Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ У ПОРІВНЯЛЬНОМУ АСПЕКТІ МЕТОДІВ ДЛЯ ПІДРАХУНКУ ЛЮДЕЙ У НАТОВПІ НА БАЗІ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ТА КЛАСИЧНИХ ПІДХОДІВ ДО АНАЛІЗУ ТЕКСТУРНИХ ЗОБРАЖЕНЬ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-1
Норматова Т.В.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Яковлева О. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Норматовій Тетяні Віталіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Реалізація та дослідження у порівняльному аспекті методів для підрахунку людей у натовпі на базі загорткових нейронних мереж та класичних підходів до аналізу текстурних зображень

затверджена наказом по університету від «23» жовтня 2020 року № 1428Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 23 листопада 2020 р.
3. Вихідні дані до роботи: Методи детектування людей у натовпі, бібліотека PyTorch, бібліотека OpenCV, мова програмування Python, згорткові нейронні мережі, алгоритм детектування текстур LBP, модель CSRNet, модель MCNN, міри оцінки точності роботи алгоритмів, датасет ShanghaiTech
4. Перелік питань, що потрібно опрацювати в роботі
1. Постановка задачі підрахунку людей у натовпі
2. Огляд існуючих підходів, що використовуються для підрахунку людей у натовпі
3. Аналіз методів підрахунку людей у натовпі на основі нейронних мереж та методи на основі виявлення структур
4. Огляд датасетів зображень для підрахунку людей у натовпі
5. Створення моделі на основі нейронної мережі CSRNet розробленої за допомоги мови програмування Python та бібліотеки PyTorch (побудувати архітектуру мережі та натренувати її)
6. Створення моделі на основі нейронної мережі MCNN розробленої за допомоги мови програмування Python та бібліотеки PyTorch (побудувати архітектуру мережі та натренувати її)
7. Дослідження методу локальних бінарних шаблонів, створити алгоритм та його програмну реалізацію на основі мови програмування Python та бібліотеки OpenCV
8. Визначення точніть результатів підрахунку усіма методами
9. Аналіз результатів проведених досліджень

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Датасет ShanghaiTech,Згорткові нейронні мережі, Створення загорткової нейронної мережі CSRNet, Створення нейронної загорткової мережі MCNN, Локальні бінарні шаблони (LBP), Використання дескриптору локальних бінарних шаблонів для підрахунку людей у натовпі, Ілюстрація роботи нейронних мереж, Ілюстрація роботи методу локальних бінарних шаблонів, Оцінка точності підрахунку людей у натовпі розглянутих методів, Оцінка швидкості навчання та роботи розглянутих методів*

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на до атестаційної роботи	23.10.2020	
2	Аналіз завдання, підбор літератури	23.10.20-25.10.20	
3	Аналіз літератури з досліджуваної проблеми	26.10.20-28.10.20	
4	Аналіз технічних засобів	29.10.20-30.10.20	
5	Розробка методів підрахунку людей у натовпі	30.10.20-01.11.20	
6	Програмна реалізація	02.11.20-10.11.20	
7	Оформлення пояснювальної записки	11.11.20-18.11.20	
8	Перевірка на плагіат	19.11.20	
9	Рецензування	20.11.20	
10	Підготовка презентації та доповіді	23.11.20	
11	Занесення роботи в електронний архів	29.11.20	
12	Попередній захист атестаційної роботи	30.11.20	

Дата видачі завдання 23 10 2020 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Яковлева О. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка атестаційної роботи: 77 с., 7 табл., 66 рис., 41 джерело.

ПІДРАХУНОК ЛЮДЕЙ У НАТОВПІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, АЛГОРИТМИ ВИЯВЛЕННЯ ТЕКСТУР, MCNN, CSRNET, ЛОКАЛЬНІ БІНАРНІ ШАБЛОНИ, PYTORCH, OPENCV.

Дана робота присвячена вирішенню проблеми підрахунку людей у натовпі на статичних зображеннях.

У роботі розглянуто методи підрахунку людей у натовпі з використанням загорткових нейронних мереж, а також методу заснованому на виявленні текстури.

Для виконання дослідження використовувались моделі загорткових нейронних мереж MCNN та CSRNet і метод локальних бінарних шаблонів, як текстурний метод.

Як середовище розробки було обрано Google Colaboratory, що дозволяє працювати у браузері та легко компілювати код, а також виводити результати кожного логічного блоку коду.

У результаті роботи здійснено порівняльний аналіз обраних методів для підрахунку людей у натовпі.

CROWD COUNTING, CONVOLUTIONAL NEURAL NETWORKS, TEXTURE DETECTION METHODS, MCNN, CSRNET, LOCAL BINARY PATTERNS, PYTORCH, OPENCV.

This work is devoted to solving the problem of crowd counting on static images.

The paper considers methods for counting people in a crowd using convolutional neural networks and a method based on texture detection.

Convolutional neural networks models MCNN and CSRNet and local binary patterns method were used to perform the study.

Google Colaboratory was chosen as the development environment, which allows you to work in a browser and easily compile code, as well as display the results of each logical block of code.

As a result we got a comparative analysis of selected methods for counting crowd counting.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN	–	Convolutional Neural Network;
CSRNet	–	Congested Scene Recognition Convolutional Neural Networks
DLR-ACD	–	DLR’s Aerial Crowd Dataset
GPU	–	Graphics Processing Unit
KRR	–	Kernel Ridge Regression
LBP	–	Local Binary Patterns
MAE	–	Mean Absolute Error
MCNN	–	Multi-Scale Convolutional Neural Networks
ML	–	Machine Learning
MSE	–	Mean Squared Error
ReLU	–	Rectified Linear Unit
SGD	–	Stochastic Gradient Descent

ВСТУП

Точна оцінка кількості об’єктів на одному зображенні – це складна, але важлива задача, що застосовується в багатьох додатках, пов’язаних з міським плануванням та громадською безпекою. Серед різних задач підрахунку об’єктів підрахунок людей у натовпі є особливо важливим через його значення для соціального забезпечення та розвитку. Підрахунок людей у натовпі та оцінка його щільності можуть допомогти кількісно визначити важливість подій та надати відповідному персоналу інформацію для підтримки прийняття рішень. Тому підрахунок людей у натовпі та оцінка його щільності стають гострими темами в галузі безпеки, які широко використовуються у відеоспостереженні, моніторингу дорожнього руху, громадській безпеці та містобудуванні. Оскільки кількість населення світу зростає, це призводить до швидкого зростання натовпу в багатьох ситуаціях, таких як паради, концерти і стадіонні заходи. У цих сценаріях підрахунок натовпу грає важливу роль для управління та контролю соціальної безпеки. Безпека натовпу в громадських місцях завжди була важливою, але клопітною справою, особливо у місцях збору людей, де щільність натовпу буває дуже високою. Чим вища щільність натовпу, тим йому легше вийти з-під контролю, і це може спричинити серйозні людські жертви.

З огляду на особливу важливість підрахунку людей у натовпі, що зазначена вище, все більше і більше дослідників намагалися розробляти різні складні проекти для вирішення проблеми підрахунку людей у нтовпі. Однак існуючі раніше системи моніторингу натовпу все ще мали багато слабких сторін, наприклад, більшість з них були обмежені сценами, до яких вони можуть примінятися, або мали низьку точність підрахунку. З появою глибокого навчання, моделі на основі згорткових нейронних мереж (CNN) переважали в різних завданнях комп’ютерного зору, включаючи підрахунок людей у натовпі.

1 СУЧАСНИЙ СТАН ЗАДАЧІ ПІДРАХУНКУ ЛЮДЕЙ У НАТОВПІ

1.1 Задача підрахунку людей у натовпі

Основною метою методів підрахунку людей у натовпі є позначення вхідних даних (кадрів у відеопослідовностях або статичних зображень) як переповнених та не переповнених. Другорядним завданням є просторове визначення та сегментація натовпу на зображенні. Підрахунок людей у натовпі може бути використано в додатках, де утворення натовпу може призвести до потенційно небезпечних ситуацій. Сегментація натовпу є корисним етапом попередньої обробки для подальшого аналізу.

Підрахунок людей та оцінка щільності натовпу – завдання, що викликають великий інтерес; завдання полягає в тому, щоб оцінити кількість або щільність об’єктів (зазвичай людей) у відеопослідовностях або статичному зображенні. На відміну від підрахунку людей, де результат – це точна кількість або передбачувана кількість людей, результат оцінки щільності точно не визначено. Його можна визначити за шкалою: 0, що позначає порожній простір; 10, що вказує на переповнений простір, або як класифікацію на класи подібної щільності (розріджені, помірні, щільні) [1], наприклад було запропоновано класифікацію на основі рівня щільності: вільний потік, обмежений потік, щільний потік і дуже щільний потік.

Кількість людей на сцені або їх щільність є корисною інформацією для багатьох цілей, включаючи оцінку використання громадського транспорту, виявлення небезпечних ситуацій або моніторинг якості послуг. Наприклад, кількість людей, які очікують перед касою, може бути показником для продавця, що слід відкрити нову касу [2].

1.2 Огляд існуючих підходів до вирішення задачі підрахунку людей у наовпі

Наразі існує чотири методи, які можна використовувати для підрахунку кількості людей у натовпі [3].

Методи на основі детектування (detection-based methods). У цих методах використовується метод ковзаючого вікна, щоб ідентифікувати людей на зображенні та підрахувати їх кількість. Методи, які використовуються для детектування, вимагають добре навчених класифікаторів, які здатні виявляти низькорівневі ознаки. Хоча такі методи добре вирішують задачі виялення облич, вони погано працюють на зображеннях, які містять багато людей, оскільки більшість цільових об’єктів не видно чітко.

Методи на основі регресії (regression-based method). Оскільки підходи, засновані на детектуванні, не можуть бути ефективно застосовані до зображень з великою кількістю людей, дослідники намагаються застосувати підходи на основі регресії, щоб вивчити взаємозв’язок між вилученими об’єктами з виділених ділянок зображення, а потім обчислити кількість конкретних об’єктів. Інформація про передній план та текстуру зобрження, використовуються для генерації низькорівневих ознак.

Методи засновані на оцінці щільності (density estimation-based methods). При використанні рішеннь, заснованих на регресії, без уваги залишається одна важлива особливість, яка називається виразністю, що спричиняє неточні результати у локальних регіонах зображення. Для вирішення цієї проблеми спочатку пропонується створити карту щільності для об’єктів. Потім алгоритм вивчає лінійне відображення між вилученими об’єктами та їх картами щільності. Також може бути використана так звана регресія random forest для вивчення нелінійного відображення замість лінійного.

Методи засновані на CNN. Замість того, щоб розглядати частини зображення, ми будуємо наскрізний метод регресії, використовуючи CNN. Вона бере все зображення як вхідне і безпосередньо генерує підрахунок натовпу. CNN дуже добре працюють із завданнями регресії або класифікації, і вони також довели свою цінність у створенні карт щільності для вирішення задач підрахунку людей у натовпі.

1.3 Глибокі нейронні мережі в задачах підрахунку людей

Було розглянуто проблему підрахунку людей у натовпі на статичних зображеннях за допомогою створення карт щільності [4]. Отримавши зображення переповненої сцени, мета полягає в тому, щоб створити карту щільності, яка має однаковий просторовий розмір, що і вхідне зображення. Кожен піксель карти щільності позначає щільність натовпу у відповідному місці на зображенні. Враховуючи приблизну карту щільності, кількість людей у натовпі можна отримати шляхом підсумовування записів карти щільності.

Попередні методи для підрахунку натовпу зазвичай трактують це завдання, як стандартну задачу навчання з учителем. Задача підрахунку людей у натовпі ще потребує дослідження. На даний час фактичним рішенням у літературі є збір набору навчальних зображень із анотаціями – ground-truth зображення. Тоді можна навчити модель, використовуючи згорткові нейронні мережі (CNN) для зіставлення вхідного зображення та карти щільності [5]. За останні кілька років для підрахунку натовпу пропонувались різні архітектури мереж на базі CNN.

1.3.1 Основні принципи глибоких нейронних мереж

Глибокі нейронні мережі відрізняються від більш загальноприйнятих нейронних мереж з одним прихованим шаром за їх глибиною; кількість шарів, через які повинні проходити дані в багатоетапному процесі розпізнавання образів, збільшується [6].

У мережах глибокого навчання кожен шар тренується на певному наборі ознак на основі вихідних даних попереднього шару. Чим далі дані просуваються по нейронній мережі, тим складнішими можуть ставати ознаки, оскільки вони об’єднують і рекомбінують ознаки з попереднього шару.

Під час тренування на даних для навчання кожен шар в глибокій мережі вивчає ознаки автоматично, неодноразово намагаючись відновити вхід, з якого він навчається прикладам, намагаючись звести до мінімуму різницю між тим, що очікувалося отримати у результаті та реальним результатом.

1.3.2 Згорткові нейронні мережі

Згорткові мережі – це нейронні мережі, які використовують згортку замість множення принаймні в одному з їх шарів [7, 8]. Формулу згортки приведено нижче:

$$(B * H_q)_{i,j} = \sum_{m,n=-(q-1)}^{q+1} H_q(q+m, q+n) B(i+m-1, j+n-1), \qquad (1.1)$$

де (w, x) – розраховує мий розмір карти;

mW – ширина попередньої карти;

mH – висота попередньої карти;

kW – ширина ядра;

kH – висота ядра.

CNN складається з різних видів шарів: згорткові (convolutional) шари, шари підвиборки (subsampling) [9] та повнозв'язаних шарів на останніх рівнях (рис. 1.1).

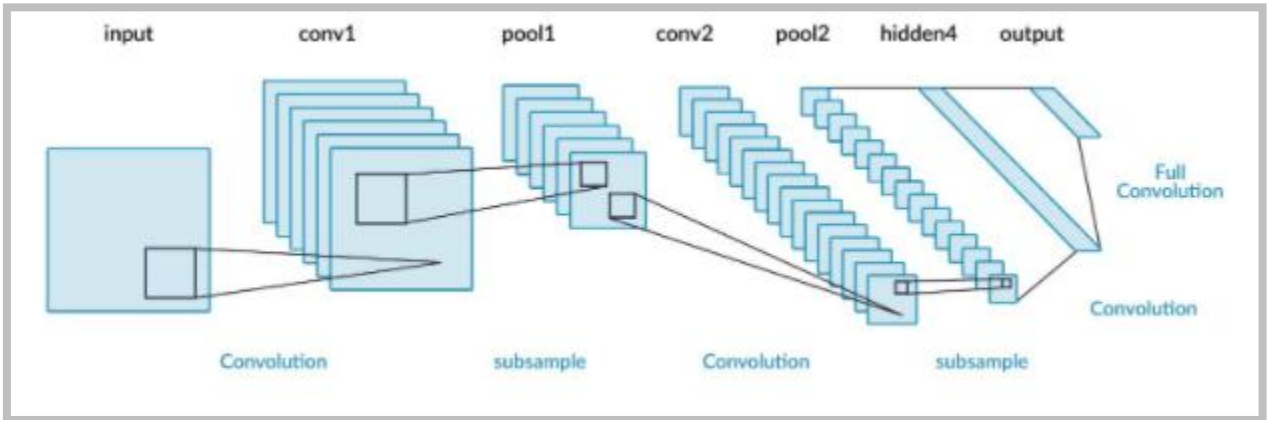

Рисунок 1.1– Топологія згорткової нейронної мережі

Згортковий шар являє собою набір карт – карт ознак або звичайних матриць. Кожна карта має синаптичне ядро, також можна назвати його фільтром (рис. 1.2).

Розмір усіх карт згорткового шару однаковий і вираховується за (1.1).

За рахунок крайових ефектів розмір початкових матриць зменшується:

$$x_j^l = f\left(\sum_i x_i^{l-1} * k_j^l + b_j^l\right), \tag{1.2}$$

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

$f()$ – функція активації;

b_j^l – коефіцієнт здвигу шару l для карти ознак j;

k_j^l – ядро згортки j карти шару l;

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

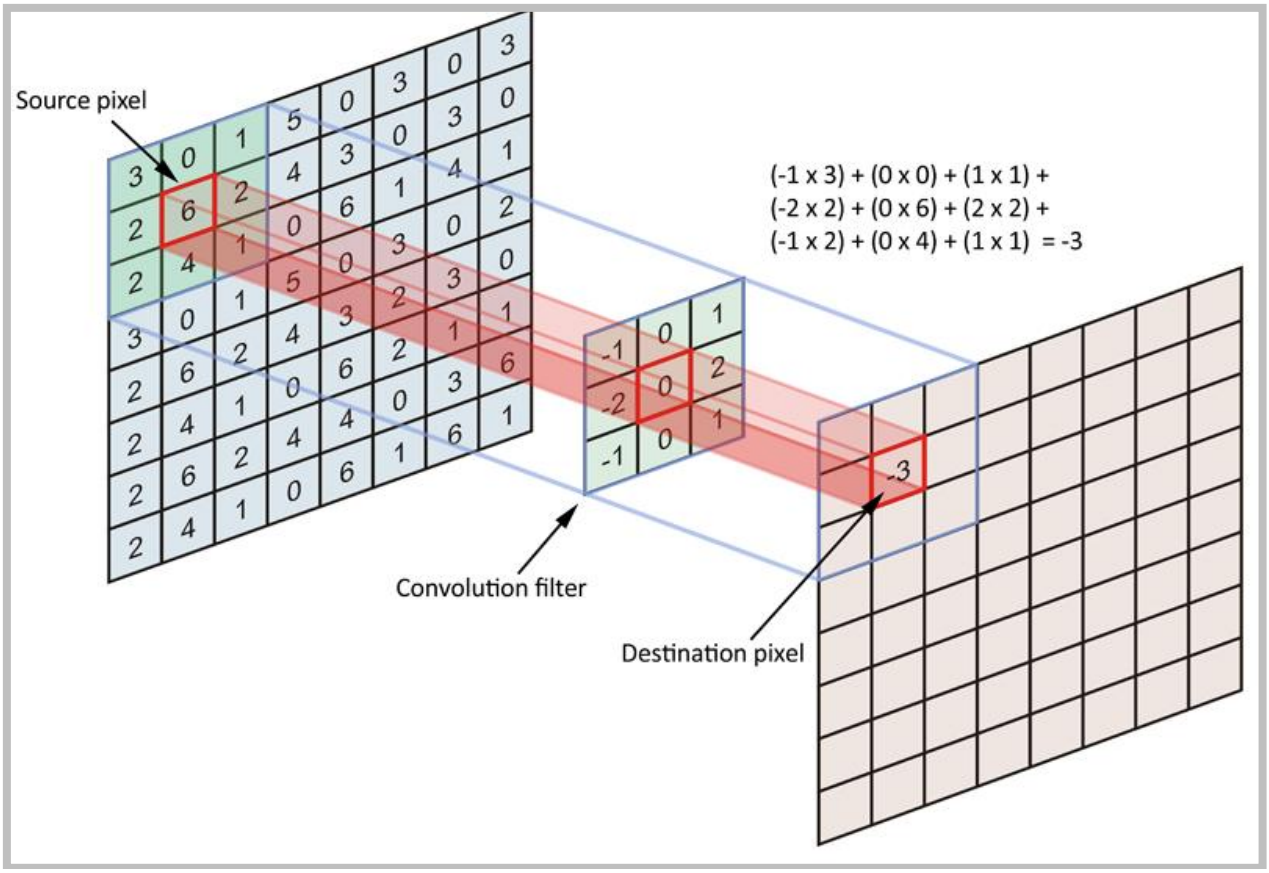

Рисунок 1.2 – Схема роботи згортки

Необхідна кількість карт встановлюється вимогами до конкретної задачі. Якщо обрати велику кількість карт, то можна підвищити якість розпізнавання, але також збільшити обчислювальну складність. Аналізуючи наукові статті та досліди, більшість випадків передбачає співвідношення один до двох. Такий підхід означає, що кожна карта попереднього шару пов’язується з двома картами згорткового шару.

Перевагами CNN є :

– спільне використання параметрів (зв’язані ваги) – можливість використовувати ті ж самі ваги кілька разів;

– еквіріантність до зсуву – зміна входу так само змінює вихід;

– не фіксований розмір форми входу (input shape).

Мета шару підвибірки – 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 моделі не перенавчатися.

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

Зазвичай в шарі підвибірки застосовується функція активації ReLU. Операція підвибірки (або MaxPooling – вибір максимального) (рис. 1.3).

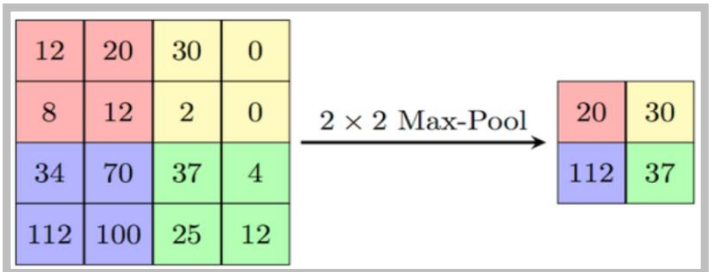

Рисунок 1.3 – Формування нової карти шару підвибірки на основі попередньої карти згорткового шару. Операція підвибірки (Max Pooling)

Формально шар може бути описаний наступною формулою:

$$x^l = f(a^l * subsample(x^{l-1}) + b^l), \tag{1.3}$$

де x^l – вихід шару l;

$f()$ – функція активації;

a^l, b^l – коефіцієнти зсуву шару l;

$subsample()$ – операція вибірки локальних максимальних значень.

Останній з шарів це – шар звичайного багатошарового персептрона – повнозв’язний шар. Його мета – класифікація, він моделює складну нелінійну функцію, оптимізація якої, поліпшує якість розпізнавання.

Нейрони кожної карти попереднього шару підвибірки пов’язані з одним нейроном прихованого шару. Таким чином кількість нейронів прихованого шару дорівнює числу карт шару підвибірки, але зв’язки можуть бути не обов’язково такими, наприклад, тільки частина нейронів будь-якої з карт шару підвибірки може бути пов’язана з першим нейроном прихованого шару, а частина, що залишилася, з другим, або все нейрони першої карти пов’язані з нейронами 1 і 2 прихованого шару. Обчислення значень нейрона можна описати формулою:

$$x_j^l = f\left(\sum_i x_i^{l-1} * w_{i,j}^{l-1} + b_j^{l-1}\right), \tag{1.4}$$

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

$f()$ – функція активації;

b_j^{l-1} – коефіцієнт зсуву шару l;

$w_{i,j}^{l-1}$ – матриця вагових коефіцієнтів шару l.

Вихідний шар пов’язаний з усіма нейронами попереднього шару. Кількість нейронів відповідає кількості класів, які необхідно розпізнати, тобто 2 для бінарної класифікації.

1.3.3 Існуючі архітектури нейронних мереж для оцінки щільності натовпу

Мультиколонна згорткова нейронна мережа (MCNN). Дана мережа була запропонована у 2016 році після того як у 2015 році у Шанхаї у результаті великої тисняви загинуло 35 людей.

Після подій у 2015 році необхідно було створити метод, який дозволяє оцінити підрахунок кількості людей у натовпі на статичному зображенні з довільною щільністю натовпу та довільною перспективою. З цією метою

було запропоновано просту, але ефективну архітектуру багатокалонної згорткової нейронної мережі (MCNN) для отримання карт щільності із вхідних зображень. MCNN повинна була стати методом, що демонструє вищу точність підрахунку людей у натовпі порівняно з попередніми методами. Запропонована мережа дозволяє вхідному зображенню мати довільний розмір або роздільну здатність. Використовуючи різні розміри ядра згортки, ознаки, вивчені кожною колонкою CNN, адаптуються до змін у розмірі людей / голів внаслідок ефекту перспективи або роздільної здатності зображення. Крім того, істинна карта щільності обчислюється точно на основі геометрично адаптивних ядер, яким не потрібно знати перспективну карту вихідного зображення. Експерименти показують, що MCNN після навчання на одному наборі даних може бути легко перенесена на новий набір даних.

Через спотворення перспективи зображення зазвичай містять голови дуже різних розмірів, отже було припущено, що фільтри з ядрами згортки однакового розміру навряд чи зможуть вловити характеристики щільності натовпу в різних масштабах. Тому більш логічно використовувати фільтри з різними розмірами ядра згортки для вивчення карти від вихідних пікселів до карт щільності. Ця мережа складається з трьох паралельних згорткових нейронних мереж (колонок), що мають однакову структуру, але відмінний розмір ядра згортки. Для кожної колонки використовуються фільтри різного розміру для моделювання карт щільності, що відповідають головам різних масштабів. Наприклад, фільтри з більшими розмірами ядра згортки є більш корисними для моделювання карт щільності, що відповідають більшим головам. Загалом кількість параметрів цієї мережі досягає майже 130 тисяч.

Згорткова нейронна мережа для розпізнавання перевантажених сцен (CSRNet). Дану нейронну мережу було запропоновано у 2018 році так як попередні методи підрахунку не давали задовільні результати, їх можна і треба було покращити. Мережа розпізнавання перевантажених сцен під назвою CSRNet розроблена для того, щоб впровадити метод глибокого

навчання заснований на даних, який може розпізнати сильно перевантажені сцени та виконати точний підрахунок людей у натовпі, а також представити високоякісні карти щільності. Кількість параметрів, що необхідно натренувати становить більше 16 мільйонів.

Запропонована CSRNet складається з двох основних компонентів: попередньо навченої згорткової нейронної мережі VGG-16 як інтерфейсу для вилучення ознак зображення, та CNN з розширеною згорткою для внутрішньої мережі, яка використовує розширене ядро згортки для заміни операцій об'єднання. CSRNet – мережа, яка легко піддається навчанню, завдяки своїй повністю згортковій бекенд структурі [10].

Нейронна мережа з роздільною здатністю для підрахунку людей у натовпі (MRCNet). Ця нейронна мережа була розроблена у 2019 році для роботи із зображеннями натовпу, знятими з висоти, наприклад, з літаку або квадрокоптеру. MRCNet складається з двох частин кодеру та декодеру. Кодер заснований на мережі VGG-16, а декодер складається з набору дволінійних шарів підвищеної вибірки та шарів згортки. Використовуючи дві функції втрат, одну на попередньому рівні та іншу на останньому рівні декодера, MRCNet оцінює підрахунок людей у натовпі та побудову карти щільності натовпу з високою роздільною здатністю як дві різні, але взаємопов'язані задачі.

1.4 Використання класичних алгоритмів аналізу текструри для підрахунку людей у натовпі

Локальні бінарні шаблони (local binary patterns, LBP). Це проста але практично корисна текстурна характеристика. Класичний LBP, що застосовується до пікселя зображення, використовує вісім пікселів околиці. Він приймає центральний пік сель, як поріг, і порівнює з ним значення інтенсивності в кожному сусідньому пікселі [11].

Енергетичні характеристики Лавса. Даний метод передбачає виявлення різних типів текстур за допомогою локальних масок. Лавс запропонував енергетичний підхід [12–14], який допомогає оцінити зміну змісту текстури в межах вікна, що має фіксований розмір. Щоб обчислити енергетичні характеристики використовується набір з дев’яти масок розмірами 5×5. Після цього енергетичні характеристики кожного пікселя зображення представляються у вигляді вектора з 9 чисел.

Щільність і напрямок країв. Виділення країв це простий у використанні спосіб виявлення характерних ознак. Тому його можна використовувати як частину алгоритму текстурного аналізу. Кількість країв присутніх у заданій області, що має фіксований розмір, являє собою характеристику заповнення цієї області. Під час виділення країв обчислюються їх напрями. Отримані значення можна застосувати для опису текстури [15].

1.5 Існуючі датасети зображень натовпу

ShanghaiTech. Великомасштабний набір даних для підрахунку людей у натовпі, який містить 1198 анотованих зображень (рис. 1.4) із загальною кількістю 330 165 людей із анотованими центрами їх голів [16].

Рисунок 1.4 – Приклади зображень з набору даних ShanghaiTech

DLR-ACD. Набір даних DLR-ACD – це колекція аерофотознімків для підрахунку людей у натовпі та оцінки щільності, а також для локалізації людей на масових заходах. Він містить 33 великі аерофотознімки (рис. 1.5), отримані в результаті 16 різних авіакампаній на різних масових заходах та над міськими сценами із залученням людей, таких як спортивні події, центри міст, ярмарки та фестивалі під відкритим небом [17].

Зображення були зроблені за допомогою стандартних фотокамер, встановлених на гвинтокрилі, і їх просторова роздільна здатність (або відстань вибірки землі – GSD) коливається від 4,5 до 15 см / піксель. Набір даних було позначено вручну точковими анотаціями для окремих людей, тому набір містить 226 291 анотацію людей, що становить від 285 до 24 368 анотацій на зображення.

Рисунок 1.5 – Приклади зображень з набору даних DLR-ACD

Mall Dataset. Набір даних Mall було зібрано з загальнодоступної веб-камери для підрахунку людей у натовпі (рис. 1.6). Датасет має 60 000 анотованих людей було позначено у 2000 сценах [18].

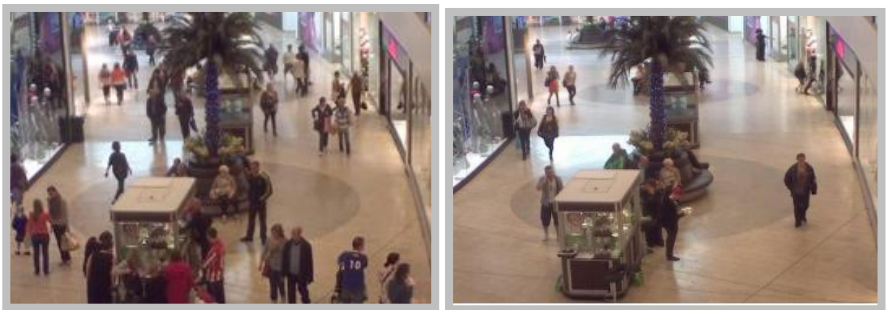

Рисунок 1.6 – Приклади зображень з набору даних DLR-ACD

UCF-CC-50. Цей набір даних містить зображення надзвичайно щільного натовпу. Зображення зібрані в основному з FLICKR фотохостингу, призначеного для зберігання і подальшого використання користувачем цифрових фотографій і відеороликів [19]. Цей датасет використовується лише для дослідницьких цілей та мість 50 анотованих зображень (рис. 1.7).

Рисунок 1.7 – Приклади зображень з набору даних UCF-CC-50

1.6 Огляд програмних засобів для вирішення задач комп’ютерного зору

1.6.1 Мова Python та можливості бібліотек SciPy library, SciKit image library, SKlearn library, OpenCV library

SciPy. Бібліотека SciPy (Scientific Python) залежить від NumPy, що забезпечує зручну і швидку маніпуляцію з N-мірними масивами [20]. Бібліотека SciPy створена для роботи з масивами NumPy і надає безліч зручних і ефективних чисельних методів, таких як процедури чисельної інтеграції та оптимізації. Разом вони працюють на всіх популярних операційних системах, швидко встановлюються і є безкоштовними. NumPy і SciPy прості у використанні, але досить потужні, щоб залежати від провідних світових вчених і інженерів.

Базова структура даних, яка використовується SciPy, являє собою багатовимірний масив, що надається модулем NumPy. NumPy надає деякі функції для лінійної алгебри, перетворень Фур’є і генерації випадкових чисел, але не з спільністю еквівалентних функцій у SciPy.

SciKit-image. Це пакет обробки зображень Python, який працює з масивами NumPy, що є набором алгоритмів для обробки зображень [21]. scikit-image – це бібліотека для обробки зображень, що реалізує алгоритми та утиліти для використання в наукових дослідженнях, освіті та галузевих додатках. Забезпечує добре задокументований API мовою програмування Python і розроблений активною міжнародною командою співавторів.

SKlearn. Scikit-learn забезпечує цілий ряд керованих та некерованих алгоритмів навчання через послідовний інтерфейс у Python [22]. Бібліотека побудована на SciPy, що потрібно встановити, перш ніж ми зможемо використовувати scikit-learn. Розширення або модулі для догляду за SciPy, умовно названі SciKits. Таким чином, модуль надає алгоритми навчання і називається scikit-learn.

OpenCV. Це бібліотека комп'ютерного зору та машинного навчання з відкритим вихідним кодом. У OpenCV входять більше 2500 алгоритмів, серед яких є як класичні, так і сучасні алгоритми для вирішення задач комп'ютерного зору та машинного навчання. Ця бібліотека має інтерфейси на різних мовах, серед яких є Python, Java, C ++ та Matlab.

1.6.2 Практичні аспекти роботи з нейронними мережами

1.6.2.1 Огляд існуючих бібліотек для роботи з нейронними мережами

Область глибокого навчання розвивається, а разом із нею і засоби розробки. Так на даний час існує велика кількість бібліотек для роботи з нейронними мережами. Безперечним лідером серед них є TensorFlow. Keras, Caffe, Microsoft Cognitive Toolkit, and PyTorch входять у першу п'ятірку (рис. 1.8).

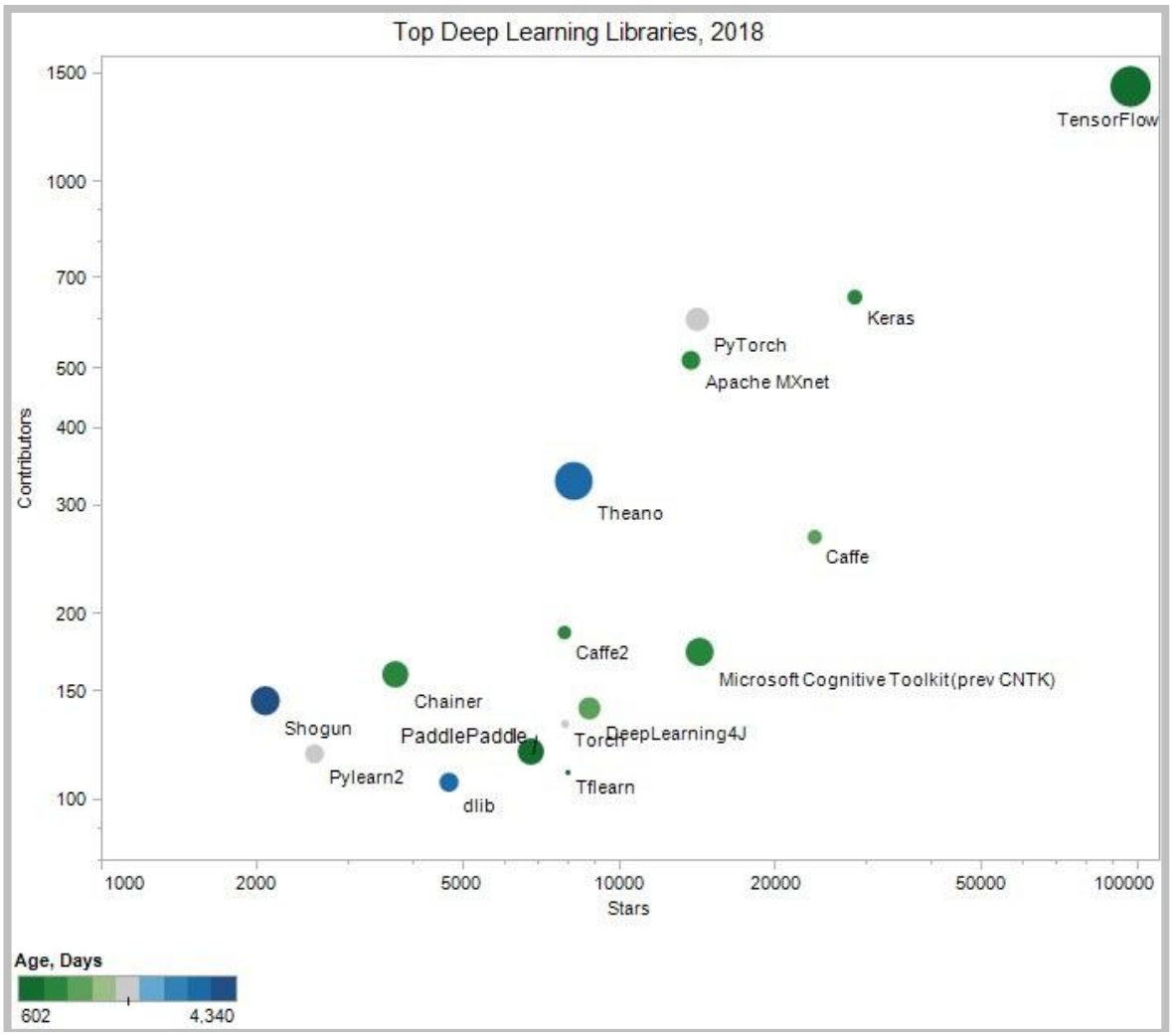

Рисунок 1.8 – Найпопулярніші бібліотеки для роботи з нейронними мережами

Необхідно зазначити, що бібліотеки розподіляються на ті, які широко використовуються у комерційних проектах і ті, з якими працюють у дослідницьких цілях.

TensorFlow. На високому рівні TensorFlow – це бібліотека Python, яка дозволяє користувачам представляти довільне обчислення, як графік потоків даних. Вузли на цьому графіку представляють математичні операції, тоді як ребра представляють дані, які передаються від одного вузла до іншого. Дані в TensorFlow представлені як тензори, які є багатовимірними масивами. TensorFlow використовується в основному для глибокого навчання на практиці у комерційних проектах та дослідженнях. Має потужну систему візуалізації [23].

Keras. Keras – це API для нейронних мережі високого рівня, написаний на Python і здатні працювати, як обгортка бекграунду, який може бути представлений TensorFlow, CNTK або Theano. Ця бібліотека не реалізує низькорівневі операції, такі як маніпуляції з тензорами і диференціювання, для цього використовується спеціалізована і оптимізована бібліотека підтримки тензорів [24].

Microsoft Cognitive Toolkit (CNTK). Цей інструментарій описує нейронні мережі, як серію обчислювальних кроків через спрямований граф [25]. CNTK реалізує стохастичний градієнтний спуск (SGD, помилка зворотного поширення) з автоматичним диференціюванням і розпаралелюванням на декількох GPU і серверах.

PyTorch. PyTorch більше підходить для некомерційних, а дослідницьких проектів [26]. Він надає гібридний інтерфейс, що дозволяє розробникам швидко переходити на свої моделі на етапі прототипування, не жертвуючи продуктивністю на стадії виробництва.

Усі ці бібліотеки пітримуються мовою програмування Python, яка є лідером для вирішення задач ML. Перевагою описаних вище бібліотек є те, що вони детально описані у різноманітних навчальних мануалах, курсах і також мають великі наукові спільноти.

1.6.2.2 Можливості бібліотеки PyTorch

Бібліотека PyTorch надає дві основні високорівневі моделі:

– тензорні обчислення (по аналогії з NumPy) з розвиненою підтримкою прискорення на GPU;

– глибокі нейронні мережі на базі системи autodiff.

Надійна екосистема. Активна спільнота дослідників та розробників створила багату екосистему інструментів та бібліотек для розширення

PyTorch та підтримки розвитку у сферах від комп’ютерного зору до навчання з підкріпленням.

Зручність для користувачів. PyTorch порівняно легше вивчати, ніж інші фреймфорки глибокого навчання. Це пов’язано з тим, що його синтаксис та застосування подібні до багатьох звичайних мов програмування, таких як Python. Документація PyTorch також дуже організована та корисна для початківців. А цілеспрямована спільнота розробників також допомагає постійно вдосконалювати PyTorch.

Підтримка хмарних платформ. PyTorch підтримується на основних хмарних платформах, забезпечуючи розробку та легке масштабування за допомогою попередньо вбудованих зображень, масштабне навчання на графічних процесорах, можливість запуску моделей у виробничому середовищі тощо.

Підтримка динамічних обчислювальних графів. PyTorch підтримує динамічні обчислювальні графи, це означає, що поведінку мережі можна програмно змінювати під час виконання. Це сприяє більш ефективній оптимізації моделі та дає PyTorch головну перевагу перед іншими фреймворками и машинного навчання, які розглядають нейронні мережі як статичні об’єкти.

Паралельність даних. Цей фреймворк має дуже корисну функцію, відому як паралельність даних. Використовуючи цю функцію, PyTorch може розподіляти обчислювальну роботу між декількома ядрами процесора або графічного процесора.

Модульність. Під моделлю розуміється послідовність або граф автономних, повністю настроюваних модулів, які можна підключити разом з якомога меншою кількістю обмежень. Зокрема, нейронні шари, функції витрат, оптимізатори, схеми ініціалізації, функції активації та схеми регуляризації – це всі окремі модулі, які можна об’єднати для створення нових моделей.

Проста розширюваність. Нові модулі легко додати (як нові класи та функції), а існуючі модулі надають велику кількість можливостей користувачу.

Робота з Python. PyTorch створений для безперебійної інтеграції з Python та його популярними бібліотеками, такими як NumPy.

1.7 Постановка задачі дослідження

Незважаючи на успіхи досягнені у вирішенні задачі підрахунку людей у натовпі окремі її випадки потребують додаткових досліджень.

Об’єктом дослідження є автоматичний підрахунок людей у натовпі на статичних зображеннях.

Метою дослідження є порівняння методів підрахунку людей у натовпі.

Основну мету роботи можна декомпозувати на декілька підзадач:

– підготовка даних для навчання нейронних мереж;

– дослідження існуючих методів підрахунку людей у натовпі на основі нейронних мереж, а також методів аналізу текстур;

– реалізація алгоритмів підрахунку на основі нейронних мереж (побудова архітектури, навчання мереж);

– реалізація алгоритму підрахунку людей у натовпі на основі методу аналізу текстури (отримання ознак зображення, побудова гістограми на основі отриманих ознак, навчання ridge regression) ;

– визначення точності результатів підрахунку усіма методами та їх швидкість;

– порівняння результатів усіх обраних методів;

– аналіз недоліків та переваг обраних методів.

Для виконання всіх пунктів необхідно вирішити и розглянути наступні теоретичні питання:

– дослідити теоретичну частину по нейронним мережам (побудова архітектури, вибір оптимізаторів, цільової функції та функції втрат);

– дослідити теоретичну частину методу локальних бінарних шаблонів (отримання ознак зображення, побудова гістограми);

– дослідження існуючих мір оцінювання точності.

Практична частина містить:

– дослідження функцій, синтаксису та можливостей бібліотеки PyTorch для роботи зі загортковими нейронними мережами;

– дослідження функцій, синтаксису та можливостей бібліотеки OpenCV для роботи з методом виявлення текстури;

– налаштування середовища розробки – Google Colaboratory ;

– створення моделей глибинного навчання для підрахунку людей у натовпі на мові Python;

– створення алгоритму на основі методу виявлення текстур для підрахунку людей у натовпі.

Для навчання мереж використано датасет ShanghaiTech [16], який містить 1198 анотованих зображень, в цілому 330 165 осіб з антонованими центрами їх голів, загальною вагою 166,11 Мбайт.

2 РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ДЛЯ ПІДРАХУНКУ ЛЮДЕЙ У НАТОВПІ

2.1 Підрахунок людей у натовпі на основі глибоких нейронних мереж

2.1.1 Розробка і навчання глибокої нейронної мережі

Розробка глибокої нейронної мережі зазвичай починається з вибору архітектури мережі – моделі. Обирати архітектуру слід спираючись на наявні дані та задачу, яку необхідно вирішити [27].

Модель повинна складатися з певної кількості шарів, які виконують різні функції (шари активації, втрат тощо), тому потрібно прийняти два важливих архітектурних рішення:

– скільки шарів використовувати;

– скільки скритих нейронів обрати для кожного шару.

Скритий нейрон – це вимір у просторі уявлень шару.

Після того, як побудовано модель, необхідно налаштувати ще три параметри, щоб підготувати мережу до тренування:

– функцію втрат, яка визначає, як мережа повинна оцінювати якість своєї роботи на данних, на яких вона навчається, і відповідно, як коректувати її у правильному напрямі. Наприклад для бінарної класифікації можно використовувати функцію бінарної перехресної ентропії;

– оптимізатор – механізм, за допомогою якого мережа буде оновлювати себе, спираючись на спостережувані дані і функцію втрат;

– метрика для моніторингу на етапах навчання та тестування – точність (частина правильно класифікованих зображень).

Після усіх налаштувань можно починати тренування мережі. У процесі навчання відображаються дві величини: втрати мережі на данних, на яких вона навчається і точність мережі на цих даних.

Процес навчання можна описати наступним чином. Мережа складається із шарів, які об’єднуються у ланцюжок і відображають початкові дані в передбаченнях (рис. 2.1). Після цього функція втрат порівнює ці передбачення з реальними результатами і повертає значення втрати: міру відповідності передбачення, що виконано мережею, очікуваному результату. Оптимізатор використовує це значення втрати для зміни вагів мережі.

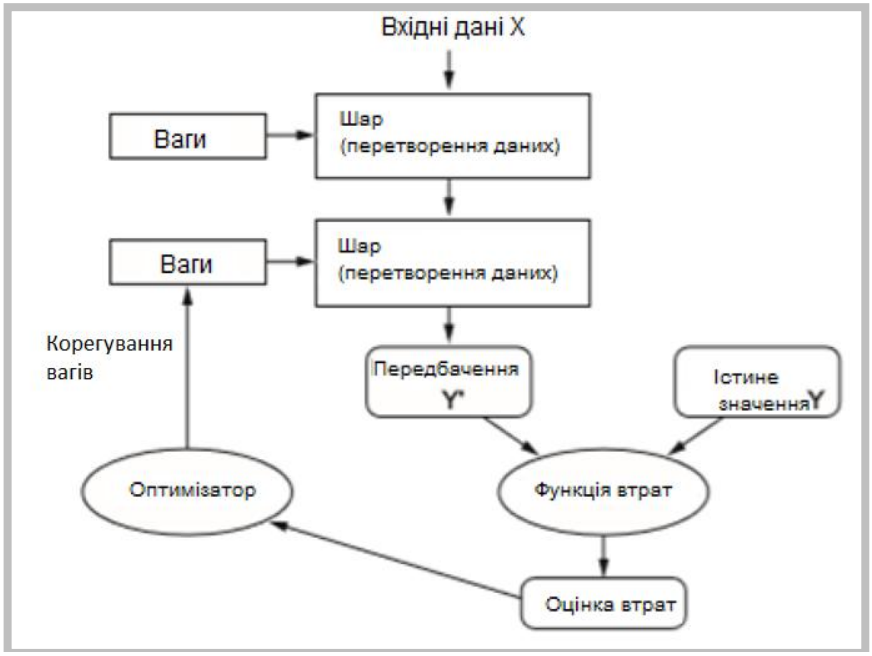

Рисунок 2.1 – Зв’язок між мережею, шарами, функцією втрат та оптимізатором

Навчати нейрону мережу можна методом зворотного поширення помилки (Backpropagation) [28]. Навчання мережі таким методом включає два проходи по всім шарам мережі: прямий та зворотній.

При прямому проході вхідний вектор подається на вхідний шар нейронної мережі, після чого поширюється від шару до шару. У результаті генерується набір вихідних сигналів, який і є фактичною реакцією мережі на вхідні дані. Під час прямого проходу всі синаптичні ваги мережі є фіксованими.

Під час зворотнього проходу ваги корегуються у відповідності з правилом коригування помилок, а саме: фактичний вихід мережі вираховується із бажаного, у результаті чого формується сигнал помилки.

У якості активаційної функції у багатошарових персептронах, як правило, використовується сигмоїдальна активаційна функція

$$OUT = \frac{1}{1+\exp(-\alpha Y)}, \tag{2.1}$$

де α – параметр нахилу сигмоїдальної функції [29].

Для коригування вагів вихідного шару введемо величину δ – різниця між очікуваним та реальним, що помножена на похідну функції активації

$$\delta_q = OUT_q(1-OUT_q)(T_q - OUT_q). \tag{2.2}$$

Тоді ваги вихідного шару після корегування дорівнюватимуть:

$$\omega_{p-q}(i+1) = \omega_{p-q}(i) + \eta\delta_q OUT_p, \tag{2.3}$$

де i – номер поточної ітерації навчання;

ω_{p-q} – величина синаптичної ваги, що поєднує нерон p з нейроном q;

η – коефіцієнт швидкості навчання, допомогає керувати середньою величиною зміни вагів;

OUT_p – вихід нейрона p.

Також необхідно коригувати ваги скритих шарів. Вводимо величину δ, що дорівнює:

$$\delta_q = OUT_q(1-OUT_q)\sum_{k=1}^{M}\delta_k\omega_{q-k}. \tag{2.4}$$

Тоді ваги скритих шарів після корекції дорівнюватимуть:

$$\omega_{p\text{-}q}(i+1) = \omega_{p\text{-}q}(i) + \eta\delta_q OUT_p . \quad (2.5)$$

2.1.2 Архітектура нейронної мережі MCNN

Ця нейронна мережа містить три паралельні CNN, фільтри яких мають ядра згортки різного розміру. Для спрощення структури мережі було використано однакові мережеві структури для всіх стовпців (тобто conv – pooling – conv – pooling), за винятком розмірів та кількості фільтрів (рис. 2.2). Max pooling застосовується для кожної області розміром 2 × 2.

Визначаємо архітектуру першої мережі:

1. Шар згортки, розмір ядра 9×9, кількість карт ознак – 16 одиниць.
2. Шар підвибірки, вибір максимального значення з квадрата 2×2.
3. Шар згортки, розмір ядра 7×7, кількість карт ознак – 32 одиниці.
4. Шар підвибірки, вибір максимального значення з квадрата 2×2.
5. Шар згортки, розмір ядра 7×7, кількість карт ознак – 16 одиниць.
6. Шар підвибірки, вибір максимального значення з квадрата 2×2.
7. Шар згортки, розмір ядра 7×7, кількість карт ознак – 8 одиниць.

Визначаємо архітектуру другої мережі:

1. Шар згортки, розмір ядра 7×7, кількість карт ознак – 20 одиниць.
2. Шар підвибірки, вибір максимального значення з квадрата 2×2.
3. Шар згортки, розмір ядра 5×5, кількість карт ознак – 40 одиниць.
4. Шар підвибірки, вибір максимального значення з квадрата 2×2.
5. Шар згортки, розмір ядра 5×5, кількість карт ознак – 20 одиниць.
6. Шар підвибірки, вибір максимального значення з квадрата 2×2.
7. Шар згортки, розмір ядра 5×5, кількість карт ознак – 10 одиниць.

Визначаємо архітектуру третьої мережі:

1. Шар згортки, розмір ядра 5×5, кількість карт ознак – 24 одиниці.
2. Шар підвибірки, вибір максимального значення з квадрата 2×2.
3. Шар згортки, розмір ядра 3×3, кількість карт ознак – 48 одиниць.
4. Шар підвибірки, вибір максимального значення з квадрата 2×2.

5. Шар згортки, розмір ядра 3×3, кількість карт ознак – 24 одиниці.
6. Шар підвибірки, вибір максимального значення з квадрата 2×2.
7. Шар згортки, розмір ядра 5×5, кількість карт ознак – 12 одиниць.

У побудованій MCNN ознаки, отримані усіма трьома мережами, сходяться на згортковий шар, розмір ядра згортки якого 1×1. Тому вхідне зображення цієї моделі може мати довільний розмір, щоб уникнути спотворень роздільної здатності зображення. Безпосередній вихід мережі – це карта щільності натовпу, завдяки якій далі отримується підрахована кількість людей на зображенні.

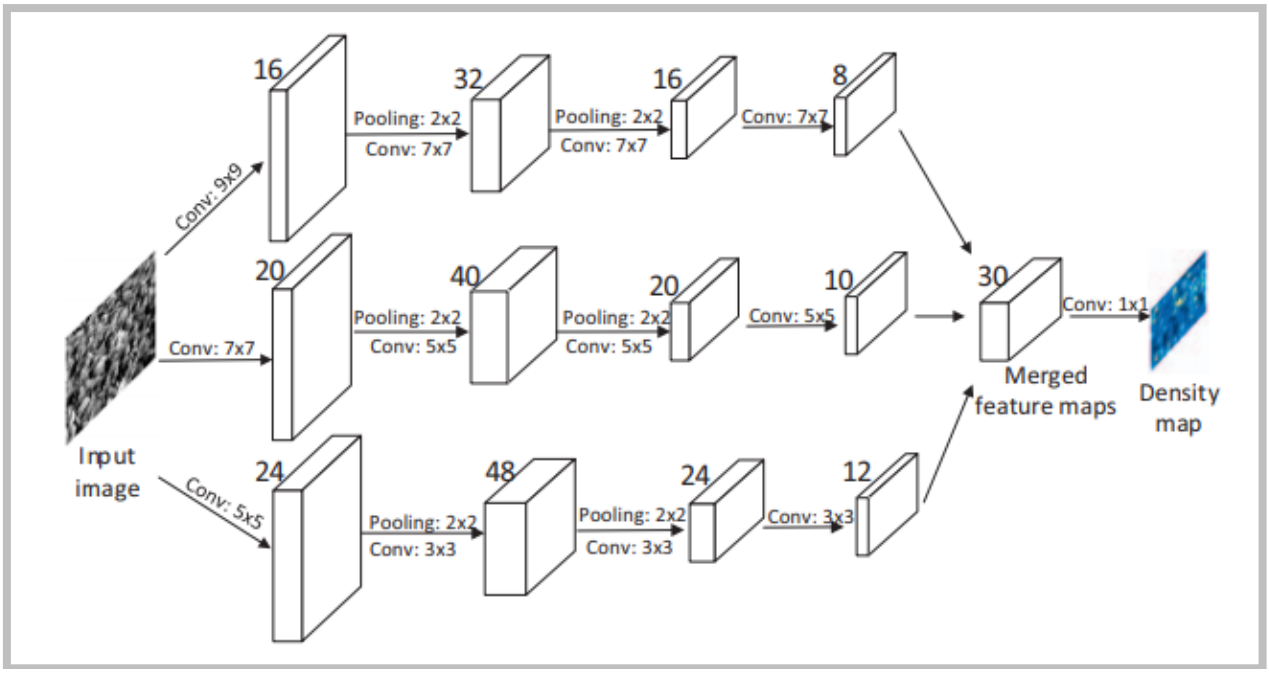

Рисунок 2.2 – Архітектура мережі MCNN

Щоб зменшити обчислювальну складність (кількість параметрів, що підлягає оптимізації), використовується менша кількість фільтрів для CNN з більшими ядрами згортки. Карти вихідних характеристик усіх CNN складаються разом та відображаються на карті щільності. Для відображення карт об’єктів на карті щільності обрано фільтр з ядром згортки 1×1.

Відображення графіків втрат на етапах навчання та перевірки для усієї мережі (рис. 2.3).

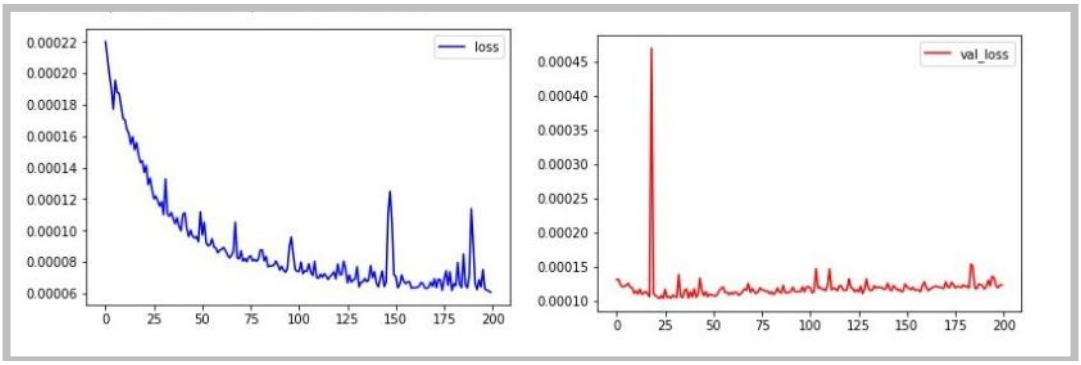

Рисунок 2.3 – Графіки втрат на етапах навчання та перевірки мережі MCNN

Відображення графіку втрат тестування для кожної з трьох колонок (рис. 2.4).

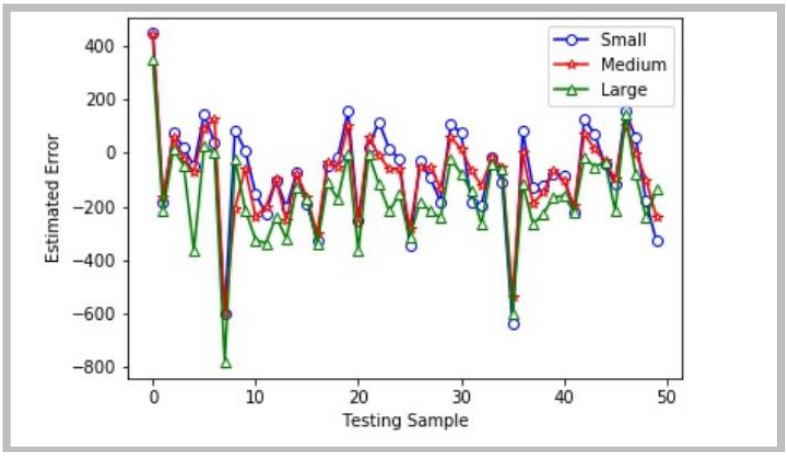

Рисунок 2.4 – Графік втрат на етапах навчання та перевірки для кожної колонки мережі MCNN

2.1.3 Архітектора нейронної мережі CSRNet

Мережа CSRNet є більш глибокою у порівнянні із попередніми загортковими нейронними мережами, які використовувалися для підрахунку людей у натовпі. CSRNet використовується для підрахунку натовпу та створення високоякісних карт щільності. На відміну від попередніх методів, які використовують глибокі CNN як допоміжний засіб, ми зосереджуємось на розробці генератора карт щільності на основі CNN. Розроблена модель використовує чисті згорткові шари у якості основи для підтримки вхідних

зображень із гнучкою роздільною здатністю. Щоб обмежити складність мережі, було використано невеликі розміри ядер згортки (наприклад, 3×3) у всіх шарах. У якості фронтенд частини використовуються перші 13 шарів з VGG-16 (10 шарів згортки та 3 шари MaxPooling) та розширені згорткові шари як бекенд, щоб збільшити ядро згортки та отримати більш глибокі ознаки зображень, не втрачаючи їх роздільну здатність (рис. 2.5).

VGG16 було як інтерфейс CSRNet через її гарну здатність до переносу навчання та гнучку архітектуру, яка легко поєднується з бекенд частиною для генерації мап щільності [30].

Для бекенд частини використано шари розширеної згортки, бо вихідний розмір фронтенд частини становить 1/8 від початкового розміру вхідного зображення. Якщо продовжувати додавати більше згорткових шарів та шарів пулінгу (основні компоненти у VGG-16), розмір вихідного зображення буде ще більше зменшений, через це буде важко створити високоякісний карти щільності. Експериментально встановлено, що коефіцієнт розширення, з яким була отримана максимальна точність, дорівнює 2.

Подаємо на вхід кольорове зображення з нефіксованою роздільною здатністю
front-end (доналаштовані шари з VGG-16)
conv3-64-1 conv3-64-1
max-pooling
conv3-128-1 conv3-128-1
max-pooling
conv3-256-1 conv3-256-1 conv3-256-1
max-pooling
conv3-512-1 conv3-512-1 conv3-512-1
back-end
conv3-512-2 conv3-512-2 conv3-512-2 conv3-256-2 conv3-128-2 conv3-64-2
conv1-1-1

Рисунок 2.5 – Архітектура мережі CSRNet

Відображення графіку втрат на етапах навчання та перевірки для усієї мережі (рис. 2.6).

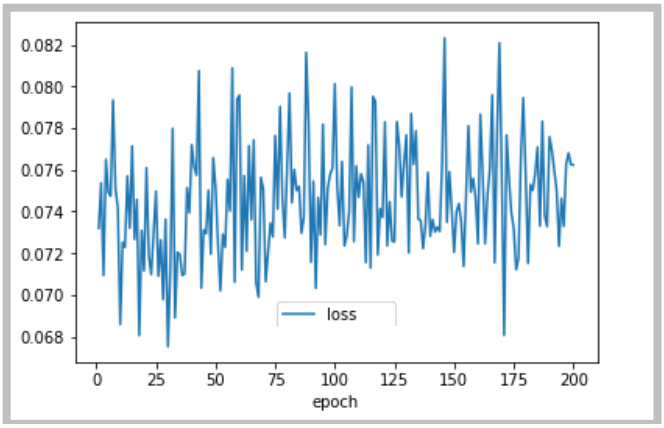

Рисунок 2.6 – Графік втрат на етапах навчання та перевірки мережі CSRNet

2.1.3.1 Розширена згортка

Однією з найважливіших складових мережі CSRNet є розширений згортковий шар [31]. Двовимірна розширена згортка може бути визначена таким чином:

$$y(m,n) = \sum_{i=1} \sum_{j=1} x(m + r \times i, n + r \times j) w(i, j), \tag{2.6}$$

де $x(m,n)$ – вхідне зображення;

$y(m,n)$ – це вихід розширеної згортки зображення $x(m,n)$;

$w(i, j)$ – фільтр з довжиною M та шириною N;

r – параметр швдкості розширення згортки.

Шари розширеної згортки були продемонстровані в завдання сегментації зі значним підвищенням точності [32] та є гарною альтернативою шарам пулінгу. Незважаючи на те, що шари пулінгу (наприклад, max pooling) широко використовуються для збереження інваріантності та контролю перенавчання, вони також різко зменшують просторову роздільну здатність, що означає, що просторова інформація карти ознак втрачається. Для

створення бекенд частини мережі CSRNet було обрано розширену згортку для адаптування ядра згортки 3×3 до розширеного ядра з коефіцієнтом = 2 (рис. 2.7). Вихідні дані мають той самий розмір, що і вхідні дані. Найголовніше, що результат розширеної згортки містить більш детальну інформацію (посилаючись на ті частини, які ми збільшуємо).

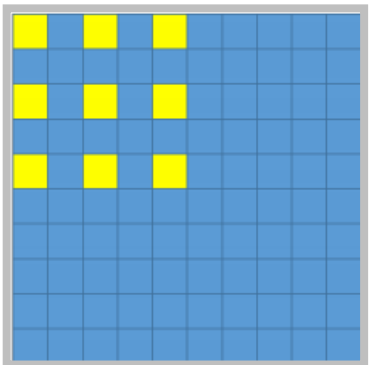

Рисунок 2.7 – Приклад розширеної згортки з ядром 3x3 та з параметром розширення 2

2.1.4 Алгоритм підрахунку людей на основі нейронних мереж

Для створення якісних методів підрахунку людей у натовпі на основі загорткових нейронних мереж необхідно виконати наступні етапи.

Попередня обробка даних. Оскільки CNN потрібно навчити оцінювати карту щільності натовпу за вхідним зображенням, якість щільності, наведена у навчальних даних, дуже впливає на ефективність нашого методу. Спочатку зображення з анотованими головами людей перетворюються на карту щільності натовпу. Якщо в пікселі x_i є голова, вона повинна бути представлена як її дельта-функція $\delta(x-x_i)$. Отже, зображення з позначеними N головами може бути представлене як функція:

$$H(x)=\sum_{i=1}^{N}\delta(x-x_i). \tag{2.7}$$

Щоб перетворити це на безперервну функцію щільності, можна згорнути цю функцію з ядром Гауса [33] $G\sigma$ так, щоб щільність становила $F(x) = H(x) * G\sigma(x)$. Однак, така функція щільності передбачає, що ці хі є незалежними зразками в площині зображення, що у дійсності не так: насправді кожен хі є зразком щільності натовпу на землі в 3D-сцені та через спотворення перспективи, а пікселі, пов'язані з різними зразками x_i, відповідають областям різного розміру у сцені. Тому, щоб точно оцінити щільність натовпу F, потрібно взяти до уваги спотворення, викликані гомографією між площиною землі та площиною зображення. На жаль, для даного завдання (і наборів даних) ми, як правило, геометрії сцени невідома. Проте, якщо взяти область навколо кожної голови, натовп розподілений дещо рівномірно, то середня відстань між головою та її найближчими k-сусідами (на зображенні) дає обгрунтовану оцінку геометричних спотворень (спричинених ефектом перспективи).

Тому необхідно визначати параметр поширення σ виходячи з розміру голови для кожної людини на зображенні. Однак на практиці майже неможливо точно отримати розмір голови через оклюзію у багатьох випадках, а також важко знайти основну залежність між розміром голови і картою щільності. Цікаво, що зазвичай розмір голови залежить від відстані між центрами двох сусідніх людей у людних сценах (рис. 2.8). Як компроміс для карти щільності цих скупчених сцен, пропонується, щоб дані адаптивно визначали параметр поширення для кожної людини виходячи із середньої відстані до сусідів.

Для кожної голови x_i на зображенні позначається відстань до її k найближчих сусідів $\{d_1^i, d_2^i, ..., d_m^i\}$. Отже, середня відстань дорівнює:

$$\bar{d}^i = \frac{1}{m}\sum_{j=1}^{m} d_j^i. \tag{2.8}$$

Таким чином, піксель, пов'язаний з x_i, відповідає області на землі у сцені приблизно в радіусі, пропорційному $\bar{d}^i$. Отже, щоб оцінити щільність натовпу навколо пікселя xi, потрібно згорнути $\delta(x - x_i)$ з ядром Гауса з дисперсією σi, пропорційною $\bar{d}^i$. Щільність *F* повинна бути

$$F(x) = \sum_{i=1}^{N} \delta(x - x_i) * G_\delta(x), \tag{2.9}$$

де $\delta = \beta \bar{d}^i$ для деякого параметра β.

Іншими словами, треба замінити мітки *H* з ядрами щільності, адаптивними до локальної геометрії навколо кожної точки даних, іменованими як геометрично адаптивні ядра. У нашому експерименті ми виявили, що емпірично $\beta = 0{,}3$ дає найкращий результат.

Рисунок 2.8 – Оригінальні зображення та відповідні карти щільності натовпу, отримані за допомогою загорткових геометрично-адаптивних ядер Гауса.

Попередня обробка даних. Щоб збільшити навчальний набір для навчання нейронних мереж, необхідно вирізати по 9 патчів з кожного зображення в різних місцях, розмір кожного патчу становить 1/4 вхідного зображення. Усі патчі використовуються для навчання моделей. Для частини A, оскільки щільність натовпу зазвичай дуже висока, використовуються геометрично-адаптивні ядра для створення карт щільності, і прогнозована щільність в області перекриття обчислюється шляхом усереднення. Для частини B, оскільки натовп відносно розріджений, використовується той самий розподіл в ядрі Гаусса для створення карт щільності (ground-truth).

Додаткова конфігурація мережі. Треба обрати функцію активації для моделей. У цьому випадку для обох мереж було обрано напівлінійну функцію активації ReLU (рис. 2.9) через її високу продуктивність для нейронних мереж. ReLU запобігає затуханню градієнта. Метод зворотного поширення помилки однаково добре працює і з нейронами, що знаходяться у кінці мережі і на самому її початку. Градієнт не стає неймовірно малим, що призводить до зупинки навчання мережі [34].

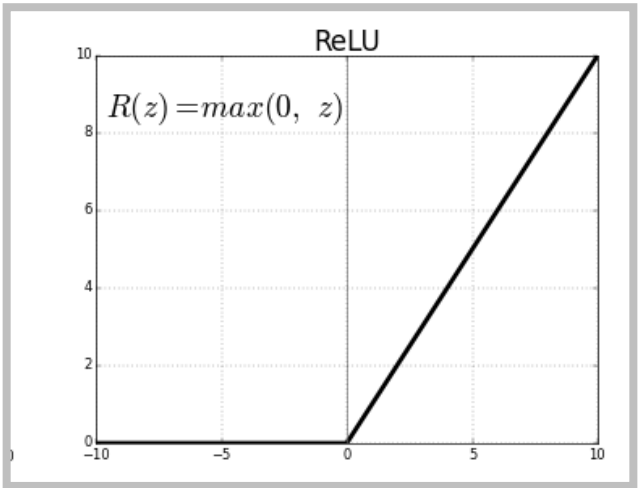

Рисунок 2.9 – Графік функції активації ReLU

Евклідова відстань використовується для вимірювання різниці між оцтриманою картою щільності та ground-truth. Функція встрат визначається наступним чином:

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} || Z(X_i;\theta) - Z_i^{GT} ||_2^2, \tag{2.10}$$

де N – розмір навчальної партії;

$Z(X_i;\theta)$ – вихід, що генерується CSRNet з параметрами, позначеними як θ;

X_i представляє вхідне зображення;

Z_i^{GT} – результат основних істин вхідного зображення X_i.

2.2 Підрахунок людей у натовпі на основі текстурних підходів

2.2.1 Математична модель підрахунку на основі локальних бінарних шаблонів

Метод локальних бінарних шаблонів порівнює інтенсивність восьми пік селів околиці з інтенсивністю обраного пікселя. Якщо це значення більше порогу або дорівнює йому, то піксель околиці приймає значення 1. Якщо ж менше, то призначаємо йому 0. Отримане восьмібітне число характеризує околицю обраного пікселя. Всього варіантів таких чисел може бути 2^8=256. Таким чином кожному пікселю зображення присвоюється одна з 256 міток, які характеризують його. За отриманими восьмибітними числами можна побудувати гістограму і використовувати її для опису текстури зображення [35].

На зображенні (рис. 2.10) можна побачити порівняння значення яскравості пікселя p і восьми його сусідів.

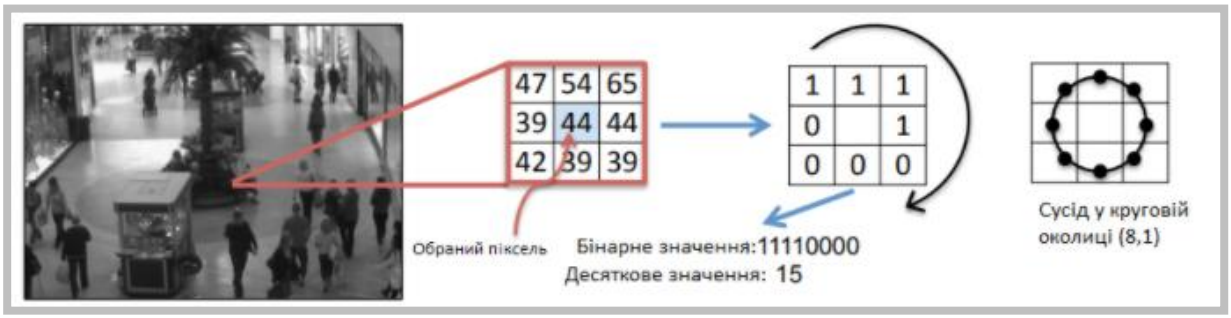

Рисунок 2.10 – Принцип роботи методу локальних бінарних шаблонів

LBP можна описати наступним чином:

$$C_i = \sum_{l=1}^{L} \sum_{w=1}^{W} z_{l,w},$$

$$s(x) = \begin{cases} 1 \ldots if \ldots x \geq \ldots 0 \\ 0 \ldots else \end{cases}, \qquad (2.11)$$

де (X_c, Y_c) – центральный піксель;

i_c – інтенсивність центрального пікселя;

i_n, де n $(n=0,1,2....7)$ – інтенсивність пікселів сусідніх пік селів;

P – кількість точок відбору.

На рисунках 2.11 – 2.13 приведено приклад роботи LBP.

Рисунок 2.11 – Вхідне зображення

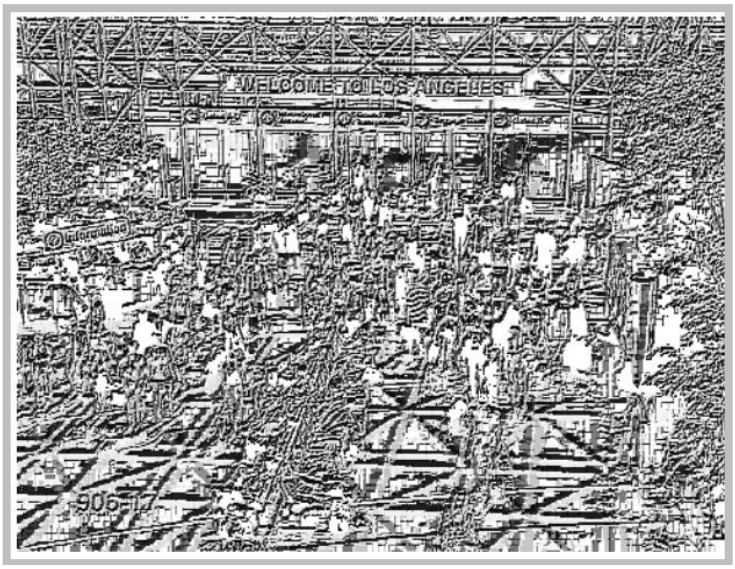

Рисунок 2.12 – Зображення LBP

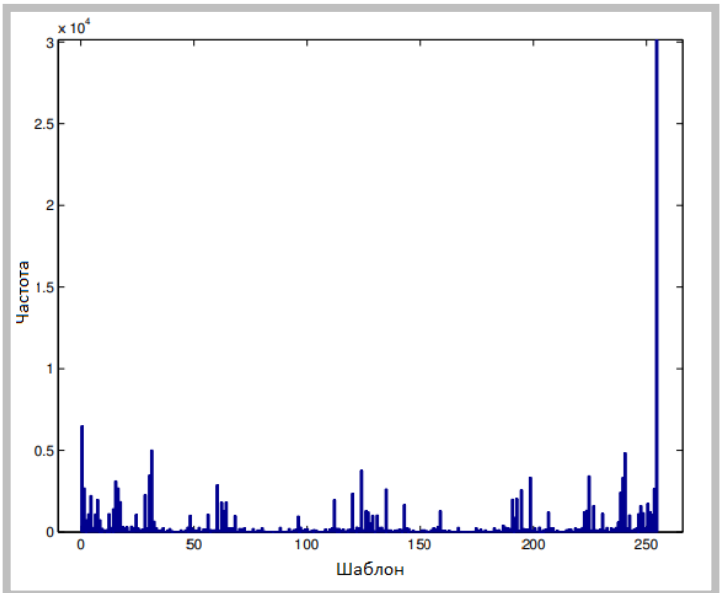

Рисунок 2.13 – Гістограма LBP

2.2.1.1 Рівномірні локальні бінарні шаблони

Корисним розширенням оператору локальних бінарних шаблонів є так званий рівномірний (uniform) шаблон [36], який може бути використаний для зменшення довжини вектору ознак та реалізації простого дескриптора інваріантного до обертання. Локальний бінарний шаблон називається рівномірним, якщо бінарний шаблон містить не більше двох переходів 0-1 або 1-0. Наприклад, 00010000 (2 переходи) є рівномірним шаблоном, а 01010100 (6 переходів) – ні. При обчисленні гістограми LBP вона має окремий інтервал для кожного рівномірного шаблону, а всі нерівномірні шаблони призначаються одному інтервалу. Кількість різних міток для околиці P-пікселів дорівнює $P(P - 1) + 3$, для околиці 3×3 – 59 унікальних шаблонів (58 для рівномірних шаблонів і 1 для всіх нерівномірних).

Отже, за допомогою рівномірних шаблонів довжина вектора ознак для однієї клітинки зменшується з 256 до 59.

На рисунках 2.14 та 2.15 показано різницю між рівномірними на нерівномірними шаблонами.

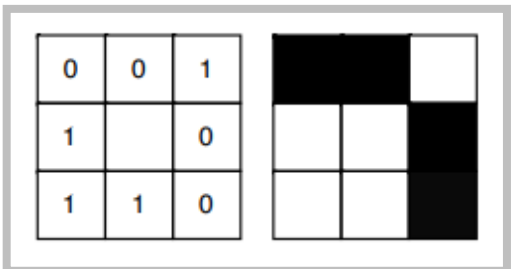

Рисунок 2.14 – Приклад нерівномірного шаблону

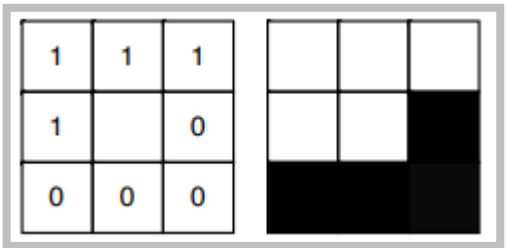

Рисунок 2.15 – Приклад рівномірного шаблону

2.2.2 Алгоритм на основі локальних бінарних шаблонів

Алгоритм передбачає роботу на основі ковзаючого вікна для підрахунку людей у натовпі. По-перше, треба розділити кожне зображення на клітинки (12 рядків та 16 стовпців), а потім обчислити особливості текстури у кожній клітинці. Далі ці характеристики об’єднуються для побудови одного вектора ознак, який є дескриптором текстури блоку зображення. Після обчислення всіх векторів характеристик блоків зображення використовується Ridge Regression для навчання моделі, що підраховує кількість людей у натовпі. Ridge Regression – це удосконалення лінійної регресії з підвищеною стійкістю до помилок, що накладає обмеження на коефіцієнти регресії для отримання куди більш наближеного до реальності результату. Ridge Regression використовує L2 регуляризацію:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2. \qquad (2.12)$$

Роботу алгоритму продемонстровано на рисунку 2.16.

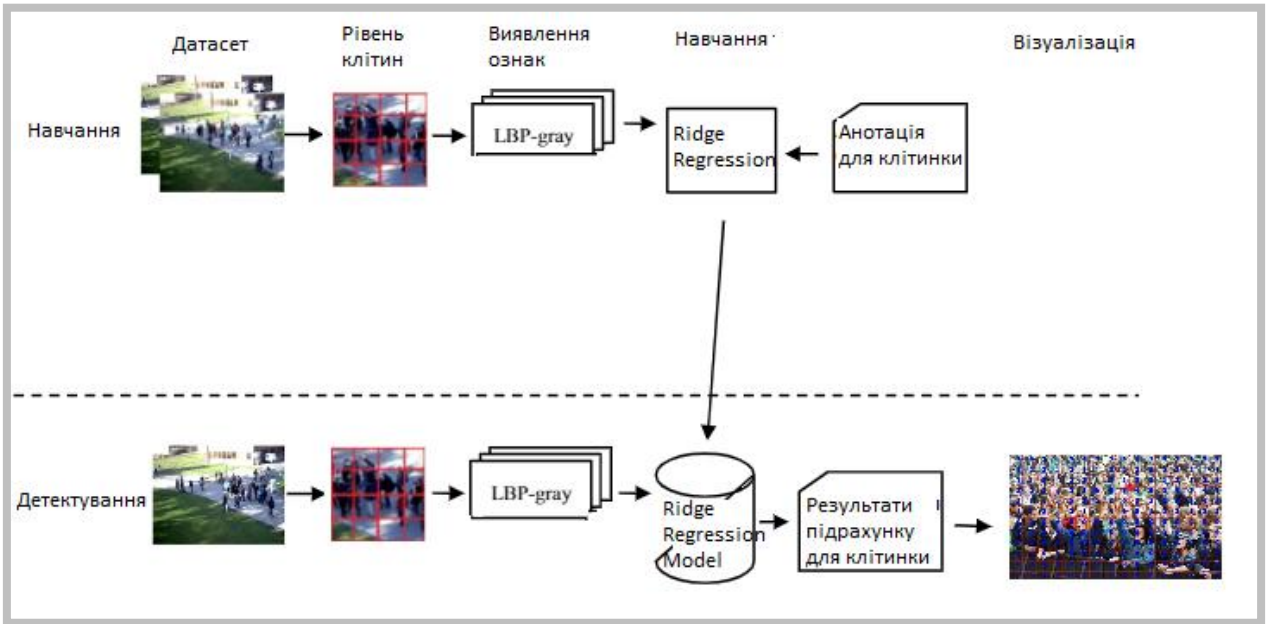

Рисунок 2.16 – Схема системи підрахунку людей у натовпі на основі LBP

2.2.2.2 Регуляризація у машинному навчанні

Одним з основних аспектів навчання моделі машинного навчання є уникнення перенавчання. Через ефект перенавчання модель матиме низьку точність. Це відбувається, тому що модель старанно намагається захопити шуми у навчальному наборі даних. Під шумами маються на увазі випадкові точки, які не відображають реальні властивості даних. Якщо мережа вивчить ці точки, то вона стане більш гнучкою та перенавчиться.

Регуляризація є одним із методів боротьби з перенавчанням [37].

Регуляризація – це форма регресії, яка стримує / регулює або зменшує оцінки коефіцієнтів до нуля. Іншими словами, цей метод перешкоджає навчанню більш складної або гнучкої моделі, щоб уникнути ризику перенавчання.

Регуляризація у загальному випадку модифікує функцію втрат таким чином, щоб мінімізувати її:

$$J(\theta) + \lambda R(\theta), \tag{2.13}$$

де $J(\theta)$ – функція втрат;

$\lambda R(\theta)$ – функція регуляризації.

Існують кілька регуляризаторів, які широко використовуються на практиці – L1-Regularization (LASSO) та L2-Regularization (Ridge).

L1-Regularization (LASSO). Вважається, що L1 регуляризація дає розріджені параметри, тобто такі моделі, у яких більшість вагів будуть нульовими. Завдяки цьому можна аналізувати, які ознаки для моделі важливі, а від яких можна позбутися: ті ознаки, які обнуляться пізно, можна вважати важливими.

Наприклад, ми маємо функцію втрат:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \ , \tag{2.14}$$

тоді її модифікація регуляризації LASSO виглядає наступним чином:

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} |\beta_j|. \tag{2.15}$$

Тепер коефіцієнти оцінюються шляхом мінімізації цієї функції. Тут λ є параметром налаштування, який визначає, наскільки ми хочемо штрафувати за прийняття великих коефіцієнтів. Це необхідно для того, щоб модель не виходила на нескінченність, де коефіцієнти стають великими. Таким чином ми дамо моделі пріорітети, що на такому нескінченному плато їй краще залишатися ближче до початку координат. Це дасть їй можливість навчатися сфокусовано у тій області, де найвірогідніше вона знайде найкращі точки. Збільшення гнучкості моделі представлено збільшенням її коефіцієнтів, і

якщо ми хочемо мінімізувати вищезгадану функцію, то ці коефіцієнти повинні бути малими.

Ця варіація регуляризації використовує | βj | (модуль), як штраф. У статистиці це відомо як норма L1. L1 – це Манхеттенська відстань до початку координат [38]. Траєкторія навчання моделі буде намагатися йти по осям і обнуляти деякі ваги (рис. 2.17).

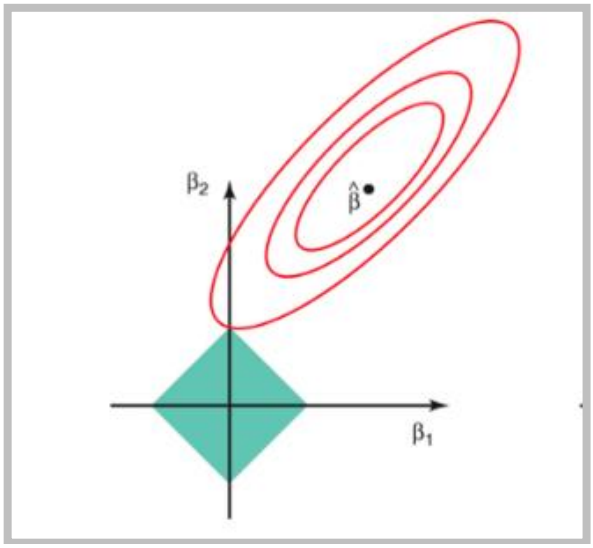

Рисунок 2.17 – Траєкторія навчання моделі при L1 регуяризації

L2-Regularization (Ridge). Регуляризація Ridge також направлена на мінімізацію функції втрат.

$$J(\beta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n}|\beta_j^{\,2}|. \qquad (2.16)$$

Такий вид регуляризації використовує квадрати β і є Евклідовою відстанню [39]. Можна побачити, що навчання моделі йде по діагональній траєкторії (рис. 2.18).

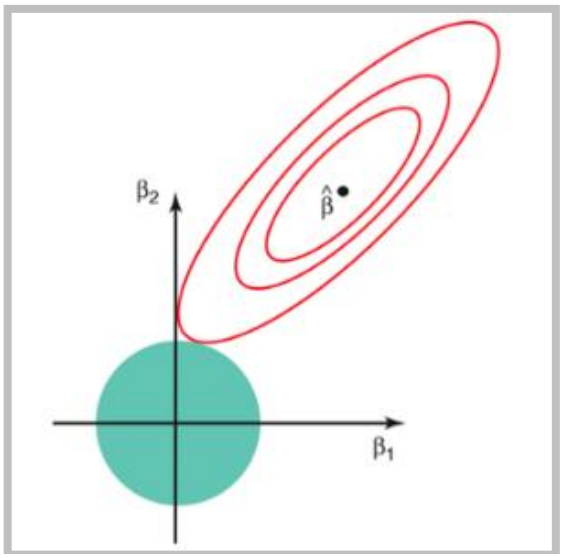

Рисунок 2.18 – Траєкторія навчання моделі при L2 регуяризації

Стандартна модель найменших квадратів має певну дисперсію, тобто ця модель не буде узагальненою для набору даних, відмінних від даних навчання. Регуляризация значно зменшує дисперсію моделі без істотного збільшення її зміщення. Таким чином, параметр налаштування λ, який використовується в описаних вище методиках регуляризації, контролює вплив на зміщення і дисперсію. При зростанні величини λ вона зменшує значення коефіцієнтів і, таким чином, зменшує дисперсію. До певної точки це збільшення λ є корисним, оскільки воно лише зменшує дисперсію (отже, ми уникаємо перенавчання), не втрачаючи важливих властивостей даних. Але після певного значення модель починає втрачати важливі властивості, що призводить до упередженості в моделі і, отже, до недосконалості. Тому значення λ повинно бути ретельно відібрано.

2.3 Міри оцінювання результатів підрахунку людей у натовпі

Для оцінки роботи методів підрахунку людей у натовпі можна використовувати середню абсолютну помилку (MAE) та середньоквадратичну помилку (MSE) [40].

MSE задається наступним чином:

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} | C_i - C_i^{GT} |^2}. \quad (2.17)$$

MAE задається наступним чином:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} | C_i - C_i^{GT} | , \quad (2.18)$$

де N – це кількість тестових зображень в одній тестовій послідовності;

C_i^{GT} – це ground_truth підрахунку;

C_i – це передбачувана кількість людей для зображення X, яка визначається наступним чином:

$$C_i = \sum_{l=1}^{L}\sum_{w=1}^{W} z_{l,w}, \quad (2.19)$$

де L та W – це довжина та ширина мапи щільності;

$z_{l,w}$ – це позначка наявності людини у точці (l,w).

Також для оцінки роботи методів можна використовувати метрики precision (точність) та recall (повнота) [41].

Precision та рекол задаютья наступним чином:

$$PRECISION = \frac{t_p}{t_p + f_p}, \quad (2.20)$$

$$RECALL = \frac{t_p}{t_p + f_n}, \quad (2.21)$$

де t_p – істинно-позитивні результати,

f_n – помилково-негативні результати.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТА АЛГОРИТМІВ ПІДРАХУНКУ ЛЮДЕЙ У НАТОВПІ ТА ЇХ ПОРІВНЯЛЬНИЙ АНАЛІЗ

3.1 Опис датасету ShanghaiTech

Набір даних ShanghaiTech складається з двох частин: у частині A є 482 зображення, які випадковим чином обрано з Інтернету, та 716 зображень у частині B, зроблені зі жвавих вулиць ті метрополітену у Шанхаї. Щільність натовпу значно варіюється між цима двома підмножинами (рис. 3.1). І Частина А, і Частина Б поділяються на набори для навчання та тестування: 300 зображень Частини А використовуються для навчання та решта 182 зображення для тестування; і 400 зображень частини В призначені для навчання та 316 для тестування.

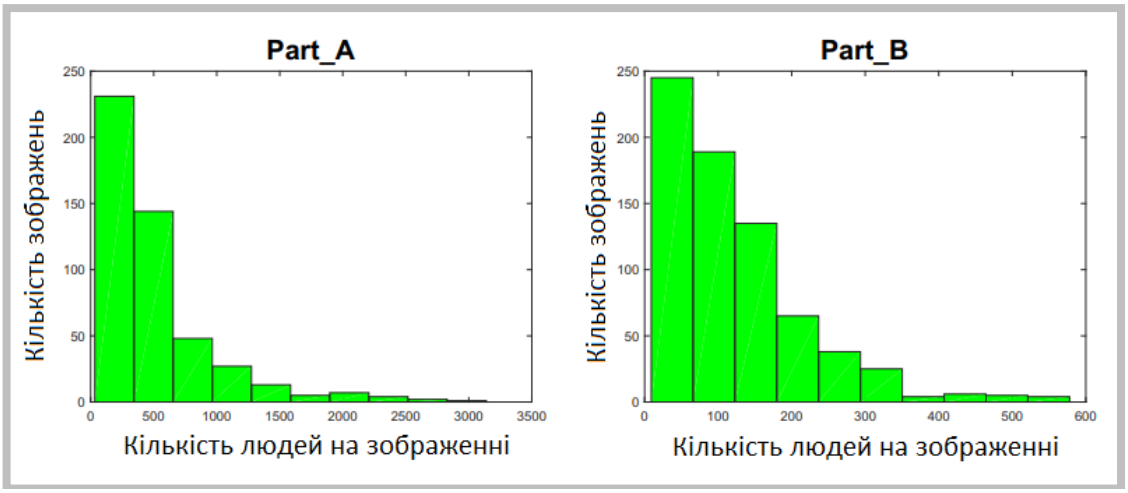

Рисунок 3.1 – Графік платності частин дата сету part_A та part_B

3.2 Налаштування програмного середовища

Google Colaboratory з режимом GPU було використано як програмне середовище, програмний код написано на мові Python 3.6.9. У таблиці 3.1 представлені основні використовувані бібліотеки та їх версії.

Таблиця 3.1 – Бібліотеки, що використовувалися для написання програмного коду

Назва бібілотеки	Версія бібліотеки
OpenCv	4.4.0
Scikit-image	0.17.2
Scikit-learn	0.23.2
SciPy	1.5.2
Torch	1.2.0 + cu92
Torchvision	0.4.0 + cu92
NumPy	1.19.1
H5py	2.10.0
Matplotlib	3.3.1

3.3 Підготовка датасету для роботи зі загортковими нейронними мережами

Для початку необхідно створити зображення ground-truth. На рисунках 3.2 – 3.4 показано створення ground-truth для зображень з частини датасету part_A.

```
part_A_train = os.path.join(root,'part_A_final/train_data','images')
part_A_test = os.path.join(root,'part_A_final/test_data','images')
path_sets = [part_A_train,part_A_test]
```

Рисунок 3.2 – Встановлення шляхів до папок з зображеннями

Створимо функцію Гаусівського розмиття, що буде примінятися до зображень для створення карт щільності (рис. 3.3). Цей код використовує

метод *k*-найближчих сусідів, він займає одну хвилину або більше, щоб створити карту щільності для зображення з тисячею людей.

```
def gaussian_filter_density(gt):
    print gt.shape
    density = np.zeros(gt.shape, dtype=np.float32)
    gt_count = np.count_nonzero(gt)
    if gt_count == 0:
        return density

    pts = np.array(zip(np.nonzero(gt)[1], np.nonzero(gt)[0]))
    leafsize = 2048
    tree = scipy.spatial.KDTree(pts.copy(), leafsize=leafsize)
    distances, locations = tree.query(pts, k=4)

    print 'розмиття для...'
    for i, pt in enumerate(pts):
        pt2d = np.zeros(gt.shape, dtype=np.float32)
        pt2d[pt[1],pt[0]] = 1.
        if gt_count > 1:
            sigma = (distances[i][1]+distances[i][2]+distances[i][3])*0.1
        else:
            sigma = np.average(np.array(gt.shape))/2./2. #case: 1 point
        density += scipy.ndimage.filters.gaussian_filter(pt2d, sigma, mode='constant')
    print 'done.'
    return density
```

Рисунок 3.3 – Гаусівське розмиття

Перетворимо зображення з анотованими центрами голів людей у ground-truth карти щільності (рис. 3.4).

```
img_paths = []
for path in path_sets:
    for img_path in glob.glob(os.path.join(path, '*.jpg')):
        img_paths.append(img_path)
for img_path in img_paths:
    print img_path
    mat = io.loadmat(img_path.replace('.jpg','.mat').replace('images','ground_truth').replace('IMG_','GT_IMG_'))
    img= plt.imread(img_path)
    k = np.zeros((img.shape[0],img.shape[1]))
    gt = mat["image_info"][0,0][0,0][0]
    for i in range(0,len(gt)):
        if int(gt[i][1])<img.shape[0] and int(gt[i][0])<img.shape[1]:
            k[int(gt[i][1]),int(gt[i][0])]=1
    k = gaussian_filter_density(k)
    with h5py.File(img_path.replace('.jpg','.h5').replace('images','ground_truth'), 'w') as hf:
            hf['density'] = k
```

Рисунок 3.4 – Перетворення зображень у карти щільності

3.4 Побудова нейронної мережі MCNN з використанням бібліотеки PyTorch

Створимо структуру загорткової нейронної мережі MCNN, використовуючи бібліотеку PyTorch (рис. 3.5, рис. 3.6).

Для початку завантажимо необхідні бібліотеки.

```
import torch
import torch.optim as optim
import torch.nn as nn
```

Рисунок 3.5 – Імпорт з PyTorch необхідних модулів для навчання мережі

Для побудови структури мережі використаємо Sequential модель. Додамо згорткові шари nn.Conv2d. Ці шари застосовують 2D-згортку до вхідного сигналу, що складається з декількох вхідних площин. Шари nn.MaxPool2d застосовуються, як шари пулінгу, що виконують 2D пулінг над вхідним сигналом. Щоб додати у структуру функцію активації, використовується nn.ReLU.

```
def __init__(self,load_weights=False):
        super(MCNN,self).__init__()
        self.branch1=nn.Sequential(
            nn.Conv2d(3,16,9,padding=4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(16,32,7,padding=3),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(32,16,7,padding=3),
            nn.ReLU(inplace=True),
            nn.Conv2d(16,8,7,padding=3),
            nn.ReLU(inplace=True)
        )
        self.branch2=nn.Sequential(
            nn.Conv2d(3,20,7,padding=3),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(20,40,5,padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(40,20,5,padding=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(20,10,5,padding=2),
            nn.ReLU(inplace=True)
        )
        self.branch3=nn.Sequential(
            nn.Conv2d(3,24,5,padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(24,48,3,padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(48,24,3,padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(24,12,3,padding=1),
            nn.ReLU(inplace=True)
        )
        self.fuse=nn.Sequential(nn.Conv2d(30,1,1,padding=0))
```

Рисунок 3.6 – Задаємо стуктуру мережі MCNN використовуючи мову Python

Виводимо для наглядності детальну структуру побудованої мережі MCNN та її параметри (рис. 3.7).

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, None, None,	0	
conv2d_1 (Conv2D)	(None, None, None, 2	1000	input_1[0][0]
conv2d_5 (Conv2D)	(None, None, None, 2	624	input_1[0][0]
conv2d_9 (Conv2D)	(None, None, None, 1	1312	input_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, None, None, 2	0	conv2d_1[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, None, None, 2	0	conv2d_5[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, None, None, 1	0	conv2d_9[0][0]
conv2d_2 (Conv2D)	(None, None, None, 4	20040	max_pooling2d_1[0][0]
conv2d_6 (Conv2D)	(None, None, None, 4	10416	max_pooling2d_3[0][0]
conv2d_10 (Conv2D)	(None, None, None, 3	25120	max_pooling2d_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, None, None, 4	0	conv2d_2[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, None, None, 4	0	conv2d_6[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, None, None, 3	0	conv2d_10[0][0]
conv2d_3 (Conv2D)	(None, None, None, 2	20020	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, None, None, 2	10392	max_pooling2d_4[0][0]
conv2d_11 (Conv2D)	(None, None, None, 1	25104	max_pooling2d_6[0][0]
conv2d_4 (Conv2D)	(None, None, None, 1	5010	conv2d_3[0][0]
conv2d_8 (Conv2D)	(None, None, None, 1	2604	conv2d_7[0][0]
conv2d_12 (Conv2D)	(None, None, None, 8	6280	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, None, None, 3	0	conv2d_4[0][0] conv2d_8[0][0] conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, None, None, 1	31	concatenate_1[0][0]

Total params: 127,953
Trainable params: 127,953
Non-trainable params: 0

Рисунок 3.7 – Детальна структура нейронної мережі MCNN та її параметри

3.5 Побудова нейронної мережі CSRNet з використанням бібліотеки PyTorch

Створимо структуру загорткової нейронної мережі CSRNet, використовуючи бібліотеку PyTorch (рис. 3.8 – рис. 3.10).

Для початку завантажимо необхідні бібліотеки.

```
import torch
import torch.optim as optim
import torch.nn as nn
```

Рисунок 3.8 – Імпорт з PyTorch необхідних модулів для навчання мережі

Задаємо дві частини структури мережі frontend_feat та backend_feat (рис. 3.9).

```
class CSRNet(nn.Module):
    def __init__(self, load_weights=False):
        super(CSRNet, self).__init__()
        self.frontend_feat = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512]
        self.backend_feat  = [512, 512, 512,256,128,64]
        self.frontend = make_layers(self.frontend_feat)
        self.backend = make_layers(self.backend_feat,in_channels = 512,dilation = True)
        self.output_layer = nn.Conv2d(64, 1, kernel_size=1)
```

Рисунок 3.9 – Задаємо стуктуру мережі CSRNet використовуючи мову Python

Далі необхідно необхідно описати виклик попередньо навченої мережі VGG-16, що виконує роль frontend.

```
if not load_weights:
           mod = models.vgg16(pretrained = True)
           self._initialize_weights()
           for i in xrange(len(self.frontend.state_dict().items())):
               self.frontend.state_dict().items()[i][1].data[:] = mod.state_dict().items()[i][1].data[:]
```

Рисунок 3.10 – Виклик попередньо навченої мережі VGG-16

Виводимо для наглядності детальну структуру побудованої мережі CSRNet та її параметри (рис. 3.11).

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, None, None, 64)	1792
conv2d_2 (Conv2D)	(None, None, None, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, None, None, 64)	0
conv2d_3 (Conv2D)	(None, None, None, 128)	73856
conv2d_4 (Conv2D)	(None, None, None, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, None, None, 128)	0
conv2d_5 (Conv2D)	(None, None, None, 256)	295168
conv2d_6 (Conv2D)	(None, None, None, 256)	590080
conv2d_7 (Conv2D)	(None, None, None, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, None, None, 256)	0
conv2d_8 (Conv2D)	(None, None, None, 512)	1180160
conv2d_9 (Conv2D)	(None, None, None, 512)	2359808
conv2d_10 (Conv2D)	(None, None, None, 512)	2359808
conv2d_11 (Conv2D)	(None, None, None, 512)	2359808
conv2d_12 (Conv2D)	(None, None, None, 512)	2359808
conv2d_13 (Conv2D)	(None, None, None, 512)	2359808
conv2d_14 (Conv2D)	(None, None, None, 256)	1179904
conv2d_15 (Conv2D)	(None, None, None, 128)	295040
conv2d_16 (Conv2D)	(None, None, None, 64)	73792
conv2d_17 (Conv2D)	(None, None, None, 1)	65

Total params: 16,263,489
Trainable params: 16,263,489

Рисунок 3.11 – Детальна структура нейронної мережі CSRNet та її параметри

3.6 Тренування нейронних мереж MCNN та CSRNet

Для того щоб натренувати побудовану мережу, необхідно задати параметри тренування.

Визначимо функцію втрат та оптимізатор (рис. 3.12). Функцією втрат виступає nn.MSELoss, що вимірює середню квадратичну помилку (квадрат L2 норми) між кожним елементом на вході *x* і цільовим *y*. Як оптимізатор для функції втрат використовується SDG (momentum). Momentum – гіперпараметр > = 0, що прискорює градієнтний спуск у відповідному напрямку та зменшує коливання. Коефіцієнт швидкості навчання становить 1e-6.

```
criterion=nn.MSELoss(size_average=False).to(device)
optimizer = torch.optim.SGD(mcnn.parameters(), lr=1e-6,
                                momentum=0.95)
```

Рисунок 3.12 – Визначення функції втрат та оптимізатору

Задамо параметри для навчання мережі (рис. 3.13). optimizer.step виконує оновлення параметрів на основі поточного градієнта (зберігається в атрибуті .grad параметра) та правила оновлення. Виклик .backward() безліч разів накопичує градієнт (шляхом додавання) для кожного параметра. Тому optimizer.zero_grad() викликається після кожного виклику .step().

```
min_mae=10000
min_epoch=0
train_loss_list=[]
epoch_list=[]
test_error_list=[]
for epoch in range(0,200):

    mcnn.train()
    epoch_loss=0
    for i,(img,gt_dmap) in enumerate(dataloader):
        img=img.to(device)
        gt_dmap=gt_dmap.to(device)
        et_dmap=mcnn(img)
        loss=criterion(et_dmap,gt_dmap)
        epoch_loss+=loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    epoch_list.append(epoch)
    train_loss_list.append(epoch_loss/len(dataloader))
    torch.save(mcnn.state_dict(),'./checkpoints/epoch_'+str(epoch)+".param")
```

Рисунок 3.13 – Параметри для тренування мережі MCNN

3.7 Створення алгоритму підрахунку людей у натовпі на основі LBP

Завантажимо необхідні бібліотеки для побудови алгоритму (рис. 3.14).

```
import scipy.io as sio
import cv2
import argparse
import utils as utils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pdb
from skimage import feature
import numpy as np
import os
```

Рисунок 3.14 – Імпорт необхідних бібліотек для створення алгоритму на основі LBP

Для початку необхідно завантажити зображення з анотованою кількістю людей і перевести їх у монохромні зображення, якщо вони такими не є (рис. 3.15).

```
height,width,numChannels=image.shape
if(numChannels==3):
    grayImage=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
else:
    grayImage=rawImage
nbPointInImage=[]
widthOfPatch=int(width/float(numberOfRectanglePerRow))
heightOfPatch=int(height/float(numberOfRectanglePerColumn))
allHistInImage=[]
```

Рисунок 3.15 – Переведення зображень у монохромні

Створюємо функцію для обчислення LBP зображення, що потім будуть використовуватися для побудови гістограми (рис. 3.16). numPoints – кількість циклічно симетричних сусідніх точок (квантування кутового простору), а radius – радіус кола (просторова роздільна здатність оператора). У параметрі method вказано «uniform» більш тонке квантування кутового простору, яке є та інваріантним до обертання та відтінків сірого.

```
class LocalBinaryPatterns:
    def __init__(self, numPoints, radius):
        self.numPoints = numPoints
        self.radius = radius

    def describe(self, image, eps=1e-7):
        lbp = feature.local_binary_pattern(image, self.numPoints,
                self.radius, method="uniform")
        (hist, _) = np.histogram(lbp.ravel(),
                bins=np.arange(0, 60))
        hist = hist.astype("float")
        hist /= (hist.sum() + eps)

        return hist
```

Рисунок 3.16 – Функція для обчислення LBP та побудови гістограми

Далі розділяємо зображення на 12 строк та 16 клітинок і підраховуємо у кожній їз них кількість людей. Потім додаємо знайдену кількість у вектор, що містить кількістю людей у кожній клітинці (рис. 3.17).

```
for i in range(numberOfRectanglePerColumn):
        for j in range(numberOfRectanglePerRow):
            nbPoint,headPointsInPatch=getNumberOfPointsInsideRectangle(points,rectangle={"anchor":(i*heightOfPatch,j*widthOfPatch),
            "width":widthOfPatch,"height":heightOfPatch})
            nbPointInImage.append(nbPoint)
```

Рисунок 3.17 – Підрахунок людей у клітинках

Підраховуємо гістограму дескриптору LBP (uniform) для кожної клітинки та об’єднуємо поточний 59D дескриптор з дескрипторами клітинок, які ми порахували раніше (рис. 3.18).

```
hist=lbp.describe(grayImage[i*heightOfPatch:(i+1)*heightOfPatch,j*widthOfPatch:(j+1)*widthOfPatch], eps=1e-7)
allHistInImage.append(hist)
```

Рисунок 3.18 – Об’єднання дескрипторів клітинок

Поєднуємо усі дескриптори усіх клітинок у один вектор (рис. 3.19).

```
allHistInImage=np.concatenate(allHistInImage, axis=0)
```

Рисунок 3.19 – Об’єднання усіх дескрипторів усіх клітинок у один вектор

На наступному кроці необхідно поєднати всі дескриптори в одну матрицю та всі вектори кількості точок в одну матрицю, а також зберегти отриману матрицю дескрипторів та матрицю ground-truth (рис. 3.20).

```
        if(allPoints is None):
            allPoints=np.array(nbPointInImage)
        else:
            allPoints=np.vstack((allPoints,nbPointInImage))

        if(allHist is None):
            allHist=allHistInImage

        else:
            allHist=np.vstack((allHist,allHistInImage))

sio.savemat(os.path.join(args.data_root,mode+".mat"),{"lbp_descriptors":allHist,"labels":allPoints})
```

Рисунок 3.20 – Збереження отриманих матриць

3.8 Навчання алгоритму підрахунку людей у натовпі на основі LBP

Підключємо необхідні модулі з бібліотеки sklearn (рис. 3.21).

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import make_scorer
```

Рисунок 3.21 – Завантаження дескрипторів і ground-truth для тренувального набору даних

Завантажуємо дескриптори і ground-truth для набору даних для навчання (рис. 3.22).

```
matFile=sio.loadmat(os.path.join(args.data_root,"train"))
descriptorsTrain=matFile["lbp_descriptors"]
labelsTrain=matFile["labels"]
```

Рисунок 3.22 – Завантаження дескрипторів і ground-truth для тренувального набору даних

Завантажуємо дескриптори і ground-truth для набору даних для тестування (рис. 3.23).

```
matFile2=sio.loadmat(os.path.join(args.data_root,"test"))
descriptorsTest=matFile2["lbp_descriptors"]
labelsTest=matFile2["labels"]
```

Рисунок 3.23 – Завантаження дескрипторів і ground-truth для тестового набору даних

Реалізуємо ridge regression. KRR поєднує RR (звичайну регресію з регуляризацією l2-норми) та застосування ядра (рис. 3.24). Таким чином, метод вивчає лінійну функцію в просторі, викликану відповідним ядром та даними. Для нелінійних ядер це відповідає нелінійній функції у вихідному просторі.

```
Grid_Dict = {"alpha": [1e-13,1e-5,1e-4,1e-3,1e-2],"gamma": np.logspace(-3, 2, 10)}
krr_Tuned = GridSearchCV(KernelRidge(kernel='rbf'), cv=4 ,param_grid=Grid_Dict,
                         scoring=make_scorer(computeError,greater_is_better=False),refit=True)
krr_Tuned.fit(descriptorsTrain, labelsTrain)
alpha=args.alpha
gamma=args.gamma

KRR = KernelRidge(kernel='rbf', alpha=alpha, gamma=gamma)
KRR.fit(descriptorsTrain, labelsTrain)

predictionsTrain=KRR.predict(descriptorsTrain)

predictionsTest=KRR.predict(descriptorsTest)
```

Рисунок 3.24 – Реалізація ridge regression

3.9 Ілюстрація підрахунку людей у натовпі за допомогою загорткових нейронних мереж MCNN,CSRNet та методу заснованому на застосуванні локальних бінарних шаблонів

Подамо на вхід до кожного методу одне й те ж саме зображення натовпу з частини дата сету part_A (рис. 3.25). На цьому зображені анотована 371 людина. Покажемо реальну кількість людей на розміченому зображенні

(рис. 3.26) та ground-truth тестового зображення (рис. 3.27). Після роботи трьох методів отримаємо результати підрахунку (рис. 3.28 – рис. 3.32).

Рисунок 3.25 – Зображенння натовпу за частини part_A, що подається на вхід

Рисунок 3.26 – Реальна кількість людей на розміченому зображенні

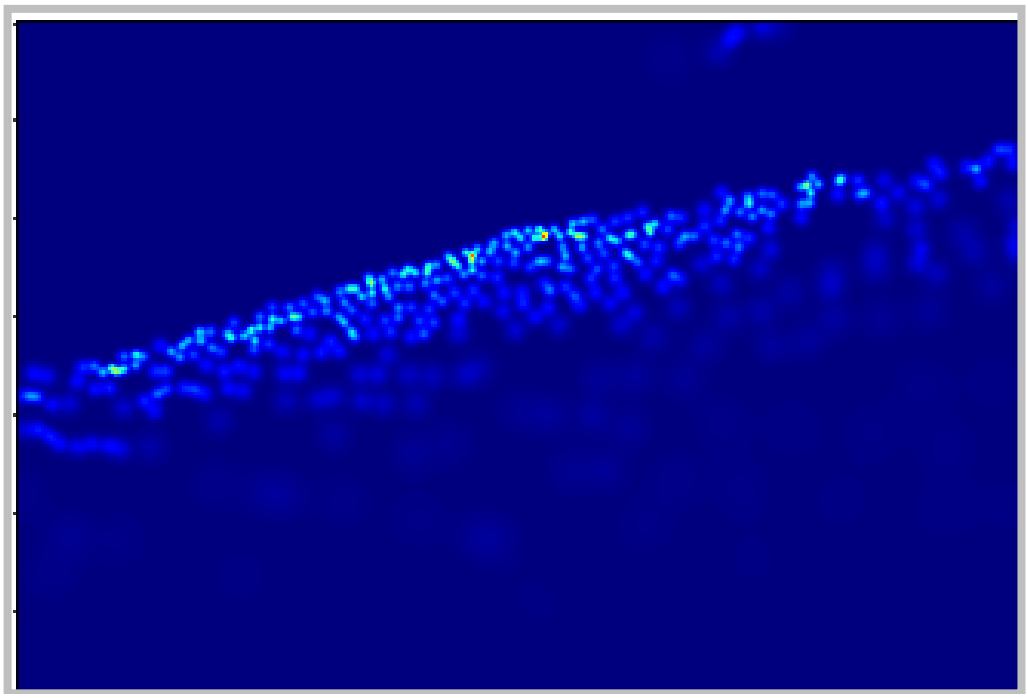

Рисунок 3.27 – Ground-truth зображенння натовпу, що подається

Спочатку розглянемо результат, який видає нейронна мережа MCNN (рис. 3.28 – рис. 3.29). Вона знайшла на зображенні 598 людей, що на 227 більше, ніж було анотовано.

Рисунок 3.28 – Результат підрахунку мережі MCNN

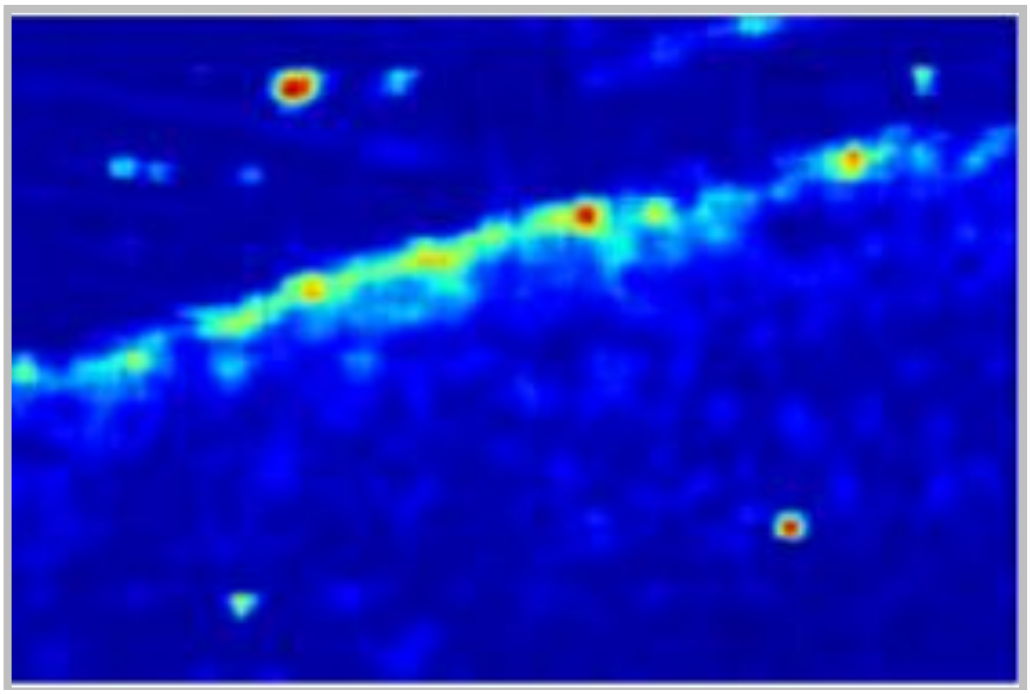

Рисунок 3.29 – Карта щільності створена мережею MCNN

Далі розглянемо нейронну мережу CSRNet, яка підрахувала на зображенні 438 людей, що більше анотованої кількості на 67 (рис. 3.30, рис. 3.31).

Рисунок 3.30 – Результат підрахунку мережі CSRNet

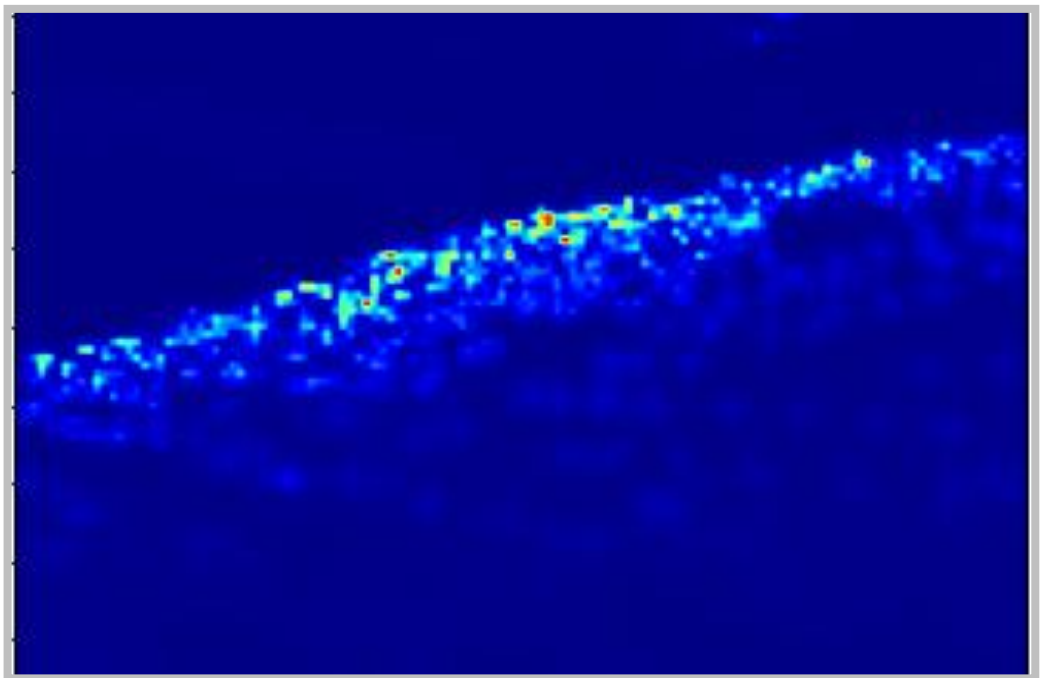

Рисунок 3.31 – Карта щільності створена мережею CSRNet

Метод, що створено на основі локальних бінарних шаблонів, дає найменш точний результат (рис. 3.32). На зображені було знайдено 630 людей, що на 259 більше, ніж дійсно є на зображенні.

Рисунок 3.32 – Результати підрахунку алгоритму на основі методу локальних бінарних шаблонів

Далі протестуємо створені методи на зображенні з частини датасету part_B (рис. 3.33). На цьому зображенні дійсно присутні 252 особи (рис. 3.34, рис. 3.35). Отримуємо результати роботи трьох методів (рис. 3.28 – рис. 3.32).

Рисунок 3.33 – Зображенння натовпу з частини part_B, що подається на вхід

Рисунок 3.34 – Реальна кількість людей на розміченому зображенні

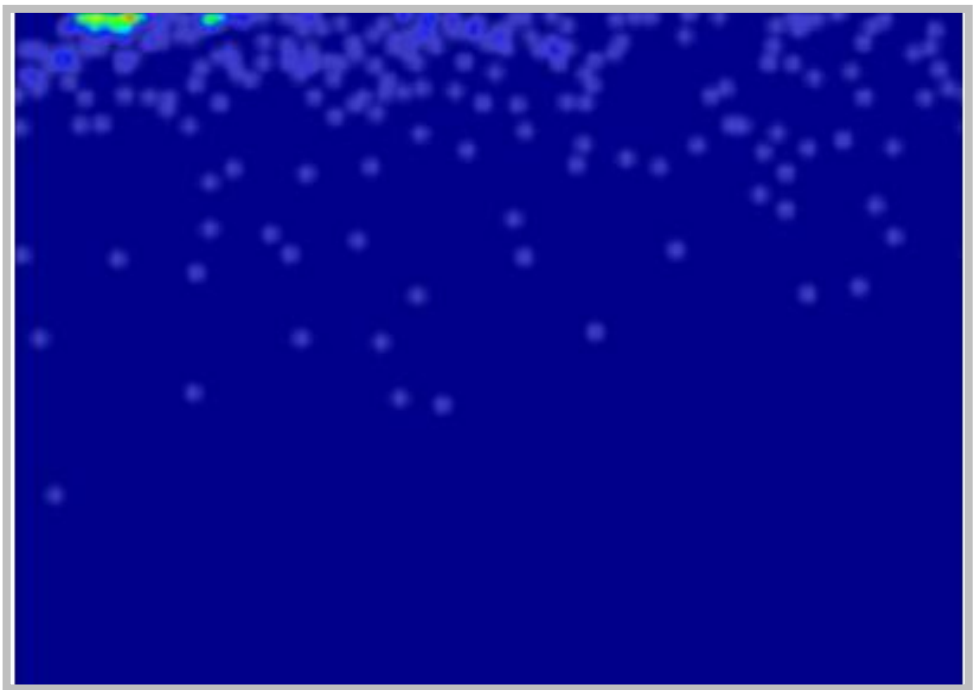

Рисунок 3.35 – Ground-truth зображенння натовпу, що подається

Розглянемо результати отримані мережею MCNN (рис. 3.36, рис. 3.37). Вихід даної мережі 221 знайдена людина, що на 30 людей менше, ніж було анотовано на зображенні.

Рисунок 3.36 – Результат підрахунку мережі MCNN

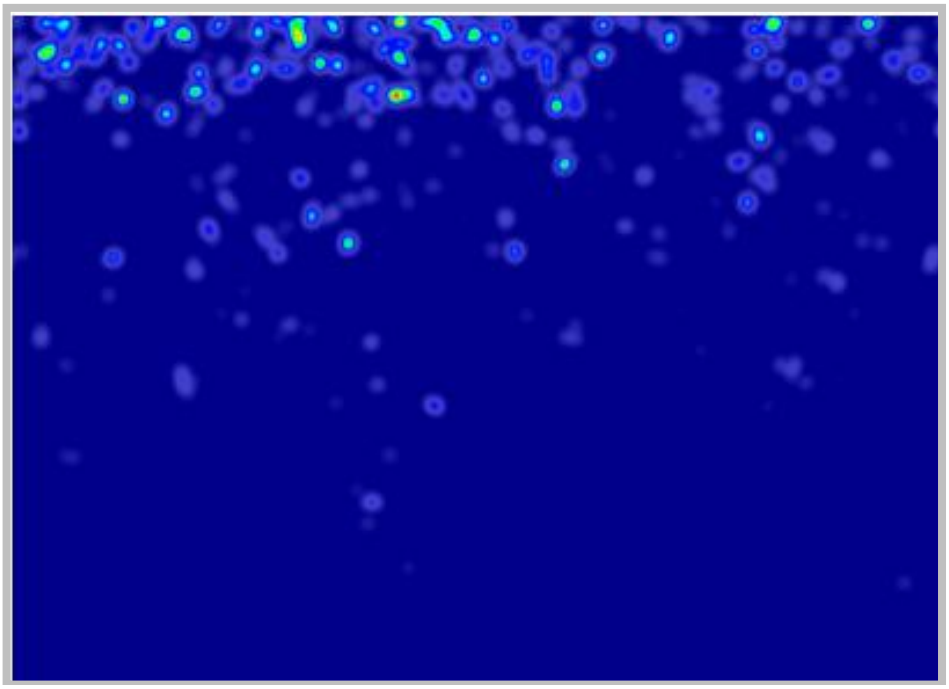

Рисунок 3.37 – Карта щільності створена мережею MCNN

Проаналізуємо, які результати показала мережа CSRNet (). Вона знову демонструє кращий результат 242 людини з 252 анотованих, усього на 10 осіб менше (рис. 3.38, рис. 3.39).

Рисунок 3.38 – Результат підрахунку мережі CSRNet

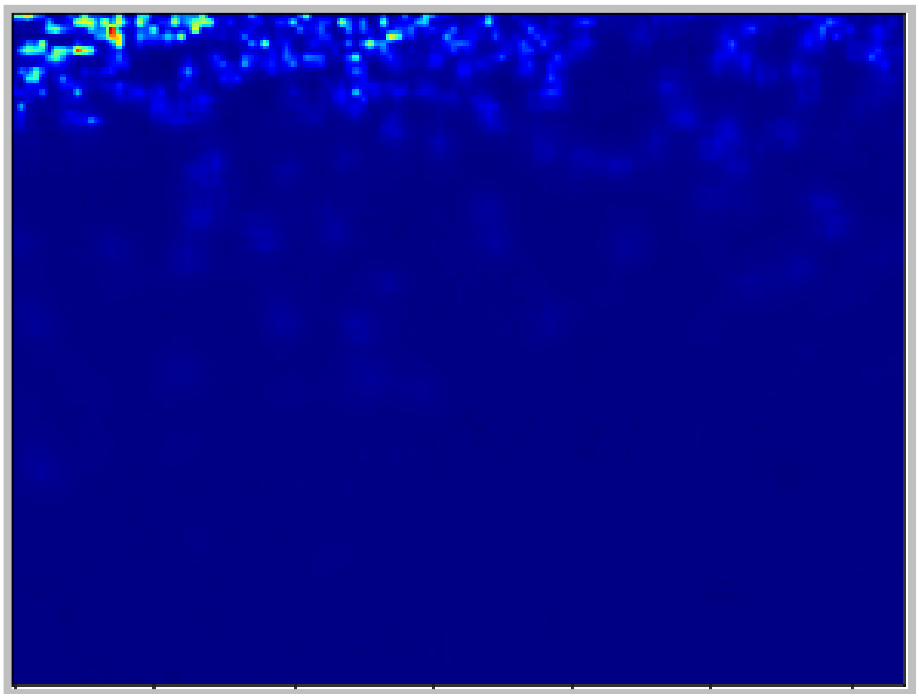

Рисунок 3.39 – Карта щільності створена мережею CSRNet

Розглянемо, які результати показав алгоритм, розроблений на основі методу локальних бінарних шаблонів. На зображенні було знайдено усього 168 людей, на 84 менше, ніж анотовано (рис. 3.40). Знову з трьох розроблених методів цей метод показує найгірші результати.

Рисунок 3.40 – Результати підрахунку алгоритму на основі методу локальних бінарних шаблонів

3.12 Аналіз у порівняльному аспекті результатів підрахунку запропонованими алгоритмами.

Після порівняння результатів було зрозуміло, що мережа CSRNet показує кращі результати серед усіх розглянутих методів. Порівняємо MAE та MSE розроблених методів. CSRNet у порівнянні з MCNN має меншу MAE на 38%, MSE на 33,6%; у порівнянні з LBP+RR MAE менша на 77,5%, MSE на 69%. У таблицях 3.2 та 3.3 наведено інформацію про точність методів, що використовувались у дослідженнях.

Таблиця 3.2 – Порівняння точності обраних методів для частини датасету part_A

	part_A	
Method	Avg.MAE	Avg.MSE
CSRNet	**68,2**	**115,0**
MCNN	110,2	173,2
LBP+RR	303,2	371,0

Таблиця 3.3 – Порівняння точності обраних методів для частини датасету part_B

	part_B	
Method	Avg.MAE	Avg.MSE
CSRNet	**10,6**	**16,0**
MCNN	26,4	41,3
LBP+RR	59,1	81,7

Також було виміряно precision (частина вірно знайдених результатів, що були головами) та recall (частина результатів, які були знайдені з усіх голів на зображенні). Кожного разу, коли голова не знайдена, це зараховується як хибно негативний результат. Коли голову виявляють у місці, де вона повинна бути, це зараховується як дійсно позитивний результат. Якщо метод виявив голову, але у дійсності це була не вона, цей результат як хибно позитивний, оскільки у цій ситуації голова не була виявлена, а методом було повернуто результат, який не був головою. У таблицях 3.4 та 3.5 показано порівння усереднених precision та recall на частинах датасету part_A та part_B. CSRNet знову має найкращий результат.

Таблиця 3.4 – Порівняння precision та recall обраних методів для частини датасету part_A

	part_A	
Method	Avg.precision	Avg.recall
CSRNet	**83,94%**	**63,50%**
MCNN	59,93%	59,18%
LBP+RR	49,4%	44,13%

Таблиця 3.5 – Порівняння precision та recall обраних методів для частини датасету part_B

	part_B	
Method	Avg.precision	Avg.recall
CSRNet	**86,1%**	**73,14%**
MCNN	65,3%	45%
LBP+RR	52,16%	47,2%

Методи також необхідно порівняти за іншою важливою характеристикою – швидкістю роботи, а також швидкістю навчання. Швідкість навчання нейронних мереж та алгоритму заснованому на методі локальних бінарних шаблонів відрізняється кардинально. У таблиці 3.6 наведено інформацію про швидкість навчання методів.

Таблиця 3.6 – Порівняння швидкості навчання обраних методів

Method	part_A	part_B
CSRNet	17 годин	17 годин
MCNN	25 годин	25 годин
LBP+RR	**5,5 хвилин**	**6 хвилин**

У результаті дослідження було помічено, що роздільная здатність зображень впливає на продуктивність мережі. Найкраща роздільна здатність зображення (300×450) пікселів. Моделі необхідно близько 4 секунд, щоб опрацювати зображення. Для зображень з більшою роздільною здатністю необхідно більше часу. У цій роботі розміри зображень не були змінені до (300×450), щоб не втратити якість вихідних карт щільності. Якщо б це було зроблено, то імовірно час навчання мережі CSRNet міг би скоротитися до 9 годин, а MCNN до 15 годин.

Далі розглянемо швидкість роботи усіх методів, коли їм на вхід передається зображення (табл. 3.7). Цікаво, що на зображенях з меншою щільністю натовпу нейронні мережі відпрацьовують трохи повільніше, ніж на зображеннях з високою щільністю. Найшвидшим методом є згорткова нейронна мережа MCNN, яка швидше за всіх відпрацьовує на зображеннях будь-якої щільності.

Таблиця 3.7 – Порівняння середньої швидкості роботи обраних методів

Method	part_A	part_B
CSRNet	0,0560 секунд	0,0764 секунд
MCNN	**0,0509 секунд**	**0,0491 секунд**
LBP+RR	1,2 секунди	2,1 секунди

ВИСНОВКИ

У рамках атестаційної роботи було проведено аналіз загорткових нейронних мереж та методу на основі детектування текстури для задачі підрахунку людей у натовпі. В ході дослідження проблематики був розглянутий такий список питань:

- способи побудови архітектури згорткової нейронної мережі;
- способи боротьби з перенавчанням;
- вплив глибини мережі на результат її роботи;
- способи навчання нейронної мережі;
- способи використання попередньо навчених нейронних мереж;
- способи розробки алгоритмів на основі текстурних методів.

Після даних досліджень можна сказати, що згорткові нейронні мережі, показують кращі результати для вирішення задачі підрахунку людей у натовпі. Перевагами використання глибоких загорткових нейронних мереж є точність результатів і швидкість роботи. Недоліки: тривалий час навчання моделей.

Було порівняно три методи: моделі на основі згорткових нейронних мереж MCNN та CSRNet, а також текстурний метод локальних бінарних шаблонів. Найкращим результат передбачення виявився у моделі CSRNet, яка містить більше усього шарів та довільну для них кількість параметрів для тренування.

Для реалізації програми на мові Python була використана бібліотека PyTorch. Це низькорівнена бібліотека для глибокого навчання. У якості середовища розробки було обрано Colaboratory, що запускається у браузері.

Результати були апробовані на XI-й Міжнародній науково-практичній конференції «Free and Open Source Software» [23], на VI-й Міжнародній науково-технічній конференції «Інформатика, управління та штучний інтелект» [24] та на 24-ому Міжнародному молодіжному форумі [2]. На конференції у рамках 24-го Міжнародному молодіжному форуму робота посіла II місце.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Polus A., Schofer, J. L., Ushpiz A. PEDESTRIAN FLOW AND LEVEL OF SERVICE. *Journal of Transportation Engineering*. 1983. Vol.109. № 1. P. 46–56. DOI: 10.1061/(asce)0733-947x(1983)109:1(46).

2. Норматова Т.В. Дослідження питання підрахунку людей у натовпі на базі нейронних загорткових мереж. *Матеріали XXV Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті* (7-9 квітня 2020 р., Харків). Харків. 2020. С.14–15.

3. Ryan D., Denman S., Sridharan S., Fookers C. An Evaluation of Crowd Counting Methods, Features and Regression Models. *Computer Vision and Image Understanding*. 2015. Vol.130, № 1(86). P. 1–17. DOI: http://dx.doi.org/10.1016/j.cviu.2014.07.008.

4. Rodriguez M., Laptev I., Sivic J. and Audibert. Density-aware person detection and tracking in crowds. 2011. P. 2423 –2430.

5. Khan A., Shah J., Kadir K., Albattah W., Khan, F. Crowd Monitoring and Localization Using Deep Convolutional Neural Network: A Review. *Applied Science*. 2020. № 10(14). P. 25-40. DOI: https://doi.org/10.3390/app10144781.

6. Ніколенко С., Кадурин А., Архангельска О. Глибоке навчання. СПб., 2018. 480 с.

7. Convolutional Neural Networks (CNNs / ConvNets). URL: http://cs231n.github.io/convolutional-networks/ (дата звернення 1.10.2020).

8. Путятин Е.П., Аверин С.И. Обработка изображений в робототехнике. М: Машиностроение, 1990. 320 с.

9. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (дата звернення: 12.10.2020).

10. The Comparison of Crowd Counting Algorithms based on Computer Vision. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1187/4/042012/pdf (дата звернення: 12.10.2020).

11. Wagner L. Video Surveillance of Crowds. 2012. 75 p. URL: https://cyber.felk.cvut.cz/theses/papers/205.pdf (дата звернення: 13.10.2020).

12. Яковлева О.В., Панченко І.А. Застосування енергетичних характеристик Лавса для сегментації зображень. *Біоніка інтелекту*: наук.-техн. журнал. 2007. №2 (67). С.94–98.

13. Яковлева О.В., Кускова І.В. Дослідження результатів сегментації зображень методом матриць співпадінь. *Вісник НТУ «ХПІ». Збірник наукових праць. Тематичний випуск «Системний аналіз, управління і інформаційні технології».* Харків: НТУ «ХПІ», 2006. № 39. С.164–171.

14. Яковлева О.В., Несторова Є.П. Порівняльний аналіз методів характеристик Лавса і матриць співпадінь у задачах сегментації текстурних зображень. *Прикладна радіоелектроніка*. 2009. №1. С. 00–00.

15. Handbook of pattern recognition and computer vision / Chen C.H., Rau L.F. and Wang P.S.P.(eds.). Singapore-New Jersey-London-Hong Kong: World Scientific Publishing Co. Pte. Ltd., 1995. 984 p.

16. ShanghaiTech. URL: https://www.kaggle.com/tthien/shanghaitech (дата звернення: 14.10.2020).

17. MRCNet: Crowd Counting and Density Map Estimation in Aerial and Ground Imagery. URL: https://www.researchgate.net/publication/335490585_MRCNet_Crowd_Counting_and_Density_Map_Estimation_in_Aerial_and_Ground_Imagery (дата звернення: 16.10.2020).

18. Mall Dataset. URL: https://pgram.com/dataset/mall-dataset/ (дата звернення: 16.10.2020).

19. Crowd Counting Data Set (UCF-CC-50). URL: https://www.crcv.ucf.edu/research/data-sets/ucf-cc-50/ (дата звернення: 17.10.2020).

20. SciPy: Scientific Library for Python. URL: https://pypi.org/project/scipy/ (дата звернення: 18.10.2020).

21. scikit-image: Image processing in Python. URL: https://pypi.org/project/scikit-image/ (дата звернення: 18.10.2020).

22. Welcome to scikit-learn. URL: https://scikit-learn.org/stable/preface.html (дата звернення: 20.10.2020).

42. Норматова Т.В., Яковлева О.В. Використання фреймворку Tensorflow для вирішення задач машинного навчання. *Матеріали X-ої Ювілейної Міжнародної науково-практичної конференції «Free and Open Source Software»* (19-21 листопада 2019 р., Харків). Харків: ХНУБА, 2019. 59 с. URL: https://foss.kn-it.info/uploads/foss-2019-theses.pdf (дата звернення: 25.10.2020).

24. Норматова Т.В., Яковлева О.В. Класифікація зображень на основі згорткових нейронних мереж з використанням бібліотеки Keras. *Тези VI-ої Міжнародної науково-технічної конференції «Інформатика, управління та штучний інтелект»* (27-29 листопада 2019 р. Харків-Краматорськ). Харків: НТУ «ХПІ», 2019. 129 с. URL: http://pim.net.ua/arch_f/tez_iyii_2019.pdf (дата звернення: 27.10.2019).

25. The Microsoft Cognitive Toolkit. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/ (дата звернення: 28.10.2019).

26. PyTorch. URL: https://ai.facebook.com/tools/pytorch/ (дата звернення: 29.10.2019).

27. Fully Convolutional Crowd Counting on Highly Congested Scenes. URL: https://www.scitepress.org/Papers/2017/60973/60973.pdf (дата звернення: 1.11.2020).

28. How Does Back-Propagation in Artificial Neural Networks Work?. URL: https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7 (дата звернення: 01.11.2020).

29. Understanding Activation Functions in Neural Networks. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (дата звернення: 02.11.2020).

30. VGG16 — сверточная сеть для выделения признаков изображений VGG16 — сверточная сеть для выделения признаков зображений. URL: https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/ (дата звернення: 03.11.2020).

31. Koltun V. Multi-scale context aggregation by dilated convolutions. 2016. URL: https://arxiv.org/abs/1511.07122 (дата звернення: 04.11.2020).

32. MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS. URL: https://arxiv.org/pdf/1511.07122.pdf (дата звернення: 05.11.2020).

33. V. Lempitsky and A. Zisserman. Learning to count objects in images. In Advances in Neural Information Processing Systems. 2010. P. 1324–1332.

34. How to understand Gradient Descent, the most popular ML algorithm. URL: https://www.freecodecamp.org/news/understanding-gradient-descent-the-most-popular-ml-algorithm-a66c0d97307f/ (дата звернення: 07.11.2020).

35. Wu X., Liang G., Lee K.K., Xu, Y. Crowd density estimation using texture analysis and learning. *International Conference on Robotics and Biomimetic* (17–20 December 2006, Kunming). Kunming: 2006, P. 214–219.

36. Towards Understanding the Formation of Uniform Local Binary Patterns. URL: https://www.hindawi.com/journals/isrn/2013/429347/ (дата звернення: 08.11.2020).

37. Regularization in Machine Learning. URL: https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a (дата звернення: 08.11.2020).

38. Проблема вибору метрики. URL: https://studbooks.net/2244344/matematika_himiya_fizika/problema_vybora_metriki (дата звернення: 10.11.2020).

39. Меры расстояния - переменные с интервальной шкалой. URL: http://www.datuapstrade.lv/rus/spss/section_20/5 (дата звернення: 11.11.2020).

40. MAE and RMSE — Which Metric is Better?. URL: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d (дата звернення: 11.11.2020).

41. Accuracy, Precision, Recall or F1?. URL: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (дата звернення: 12.11.2020).