення. Вихід фільтрів в кожному шарі є картою ознак, яка переходить в наступний згортковий шар в якості вхідного сигналу. Фільтри проходять увесь попередній шар, переміщаючись на один крок. У випадку якщо розмір попереднього шару не ділиться на розмір рецептивного поля, то фільтр пропускатиме інформацію на краях вхідної карти ознак. Для вирішення цієї проблеми можна використати нульове заповнення шляхом додавання нулів до країв вхідної карти [16].

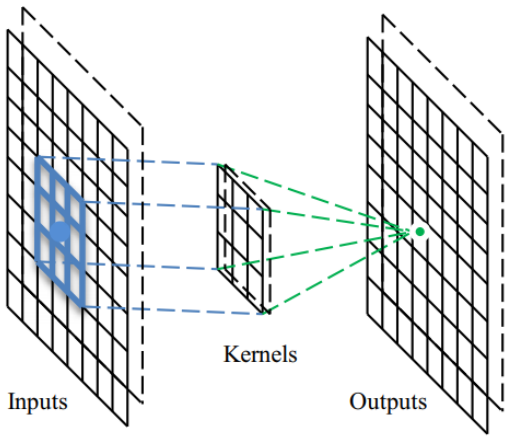


Рисунок 1.8 – Ілюстрація роботи згорткового шару.

Як правило, об'єднуючий шар йде за згортковим і може використовуватися для зменшення розмірів карт ознак і параметрів мережі. Як і згорткові шари, об’єднуючі – інваріантні до перетворення, оскільки при їх обчисленнях враховуються сусідні пікселі. Найчастіше використовуваними стратегіями є усереднювання і максимізаційне об'єднання (рис. 1.9).

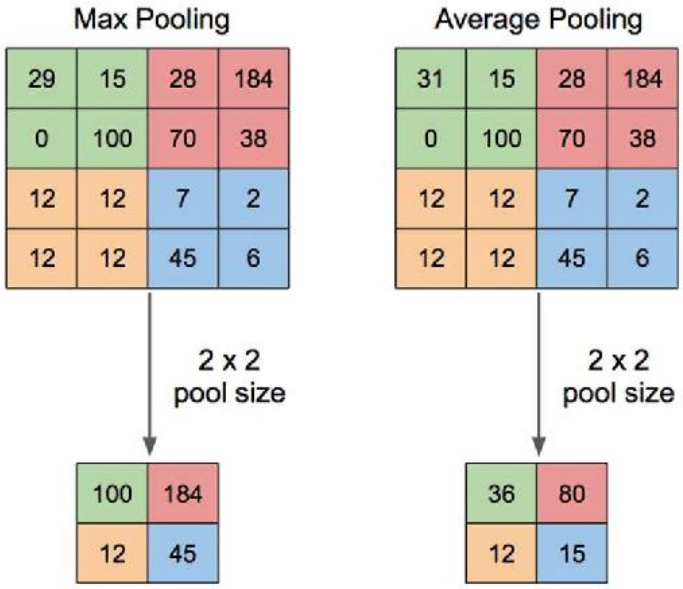


Рисунок 1.9 – Ілюстрація роботи стратегій усереднювання і максимізаційного об'єднання.

Повноз’єднаний шар в основному використовується у кінці мережі для класифікації. На відміну від об'єднання і згортки, він є глобальною операцією. Він отримує вхідні дані з етапів витягання ознак і аналізує вихід усіх попередніх шарів. В результаті виходить нелінійна комбінація вибраних ознак, яка використовується для класифікації даних [17].

Впродовж усієї історії розвитку цієї області було розглянуто ряд ідей для поліпшення згорткових нейронних мереж: використання різних функцій активації і втрат, оптимізація параметрів, регуляризація.

В праці [17] особливо виділяються архітектурні модифікації на основі моделей проектування блоків обробки. Відзначається, що основний приріст продуктивності CNN був досягнутий за рахунок заміни класичної структури шарів на блоки. Блок в мережі може грати роль допоміжного навчального

елементу. Ці допоміжні елементи використовують або просторову інформацію, або інформацію про карту ознак, або навіть посилюють вхідні канали для підвищення продуктивності. Більше того, блокова архітектура CNN сприяє модульному навчанню і тим самим робить архітектуру простішою і зрозумілішою.

Типи модифікацій архітектури згорткових нейронних мереж можна розділити на сім категорій (рис. 1.10) [17]:

– експлуатація простору: використання фільтрів різного маштабу;
– використання глибини: збільшення кількості шарів та максимізація незатухаючого градієнту;
– використання мульти-шляхів: використання з’єднань між шарами;
– використання ширини: максимізація ширини нейронної мережі, використовуючи мульти-з’єднання;
– експлуатація карти ознак: збільшення інформаційного насичення ознак методом селекції, використовуючи модуль Squeeze-Excitation;
– експлуатація канального підсилення: додавання в отримані методом transfer learning канали нових штучних каналів або застосування генеративних моделей;
– використання модулів уваги: фокусування моделі на областях інтересу зображення, за допомогою накладання масок на карту ознак.

Розвиток моделей згорткових мереж, призначених для детектування об'єктів на зображеннях, проходить в двох напрямах (рис. 1.7). Перший дотримується класичної технології детектування об'єктів, генеруючи пропозиції регіонів інтересу з подальшою класифікацією кожної пропозиції на різні категорії об'єктів. Другий розглядає детектування як проблему регресії або класифікації, використовуючи єдину структуру для визначення категорій та локалізації об’єктів на зображенні [18].

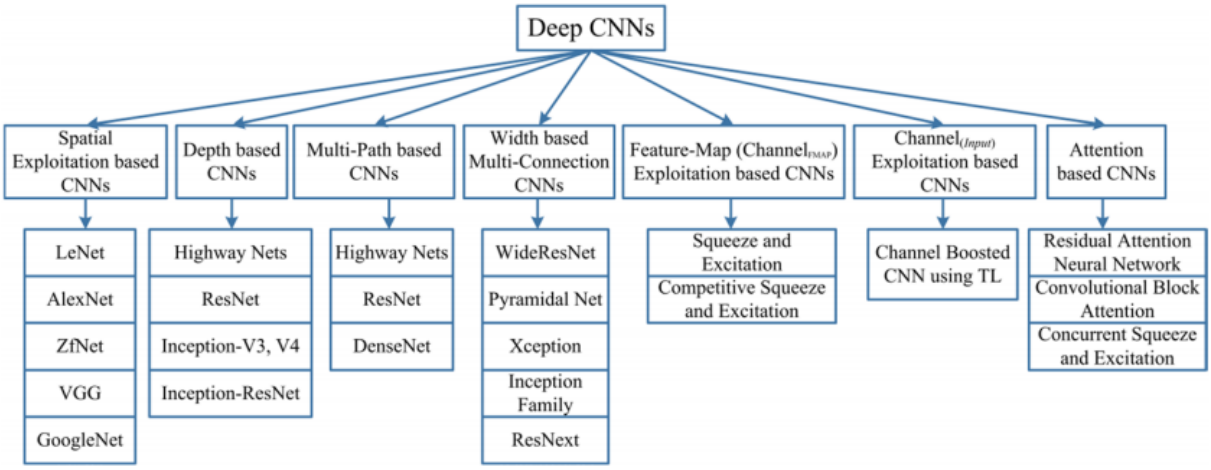


Рисунок 1.10 – Таксономія глибоких згорткових мереж, що показує сім категорій архітектурних модифікацій [17].

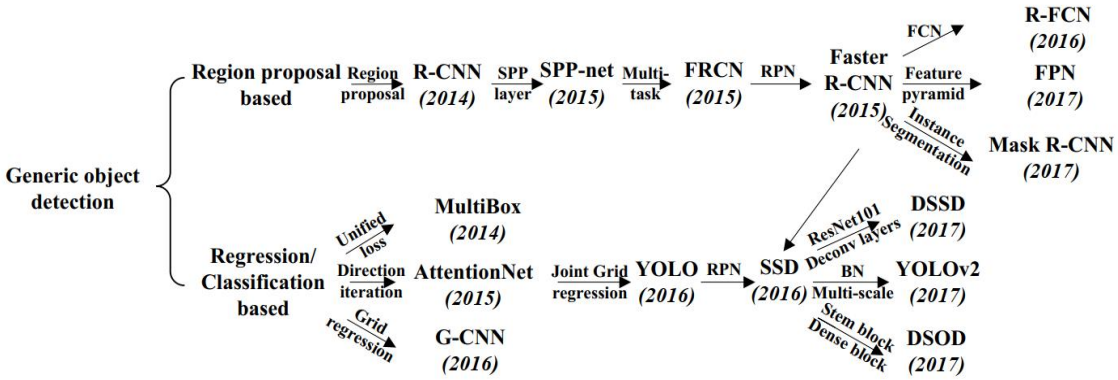


Рисунок 1.7 – Сучасні напрямки розвитку моделей згорткових нейронних мереж для детектування об’єктів [18].

Еволюція згорткових мереж, показана на схемі (рис. 1.7), призвела до розробки трьох, найбільш популярних на сьогодні, архітектур детектування об’єктів: Faster R-CNN, SSD, YOLO.

Попри високу обчислювальну вартість глибоких нейромереж, вони успішно використовується при розробці різних вбудовуваних систем. Такі архітектури, як MobileNet, ShufeNet, ANTNets, розроблені для ефективної роботи на мобільних пристроях. Так, поєднання архітектури MobileNet та SSD, легко розгортається на апаратному забезпеченні з обмеженими ресурсами і може застосовуватися для класифікації та локалізації об’єктів на відео в реальному часі [17].

1.3 Постановка задачі

В результаті проведеного огляду інформації, що стосується сучасних технологій проведення інспекції стічних труб і аналізу моделей та методів обробки візуальних даних, було виявлено, що застосування методів навчання глибоких нейронних мереж значно полегшує процес інспектування. Крім того, особливу увагу при діагностиці каналізаційних систем приділяють місцям стику труб.

Критично важливим для підтримання належного стану підземної інфраструктури є регулярність оцінювання мережі трубопроводів. Тому муніципалітети зацікавлені в отриманні карт мереж, для того щоб мати можливість відслідковувати місця з’єднання труб та дефекти, що на них з’являються.

У зв’язку з цим постановкою задачі можна вважати розробку інтелектуальної системи для визначення місць розташування стиків труб, на основі згорткової нейромережі.

Виконання поставленої задачі передбачає наступні етапи:

1) вибір моделі нейронної мережі для аналізу зображень;
2) підготовка даних та навчання моделі;
3) розробка алгоритму обробки відеоданих інспекції для детектування місць з’єднання труб та формування звіту з картою їх розташування.

2. ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АНАЛІЗУ ВІДЕОІНСПЕКЦІЇ СТІЧНИХ ТРУБ

2.1 Модель нейромережі для розпізнавання функціонального стану стічних труб

Ефективність згорткових нейронних мереж (CNN) в різних завданнях класифікації зображень, розпізнавання і локалізації об'єктів, вказує на високий потенціал для інспектування громадської інфраструктури. Нині існує ряд досліджень, спрямованих на застосування підходів глибокого навчання при автоматизації оцінки функціонального стану муніципальних об’єктів. Проте, застосування CNN для обстеження інфраструктури знаходиться на початковому етапі, і досліджень, що використовують дану технологію для створення системи інспекції стічних труб, досить мало [1].

Алгоритми глибоких нейронних мереж вимагають великих обчислювальних ресурсів. Тому гостро постає питання розробки моделей глибокого навчання для середовищ з обмеженими ресурсами. Дослідники поступово поліпшують рівень технології, так у роботі [19] представили модифікацію архітектури MobileNet – MobileNetV2, що значно зменшую об’єм необхідної пам’яті та кількість операцій, зберігаючи точність, навіть при навчанні на відносно малому наборі даних.

Відмінність модифікації MobileNet від базової структури CNN (рис. 1.7) полягає в розділенні блоку згортки (рис. 2.1) на окремі шари. Перший шар називається точковою згорткою, він відповідає за побудову нових характеристик шляхом обчислення лінійних комбінацій вхідних каналів (рис. 2.2). Другий шар – згортка за глибиною (depthwise convolution), виконує легку фільтрацію, застосовуючи один згортковий фільтр на один вхідний канал (рис. 2.3).

Основний будуючий блок мережі MobileNetV2 складається із розширювальних згорткових блоків з кроком фільтру 1 та кроком 2 (рис. 2.4). Останні використовуються для зниження просторової розмірності тензора. На

відміну від блоків з кроком 1, вони не мають залишкових з'єднань (residual connections), ідея яких полягає в тому, щоб дати активаціям попередніх шарів можливість проходити не тільки до наступного шару, але і на декілька шарів далі [19]. Переваги такого підходу детально описані в роботі [20].

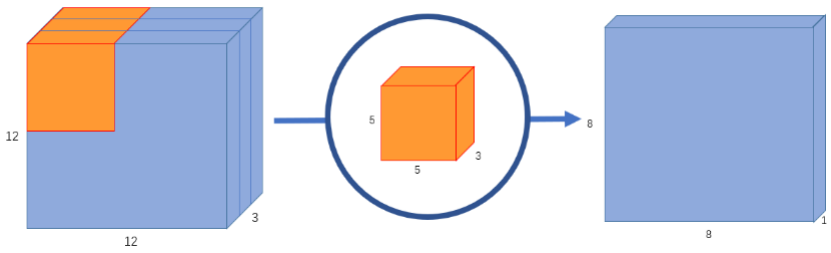


Рисунок 2.1 – Ілюстрація процесу звичайної згортки.

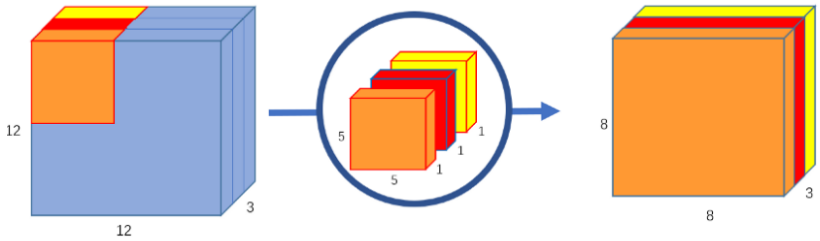


Рисунок 2.2 – Ілюстрація згортки з розділенням за глибиною.

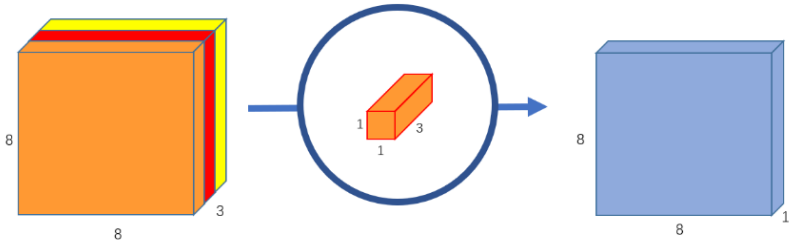


Рисунок 2.3 – Ілюстрація точкової згортки.

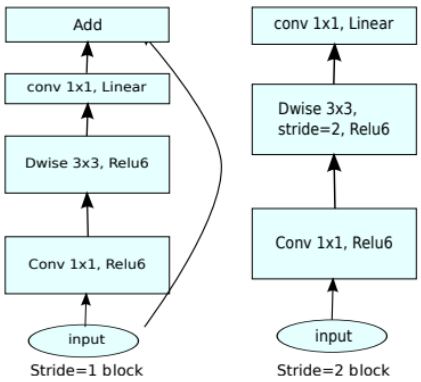


Рисунок 2.4 – Згорткові блоки архітектури MobileNetV2 з кроками 1 і 2.

Розширювальний загортковий блок, показаний на рис. 2.4, складається із трьох шарів. Спочатку йде шар точкової згортки з великим числом каналів. Він приймає на вхід тензор вигляду $h \times w \times k$, де h та w – розмір ядра згортки, k – кількість вхідних каналів, а повертає $h \times w \times (tk)$, де t – рівень розширення (expansion factor). Цей шар створює відображення вхідного тензору в просторі більшої розмірності. Далі йде згортка за глибиною з функцією активації ReLU6. Після проходження цього шару, вхідний тензор $h \times w \times (tk)$ перетворюється на $\frac{h}{s} \times \frac{w}{s} \times (tk)$, де s – крок ядра (stride). Останнім шаром є точкова згортка з лінійною функцією активації, що знижує число каналів. Отримуючи вихідний тензор попереднього шару він повертає $\frac{h}{s} \times \frac{w}{s} \times k'$, де k' – кількість вихідних каналів блоку [19].

Результати експериментів цілком підтверджують гіпотезу, представлена авторами дослідження [19], яка полягає в тому, що відображення вхідного тензору високої розмірності, створене першими двома шарами, можна вкласти в простір меншої розмірності, без втрати важливої інформації. Крім того, автори дослідження описують високу ефективність поєднання архітектури SSD та згорткових блоків MobileNetV2, в задачах детектування об'єктів. Згідно проведених тестувань, така модель перевершує в точності, швидкості та розмірах, одну із найпопулярніших нейронних мереж YOLOv2 [19].

Таким чином, поєднання архітектур SSD та MobileNetV2 можна використати, як ефективну та легку модель нейронної мережі, для вирішення завдання детектування об'єктів, і зокрема локалізації стиків стічних труб.

2.2 Навчання нейромережі

При навчанні моделей штучного інтелекту перше, що необхідно зробити – підготувати навчальні дані. Процес модифікації навчальних даних, з метою розширення набору, шляхом додавання змінених оригінальних зображень називається аугментацією. Її часто використовують в умовах тренування моделі на обмеженій вибірці даних. Якщо модель була навчена, наприклад, тільки на

точному русі камери, в майбутньому вона не зможе правильно обробляти кадри, які не були точними. Для аугментації навчальної вибірки в задачах розпізнавання та детектування об’єктів, використовують різні зміни зображення, такі як оберти по горизонталі та вертикалі, шуми, розмиття, розтягування та вирізання окремих частин зображення. Але види модифікації зображення, для аугментації набору, потрібно підбирати обережно, проаналізувавши які з них будуть доречними для своєї задачі. Наприклад, вирізання випадкових частин може стати причиною втрати корисної інформації при навчанні моделі для класифікації та локалізації об’єктів.

Зазвичай дані діляться на навчальну і тестову вибірки. Тестовий набір прихований від мережі під час навчання. Навчальний і тестовий набори даних мають бути розділені так, щоб розподіл класів в обох наборах даних залишався відносно однаковим. Цього можна досягти за допомогою стратифікованого розбиття. Але, при оптимізації гіперпараметрів, тестового набору недостатньо, оскільки ми можемо втрачати узагальнення.

Ще один важливий етап підготовки даних – нормалізація. Ключова мета нормалізації – приведення різних даних в самих різних одиницях виміру і діапазонах значень до єдиного виду, який дозволить порівнювати їх між собою або використати для розрахунку схожість об'єктів. Використовуючи нормалізацію, ми можемо добитися спрощення ландшафту функції, що дає кращі результати, оскільки знайти мінімум простіше. Крім того, спрощуючи ландшафт функції, модель може навчатися швидше, оскільки їй не треба досліджувати різні локальні мінімуми [21]. Стандартизація часто використовується як метод нормалізації, оскільки вона виявилася дуже ефективною. Вона створює середовище, в якому кожна ознака має нульове середнє і дисперсію, рівну одиниці. Стандартизацію можна описати наступним чином:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2.1}$$

де x – початковий вектор ознак, $\bar{x}$ – середнє значення вектору ознак, а σ - стандартне відхилення.

Під час навчання моделі необхідно вимірювати її продуктивність. Функцію втрат при навчанні використовують для оцінки того, на скільки сильно передбачення відхиляється від оптимального рішення. Вона дає моделі інформацію про точність передбачень і як продовжити навчання. Враховуючи специфіку основного завдання моделі в даній роботі, обчислення втрат повинно складатися із двох частин: втрати локалізації для передбачення зміщення обмежуючої рамки (bounding box) та втрати класифікації об'єктів. Обидві частини можна обчислити як суму квадратів помилок. Середньоквадратична помилка є однією із найпоширеніших функцій втрат і може обчислюватись як:

$$L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{2.2}$$

де x – прогноз моделі, y – очікуваний прогноз. Функція втрат повинна бути мінімальною в тих випадках, коли точність передбачення моделі висока, і максимальною, коли точність низька.

Навчання потребує деякої форми оптимізації функції втрат. Оскільки метою є її мінімізація, гарним способом буде обчислення градієнту і подальший спуск по ньому. Градієнт обчислюється як похідна функції втрат L, по кожному параметру моделі θ. Градієнтний спуск для m навчальних прикладів можна записати як:

$$g = \Delta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \Delta_\theta L\left(x^i, y^i, \theta\right) \tag{2.3}$$

де $\Delta_\theta L\left(x^i, y^i, \theta\right)$ – обчислений градієнт для прогнозу x [21].

Функція активації нейронів – ще один важливий елемент процесу навчання моделі. Приховані шари нейронної мережі використовують їх для регулювання

ваги нейронів в процесі навчання. Ця функція описує залежність вхідного та вихідного сигналів нейрону. В сучасних нейронних мережах часто використовується функція активації ReLU (рис. 2.5), що описується як:

$$f(x) = \max(0, x) \tag{2.4}$$

Згортковий блок архітектури MobileNetV2 використовує її модифікацію – ReLU6 (рис. 2.6), яка виглядає наступним чином:

$$f(x) = \min(\max(0, x), 6) \tag{2.5}$$

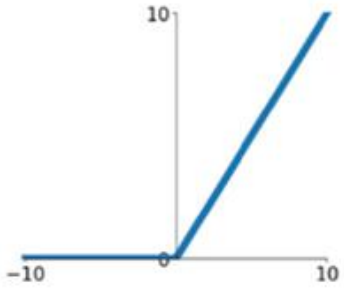


Рисунок 2.5 – Графік функції активації ReLU.

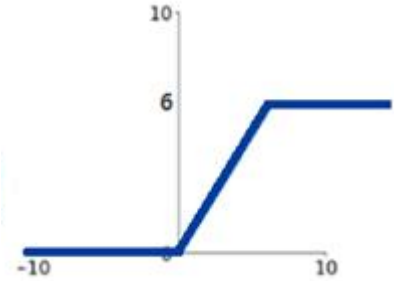


Рисунок 2.6 – Графік функції активації ReLU6.

Модель може мати безліч параметрів. У машинному навчанні параметри, які задаються до навчання моделі, називаються гіперпараметрами. Це може бути швидкість навчання градієнтного спуску, різні методи, що також називаються оптимізаторами, кількість згорткових, об'єднуючих і повнозв'язаних шарів, кількість нейронів в повнозв'язаному шарі та інші. Модель не може змінити ці параметри, хоча вони можуть бути встановлені автоматично за допомогою таких методів, як пошук по сітці (Grid Search). Даний метод можна використовувати,

коли ми точно не знаємо, як встановити гіперпараметр, але у нас є підозра, що він може знаходитися на якомусь інтервалі. Далі навчання проводиться кілька разів, кожного разу з різними параметрами. Це можна зробити навіть для їх комбінації, але час дослідження росте експоненційно, тому що кожного разу, коли ми хочемо автоматично знайти інше значення, ми додаємо ще один вимір. Більшість гіперпараметрів мають значення за замовчуванням. Інший метод, запропонований в [21], полягає в тому, щоб задати простір точно так, як і в пошуку по сітці, але шукати випадковим чином. Пошук по сітці може пропустити деякі значення, які може знайти випадковий пошук.

Архітектура MobileNet має два гіперпараметри: α – множник ширини, ρ – множник глибини. Перший відповідає за кількість каналів у кожному шарі. Другий – за розміри вхідних тензорів, тобто, наприклад, значення параметра $\rho = 0.25$ призводить до зменшення висоти та ширини карти ознак в 4 рази. Ці параметри дозволяють контролювати розміри мережі. Зменшуючи їх значення, ми знижуємо точність розпізнавання, але при цьому збільшується швидкість роботи і оптимізується обсяг витрат пам'яті.

Отже, навчання нейронної мережі для детектування об’єктів складається з декількох етапів, а саме: підготовка набору зображень, що включає розмітку, аугментацію і розділення на тестову та навчальну вибірки, вибір функції втрат та методу їх оптимізації, визначення оптимальних функцій активації та гіперпараметрів мережі. Для поєднання архітектури SSD та згорткового блоку MobileNet ефективною конфігурацією навчання є обчислення втрат локалізації та класифікації з використанням методу градієнтного спуску.

2.3 Критерії валідації навчених моделей

Моделі можуть бути оцінені за допомогою різних методів. Точність використовується для категоріальних та бінарних проблем, і може бути обчислена як:

$$acc = \frac{n_c}{n} \tag{2.6}$$

де n_c – кількість правильно класифікованих зображень, а n – кількість усіх зображень. Іноді модель теж не може бути однозначною, тому в якості методу оцінки можна використати перші k елементів. У разі великих наборів даних і класів розпізнавання, правильна відповідь може бути на k-му місці. Хоча k не має бути великим числом по відношенню до кількості класів, це знецінило б результати і давало б майже 100% результат кожного разу [21].

Однією з мір точності моделі є F-міра (F-score), яка обчислюється через влучність і повноту (рис. 1.8).

Рисунок 2.7 – Візуалізація понять влучності і повноти.

Влучність (precision) – це доля релевантних знайдених зразків, про які повідомила модель, або математично:

$$p = \frac{t_p}{t_p - f_p} \tag{2.7}$$

де t_p – істинно позитивні, а f_p – псевдопозитивні.

Повнота (recall) – доля t_p, яку було знайдено:

$$r = \frac{t_p}{t_p - f_n} \tag{2.8}$$

де f_n – всі релевантні елементи.

Об'єднуючи влучність і повноту, можна обчислити F-міру[21]:

$$F = 2 \times \frac{p \times r}{\rho + r} \tag{2.9}$$

Для валідації моделей класифікації об'єктів поширено використання матриці невідповідностей [15], що дозволяє візуально оцінити результати прогнозів мережі. Рядки матриці описують всі екземпляри класів, а стовбці – прогнозовані. На рис. 2.8 показано приклад такої матриці невідповідностей для 10 класів розпізнавання

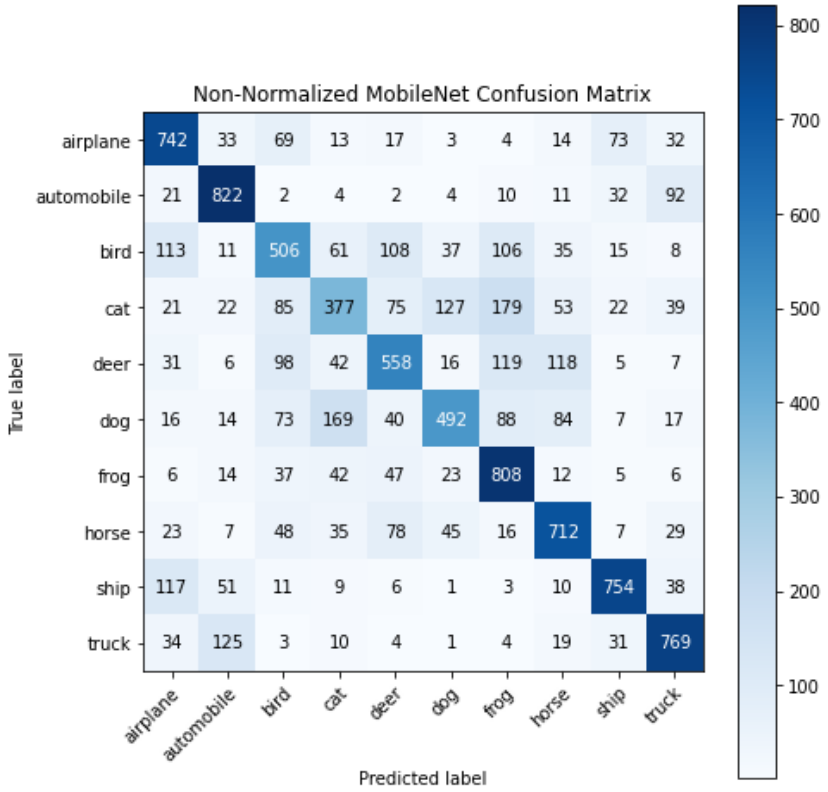


Рисунок 2.8 – Матриця невідповідностей.

Елементи на діагоналі нормалізованої матриці невідповідностей, зображеної на рис. 2.9 – це значення повноти (recall), а інші – помилкові прогнози. Використовуючи матрицю невідповідностей можна обчислити влучність та повноту.

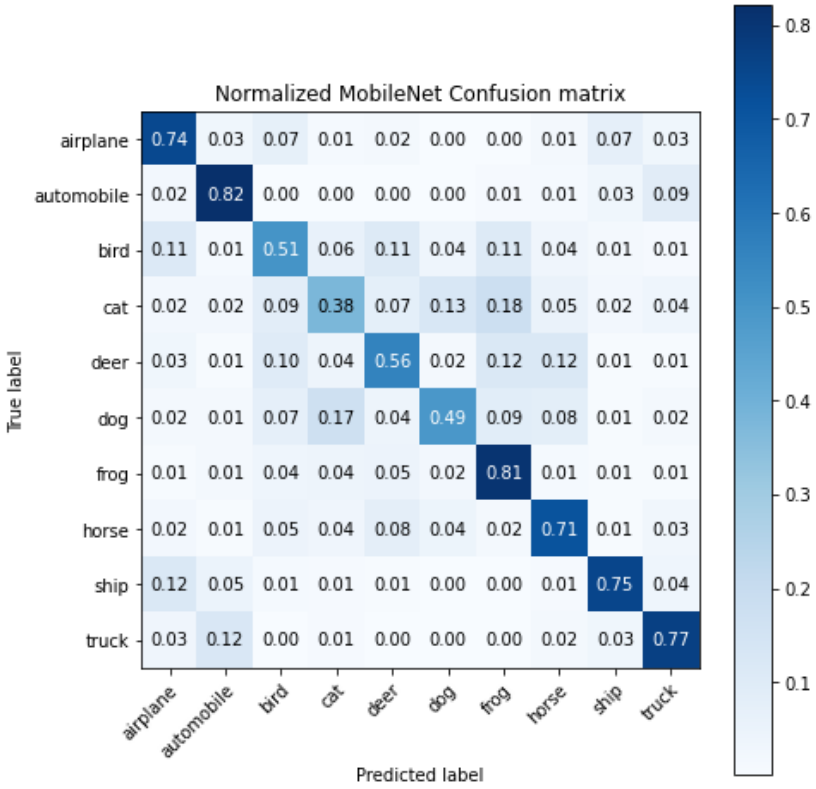


Рисунок 2.9 – Нормалізована матриця невідповідностей.

Таким чином, серед великої кількості метрик валідації моделі, однією з найбільш інформативних є оцінка прогнозів з тестової вибірки через влучність та повноту. Для візуалізації результатів прогнозування в задачі класифікації, можна використати матрицю невідповідностей.

3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ФУНКЦІОНАЛЬНОГО СТАНУ СТІЧНИХ ТРУБ

3.1 Формування навчальних та тестових даних

Для навчання моделі сформовано набір даних, зібраних з відеоінспекцій, що складається з 3300 зображень стиків стічних труб розміром 640x640 пікселів (рис. 3.1).

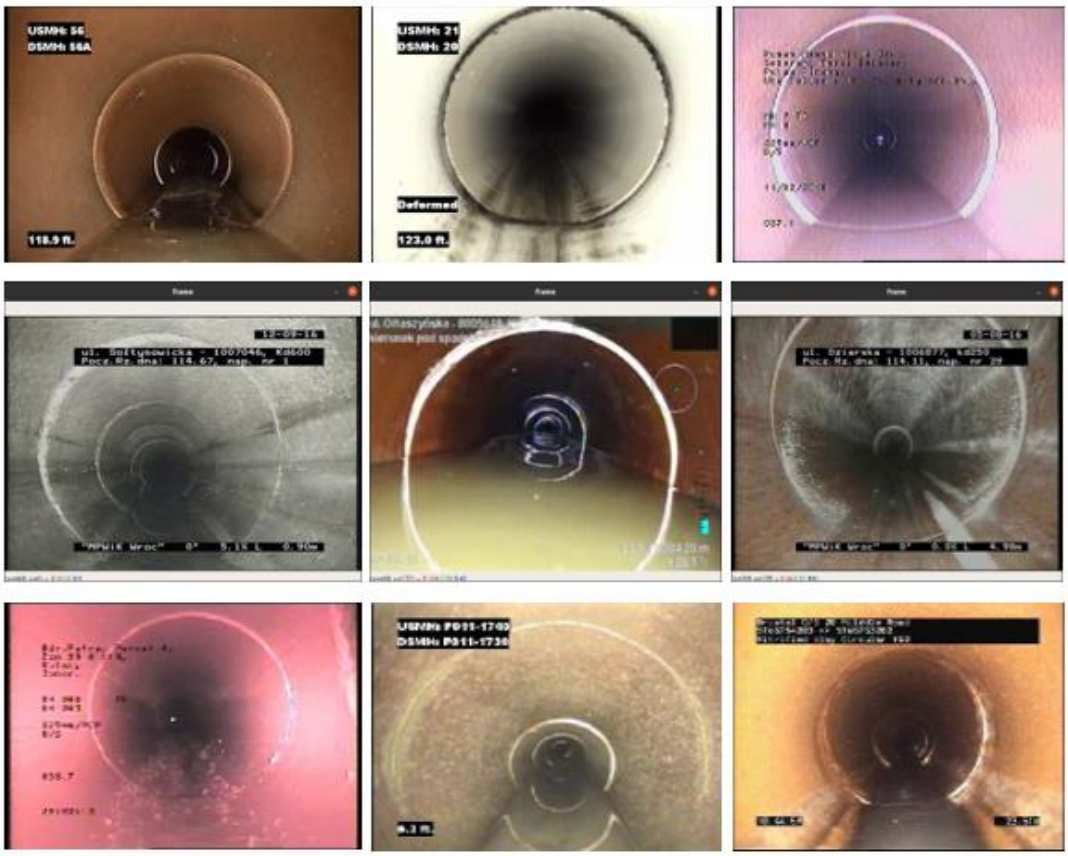


Рисунок 3.1 – Приклад зображень із використаного набору.

Дані розмічені вручну, за допомогою програми labelimg. Анотації зроблені у форматі Pascal VOC XML. Специфіка задачі потребує виділення найближчих до камери помітних стиків, але для збільшення обсягу корисної інформації також помічаються деякі наступні. Крім того, на зображеннях, де спостерігається підвищений рівень води в трубі, виділяється тільки видима частина місця з’єднання. Приклад розмітки представлено на рис. 3.2

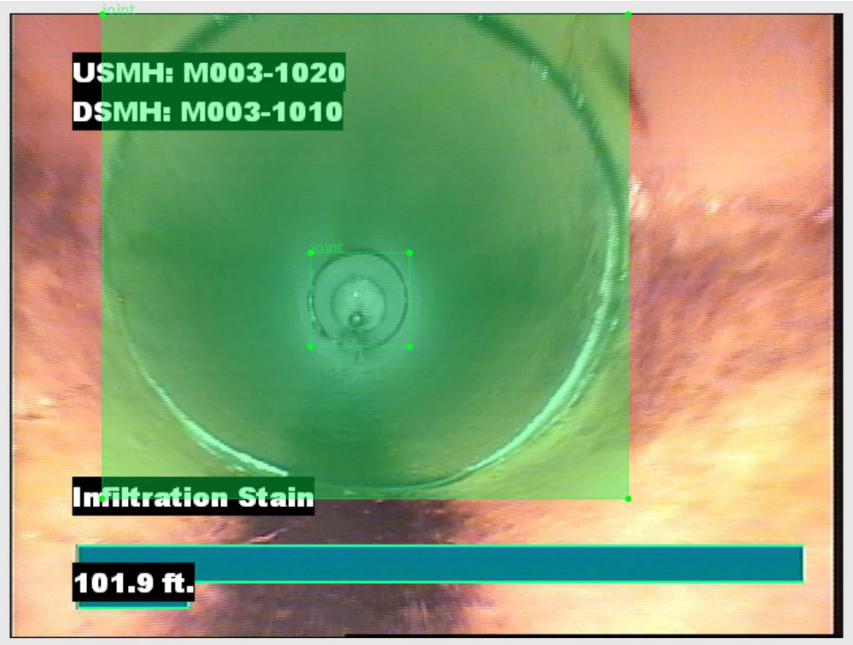


Рисунок 3.2 – Приклад ручної анотації зображення.

Для аугментації даних використані наступні перетворення оригінальних зображень: оберти по вертикалі та горизонталі, невеликі повороти та здвиги на 10-15%, розмиття, накладання областей гаусового шуму випадкового розміру, шумів типу «сіль та перець». Приклади використаних маніпуляцій із зображеннями для розширення набору можна побачити на рис. 3.3.

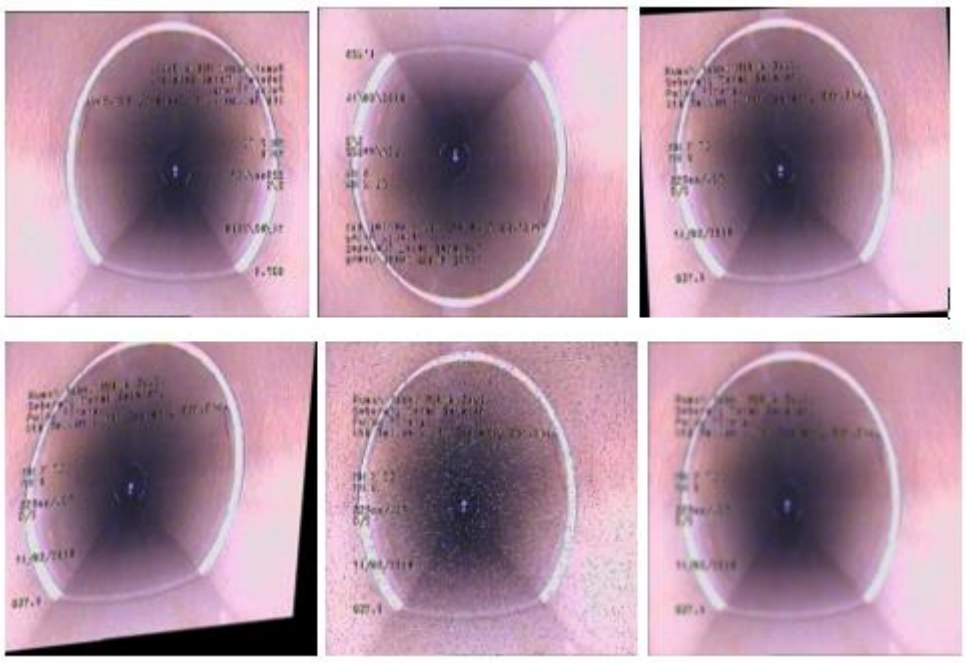


Рисунок 3.3 — Приклад перетворень зображення.

Описані перетворення проводяться з вірогідністю 50% на кожному зображенні, адже для навчання моделі необхідно використати якомога більше різноманітних даних, щоб мережа навчилась виділяти потрібні для прогнозування ознаки. Для розширення набору використовується модуль аугментації Tensorflow Object Detection API, за допомогою якого ображення змінюється разом із обмежуючими рамками розмітки.

Останнім етапом перед навчанням моделі є розділення даних на тестовий та навчальний набори. В результаті поділу, 75% зображень будуть використані для навчання, а інші 25% виступатимуть у якості вибірки для оцінки точності мережі.

3.2 Результати машинного навчання нейромережі

Для класифікації та локалізації стиків стічних труб, було обрано модифікацію архітектури згорткової нейронної мережі SSD MobileNetV2. Навчання нейромережі проводилось з використанням фреймворку TensorFlow Object Detection API. Файл конфігурації моделі знаходиться в додатку А.

При навчанні моделі використовуються функції втрат локалізації та класифікації, що обчислюються як середньоквадратична похибка, та градієнтний спуск, як алгоритм оптимізації. Коефіцієнт швидкості навчання (learning rate) на першому кроці становить 0.0266 та лінійно збільшується через 1000 кроків до показника 0.0799. Цей метод в навчанні називають warm-up, він використовується для зменшення ефекту перших навчальних прикладів. Обраний розмір одного пакету зображень (batch) становить 32.

Під кроком розуміється одна операція по оновленню ваг моделі. Вхідні дані обробляються пакетами зображень, для кожного з яких ваги оновлюються один раз. Етап навчання, коли вхід охоплює увесь набір зображень, називається однією епохою. Кількість пройдених кроків в епоху пов'язана з розміром набору даних і розміром пакету.

Після 25000 кроків навчання моделі, можемо отримати графіки зміни функцій втрат локалізації (рис. 3.4), класифікації (рис. 3.5) та загальних втрат на навчальному і тестовому наборах даних.

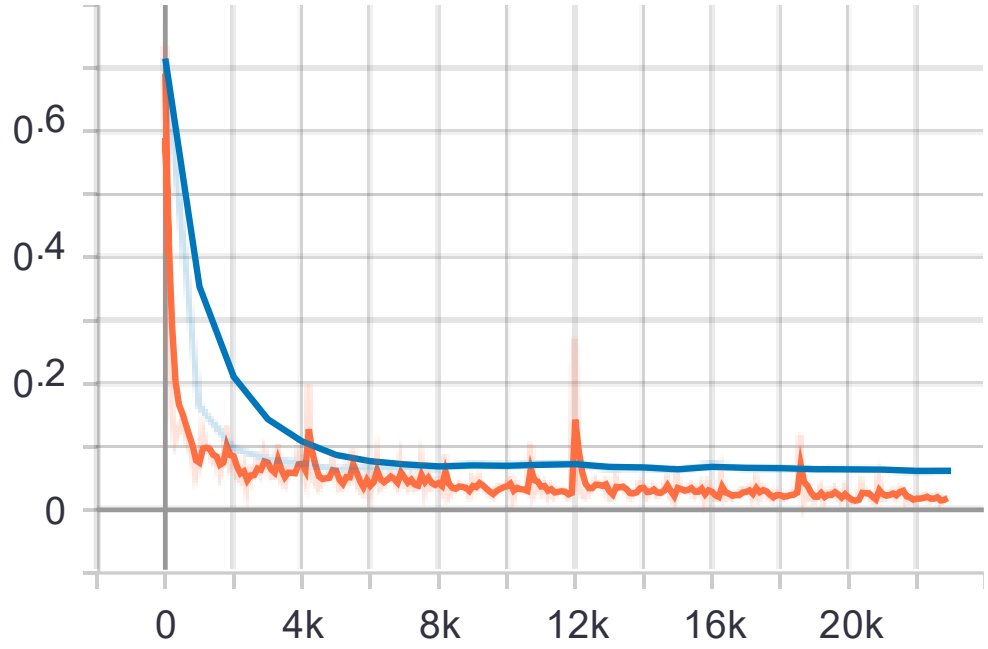


Рисунок 3.4 – Графік залежності зміни втрат локалізації від кількості кроків, де крива синього кольору відповідає за втрати на тестовому наборі

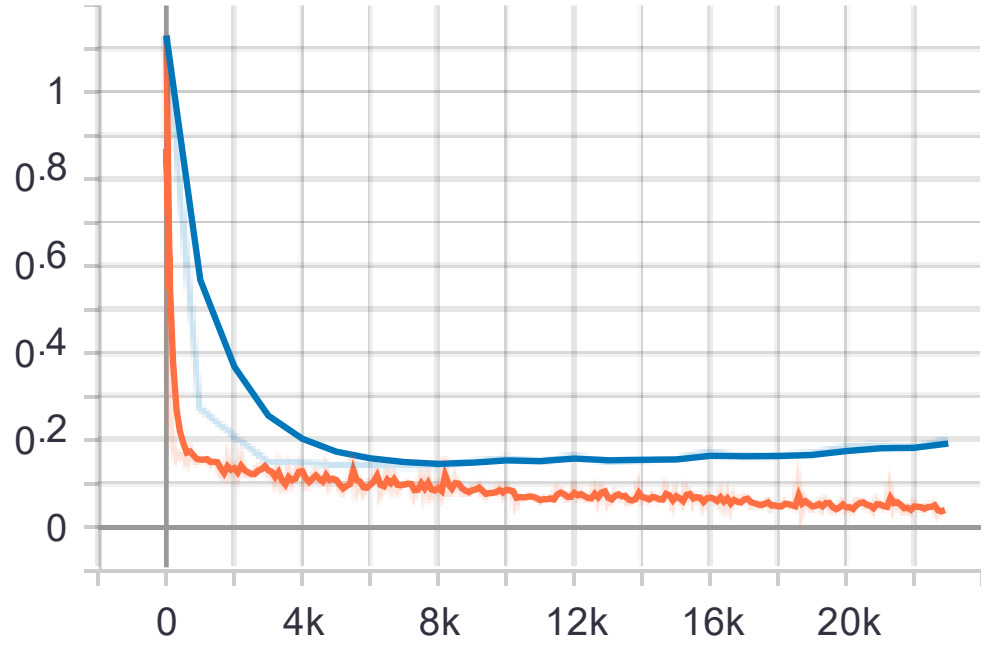


Рисунок 3.5 – Графік залежності зміни функції втрат класифікації від кількості кроків.

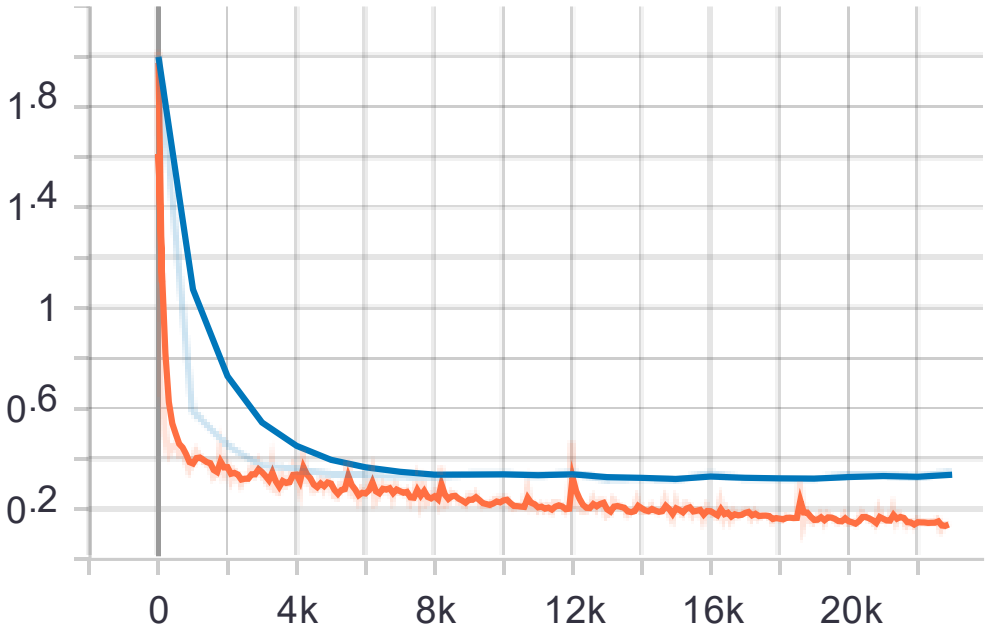


Рисунок 3.6 – Графік залежності зміни загальних втрат від кількості кроків.

Як показує аналіз рис. 3.6, зменшення функції втрат значно сповільнюється на 10000 кроці, при цьому, відповідно до графіку на рис. 3.7, точність навченої моделі становить 90%

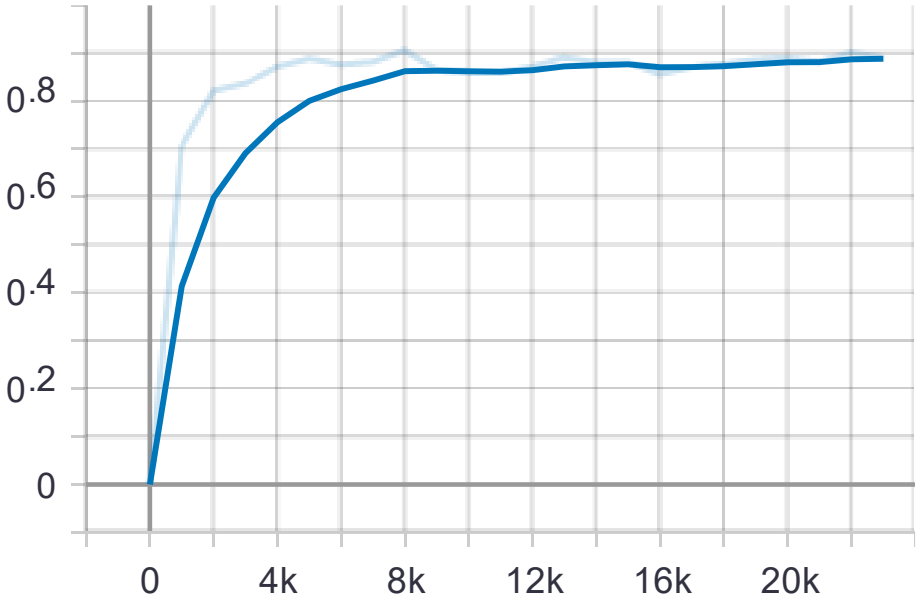


Рисунок 3.7 – Графік залежності зміни точності від кількості кроків для тестового набору в процесі навчання моделі.

На рис. 3.8 показано приклад послідовного покращення результатів прогнозування моделі для тестової вибірки даних.

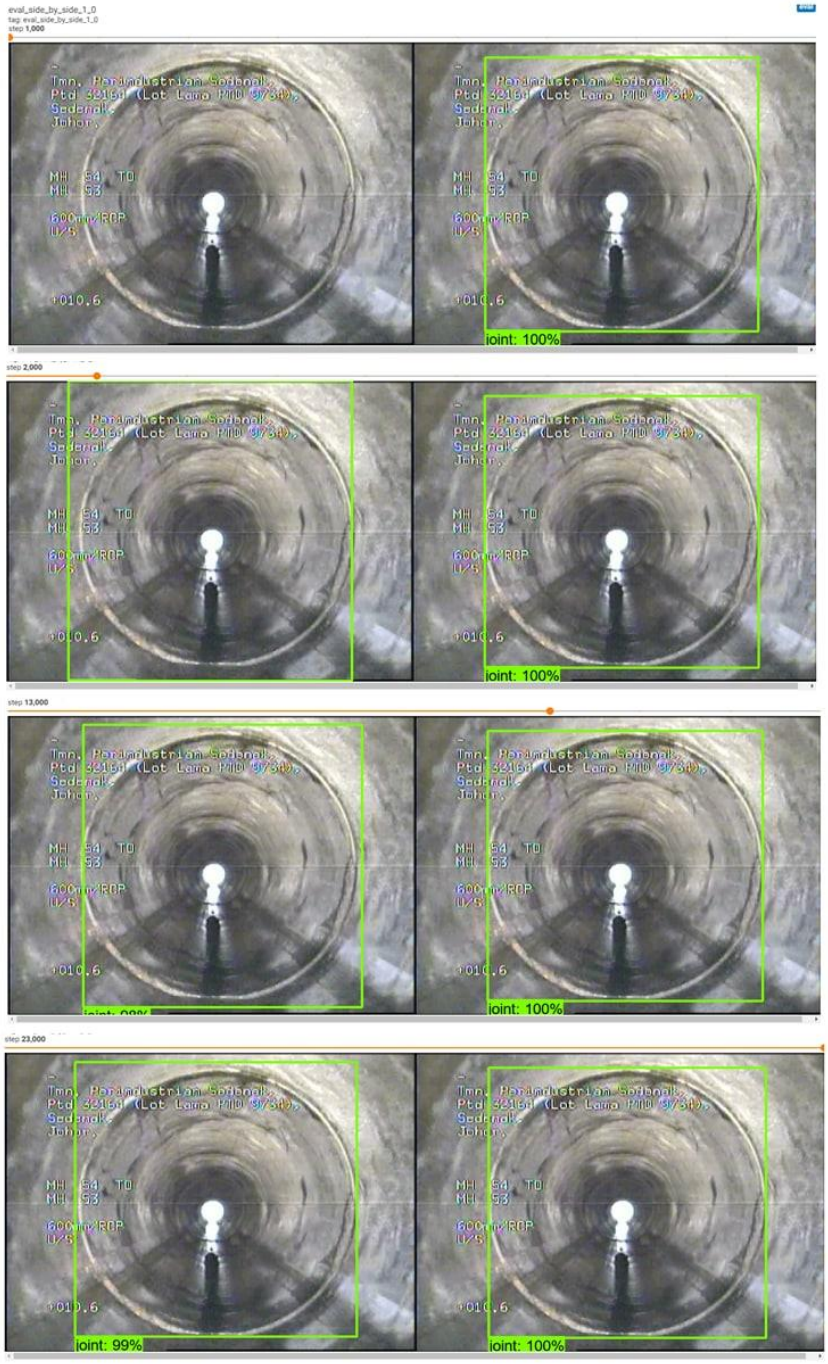


Рисунок 3.8 – Приклад послідовного покращення прогнозів моделі, праворуч знаходиться розмічене зображення із тестового набору, ліворуч – прогноз моделі на обраному кроці навчання.

Отже, згорткова нейронна мережа, побудована на архітектурі SSD MobileNetV2, в результаті навчання протягом 25000 кроків має точність 90%, тому може бути використана як достатньо ефективна модель детектування стиків труб.

3.3 Опис алгоритму роботи інтелектуальної системи

Для формування звіту системі детектування з'єднань труб, окрім відеофайлу інспекції, потрібна інформація про направлення об'єктиву камери та показники одометра. Щоб отримати ці дані система використовує дві додаткові моделі класифікації зображень. Алгоритм створеної інтелектуальної системи складається із декількох основних етапів обробки кадрів вхідного відео, які показані на рис. 3.9.

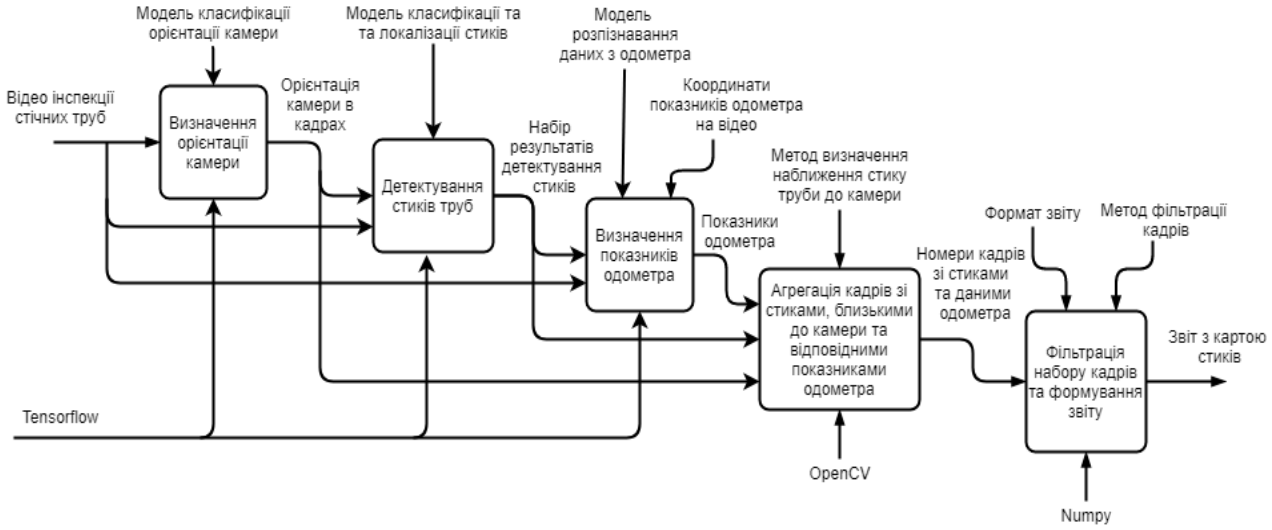


Рисунок 3.9 – Схема інтелектуальної системи детектування стиків труб

Перший етап – визначення орієнтації камери в трубі. Із всіх можливих варіантів орієнтації, для визначення локації стику потрібно розглядати кадри, на яких камера направлена вздовж труби, тобто дивиться прямо в центр, або частково відхиляється від центру вліво, вправо, вгору чи вниз, оскільки тільки в цих положеннях на зображенні можна помітити та вірно класифікувати стик.

Наступний крок конвеєра обробки зображень отриманих з відео, полягає в отриманні результатів прогнозування моделі класифікації та локалізації стиків труб. Показники одометра відомі для з'єднань, що знаходься близько до об'єктиву камери. Тому кадри, на яких обмежуючі рамки об'єкту, створені моделлю детектування, входять в область 10% від країв зображення, співвідносяться з пройденою під час інспекції відстанню.

На відео інспекції стічних труб вказується значення одометру для кожного кадру. Відстань пройдена камерою береться із зображення за допомогою моделі розпізнавання цифр та символів. Для цього потрібно визначити координати лічильника пройденої дистанції на зображенні. Враховуючи, що його положення не змінюється протягом всієї інспекції, перед початком обробки відео, локація одометра в кадрі визначається вручну (рис. 3.10).

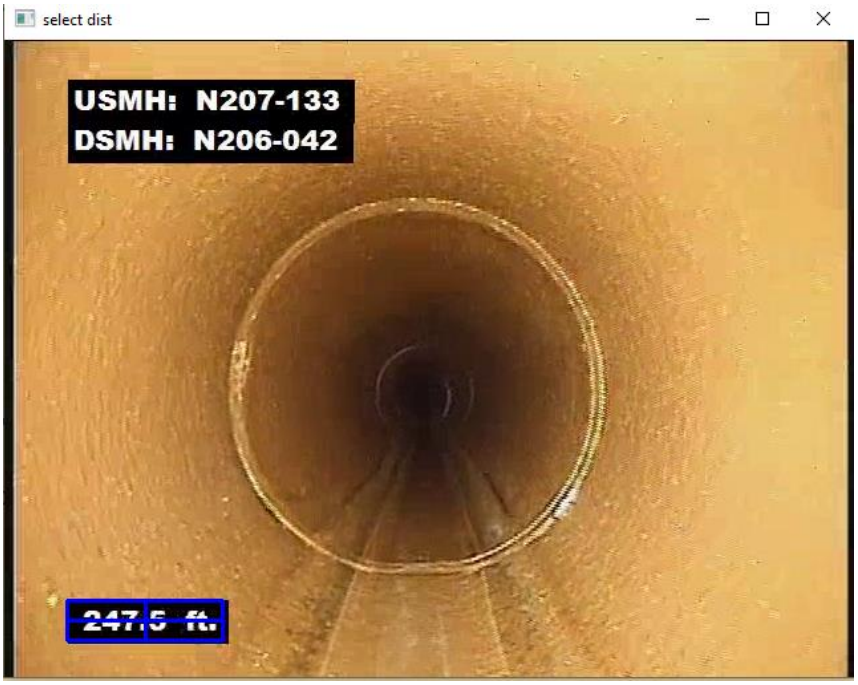


Рисунок 3.10 – Позначення координат одометра на випадковому кадрі відеоінспекції

Отриманий набір прогнозів необхідно фільтрувати, адже частина кадрів, на яких системою були знайдені близькі до об’єктиву камери стики труб, мають однакове або близьке значення показників одометра. Для цього прогнози, які мають близькі та однакові відстані, групуються та вважаються даними одного і того ж стику. Із отриманих груп обираються максимальні показники одометра, як найбільш близькі до камери.

Останнім етапом алгоритму є формування звіту інспекції у вигляді таблиці з номерами кадрів, де було знайдено стики труб, мітками часу і відповідними показниками одометра (рис. 3.11).

Крім того, система створює відео файл з візуалізацією результатів прогнозування використаних моделей та алгоритму пошуку близьких до камери стиків (рис. 3.12).

frame	time	odom value
55	1.83 s	8 ft.
387	12.9 s	13.8 ft.
694	23.13 s	19 ft.
1018	33.93 s	24.5 ft.
2191	1:13 m	30 ft.
2855	1:24 m	35.2 ft.
2869	1:35 m	40.5 ft.
3196	1:46 m	45.9 ft.
3531	1:57 m	51.2 ft.

Рисунок 3.11 – Приклад звіту системи локалізації стиків стічних труб

Рисунок 3.12 – Приклад візуалізації прогнозів використаних нейронних мереж

Для програмної реалізації системи використовувалась мова програмування Python версії 3.8 та середовище розробки PyCharm. Програмний код знаходиться в додатку А. Основні класи програмы наведено в таблиці 3.1.

Обробка відеофайлів здійснюється за допомогою бібліотек OpenCV та NumPy. Зчитані дані у вигляді тензорних масивів аналізуються нейромережами використовуючи бібліотеку TensorFlow. Список всіх використаних бібліотек описано в таблиці 3.2.

Таблиця 3.1 – Основні класи програми

Клас	Опис
AIModel	Абстрактний клас, що описує сигнатури методів для роботи з моделями машинного навчання.
AIPredictor	Клас, об’єкт якого використовується для роботи з масивом класів моделей. Забезпечує обробку наборів вхідних даних та агрегацію прогнозів для всіх нейромереж, класи яких були передані при ініціалізації об’єкту.
JointsDetector	Клас-нащадок класу AIModel, екземпляр якого використовується для завантаження моделі детектування стиків стічних труб, обробки вхідних даних та набору прогнозів.
ContextClassifier	Клас-нащадок AIModel, екземпляр якого забезпечує завантаження моделі класифікації орієнтації камери, обробки вхідних даних та набору прогнозів.
OdometerTextRecognizer	Клас-нащадок AIModel, використовується для завантаження моделі розпізнавання даних одометра, обробки вхідних даних та набору результатів розпізнавання.

Таблиця 3.2 – Опис використаних бібліотек та фреймворків

Назва	Опис
OpenCV	OpenCV – бібліотека комп'ютерного зору з відкритим кодом під ліцензією BSD. Вона містить інтерфейси Python, C++, Java і має високу обчислювальну ефективність. Містить методи обробки зображень, відео та інші чисельні алгоритми
NumPy	Бібліотека розширення мови Python. Додає підтримку матриць, багатовимірних масивів, та колекцію математичних функцій для роботи з ними.
TensorFlow	Безкоштовна бібліотека з відкритим кодом для машинного навчання. Може використовуватись в різних задачах, але особлива увага приділяється навчанню глибоких нейронних мереже. Алгоритми бібліотеки здатні працювати на різних платформах(CPU, GPU, TPU). TensorFlow створено командою Google Brain з організації Google AI, підтримує мови програмування C++, python, java script, java, і повністю підтримується в Google Colab.
TensorBoard	Набір інструментів для візуалізації та експериментів з машинним навчанням. Він має наступні можливості: – візуалізація показників навчання моделі, таких як точність і втрати; – візуалізація шарів моделі; – виведення зображень; – створення гістограм ваг, зміщень та інших тензорів; – проеціювання вкладень у простір меншої розмірності;

Продовження таблиці 3.2

Назва	Опис
TensorFlow Object Detection API	Фреймворк з відкритим кодом, побудований на TensorFlow, який спрощує створення, навчання і розгортання моделей машинного навчання, здатних локалізувати та ідентифікувати кілька об'єктів на одному зображенні.
ABC	Стандартна бібліотека, що надає інфраструктуру для визначення абстрактних класів в мові Python
pandas	Бібліотека для аналізу та маніпулювання даними з відкритим кодом під ліцензією BSD. Включає структури даних та методи для роботи з часовими рядами і таблицями

Отже, розроблена система дозволяє визначати відстані стиків стічних труб, шляхом об’єднання прогнозів набору моделей машинного навчання, що в майбутньому дозволяє використати описаний алгоритм для створення повнофункціональної інтелектуальної системи оцінки функціонального стану каналізаційної чи дренажної мережі трубопроводів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було проаналізовано сучасні методи інспекції каналізаційних систем. Підтверджено важливість та актуальність автоматизації оцінки функціонального стану стічних труб. Аналіз літератури показав, що стики труб є найбільш уразливими елементами мереж трубопроводів, яким необхідно приділяти особливу увагу під час інспекції.

Проведено огляд моделей та методів машинного навчання для обробки візуальних даних. Обрано оптимальну архітектуру багатошарової згорткової нейронної мережі для дослідження її ефективності в задачі детектування стиків труб на відеоданих інспекції. Створено набір навчальних та тестових даних, на яких проведено навчання та оцінку моделі. Прогнози отриманої нейронної мережі мають достатню точність для створення системи класифікації та локалізації стиків стічних труб.

Створено інтелектуальну систему, що визначає положення місць з'єднання труб, відносно даних лічильника пройденого камерою шляху на відеоданих інспекції. Для цього використовується об'єднання прогнозів моделей класифікації орієнтації камери та визначення показників одометра. В результаті створюються звіти в форматі відео і таблиці з мітками відстані та часу. В поєднанні з моделлю класифікації дефектів, реалізований алгоритм можна використати для створення повнофункціональної інтелектуальної системи оцінки функціонального стану стічних труб.

СПИСОК ЛІТЕРАТУРИ

1. X. Yin, Y. Chen, A. Bouferguene "A deep learning-based framework for an automated defect detection system for sewer pipes", Automation in Construction, vol. 109, no. 102967, pp. 1-16, 2020.
2. J. C.P. Cheng, M. Wang, "Automated detection of sewer pipe defects in closed-circuit television images using deep learning techniques", Automation in Construction, vol. 95, pp. 155-171, 2018. doi:10.1016/j.autcon.2018.08.006.
3. NASSCO, Pipeline Assessment and Certification Program (PACP) v.7, 2016.
4. J. Haurum, B. Bahnsen, C. H. Pedersen, T. B. Moeslund, "Water Level Estimation in Sewer Pipes Using Deep Convolutional Neural Networks", Water, vol. 12, no. 3412, pp. 2-14, 2020, doi:10.3390/w12123412
5. S. Moradi, T. Zayed, F. Golkhoo, "Review on Computer Aided Sewer Pipeline Defect Detection and Condition Assessment" Infrastructures, vol.4, no.10, pp. 1-13, 2019.
6. Y. Bengio, A. Courville, P. Vincent, "Representation Learning: A Review and New Perspectives", IEEE, vol. 35, pp 1799-1820, 2014.
7. Y. Bengio, M. Monperrus, "Non-Local Manifold Tangent Learning", Neural Information and Processing Systems. Downtown Branch, Montreal, pp.129-136, 2004.
8. B. Suzanna, H. Geoffrey, "A self-organizing neural network that discovers surfaces in random-dot stereograms", Nature, vol. 355, pp. 161-163, 1992.
9. L. Wiskott, T. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances", Neural Computation, vol. 14, pp. 715-770, 2002.
10. H. Mobahi, R. Collobert, J. Weston, Deep learning from temporal coherence in video, ICML, 2009.
11. G. Hinton, A. Krizhevsky, S. Wang, Transforming autoencoders, IICANN, 2011.

12. F. Bach, R. Jenatton, J. Mairal, G. Obozinski, Structured sparsity through convex optimization, CoRR, 2011.
13. X. Chen, C. Gong, C. Ma, X. Huang, J. Yang, “Privileged Semi-Supervised Learning”, IEEE, 2018, doi:10.1109/icip.2018.8451098
14. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, “Natural language processing (almost) from scratch”, Journal of Machine Learning Research, vol.12, pp. 2493–2537, 2011.
15. C. Davide, Siamese neural networks: an overview, Artificial Neural Networks. New York City, 2020, pp. 73-94.
16. Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, M. Lew, “Deep learning for visual understanding: A review”, Neurocomputing, vol.187, pp. 27–48, 2015. doi:10.1016/j.neucom.2015.09.116.
17. A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi, “A Survey, of the Recent Architectures of Deep Convolutional Neural Networks”, 2019, pp. 1-62. doi: https://doi.org/10.1007/s10462-020-09825-6.
18. Z. Zhao, P. Zheng, S. Xu, X. Wu, “Object Detection With Deep Learning: A Review”, IEEE, vol. 30, no. 11, 2019, doi: 10.1109/TNNLS.2018.2876865.
19. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks”, IEEE/CVF, Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT, USA: IEEE, 2018. pp. 4510-4516. doi: 10.1109/CVPR.2018.00474
20. H. Kaiming, Z. Xiangyu, R. Shaoqing, S. Jian, “Deep Residual Learning for Image Recognition”, Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA: IEEE, 2016. pp. 778.
21. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, The MIT Press. 2016, pp. 326-366. URL: https://www.deeplearningbook.org/

ДОДАТОК А

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
```

```
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 128
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
    share_prediction_tower: true
    use_depthwise: true
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
    use_static_shapes: false
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
```

```
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint:                          "Tensorflow/workspace/pre-trained-
models/ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8/checkpoint/ckpt-0"
  fine_tune_checkpoint_type: "detection"
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_vertical_flip {
    }
  }
  data_augmentation_options {
    random_rotation90  {
    }
  }
  data_augmentation_options {
      random_patch_gaussian {
      min_patch_size: 1
      max_patch_size: 250
      min_gaussian_stddev: 0.0
      max_gaussian_stddev: 0.4
      random_coef: 0.5
    }
  }
  data_augmentation_options {
    ssd_random_crop{
    }
  }
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.07999999821186066
          total_steps: 50000
          warmup_learning_rate: 0.026666000485420227
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
```

```
  sync_replicas: true
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  num_steps: 50000
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}
train_input_reader {
  label_map_path: "Tensorflow/workspace/annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "Tensorflow/workspace/annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "Tensorflow/workspace/annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "Tensorflow/workspace/annotations/test.record"
  }
}
```

ДОДАТОК Б

```
import cv2
import random as rnd
import numpy as np
import os
import tqdm
import glob
import tensorflow as tf
from pipe_ai.pipe_processing.ai_model import AIModel
from pipe_ai.pipe_processing.ai_predictor import AIPredictor
from pipe_ai.pipe_processing.odometer.pipe_odom_recognizer import
OdometerTextRecognizer
from pipe_ai.pipe_processing.context.context_classificatory import
ContextClassifier
from pipe_ai.pipe_processing.joints.joints_detector import JointsDetector
from pipe_ai.pipe_utils.utils import manual_select_odom_roi, write_to_json_file,
read_from_json_file, create_out_table
from object_detection import export_inference_graph as exporter
from object_detection.utils import visualization_utils as viz_utils
import pipe_ai.auto_annotation.auto_annotation as aan
import pipe_ai.frames_extractor as fe
from datetime import datetime


def get_roi(video_path=None):
    if video_path is None:
        return [48, 423, 113, 450]
    return manual_select_odom_roi(video_path)


def get_prediction_result(predictor, images, odom_roi=None):
    image_np = np.array(images)
    odom_roi = np.array(odom_roi, dtype=np.int)
    return predictor.predict(image_np, odometer_frame_pos=odom_roi)


def draw_prediction(image, detections, id_offset, category_index, max_boxes,
threshold):
    out = viz_utils.visualize_boxes_and_labels_on_image_array(
        image,
        np.squeeze(np.array(detections['detection_boxes'])),
        np.squeeze(np.array(detections['detection_classes']) + id_offset),
        np.squeeze(np.array(detections['detection_scores'])),
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=max_boxes,
        min_score_thresh=threshold,
        agnostic_mode=False)
    img_width, img_height, c = image.shape
    font = cv2.FONT_HERSHEY_PLAIN
    odom_value_string =
'Odom_value:{}'.format(detections['distance_field']['label'])
    (odom_text_width, odom_text_height) = cv2.getTextSize(odom_value_string,
font, 1, 2)[0]
    cv2.putText(out, odom_value_string, (img_width - odom_text_width,
odom_text_height),
                font, 1, (0, 255, 0), 2, cv2.LINE_AA)
    context = detections['context_field']['label']
    cv2.putText(out, 'Context: ' + context, (img_width - odom_text_width, 3 *
```

```
odom_text_height),
                font, 1, (0, 255, 0), 2, cv2.LINE_AA)
    return out


video_path = 'path\\to\\video'
extracted_frames_path = 'pipe_ai\\auto_annotation\\out\\'
label_id_offset = 0
threshold = 0.7
category_index = {1: {'id': 1, 'name': 'joint'}}
roi = get_roi(video_path)

# clear image folder
old_images = os.listdir(extracted_frames_path)
for f in old_images:
    os.remove(os.path.join(extracted_frames_path, f))

# frames extraction
frames = fe.extract(video_path, 3 * 60 + 47, 1, 2 * 60, extracted_frames_path)
frames_batches = np.array_split(frames, len(frames) / 10)
print("Wait GPU...")

# prediction
ai_models = [
    JointsDetector(),
    OdometerTextRecognizer(),
    ContextClassifier()
]
_predictor = AIPredictor(ai_models)
predictions = []
out_images = []
with tqdm.tqdm(total=len(frames_batches), desc="Batches predicted") as pbar:
    for frames_batch in frames_batches:
        results = get_prediction_result(_predictor, frames_batch, roi)
        for i in range(len(results)):
            out_image = draw_prediction(frames_batch[i], results[i],
label_id_offset, category_index, 1, threshold)
            out_images.append(out_image)
            predictions.append(results[i])
        pbar.update(1)

# writing to json
result_path = 'result.json'
write_to_json_file(predictions, result_path)

# read json
# predictions_json = read_from_json_file(result_path)

detected_joints = {'frame_index': [], 'odom_value': []}

def main(predictions):
    images = glob.glob(extracted_frames_path + '*.jpg')
    img = cv2.imread(images[0])
    h, w, c = img.shape
    print(w, h, c)
    video = cv2.VideoWriter('output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 30.0,
(w, h))
    images.sort(key=os.path.getmtime)
    i_offset = 0
    last_joint = 0
```

```
    for i, path in enumerate(images[i_offset:]):
        img = np.array(cv2.imread(path))
        detection = predictions[i + i_offset]
        out = draw_prediction(img, detection, label_id_offset, category_index,
1, threshold)
        boxes = detection['detection_boxes']
        scores = detection['detection_scores']
        odom = detection['distance_field']['label']
        context = detection['context_field']['label']
        box = aan.get_box(boxes, scores)

        font = cv2.FONT_HERSHEY_PLAIN
        if box.size:
            aan.write(path, detection, category_index, threshold)
            if context == 'semi_left' or context == 'forward' or context ==
'semi_right' or context == 'semi_up' or context == 'semi_down':
                x1, y1, x2, y2 = aan.normalize_box(box, w, h)
                pw = w * 0.10
                ph = h * 0.10
                if (x1 < pw and x2 > w - pw) or (y1 < ph and y2 > h - ph):
                    detected_joints['frame_index'].append(i)
                    detected_joints['odom_value'].append(odom)
                    # print('joint on', odom)
                    last_joint = odom
        cv2.putText(out, 'last joint: ' + str(last_joint), (w - 300, h), font,
2, (0, 255, 0), 2, cv2.LINE_AA)
        video.write(out)
    video.release()

main(predictions)
create_out_table(detected_joints)
# main(predictions_json)
```

```
import abc
from numpy import argmax
import tensorflow as tf
from pipe_ai.pipe_utils.models_prep import load_labels_pbtxt, load_param


class AIModel(object):

    def __init__(self, model_description_path, model_name, frame_field,
thresholds=None):
        __metaclass__ = abc.ABCMeta
        self.__model_description = load_param(model_description_path)

        self.__model_name = model_name
        self.__frame_field = frame_field

        self.__weights_path = self.__model_description['weights_path']

        if 'model_config' in self.__model_description:
            self.__model_config = self.__model_description['model_config']
        else:
            self.__model_config = None

        self.__shape = self.__model_description['model_shape']

        if self.__model_description['model_labels'] is not None:
            self.__labels =
load_labels_pbtxt(self.__model_description['model_labels'])

        if thresholds is not None:
            self.__thresholds = thresholds
        else:
            self.__thresholds = self.__model_description['model_thresholds']

        self.__input_tensor_name = self.__model_description['model_input']

        self.__output_tensors_names = self.__model_description['model_output']

        self.__graph = self.load_graph()

    @property
    def model_type(self):
        return self.__model_name

    @property
    def frame_field(self):
        return self.__frame_field

    @property
    def weights_path(self):
        return self.__weights_path

    @property
    def model_config(self):
        return self.__model_config

    @property
    def labels(self):
        return self.__labels

    @labels.setter
```

```
    def labels(self, set):
        self.__labels = set

    @property
    def shape(self):
        return self.__shape

    @property
    def thresholds(self):
        return self.__thresholds

    @property
    def input_tensor_name(self):
        return self.__input_tensor_name

    @property
    def output_tensors_names(self):
        return self.__output_tensors_names

    @property
    def model_description(self):
        return self.__model_description

    @property
    def graph(self):
        return self.__graph

    @property
    def original_shape(self):
        return self.__original_shape

    @original_shape.setter
    def original_shape(self, shape):
        self.__original_shape = shape

    def basic_prediction_record(self, box=None, label=-1, conf=-1.0):
        if box is None:
            box = -1
        return {
            'box': box,
            'label': label,
            'conf': conf,
            'overlap': -1
        }

    @abc.abstractmethod
    def load_graph(self):
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(open(self.weights_path, 'rb').read())
        import_fucn = lambda: tf.graph_util.import_graph_def(graph_def, name="")
        wrapped_import = tf.compat.v1.wrap_function(import_fucn, [])
        import_graph = wrapped_import.graph
        return wrapped_import.prune(
            tf.nest.map_structure(import_graph.as_graph_element,
self.input_tensor_name),
            tf.nest.map_structure(import_graph.as_graph_element,
self.output_tensors_names))

    @abc.abstractmethod
    def pre_processing(self,
                       img_batch,
```

```
                        roi_odom=None):
        pass

    @abc.abstractmethod
    def post_processing(self, graph_out, odom_roi=None):
        pass

    @abc.abstractmethod
    def apply_threshold(self, frame_res):
        pass

    def get_graph_out(self, input_batch):
        return self.__graph(input_batch)

    def _basic_decode_classifier(self, graph_out):
        batch = []
        for cntx_out in graph_out[0]:
            batch.append({self.frame_field:
self.basic_prediction_record(label=self.labels[int(argmax(cntx_out))],

conf=cntx_out.numpy().tolist())})
        return batch

    def _basic_decode_detector_ssd(self, graph_out):
        batch = []
        for id_b in range(graph_out[1].shape[0]):

            num_detections = int(graph_out[0][id_b])
            predictions = []
            for i in range(num_detections):
                classId = int(graph_out[3][id_b][i])
                label = self.labels[classId]
                confidence = float(graph_out[1][id_b][i])
                bbox = [float(v) for v in graph_out[2][id_b][i]]
                if self.original_shape is None:
                    b = [bbox[1], bbox[0],
                         bbox[3], bbox[2]]
                else:

                    b = [int(bbox[1] * self.original_shape[2]), int(bbox[0] *
self.original_shape[1]),
                         int(bbox[3] * self.original_shape[2]), int(bbox[2] *
self.original_shape[1])]

                predictions.append(self.basic_prediction_record(box=b,
                                                                conf=confidence,
                                                                label=label))

            batch.append({self.frame_field: predictions})

        return batch

    def _basic_decode_regressor(self, graph_out):
        batch = []
        for out in graph_out[0]:
            batch.append({self.frame_field: self.basic_prediction_record(
                conf=out)})
        return batch
```

```
import numpy as np
import tensorflow as tf
from pipe_ai.pipe_processing.ai_model import AIModel


class AIPredictor:

    def __init__(self, ai_models: [AIModel]):
        self.ai_models = ai_models

    # @tf.function
    def inference_models(self, img_batch, odom_roi):
        bgr = tf.unstack(img_batch, axis=-1)
        b, g, r = bgr[0], bgr[1], bgr[2]
        img_batch_tensor = tf.stack([r, g, b], axis=-1)
        model_outs = []
        for model in self.ai_models:
            model_outs.append(
                model.get_graph_out(model.pre_processing(img_batch_tensor,
odom_roi))
            )
        return model_outs

    def predict(self, img_batch, odometer_frame_pos=None):

        if odometer_frame_pos is not None:
            odometer_frame_pos = np.array(odometer_frame_pos, dtype=np.int)

        for model in self.ai_models:
            model.original_shape = img_batch[0].shape

        img_batch_tensor = tf.convert_to_tensor(np.array(img_batch),
dtype=tf.float32)

        graphs_out = self.inference_models(img_batch_tensor, odometer_frame_pos)

        decoded_res = []
        for id_, model in enumerate(self.ai_models):
            model.original_shape = img_batch[0].shape
            decoded_res.append(model.post_processing(graphs_out[id_],
odometer_frame_pos))

        res = [{}] * len(img_batch)
        for id_b in range(len(img_batch)):
            for ai_h_out in decoded_res:
                if any([k in list(res[id_b].keys()) for k in
ai_h_out[id_b].keys()]):
                    for k in list(ai_h_out[id_b].keys()):
                        if isinstance(res[id_b][k], list):
                            res[id_b][k].extend(ai_h_out[id_b][k])
                else:
                    res[id_b] = {**res[id_b], **ai_h_out[id_b]}
        return res

    def close_graph(self):
        pass
```

```
import tensorflow as tf
from pipe_ai.interfaces import FrameField, ModelType
from pipe_ai.pipe_processing.ai_model import AIModel
import numpy as np


class JointsDetector(AIModel):

    def __init__(self,
model_description_path='models/my_ssd_mobnet/model_description.yaml'):
        super().__init__(model_description_path, ModelType.JOINT_DETECT,
FrameField.JOINT)

    def load_graph(self):
        graph = tf.saved_model.load(self.weights_path)
        return graph

    def pre_processing(self,
                       img_batch,
                       roi_odom=None):

        img_batch = tf.image.resize(img_batch,
                                    [self.shape[0],
                                     self.shape[1]],
                                    name="resize_{}".format(self.model_type),

method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        img_batch = tf.cast(img_batch, dtype=tf.uint8)
        return img_batch

    def post_processing(self, graph_out, odom_roi=None):
        batch = []
        for out in graph_out:
            num_detections = int(out.pop('num_detections'))

            classes = np.array(out.pop('detection_classes'))
            boxes = np.array(out.pop('detection_boxes'))
            scores = np.array(out.pop('detection_scores'))
            detections = {'num_detections': num_detections,
                          'detection_classes': classes.astype(np.int64),
                          'detection_boxes': boxes,
                          'detection_scores': scores}
            batch.append(detections)
        return batch

    def apply_threshold(self, frame_res):
        if self.frame_field not in frame_res:
            return frame_res
        record = frame_res[self.frame_field]
        record['conf'] = float(max(record['conf']))
        frame_res[self.frame_field] = record
        return frame_res

    def get_graph_out(self, input_batch):
        graph_out = []
        for tensor in input_batch:
            graph_out.append(
                self.graph(tf.expand_dims(tensor, axis=0))
            )
        return graph_out
```

```
import tensorflow as tf
from pipe_ai.interfaces import FrameField, ModelType
from pipe_ai.pipe_processing.ai_model import AIModel
from numpy import argmax


class ContextClassifier(AIModel):

    def __init__(self,
model_description_path='models/clf_context_model/model_description.yaml'):
        super().__init__(model_description_path, ModelType.CONTEXT_CLASSFY,
FrameField.CONTEXT)

    def load_graph(self):
        graph = tf.saved_model.load(self.weights_path)
        return graph

    def pre_processing(self,
                       img_batch,
                       roi_odom=None):
        img_batch = tf.image.resize(img_batch,
                                    [self.shape[0],
                                     self.shape[1]],
                                    name="resize_{}".format(self.model_type),

method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        img_batch /= 127.5
        img_batch -= 1.

        return img_batch

    def post_processing(self, graph_out, odom_roi=None):
        batch = []
        for cntx_out in graph_out.numpy():
            batch.append({self.frame_field:
self.basic_prediction_record(label=self.labels[int(argmax(cntx_out))],

conf=cntx_out.tolist())})
        return batch

    def apply_threshold(self, frame_res):
        if self.frame_field not in frame_res:
            return frame_res
        record = frame_res[self.frame_field]
        record['conf'] = float(max(record['conf']))
        frame_res[self.frame_field] = record
        return frame_res
```

```
from pipe_ai.pipe_processing.ai_model import AIModel
from pipe_ai.pipe_processing.odometer import tf_io_pipline_fast_tools
import tensorflow as tf
from pipe_ai.interfaces import FrameField, ModelType
from pipe_ai.pipe_utils.regular_expresions import correction_odom_lables

class OdometerTextRecognizer(AIModel):
    def __init__(self,
model_description_path='models/text_recodnt3/model_description.yaml'):
        super().__init__(model_description_path, ModelType.ODOMETR_RECOGNIZER,
FrameField.DISTANCE)
        self.__codec_odometer_recognize =
tf_io_pipline_fast_tools.CrnnFeatureReader(
            char_dict_path=self.model_description['char_dict'],
            ord_map_dict_path=self.model_description['ord_map_dict']
        )
    def pre_processing(self,
                       img_batch,
                       roi_odom=None):
        odom_region = tf.image.crop_to_bounding_box(
            img_batch,
            roi_odom[1],
            roi_odom[0],
            roi_odom[3] - roi_odom[1],
            roi_odom[2] - roi_odom[0]
        )
        # crop_img = img[y:y + h, x:x + w]
        # print(img_batch[0])

        odom_region = tf.image.resize(odom_region, [self.shape[1],
                                                    self.shape[0]],
                                      name="resize_{}".format(self.model_type),

method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        odom_region = tf.cast(odom_region, tf.float32) / 127.5 - 1.0
        return odom_region

    def post_processing(self, graph_out, odom_roi=None):

        prediction_val =
self.__codec_odometer_recognize.sparse_tensor_to_str_for_tf_serving(
            decode_indices=graph_out[0].numpy(),
            decode_values=graph_out[1].numpy(),
            decode_dense_shape=graph_out[2].numpy()
        )
        batch = []
        for recng_txt in prediction_val:
            res =
correction_odom_lables(self.basic_prediction_record(box=odom_roi,

label=recng_txt)
                                           )
            if res is None:
                res = self.basic_prediction_record(box=odom_roi)
                res['units'] = "None"
            batch.append({self.frame_field: res})
        return batch
    def apply_threshold(self, frame_res):
        if self.frame_field not in frame_res:
            return frame_res
        return frame_res
```