Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ящук Т. Є.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф. каф.КСУ
Биков М. М.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри КН
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«06» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ящук Тетяні Євгеніївні

1. Тема роботи: Програмний модуль розпізнавання людей у касці на основі згорткової нейронної мережі.
керівник роботи: Колесницький Олег Костянтинович, к.т.н., проф.
затверджені наказом вищого навчального закладу "20" березня 2025 року № 97

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи :
Формат вхідних зображень – jpeg, вид класифікатора – нейронна мережа, кількість навчальних прикладів – не менше 1000, кількість тестових прикладів – не менше 100, розмір вхідного зображення – не менше 416x416, використання об'єктно-орієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи розпізнавання людей у касці та вибрати найбільш перспективний;
вибрати нейронну мережу та її структуру для модуля розпізнавання людей у касці;
розробити алгоритм роботи модуля розпізнавання людей у касці;
спроектувати та програмно реалізувати модуль розпізнавання людей у касці за допомогою нейронної мережі;
протестувати роботу програми розпізнавання людей у касці на основі нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема алгоритму роботи програми,
структура нейронної мережі,
фрагмент набору даних,
результати роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесницький О. К., к.т.н., проф. каф. КН		

7. Дата видачі завдання 05 Травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	07.05.25	
2	Обгрунтування методу розв'язання задачі, проектування модуля розпізнавання людей у касці на основі згорткової нейронної мережі	08.05.25	15.05.25	
3	Програмна реалізація модуля розпізнавання людей у касці на основі згорткової нейронної мережі	16.05.25	29.05.25	
4	Аналіз результатів тестування модуля розпізнавання людей у касці на основі згорткової нейронної мережі	30.05.25	02.06.25	
5	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Ящук Т. Є.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Колесницький О. К**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ящук Т. Є. Програмний модуль розпізнавання людей у касці на основі згорткової нейронної мережі. Бакалаврська кваліфікаційна робота скаладається з 87 сторінок формату А4, на яких є 22 рисунки, 1 таблиця, список використаних джерел містить 31 найменувань.

Метою роботи є підвищення точності розпізнавання людей у касці за рахунок використання згорткової нейронної мережі.

У роботі було обґрунтовано вибір типу нейронної мережі для модуля розпізнавання людей у касці – згорткова нейронна мережа YOLOv3, як найбільш перспективну за сполученням параметрів точність-швидкодія.. Для програмної реалізації модуля було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек ImageAI та OpenCV. Модуль обводить рамкою кожну людину у касці та виводить координати рамки та відсоток рівня довіри. Для оцінки точності роботи модуля використовувалась така метрика як середня точність mAP. Розроблений програмний модуль на основі згорткової нейронної мережі YOLOv3 має середню точність розпізнавання 84,4%, що порівняно з кращим аналогом збільшено на 1,5%.

Ключові слова: розпізнавання об’єктів, каска, людина у касці, зображення, згорткова нейронна мережа.

ABSTRACT

Yaschuk T. Ye. Software module for recognizing people wearing helmets based on a convolutional neural network. The bachelor's qualification paper consists of 87 pages of A4 format, on which there are 22 figures, 1 table, the list of used sources contains 31 names.

The aim of the work is to increase the accuracy of recognizing people in helmets by using a convolutional neural network.

The work justified the choice of the type of neural network for the module for recognizing people in helmets - the YOLOv3 convolutional neural network, as the most promising in terms of the combination of accuracy and speed. For the software implementation of the module, the choice of the Python programming language and the specialized ImageAI and OpenCV libraries was justified. The module frames each person in a helmet and displays the coordinates of the frame and the percentage of the confidence level. To assess the accuracy of the module, a metric such as the average mAP accuracy was used. The developed software module based on the YOLOv3 convolutional neural network has an average recognition accuracy of 84.4%, which is increased by 1.5% compared to the best analogue.

Keywords: object recognition, helmet, person in a helmet, image, convolutional neural network.

ЗМІСТ

ВСТУП

Актуальність досліджень. В останні роки через бурхливий розвиток технологій та істотне збільшення потужності обчислювальних систем, стала актуальною автоматизація процесів інтелектуальної обробки інформації із залученням систем штучного інтелекту (ШІ).

Одним із напрямків ШІ, який зараз активно розвивається, є штучні нейронні мережі (ШНМ), що працюють за аналогією з біологічними нейромережами і являють собою систему взаємодіючих між собою штучних нейронів. Серед основних галузей застосування ШНМ можна виділити: класифікацію зображень, обробку звукової та текстової інформації, прогнозування, оптимізацію, прийняття рішень, аналіз даних.

Розпізнавання, класифікація зображень та комп'ютерний зір є одними з найперспективніших у розвитку серед галузей сучасних інформаційних технологій. Існує багато методів та запропоновано багато алгоритмів вирішення задачі виявлення об'єктів на зображенні, проте усі ці ідеї поступаються у точності результату, швидкодії та простоті штучним нейромережам. В основі сучасних глибоких нейромереж, як правило, лежать архітектури мереж згорткового типу. Їх ефективність і стрімкий розвиток обумовлено гібридним підходом до архітектурних рішень, розвитком методів навчання, додаткових методів захисту від перенавчання. Внаслідок зростаючої популярності глибоких згорткових нейромереж досягаються суттєві успіхи у розпізнаванні об'єктів на зображенні.

В даній роботі буде розглянута програмна реалізація згорткової нейронної мережі, завданням якої є виявлення людей у касці на зображенні. Ця задача є актуальною при автоматизованому відеоспостереженні будівельних та інших промислових об'єктів, де всі працівники мають носити каски через вимоги техніки безпеки. Розроблювана програма повинна виявляти чи всі люди на зображенні вдягнуті у каски. І якщо ні, то це є сигналом до відповідних служб відновити дисциплінуносіння касок.

Метою дослідження є підвищення точності розпізнавання людей у касці за рахунок використання згорткової нейронної мережі.

Об’єктом дослідження є процес комп’ютеризованого розпізнавання людей у касці на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розпізнавання людей у касці на основі нейронної мережі та точність їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи розпізнавання об’єктів на зображенні та обрати напрямок досліджень;
2. Обгрунтувати вибір архітектури нейромережі;
3. Розробити алгоритм роботи програмного модуля розпізнавання людей у касці;
4. Спроектувати модуль розпізнавання людей у касці на основі нейронної мережі;
5. Здійснити програмну реалізацію модуля розпізнавання людей у касці;
6. Провести тестування програмного модуля розпізнавання людей у касці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ

1.1 Постановка задачі

Задача розпізнавання людей у касці це окремий випадок загальної задачі виявлення об’єктів на зображенні.

Виявлення об'єктів (Object Detection) - це завдання, що полягає у визначенні розташування та класу об'єктів у зображенні чи відеопотоці. Результатом роботи детектора об'єктів є набір обмежувальних рамок, які містять об'єкти на зображенні, а також мітки класів і бали довіри для кожної рамки.

Виявлення об’єктів - це просто визначення об'єктів на зображенні/кадрі. Тобто алгоритм або нейронна мережа визначають об'єкт і записують його позицію і обмежувальні рамки (параметри прямокутників навколо об'єктів). Алгоритм працює тільки з одним кадром. Приклад виявлення об’єктів (Object Detection) показано на рисунку 1.1.

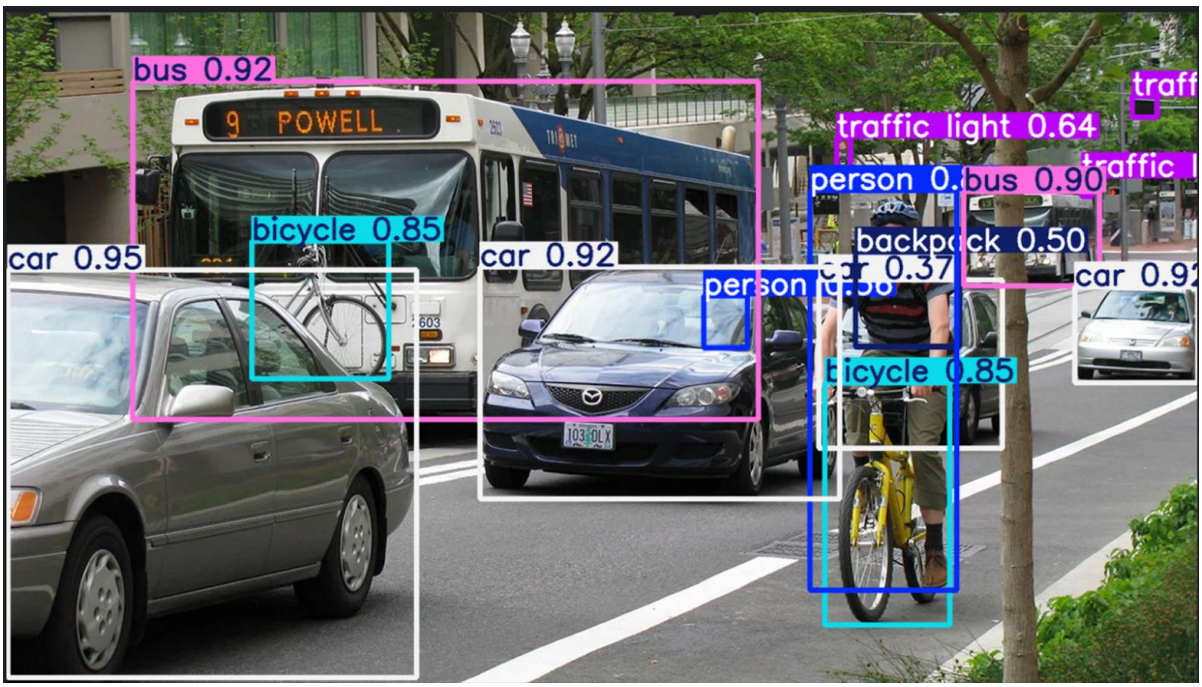


Рисунок 1.1 – Приклад виявлення об’єктів (Object Detection)

У цій роботі ми використаємо Python, OpenCV (бібліотеку комп'ютерного зору з відкритим кодом) та ImageAI (бібліотеку глибокого навчання для зору) для навчання ШІ виявляти, чи носять працівники каски. У процесі буде створено комплексне рішення, яке можна використовувати в реальному житті. Це важливий варіант використання, оскільки багато компаній повинні забезпечити працівників належним захисним спорядженням.

1.2 Огляд відомих методів розпізнавання об’єктів на зображенні

Виявлення об’єктів – це завдання, в якому треба виконати локалізацію та класифікацію об’єктів на зображенні чи послідовності відеокадрів.

Виявлення об'єктів, як одна з найфундаментальніших і найскладніших проблем комп'ютерного зору, отримало велику увагу в останні роки. Протягом останніх двох десятиліть спостерігалась швидка технологічна еволюція виявлення об'єктів та її глибокий вплив на всю галузь комп'ютерного зору.

Розвиток галузі виявлення об’єктів [1] за останні 20 років загалом розділяють на 2 періоди (рис. 1.2):

– період традиційних методів виявлення об’єктів (до 2014 року);
– період методів на основі глибокого навчання для виявлення об’єктів (після 2014 року).

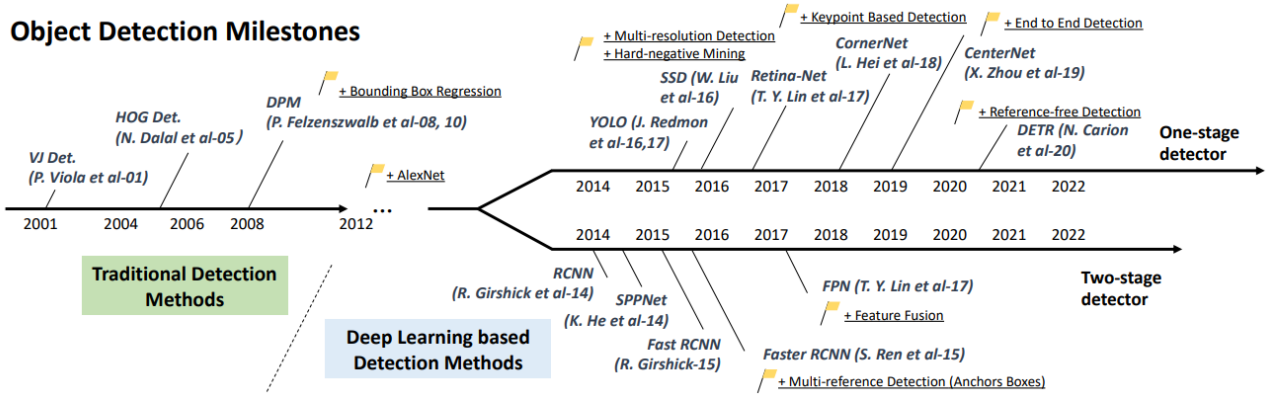


Рисунок 1.2 – Розвиток галузі виявлення об’єктів на зображенні

Давайте розгляемо ключові методи детектування (детектори) цього періоду, де час появи та продуктивність слугуватимуть головною підказкою для висвітлення рушійної сили технології (рис. 1.2).

1) Ключові етапи: Традиційні детектори

Якщо розглядати сучасну техніку виявлення об'єктів як революцію, зумовлену глибоким навчанням, то ще в 1990-х роках ми побачили б геніальний дизайн та довгострокову перспективу раннього комп'ютерного зору.

Більшість ранніх алгоритмів виявлення об'єктів були побудовані на основі отриманих вручну ознак. Через відсутність ефективного представлення зображень у той час людям доводилося розробляти складні представлення ознак та різноманітні методи прискорення.

1а) Детектори Віоли Джонса: у 2001 році П. Віола та М. Джонс вперше досягли виявлення людських облич у реальному часі без будь-яких обмежень (наприклад, сегментації кольору шкіри) [2, 3]. Детектор VJ використовує найпростіший спосіб виявлення, тобто ковзні вікна: пройтися по всіх можливих місцях та масштабах зображення, щоб побачити, чи містить якесь вікно людське обличчя.

Хоча це здається дуже простим процесом, обчислення, що лежали в його основі, значно перевершували можливості комп'ютерів того часу. Детектор VJ значно покращив швидкість виявлення, включивши три важливі методи: «інтегральне зображення», «вибір ознак» та «каскади виявлення».

1б) Детектор HOG: У 2005 році Н. Далал та Б. Тріггс запропонували дескриптор ознак «Гістограма орієнтованих градієнтів» (HOG) [4]. HOG можна розглядати як важливе вдосконалення масштабно-інваріантного перетворення ознак та контекстів форми свого часу. Щоб збалансувати інваріантність ознак (включаючи переміщення, масштаб, освітлення тощо) та нелінійність, дескриптор HOG розроблено для обчислення на щільній сітці рівномірно

розташованих комірок та використання перекриваючої нормалізації локального контрасту (на «блоках»). Хоча HOG можна використовувати для виявлення різноманітних класів об'єктів, його використання було мотивоване, головним чином, проблемою виявлення пішоходів. Для виявлення об'єктів різних розмірів детектор HOG змінює масштаб вхідного зображення кілька разів, зберігаючи при цьому розмір вікна виявлення незмінним.

1в) Модель на основі деформованих деталей (DPM): DPM, як переможець конкурсів VOC-07, -08 та -09, була втіленням традиційних методів виявлення об'єктів. DPM була спочатку запропонована П. Фельценшвальбом [5] у 2008 році як розширення детектора HOG. Наприклад, проблему виявлення «автомобіля» можна розкласти на виявлення його вікна, кузова та коліс. Хоча сучасні детектори об'єктів значно перевершили DPM за точністю виявлення, багато з них все ще глибоко залежать від його цінних висновків, наприклад, моделі сумішей, жорсткий негативний аналіз, регресія обмежувального прямокутника, контекстне праймування тощо.

2) Ключові етапи: Двоступеневі детектори на основізгорткових нейронних мереж CNN

Оскільки продуктивність вручну створених ознак насичувалася, дослідження виявлення об'єктів досягли плато після 2010 року. У 2012 році світ став свідком відродження згорткових нейронних мереж [6]. Оскільки глибока згорткова мережа здатна навчатися надійним та високорівневим представленням ознак зображення, виникає природне питання: чи можемо ми впровадити її у виявлення об'єктів?

Р. Гіршік та ін. взяли на себе ініціативу у вирішенні цієї проблеми у 2014 році, запропонувавши Регіони з ознаками CNN (RCNN) [7, 8]. Відтоді виявлення об'єктів почало розвиватися з безпрецедентною швидкістю. В еру глибокого навчання існують дві групи детекторів: «двоступеневі детектори» та «одноступеневі детектори», де перші розглядають виявлення як процес «від

грубого до точного», тоді як другі розглядають його як «завершення за один крок».

2а) Ідея **RCNN** проста: вона починається з вилучення набору пропозицій об'єктів (полей кандидатів об'єктів) за допомогою вибіркового пошуку [9]. Потім кожна пропозиція масштабується до зображення фіксованого розміру та подається в модель CNN, попередньо навчену на ImageNet (скажімо, AlexNet) для вилучення ознак. Нарешті, лінійні SVM-класифікатори використовуються для прогнозування наявності об'єкта в кожній області та розпізнавання категорій об'єктів. RCNN забезпечує значне підвищення продуктивності на VOC07, зі значним покращенням середньої точності (mAP) з 33,7% (DPM-v5) до 58,5%. Хоча RCNN досягла значного прогресу, її недоліки очевидні: надлишкові обчислення ознак для великої кількості перекритих пропозицій (понад 2000 блоків з одного зображення) призводять до надзвичайно низької швидкості виявлення (14 секунд на зображення з графічним процесором). Пізніше того ж року була запропонована SPPNet [10], яка вирішила цю проблему.

2б) **SPPNet**: у 2014 році К. Хе та ін. запропонували мережі просторового пірамідального об'єднання (SPPNet) [10]. Попередні моделі CNN вимагають вхідних даних фіксованого розміру, наприклад, зображення 224x224 для AlexNet. Основним внеском SPPNet є введення шару просторового пірамідального об'єднання (SPP), який дозволяє CNN генерувати представлення фіксованої довжини незалежно від розміру зображення/області інтересу без його масштабування. Під час використання SPPNet для виявлення об'єктів, карти ознак можна обчислювати з усього зображення лише один раз, а потім можна генерувати представлення фіксованої довжини довільних областей для навчання детекторів, що дозволяє уникнути повторного обчислення згорткових ознак. SPPNet більш ніж у 20 разів швидший за R-CNN без шкоди для точності виявлення (VOC07 mAP=59,2%). Хоча SPPNet ефективно покращив швидкість виявлення, він все ще має деякі недоліки: по-перше, навчання все ще є

багатоетапним, по-друге, SPPNet лише налаштовує свої повністю зв'язані шари, просто ігноруючи всі попередні шари. Пізніше наступного року був запропонований Fast RCNN [11], який вирішив ці проблеми.

2в) **Fast RCNN**: У 2015 році Р. Гіршик запропонував детектор Fast RCNN [11], який є подальшим удосконаленням R-CNN та SPPNet [7, 10]. Fast RCNN дозволяє нам одночасно навчати детектор та регресор обмежувальної рамки в однакових конфігураціях мережі. На наборі даних VOC07, Fast RCNN збільшив mAP з 58,5% (RCNN) до 70,0%, при цьому швидкість виявлення була більш ніж у 200 разів вищою, ніж у R-CNN. Хоча Fast-RCNN успішно інтегрує переваги R-CNN та SPPNet, швидкість його виявлення все ще обмежена виявленням пропозицій. Тоді природно виникає питання: «Чи можемо ми генерувати пропозиції об'єктів за допомогою моделі CNN?» Пізніше Faster R-CNN [12] відповів на це питання.

2г) **Faster RCNN:** у 2015 році С. Рен та ін. запропонували детектор Faster RCNN [12, 13] невдовзі після Fast RCNN. Faster RCNN — це перший детектор глибокого навчання майже в реальному часі (COCO mAP@.5=42,7%, VOC07 mAP=73,2%, 17 кадрів/с з ZF-Net). Основним внеском Faster-RCNN є впровадження мережі пропозицій регіонів (RPN), яка дозволяє майже безкоштовні пропозиції регіонів. Від R-CNN до Faster RCNN, більшість окремих блоків системи виявлення об'єктів, наприклад, виявлення пропозицій, вилучення ознак, регресія обмежувальної рамки тощо, поступово інтегрувалися в єдину, наскрізну систему навчання. Хоча Faster RCNN долає вузьке місце швидкості Fast RCNN, на наступному етапі виявлення все ще існує надмірність обчислень.

2д) **Мережі пірамід ознак (FPN):** у 2017 році Т.-Ю. Лін та ін. запропонували FPN [14]. До FPN більшість детекторів на основі глибокого навчання виконували виявлення лише на картах ознак верхнього шару мереж. Хоча ознаки в глибших шарах CNN корисні для розпізнавання категорій, це не

сприяє локалізації об'єктів. З цією метою в FPN розроблена низхідна архітектура з латеральними зв'язками для побудови високорівневої семантики на всіх масштабах. Оскільки CNN природним чином формує піраміду ознак через своє пряме поширення, FPN демонструє значні досягнення у виявленні об'єктів з широким спектром масштабів. Використовуючи FPN у базовій системі Faster R-CNN, вона досягає найсучасніших результатів виявлення однієї моделі на наборі даних COCO без додаткових зайвих зусиль (COCO mAP@.5=59.1%). FPN тепер стала основним будівельним блоком багатьох новітніх детекторів.

3) Ключові етапи: Одноступінчасті детектори на основі CNN

Більшість двоступінчастих детекторів дотримуються парадигми обробки від грубої до точної. Грубий метод прагне покращити здатність до запам'ятовування, тоді як точний метод уточнює локалізацію на основі грубого виявлення та робить більший акцент на дискримінаційній здатності. Вони можуть легко досягти високої точності без будь-яких додаткових функцій, але рідко використовуються в інженерії через низьку швидкість та величезну складність. Навпаки, одноступеневі детектори можуть виявляти всі об'єкти за допомогою однокрокового виведення. Вони добре використовуються мобільними пристроями з функціями реального часу та простим розгортанням, але їхня продуктивність помітно погіршується під час виявлення щільних та малих об'єктів.

3а) **You Only Look Once (YOLO** - Ви тільки дивитеся один раз): YOLO був запропонований Р. Джозефом та ін. у 2015 році. Це був перший одноступеневий детектор в еру глибокого навчання [15]. YOLO надзвичайно швидкий: швидка версія YOLO працює зі швидкістю 155 кадрів/с з VOC07 mAP=52,7%, тоді як його покращена версія працює зі швидкістю 45 кадрів/с з VOC07 mAP=63,4%. YOLO дотримується зовсім іншої парадигми, ніж двоступеневі детектори: застосовувати одну нейронну мережу до повного зображення. Ця мережа розділяє зображення на області та прогнозує обмежувальні рамки та ймовірності

для кожної області одночасно. Незважаючи на значне покращення швидкості виявлення, YOLO страждає від зниження точності локалізації порівняно з двоступеневими детекторами, особливо для деяких малих об'єктів. Наступні версії YOLO та запропонований останнім SSD [16] приділили більше уваги цій проблемі. Нещодавно було запропоновано YOLOv7, подальшу роботу команди YOLOv4. Він перевершує більшість існуючих детекторів об'єктів за швидкістю та точністю (діапазон від 5 FPS до 160 FPS) завдяки впровадженню оптимізованих структур, таких як динамічне призначення міток та перепараметризація структури моделі.

3б) **Single Shot MultiBox Detector (SSD)** - Однокадровий багатобоксовий детектор: SSD [16] був запропонований В. Лю та ін. у 2015 році. Основним внеском SSD є впровадження методів багатоетапного та багатороздільного виявлення, що значно покращує точність виявлення одноступеневого детектора, особливо для деяких малих об'єктів. SSD має переваги як у швидкості виявлення, так і в точності (COCO mAP@.5=46,5%, швидка версія працює зі швидкістю 59 кадрів/с). Основна відмінність між SSD та попередніми детекторами полягає в тому, що SSD виявляє об'єкти різного масштабу на різних рівнях мережі, тоді як попередні моделі виконують виявлення лише на своїх верхніх рівнях.

3в) **RetinaNet:** незважаючи на високу швидкість та простоту, одноступеневі детектори роками відставали за точністю від двоступеневих. Т.-Ю. Лін та ін. дослідили причини цього та запропонували RetinaNet у 2017 році [17]. Вони виявили, що основною причиною є екстремальний дисбаланс класів переднього плану та фону, який виникає під час навчання щільних детекторів. З цією метою в RetinaNet було введено нову функцію втрат під назвою «фокальна втрата» шляхом зміни стандартної перехресної втрати ентропії таким чином, щоб детектор більше зосереджувався на складних, неправильно класифікованих прикладах під час навчання. Фокальна втрата дозволяє одноступеневим детекторам досягати порівнянної точності з двоступеневими детекторами,

зберігаючи при цьому дуже високу швидкість виявлення (COCO mAP@.5=59.1%).

3г) **CornerNet:** попередні методи в основному використовували опорні рамки для забезпечення класифікації та регресійних посилань. Об'єкти часто демонструють варіації щодо кількості, розташування, масштабу, співвідношення тощо. Вони повинні йти шляхом встановлення великої кількості опорних рамок для кращого узгодження з базовими даними, щоб досягти високої продуктивності. Однак мережа страждатиме від подальшого дисбалансу категорій, великої кількості вручну розроблених гіперпараметрів та тривалого часу конвергенції. Щоб вирішити ці проблеми, Х. Ло та ін. [18] відкинули попередню парадигму виявлення та розглядають завдання як задачу прогнозування ключових точок (кутів прямокутника). Після отримання ключових точок, система розділяє та перегруповує кутові точки, використовуючи додаткову інформацію про вбудовування для формування обмежувальних прямокутників. CornerNet перевершує більшість одноступеневих детекторів на той час (COCO mAP@.5=57.8%).

3д) **CenterNet:** X. Zhou та ін. запропонували CenterNet [19] у 2019 році. Він також дотримується парадигми виявлення на основі ключових точок, але виключає дорогі пост-процеси, такі як групове призначення ключових точок (у CornerNet [18], ExtremeNet тощо) та NMS, що призводить до повністю наскрізної мережі виявлення. CenterNet розглядає об'єкт як одну точку (центр об'єкта) та регресує всі його атрибути (такі як розмір, орієнтація, розташування, поза тощо) на основі опорної центральної точки. Модель проста та елегантна, і вона може інтегрувати 3D-виявлення об'єктів, оцінку пози людини, навчання оптичного потоку, оцінку глибини та інші завдання в єдину структуру. Незважаючи на використання такої лаконічної концепції виявлення, CenterNet також може досягати порівняльних результатів виявлення (COCO mAP@.5=61.1%).

3е) **DETR**: в останні роки трансформери глибоко вплинули на всю галузь глибокого навчання, зокрема на галузь комп'ютерного зору. Трансформери відмовляються від традиційного оператора згортки на користь обчислення лише з увагою, щоб подолати обмеження CNN та отримати рецептивне поле глобального масштабу. У 2020 році Н. Каріон та ін. запропонували DETR [20], де вони розглядали виявлення об'єктів як проблему прогнозування множин та запропонували наскрізну мережу виявлення з трансформерами. Наразі виявлення об'єктів вступило в нову еру, в якій об'єкти можна виявляти без використання опорних блоків або опорних точок. Пізніше X. Zhu та ін. запропонували Deformable DETR [21] для вирішення проблеми тривалого часу збіжності DETR та обмеженої продуктивності у виявленні малих об'єктів. Він досягає найсучаснішої продуктивності на наборі даних MSCOCO (COCO mAP@.5=71.9%).

Таким чином, серед розглянутих методів виявлення об’єктів на зображенні:

- традиційні методи;
- двоступеневі методи на основі глибокого навчання;
- одноступеневі методи на основі глибокого навчання;

найбільш перспективним є одноступеневі методи на основі глибокого навчання. Вони будуються на таких нейронних мережах як YOLO, SSD, RetinaNet, CornerNet, CenterNet, DETR та інши. В 2-му розділі нам потрібно буде із цих мереж обрати найперспективнішу для нашої задачі розпізнавання людей у касках.

1.3 Обгрунтування вибору аналогу до програмного модуля розпізнавання людей у касці

Традиційно методи виявлення носіння каски можна розділити на дві категорії: виявлення на основі датчиків та виявлення на основі зору.

Методи виявлення на основі датчиків [22, 23] зосереджені на методах дистанційного визначення місцезнаходження та відстеження, таких як радіочастотна ідентифікація (RFID) та бездротові локальні мережі (WLAN). Однак RFID-зчитувачі, розташовані на в'їзді на будівельні майданчики, не здатні перевіряти зони, що не входять до складу. Крім того, вони не можуть визначити, чи носяться каски потім. Щоб визначити, чи носить каску, датчик тиску розміщувався в касці, а потім інформація про тиск передавалася через Bluetooth для моніторингу. У роботі [22] розробили інтелектуальну систему касок, використовуючи архітектуру на основі Інтернету речей (IoT). Для визначення статусу використання каски всередині неї було розташовано детектор інфрачервоного випромінювання та тепловий інфрачервоний датчик. Використання каски працівником підтверджувалося активацією як детектора інфрачервоного випромінювання, так і теплового інфрачервоного датчика. Як правило, існуючі методи на основі датчиків, що базуються на фізичних мітках або датчиках, що використовуються в касках, мають труднощі з визначенням того, чи носять хтось на будівельних майданчиках каски, чи ні. Крім того, практичне використання міток або датчиків призведе до значних витрат при масовому виробництві.

Порівняно з методами виявлення на основі датчиків, методи на основі зору отримали більшу увагу. У зв'язку з цим, використання звичайних камер та досягнення в галузі комп'ютерного зору та методів розпізнавання образів створили міцну основу для виявлення носіння касок на основі зору. Наприклад, в роботі [24] запропонували структуру для моніторингу носіння касок шляхом спочатку виявлення рухомих об'єктів за допомогою матриці стандартного відхилення (SDM), а потім класифікації людей за допомогою детектора об'єктів на основі агрегованих ознак каналу. Після цього каскадний детектор об'єктів на основі ознак гістограми орієнтованих градієнтів (HOG) шукав каски у верхній області ідентифікованого персоналу, які потім вносилися до компонента класифікації на основі кольору. Загалом, ці багатоетапні методи значною мірою покладаються на вручну створені ознаки для виявлення осіб на будівельних

майданчиках. Отже, вони можуть дати збій у випадках складних сцен з мінливістю погоди, різними ракурсами та перекриттями.

Останнім часом зростаюча популярність виявлення об'єктів на основі глибокого навчання мотивувала виявлення носіння касок на основі згорткових нейронних мереж (CNN). Фанг та ін. [25] запропонували метод на основі Faster R-CNN для автоматичного виявлення невикористання касок будівельниками (NHU). Загалом було зібрано 81 000 кадрів зображень з різних будівельних майданчиків як навчальний набір даних для навчання моделі Faster R-CNN. На етапі навчання працівник, що цікавить (WOI) на зображенні, був анотований як базовий показник для навчання. На етапі тестування будуть виявлені працівники NHU, а решта буде вважатися фоном. Модель Faster R-CNN сильно залежить від інформації, отриманої з верхніх ознак, не повністю використовує нижні деталі, що може погіршити продуктивність виявлення працівників в різних масштабах на зображеннях.

Також відомий автоматичний алгоритм виявлення носіння касок [26], заснований на згортковій нейромережі SSD. У цій роботі пропонується одноетапна система на основі згорткової нейронної мережі для автоматичного моніторингу того, чи носять будівельники каски, та ідентифікації відповідних кольорів. Для полегшення дослідження в цій роботі створено новий загальнодоступний набір даних для виявлення носіння касок, який складається з 3174 зображень, що охоплюють різні умови на будівельному майданчику. Потім ознаки з різних шарів з різним масштабом об'єднуються вибірково за допомогою запропонованої зворотної прогресивної уваги для створення нової піраміди ознак, яка буде передана в однопотоковий багатобоксовий детектор (SSD) для прогнозування кінцевих результатів виявлення. Запропонована система навчається за наскрізною схемою. Експериментальні результати показують, що запропонована система ефективна за будь-яких умов на будівельному майданчику, що дозволяє досягти 83% mAP (середньої точності) з розміром вхідних даних 512×512.

Оскільки у даній роботі розробляється програмний модуль на основі згорткової нейромережі YOLOv3, то як перший аналог було обрано «Детектор випадків невикористання каски на основі Faster R-CNN» [25], а як другий аналог було обрано «Автоматичний детектор носіння касок на основі SSD» [26].

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі розпізнавання людей у касці. Також розглядаються різні методи розв'язання задачі розпізнавання об’єктів на зображенні і оцінюються їх переваги і недоліки. Серед таких методів як традиційні методи; двоступеневі методи на основі глибокого навчання; одноступеневі методи на основі глибокого навчання як найбільш перспективний було обрано одноступеневі методи на основі глибокого навчання. Вони будуються на таких нейронних мережах як YOLO, SSD, RetinaNet, CornerNet, CenterNet, DETR та інши. На основі недоліків відомих методів виявлення об’єктів на зображеннях було сформульовано мету роботи – підвищення точності розпізнавання людей у касці. Крім цього, було обгрунтовано для визначення переваг розробленого модуля порівнювати його з двома аналогами: перший на основі згорткової нейронної мережі Faster R-CNN, а другий аналог на основі згорткової нейронної мережі SSD.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Обґрунтування вибору типу нейронної мережі для модуля розпізнавання людей у касці

Для виявлення об’єктів на зображенні можуть використовуватись такі згорткові нейромережі:

- R-CNN,
- Fast R-CNN,
- Faster R-CNN,
- SSD,
- YOLO.

Давайте розглянемо їх, порівняємо їх переваги та недоліки та оберемо найбільш підходящу для модуля розпізнавання людей у касці. Із трьох моделей R-CNN, Fast R-CNN та Faster R-CNN розглянемо тільки найкращу Faster R-CNN.

2.1.1 Faster R-CNN [12, 13].

Взагалі R-CNN – region based CNN, тобто ЗНМ на основі регіонів. У перших моделях R-CNN пропозиції регіонів виявлялись за допомогою методу вибіркового пошуку, що вимагало значних обчислювальних ресурсів. У [12] представили Мережу пропозицій регіонів (RPN – regional propoition network) для прямого створення пропозицій регіонів, прогнозування обмежувальних рамок та виявлення об'єктів. Швидша мережа на основі регіонів (Faster R-CNN) являє собою комбінацію RPN і моделі Fast R-CNN.

Модель CNN приймає як вхідні дані все зображення і створює карти характеристик. Вікно розміром 3x3 ковзає по всіх картах об'єктів і виводить вектор ознак, пов'язаний з двома повністю зв'язаними шарами, один для блокової регресії та один для блокової класифікації. Пропозиції кількох регіонів

передбачаються повністю зв'язаними шарами. Фіксується максимум k областей (рис. 2.1), тому вихідні шари регресії блоків мають розмір 4k (координати блоків, їх висота і ширина), а вихідні дані шару класифікації блоків мають розмір 2k (бал впевненості, що виявлений об'єкт знаходиться в рамці). Пропозиції області k, виявлені ковзним вікном, називаються якорями. [12].

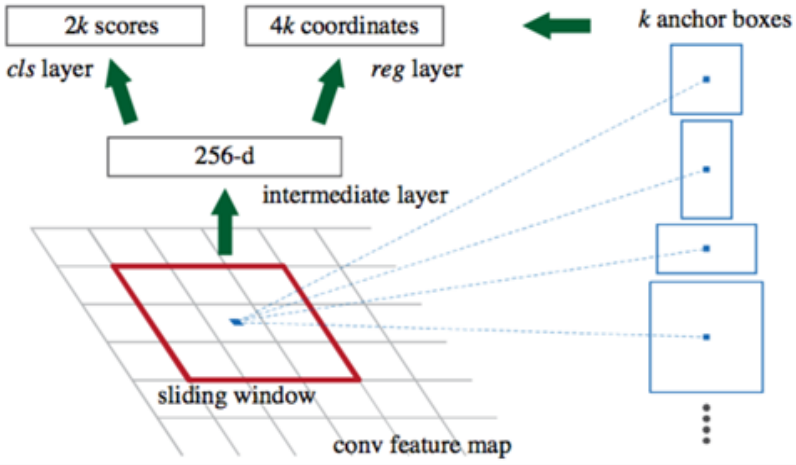


Рисунок 2.1 – Виявлення блоків прив'язки для одного вікна 3x3 у Faster RCNN

Коли блоки прив'язки виявлені, вони вибираються шляхом застосування порога до показника об'єктивності, щоб залишити тільки відповідні блоки. Faster R-CNN використовує RPN, щоб уникнути методу вибіркового пошуку, прискорити процеси навчання та тестування та підвищити продуктивність (рис. 2.2). RPN використовує попередньо вивчену модель набору даних ImageNet для класифікації і точно налаштовує набір даних PASCAL VOC. Потім згенеровані пропозиції регіонів з якірними полями використовують для навчання Fast R-CNN. Цей процес є ітеративним.

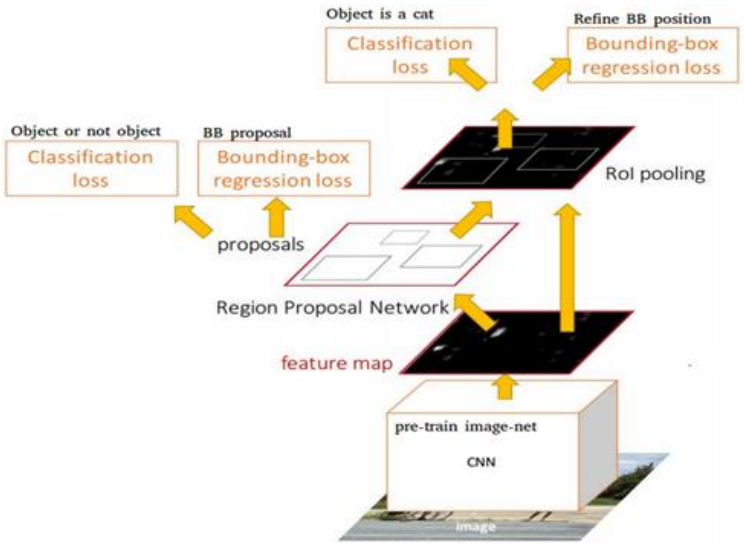


Рисунок 2.2 –Створення блоків прив'язки як пропозиції області з упевненістю, що вона містить об'єкт у Faster RCNN

2.1.2. SSD (Single-Shot Detector).

W. Liu et al. (2016) Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). “SSD: Single shot multibox detector”. In: European conference on computer vision. Springer, pp. 21–37] розробили однопрохідний детектор (SSD) для одночасного прогнозування всіх обмежувальних рамок та ймовірностей класів за допомогою наскрізної архітектури CNN.

Як вхідні дані модель (рис. 2.3) приймає зображення, яке проходить через кілька згорткових шарів з різними розмірами фільтрів (10x10, 5x5 і 3x3). Карти об'єктів зі згорткових шарів у різних шарах мережі використовуються для прогнозування обмежувальних рамок. Вони обробляються спеціальними шарами згортки з фільтрами 3x3, так званими додатковими шарами об'єктів, для створення набору обмежувальних рамок, подібних до якірних рамок Fast R-CNN.

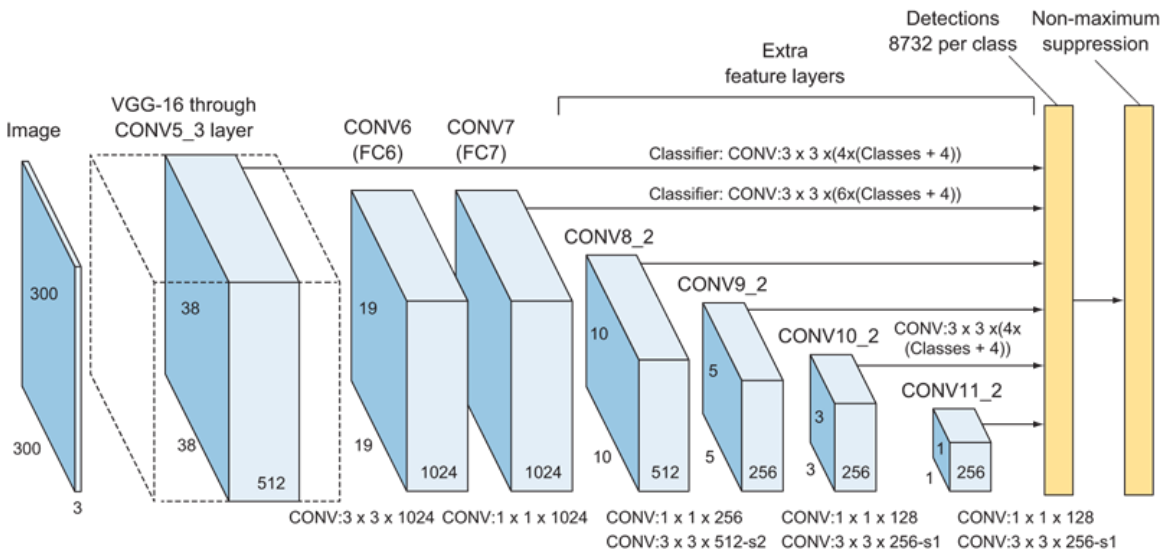


Рисунок 2.3 –Структура моделі SSD

Кожна обмежувальна рамка має 4 параметри: координати центру, ширину та висоту (рис. 2.4).. На рисунку 2.4(a) модель бере зображення та його обмежувальні рамки. Невеликі набори блоків із різним співвідношенням сторін фіксуються іншою картою ознак (рис. 2.4(b) та (c)). Під час навчання локалізація обмежувальних рамок змінюється, щоб максимально відповідати дійсності.

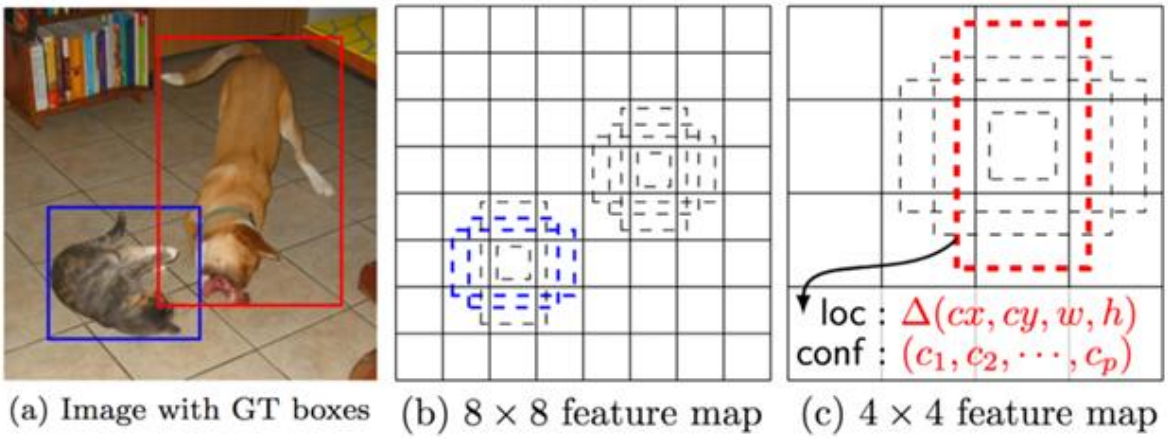


Рисунок 2.4 – Фреймворк SSD

Метод немаксимального придушення також використовується в кінці моделі SSD, щоб зберегти найбільш релевантні обмежувальні рамки. Потім використовується Hard Negative Mining (HNM), оскільки все ще прогнозується багато негативних полів. Він полягає у виборі лише частини цих блоків під час

навчання. Обмежувальні рамки впорядковані за достовірністю, а вершина вибирається залежно від співвідношення між негативним та позитивним значенням, яке не перевищує 1/3.

Розрізняють модель SSD300 (архітектура докладно показана на рисунку 2.3 вище) і модель SSD512, яка є SSD300 з додатковим шаром для згортки для підвищення продуктивності прогнозування. Найкращі SSD алгоритми навчаються на наборах даних PASCAL VOC 2007, 2012 та наборі даних COCO 2015 з доповненням даних.

Мінуси мережі SSD:

- Ступінь точності SSD трохи знижується при ідентифікації дрібніших речей. Якщо модель дуже велика, швидкість може значно впасти.

2.1.3. YOLO (You Only Look Once).

Модель YOLO [15] безпосередньо передбачає обмежувальні рамки та ймовірності класів за допомогою однієї мережі в одній оцінці. Простота моделі YOLO дозволяє робити прогнози у реальному часі.

Спочатку модель приймає зображення як вхідні дані. Далі поділяє його на сітку SxS (рис.2.5). Кожен осередок цієї сітки передбачає B обмежувальних прямокутників з показником впевненості. Ця впевненість є просто ймовірністю виявлення об'єкта, помножену на IoU між передбаченим і дійсним об’єктом. [15].

Створення YOLO черпало натхнення з моделі GoogLeNet, в якій представлені початкові модулі. Мережа має 24 згорткових шари, за якими слідують 2 повнозв'язні шари. Шари скорочення з фільтрами 1x1, за якими йдуть згорткові шари 3x3, замінюють початкові модулі. Модель Fast YOLO - це легша версія, в якій всього 9 згорткових шарів і менше фільтрів. Більшість згорткових шарів попередньо навчені з використанням набору даних ImageNet із класифікацією. До попередньої мережі додаються чотири згорткові шари, за якими йдуть два повнозв'язні шари, і вона повністю перенавчається з наборами даних PASCAL VOC 2007 і 2012 років.

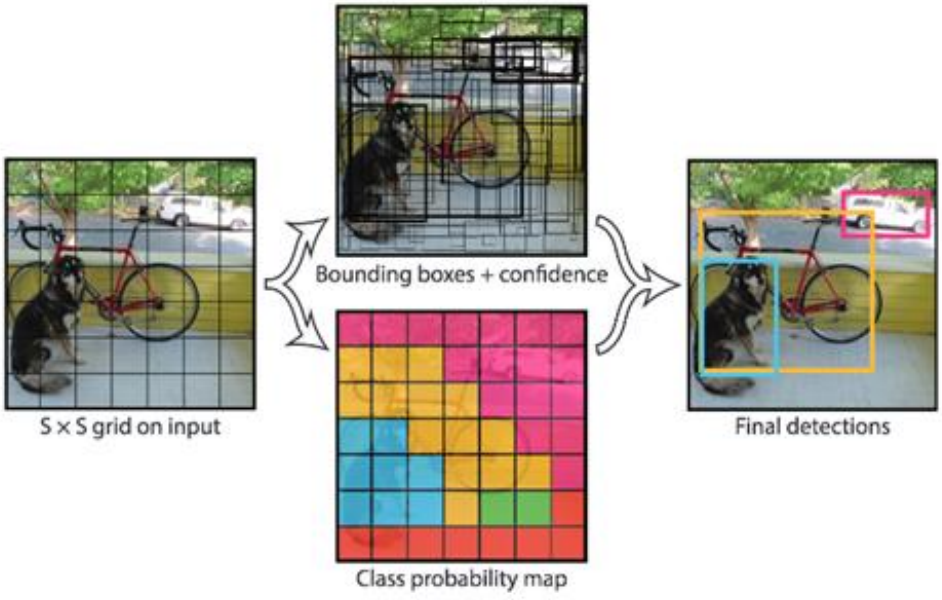


Рисунок 2.5 – Приклад застосування. Вхідне зображення ділиться на сітку SxS

Останній шар виводить тензор S * S * (C + B * 5), що відповідає прогнозам для кожної комірки сітки. C - кількість ймовірностей для кожного класу. B - фіксована кількість блоків прив'язки на комірку, кожен з цих блоків пов'язаний з 4 координатами (координати центру блоку, ширина та висота) та довірчим значенням.

У попередніх моделях передбачені рамки, що обмежують, часто містили об'єкт. Однак модель YOLO передбачає велику кількість рамок, що обмежують. Таким чином, є багато рамок, що обмежують, без будь-якого об'єкта. Метод максимального придушення (NMS) застосовується в кінці мережі. Він полягає в об'єднанні обмежуючих рамок одного і того ж об'єкта в одну, що сильно перекриваються. Автори помітили, що хибних спрацьовувань, як і раніше, мало.

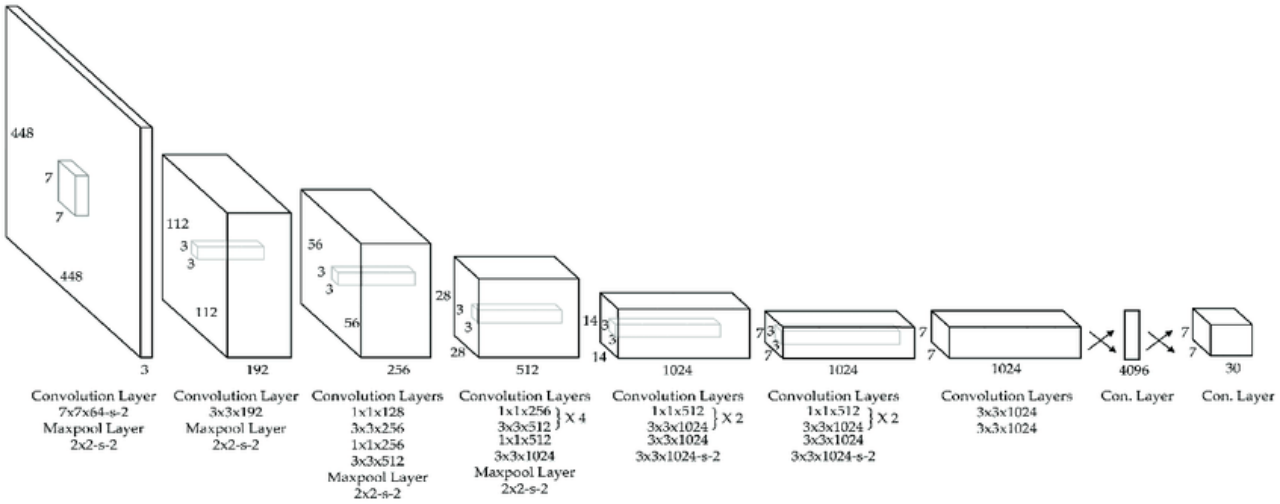


Рисунок 2.6 – Архітектура YOLO: 24 згорткових шари та 2 повнозв’яних шари

Плюси YOLO алгоритму:

- Швидкість: Цей алгоритм підвищує швидкість виявлення, оскільки може прогнозувати об'єкти у режимі реального часу.
- Висока точність: YOLO - це метод прогнозування, який забезпечує точні результати з мінімальними помилками фону.
- YOLO може узагальнювати зображення без навантаження пам'яті обробки.

Мінуси YOLO алгоритму:

- YOLO страждає від значно більшої кількості помилок локалізації та має проблеми з ідентифікацією найближчих предметів.

Наочне порівняння розглянутих архітектур ЗНМ показано на рисунку 2.7. Із цієї діаграми [27] можна побачити, що при справедливому порівнянні різні версії YOLO-V3 (червоні точки), навчені GluonCV, досягають чудової точності, поступаючись лише набагато повільнішим, Faster RCNN.

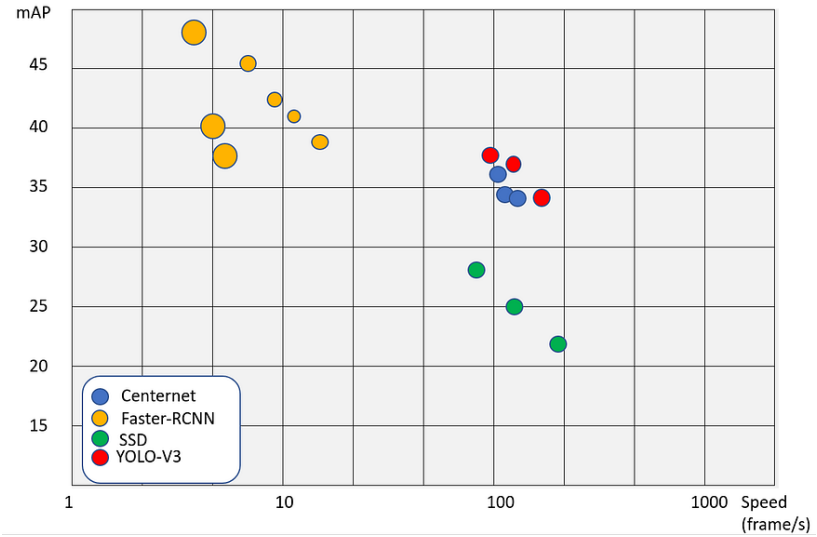


Рисунок 2.7 – Діаграма порівняння різних архітектур ЗНМ

Таким чином, бачимо, що по сукупності параметрів точність/швидкодія найкращою є модель YOLO, тому саме її оберемо для реалізації у нашому модулі розпізнавання людей у касці на основі згорткової нейронної мережі.

2.2 Архітектура згорткової нейронної мережі YOLOv3 для розпізнавання людей у касці

YOLO переосмислила завдання виявлення об'єктів як завдання регресії. Вона йде від пікселів зображення до координат обмежувальних рамок та ймовірностей класів. Тим самим, єдина мережа згортки передбачає кілька обмежувальних рамок і ймовірності класів, що знаходяться у цих областях.

Так як YOLO необхідно лише один погляд на зображення, то метод ковзного вікна не підходить у цій ситуації. Замість цього зображення буде поділено на сітку з комірками розміром S x S. Кожна комірка може містити декілька різних об'єктів для розпізнавання.

По-перше, кожна комірка відповідає за прогнозування кількох обмежувальних рамок. Також кожна комірка прогнозує довірче значення (confidence value) для кожної області, обмеженої рамкою. Іншими словами, це значення визначає можливість знаходження того чи іншого об'єкта в даній

ділянції. Тобто у випадку, якщо якась комірка сітки не має певного об'єкта, важливо, щоб довірче значення для цієї області було низьким.

Коли ми візуалізуємо всі передбачення, ми отримуємо карту об'єктів та впорядковані за довірчим значенням рамки (рис. 2.8).

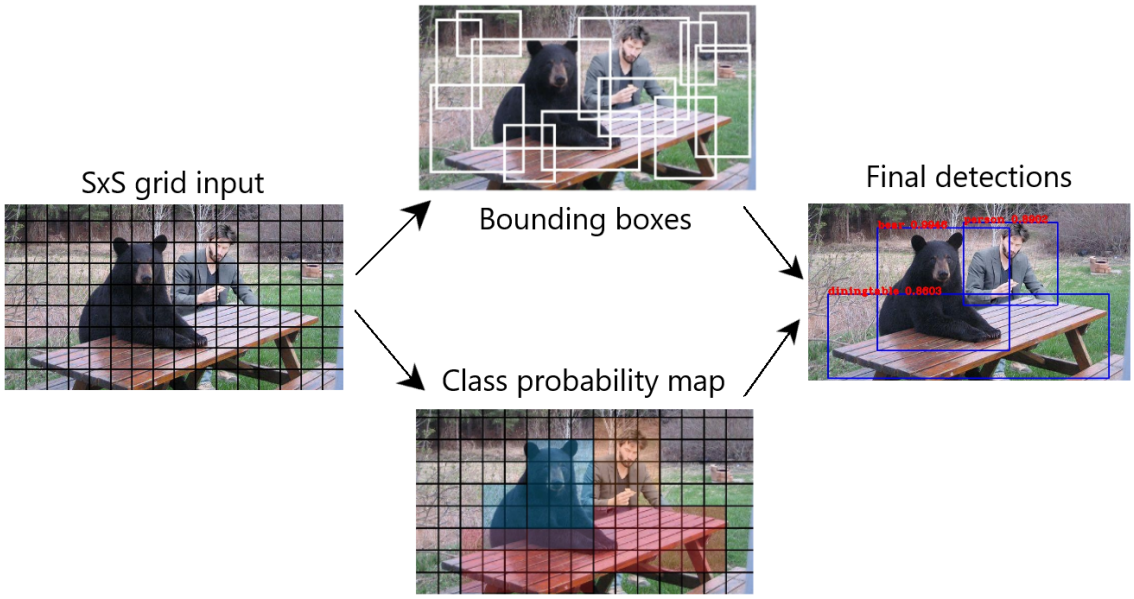


Рисунок 2.8 – Карта об'єктів та впорядковані за довірчим значенням рамки

По-друге, кожна комірка відповідає за передбачення ймовірностей класів. Це не говорить про те, що якась комірка містить якийсь об'єкт, лише ймовірність знаходження об'єкта. Припустимо, якщо комірка передбачає автомобіль, це не гарантує, що автомобіль насправді присутній в ній. Це говорить лише про те, що якщо є об'єкт, то цей об'єкт швидше за все автомобіль.

У YOLO використовуються anchor boxes (якорні рамки / фіксовані рамки) для прогнозування обмежувальних рамок. Ідея anchor box'ів зводиться до попереднього визначення двох різних форм. І таким чином, ми можемо об'єднати два передбачення з двома anchor box'ами (загалом, ми могли б використовувати навіть більшу кількість anchor box'ів). Ці якорі були розраховані за допомогою датасету COCO (Common Objects in Context) та кластеризації k-середніх (K-means clustering).

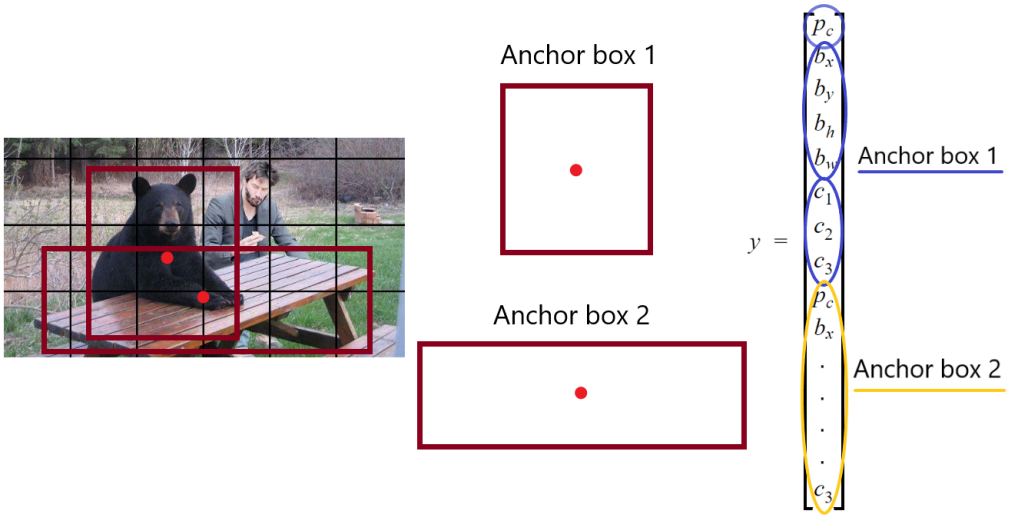


Рисунок 2.9 – Отримання якірних рамок

У нас є сітка, де кожна комірка передбачає:

– Для кожної обмежувальної рамки:
  - 4 координати (t_x, t_y, t_w, t_h),
  - 1 objectness error (помилка об'єктності), яка є показником впевненості у присутності того чи іншого об'єкта.
– Декілька ймовірностей класів.

Якщо ж є деяке зміщення від верхнього лівого кута на c_x , c_y то прогнози будуть відповідати:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{2.1}$$

де p_w (ширина) і p_h (висота) відповідають ширині та висоті обмежувальної рамки. Замість того, щоб передбачати зміщення як у минулій версії YOLOv2, автори прогнозують координати розташування щодо розташування комірки.

Цей висновок є висновок нашої нейронної мережі. Загалом тут S x S x [B * (4+1+C)] висновків, де B – це кількість обмежувальних рамок, яка може

передбачити комірку на карті об'єктів, C – це кількість класів, 4 – для обмежувальних рамок, 1 – для objectness prediction (прогнозування об'єктності). За один прохід ми можемо пройти від вхідного зображення до вихідного тензора, який відповідає виявленим об'єктам на зображенні. Також варто відзначити, що YOLOv3 прогнозує обмежувальні рамки у трьох різних масштабах.

Тепер, якщо ми візьмемо ймовірність і помножимо їх на довірчі значення, ми отримаємо всі обмежувальні рамки, зважені на ймовірність утримання цього об'єкта. Просте знаходження порогового значення позбавить нас прогнозів з низьким довірчим значенням. Для наступного кроку важливо визначити метрику IoU (Intersection over Union/Перетин над об'єднанням). Ця метрика (рис. 2.10) дорівнює співвідношенню площі областей, що перетинаються, до площі областей об'єднання.

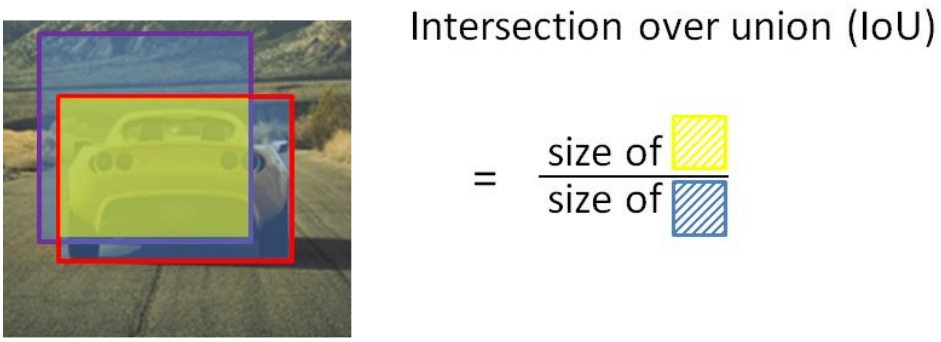


Рисунок 2.10 – Визначення метрики IoU

Після цього все одно можуть залишитися дублікати, і щоб їх позбутися потрібно використовувати “пригнічення не-максимумів” (non-maximum suppression). Пригнічення не-максимумів полягає в наступному: алгоритм бере обмежувальну рамку з найбільшою ймовірністю приналежності до об'єкта, потім, серед інших суміжних обмежувальних рамок з даної області, бере один з найвищим IoU і пригнічує його.

Зважаючи на те, що все робиться за один прогін, ця модель працюватиме майже так само швидко, як і класифікація. До того ж, всі виявлення передбачаються одночасно, що означає, що модель неявно враховує глобальний

контекст. Простіше кажучи, модель може дізнатися, які об'єкти зазвичай зустрічаються разом, їх відносний розмір і розташування об'єктів і так далі.

Структура згорткової нейронної мережі YOLOv3 зображена на рисунку 2.11, а деталізація її шарів подана на рисунку.2.12

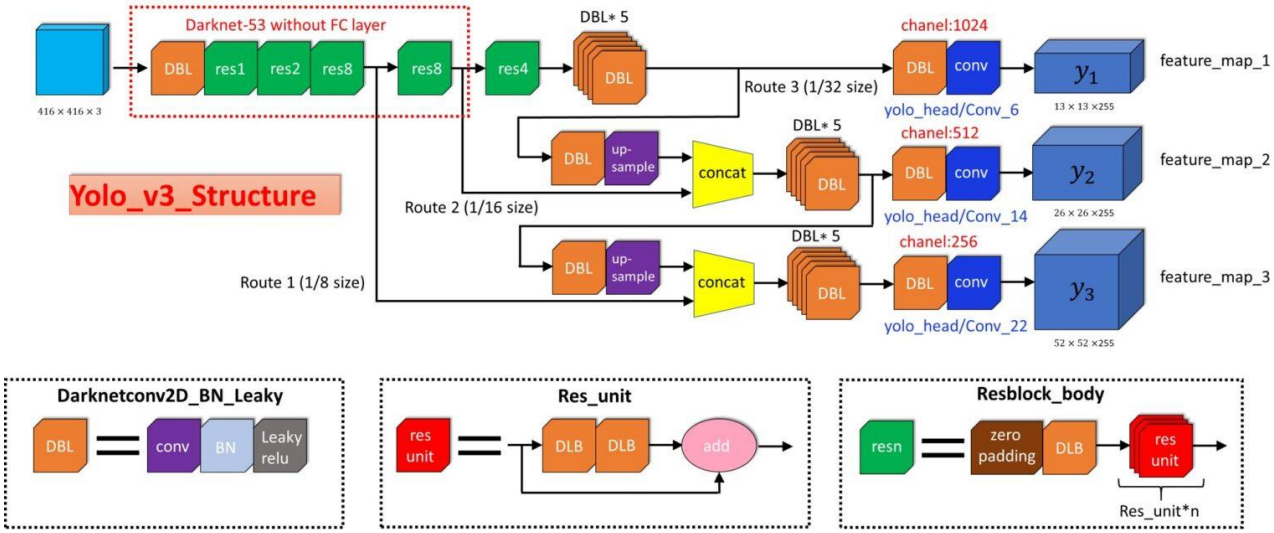


Рисунок 2.11 – Структура згорткової нейронної мережі YOLOv3

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

Рисунок 2.12 – Деталізація шарів згорткової нейронної мережі YOLOv3

По структурі ЗНМ YOLOv3 згідно рисунку.2.11 та 2.12 видно, що вона містить 53 згорткові шари, то найпростішим способом при її програмній реалізації є створення функції, в яку ми будемо передавати важливі параметри, що змінюються від шару до шару.

2.3 Розробка алгоритму роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі

Для навчання моделей комп'ютерного зору на основі штучного інтелекту потрібен набір навчальних даних, що містять зображення людей, деякі з яких у касках, а деякі без них. Ці дані дозволять нам навчити нашу модель розрізняти людей у касках і людей без касок.

Існує три загальні кроки для створення моделі виявлення об'єктів:

- Завантаження зразків даних для класифікації
- Навчання моделі на зразках даних
- Тестування моделі на різних зразках даних, що містять як збіги, так і незбіги

Відповідно до мети роботи та постановкою задачі було розроблено алгоритм роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3, представлений на рисунку 2.13.

Першим кроком в програмному модулі розпізнавання людей у касці буде завантаження бібліотек (блок 1). Основні бібліотеки, що необхідні нам, це OpenCV [28] та ImageAI [29]. Бібліотека ImageAI крім іншого, включає в себе також бібліотеки, необхідні для нейромереж – TensorFlow та Keras.

Потім йде завантаження датасету зображень (блок 2) із репозиторію ImageAI. Після цього із завантажених даних ми відбираємо зображення з касками (блок 3) та зображення з людьми (блок 4).

Далі потрібно завантажити попередньо навчену модель згорткової нейронної мережі YOLOv3 (блок 5). Завантажується вона також з репозиторію

ImageAI. Оскільки YOLOv3 навчена на великій базі даних зображень MS COCO розпізнавати близько 80 класів різних об’єктів, а нам потрібно розпізнавати тільки людей (у касці чи без каски), то наступний крок алгоритму – це налаштування моделі YOLOv3 на детектування тільки людей (блок 6).

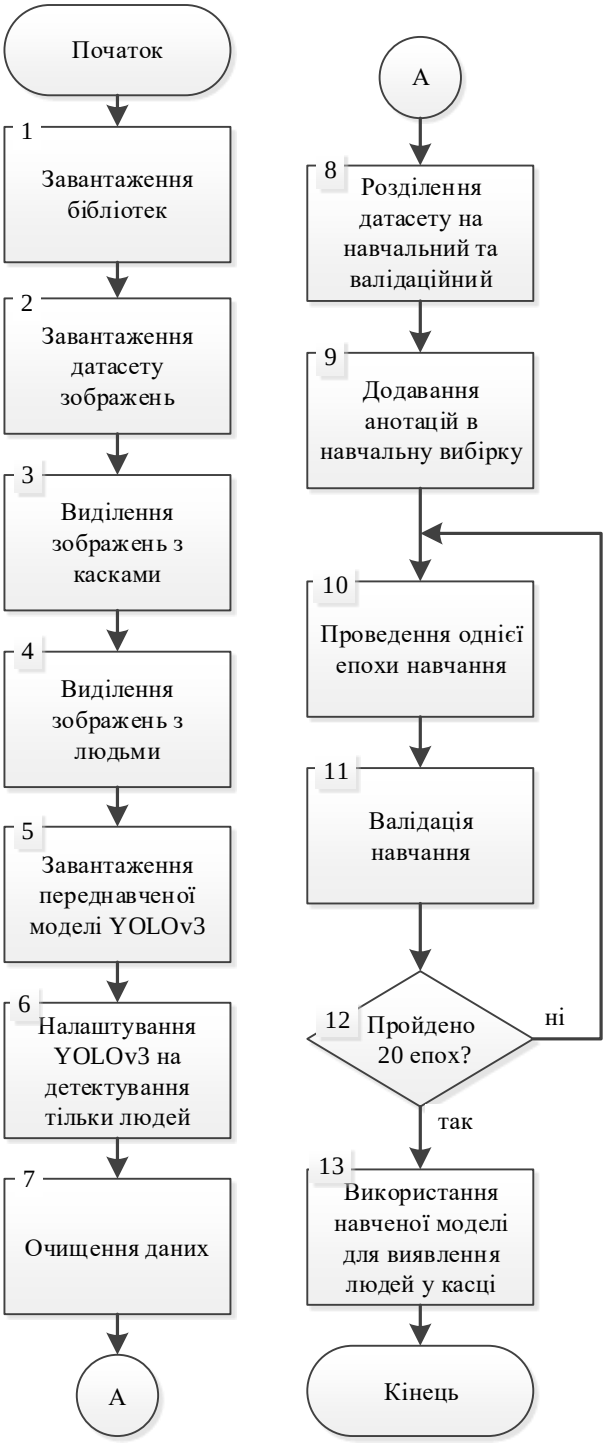


Рисунок 2.13 – Схема алгоритму роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3

Потім потрібно виконати очищення даних (блок 7), тобто із зображень , завантажених для навчання нейромережі, видалити ті, на яких немає людей. Далі потрібно цей очищений набір даних розділити (блок 8) на навчальний (90%) та валідаційний (10%). Після цього потрібно продивитись чи для зображень, що увійшли до навчальної вибірки, присутні анотації до всіх об’єктів, які там визначаються як люди (в касках чи без). Якщо для деяких об’єктів анотації відсутні, то їх потрібно додати (блок 9).

Тепер все готово до запуску процесу навчання. Тому здійснюється проведення однієї епохи навчання (блок 10), після якої відбувається валідація якості навчання на валідаційному наборі (блок 11). Результат валідації зберігається для того, щоб контролювати зменшення втрат від епохи до епохи з метою уникнення перенавчання мережі. Умовний блок 12 визначає чи пройдено потрібну кількість епох (наприклад, 20 епох) і якщо пройдено, то нейромережа вже навчена і її можна використовувати для виявлення людей у касці(блок 13).

Згідно цього алгоритму у п.3.3 описано процес програмної реалізації модуля розпізнавання людей у касці на основі згорткової нейронної мережі.

2.4 Висновок до розділу 2

У розділі було обґрунтувано вибір типу згорткової нейронної мережі для модуля розпізнавання людей у касці. Із таких типів згорткових нейронних мереж як R-CNN, Fast R-CNN, Faster R-CNN, SSD, YOLO для практичної реалізації програмного модуля було обрано нейромережу YOLOv3 як найбільш перспективну за сполученням параметрів точність-швидкодія. Також було проаналізовано архітектуру моделі YOLOv3 та принципи її функціонування.. Крім цього, було розроблено алгоритм роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови та спеціалізованих бібліотек

Для реалізації програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3 обрано мову програмування Python та програмне середовище Anaconda.

Python [30] (читається — «Пайтон») є об'єктно-орієнтованою мовою програмування високого рівня. Наявність у Пайтоні структур даних високого рівня поряд із динамічною семантикою та динамічним зв'язуванням робить її придатною для зручної та швидкої розробки програм. Python підтримує модулі та пакети модулів та дозволяє повторне використанню кодів. Інтерпретатор Python має багато спеціалізованих бібліотек, які доступні і у скомпільованій, і у початковій формах на майже усіх платформах. Мова Python підтримує низку парадигм програмування: функціональне, об'єктно-орієнтоване, процедурне та аспектно-орієнтоване.

Гідні риси мови програмування Python:

- зручний синтаксис (для виділення блоків застосовуються відступи);
- можливість перенесення програм;
- стандартний дистрибутив має значну кількість корисних модулів (включно - модуль розробки графічного інтерфейсу);
- властивість використання мови програмування Python у діалоговому режимі (корисно для простих завдань та експериментування з програмами);
- зручний для вирішення математичних задач (має засоби роботи із цілими та комплексними числами, може використовуватись як потужний калькулятор);
- має відкритий код (доступність коду та можливість його редагувати іншим користувачам).

Так як Python наразі дуже популярний серед розробників систем штучного інтелекту і має багаті функціональні можливості, то обираємо саме його.

Оскільки ми будемо використовувати нейромережі і нам потрібно десь брати датасети зображень та попередньо навчені фреймворки нейронних мереж, то для цих задач нам стане у пригоді така бібліотека як ImageAI [29].

ImageAI створений з урахуванням простоти та підтримує список найсучасніших алгоритмів машинного навчання для прогнозування зображень, прогнозування користувацьких зображень, виявлення об'єктів, виявлення відео, відстеження відеооб'єктів та навчання прогнозуванню зображень. ImageAI наразі підтримує прогнозування та навчання зображень, використовуючи 4 різні алгоритми машинного навчання, навчені на наборі даних ImageNet-1000: MobileNetV2, ResNet50, InceptionV3 and DenseNet121. ImageAI також підтримує виявлення об'єктів, виявлення відео та відстеження об'єктів за допомогою RetinaNet, YOLOv3 та TinyYOLOv3, навчених на наборі даних COCO. Нарешті, ImageAI дозволяє навчати користувацькі моделі для виконання виявлення та розпізнавання нових об'єктів.

Зрештою, ImageAI забезпечує підтримку ширших та більш спеціалізованих аспектів комп'ютерного зору.

ImageAI надає класи та методи для виявлення та розпізнавання власних об'єктів на зображеннях за допомогою власної моделі, навченої за допомогою класу DetectionModelTrainer. Користувачі можуть використовувати свою власну навчену модель YOLOv3 або TinyYOLOv3 та файл **.json**, згенерований під час навчання.

ImageAI забезпечує абстрактні та зручні реалізації найсучасніших технологій комп'ютерного зору. Усі реалізації та код ImageAI можуть працювати на будь-якій комп'ютерній системі з помірною потужністю процесора. Однак швидкість обробки таких операцій, як прогнозування зображень, виявлення об'єктів та інші, на процесорі є низькою та не підходить для програм реального часу. Для виконання операцій комп'ютерного зору в реальному часі з високою

продуктивністю необхідно використовувати технології з підтримкою графічного процесора.

ImageAI використовує основу PyTorch для своїх операцій комп'ютерного зору. PyTorch підтримує як центральні, так і графічні процесори (зокрема, графічні процесори NVIDIA.

Так як розроблюваний програмний модуль повинен обробляти зображення, то для цієї роботи знадобиться бібліотека еомп’ютерного зору OpenCV [28].

OpenCV [28] — це найбільша бібліотека з відкритим вихідним кодом для комп’ютерного зору, яка містить майже всі можливі алгоритми обробки зображень. Використання OpenCV для відстеження об’єктів YOLOv3 поєднує розширені можливості виявлення YOLOv3 із надійними функціями бібліотеки OpenCV, пропонуючи інноваційні рішення для складного виявлення та розпізнавання об’єктів у реальному часі.

3.2 Опис набору даних для навчання та тестування модуля

Для навчання моделей комп'ютерного зору на основі штучного інтелекту потрібен набір навчальних даних, що містять зображення людей, деякі з яких у касках, а деякі без них. Ці дані дозволять нам навчити нашу модель розрізняти людей, які носять каски, і тих, хто їх не носить.

Існує три загальні кроки для створення моделі виявлення об'єктів:

- Завантаження зразків даних для класифікації
- Навчання моделі на зразках даних
- Тестування моделі на різних зразках даних, що містять як збіги, так і незбіги

Пошук навчальних зображень/

Ми використовуємо базу даних зображень під назвою ImageNet, щоб отримати більшість наших вихідних даних для цього етапу. ImageNet підтримує

базу даних зображень, організовану за допомогою ієрархії іменників WordNet. Ця база даних, наприклад, дозволяє нам отримувати всі зображення в ієрархії:

артефакт -> покриття -> одяг -> головний убір -> шолом -> каска

ImageNet також має публічний API, який повертає список URL-адрес зображень на основі ідентифікатора іменника. Ми використовуємо цей API для завантаження великої кількості зображень людей у касках. Форма цього API:

```
http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=<WordNet ID>
```

Ми працювали з двома іменниками WordNet: іменником hardhat (як згадувалося вище) з ідентифікатором n03492922; та іменником misc -> people з ідентифікатором n07942152.

Завантаження зображень людей у касках.

Щоб завантажити всі зображення «hardhat» з API, створюємо наступний блок коду:

```
hardhatLoc = 'http://www.image-
net.org/api/text/imagenet.synset.geturls?wnid=n03492922'

hardhatImages = req.get(hardhatLoc).text
noOfImages = 0

if not os.path.exists('hardhat'):
    os.makedirs('hardhat')

for i in hardhatImages.split('\n'):
    try:
        r = req.get(i, timeout=0.5)
        file = i.split("/")[-1].split('\r')[0]
        if 'image/jpeg' in r.headers['Content-Type']:
            if len(r.content) > 8192:
                with open('hardhat\\' + file, 'wb') as outfile:
                    outfile.write(r.content)
                    noOfImages += 1
                    print('Success: ' + file)
            else:
                print('Failed: ' + file + ' -- Image too small')
        else:
            print('Failed: ' + file + ' -- Not an image')

    except Exception as e:
        print('Failed: ' + file + ' -- Error')

print('********** Download Finished **********')
```

Давайте розглянемо це крок за кроком. Спочатку ми встановлюємо URL-адресу API, який ми хочемо викликати, за допомогою змінної hardhatLoc. Потім ми запитуємо API, використовуючи змінну req, яка вказує на нашу бібліотеку запитів, і ми об'єднуємо два методи разом, використовуючи цю бібліотеку:

– .get – приймає рядкову URL-адресу та повертає відповідь,

– .text – приймає двійкову відповідь та перетворює її на рядкове значення.

Після того, як ми маємо список вхідних зображень з API, ми перевіряємо, чи є у нас папка для завантаження зображень, за допомогою методу os.path.exists. Якщо у нас немає каталогу для їх розміщення, метод .makedirs() створить каталог.

Далі ми запускаємо наш цикл for, який використовує URL-адреси, які ми завантажили з API. Ми розділяємо рядок на кожному символі нового рядка (\n) і поміщаємо цей рядок у змінну i. Щоб виявити будь-які помилки під час завантаження файлу, ми також відкриваємо блок try.

У блоці try цей процес буде перебирати всі URL-адреси, які ми завантажили з API. Для кожного циклу або URL-адреси відбувається наступний процес:

– Використовуємо .get(), щоб спробувати завантажити файл. Ми також використовуємо необов'язковий параметр timeout, щоб зупинити спроби завантажити файл через 0,5 секунди.

– Потім розділяємо рядок URL-адреси двічі за допомогою .split().

– Перший поділ відбувається кожні "/", а значення -1 у масиві отримує останнє значення.

– Другий поділ видаляє кінцевий символ повернення каретки.

– На наступному кроці перевіряється, чи є завантажений файл jpeg, використовуючи масив .headers. Цей масив містить усі HTML-заголовки з відповіді.

– Після підтвердження того, що це jpeg, перевіряємо, чи файл має пристойний розмір, щоб ми не обробляли жодних зображень помилок.

– Якщо завантажений файл відповідає нашим вимогам, ми використовуємо блок оператора with, щоб зберегти файл у папці hardhat.

– Нарешті, закриваємо всі наші блоки if та try з операторами, щоб вивести будь-які помилки.

Після запуску цього блоку коду, бачимо, що код виконується та виводить інформацію про успішне або невдале завантаження кожного зображення через API. Це може зайняти кілька хвилин, тому для завершення чекаємо на завершальне повідомлення.

Перевірка завантажених даних.

Після запуску блоку коду завантажувача можливо з’являться кілька повідомлень про помилку. Це добре, оскільки ми хочемо мати точні зображення в нашій папці для використання класифікатором. Після цього можна відкрити вікно провідника та переглянути папку, яка містить даний Jupyter notebook. Тепер у нас має бути підпапка під назвою "hardhat".

Можна відкрити цю папку hardhat та перевірити завантажені зображення (рис.3.1).

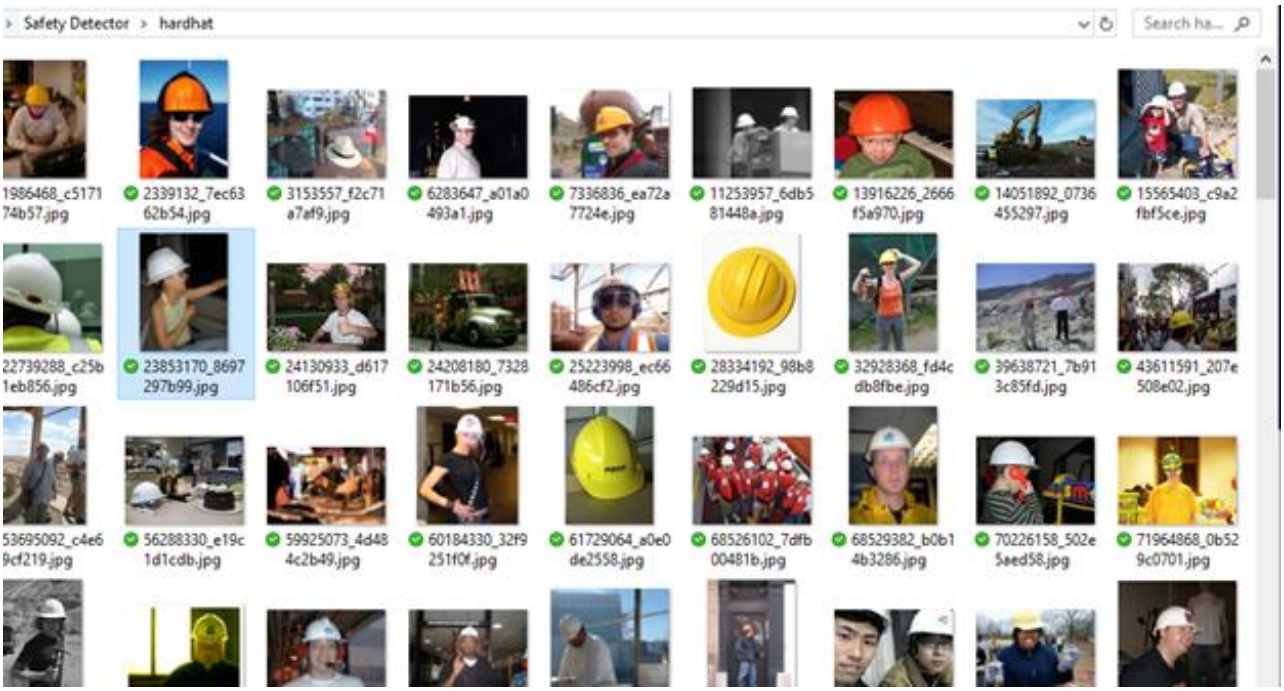


Рисунок 3.1 – Приклади отримання зображень людей з каскою

У нас вийшло 687 зображень hardhat, переважно людей, які носять каску, але деякі зображення мають окремо каску.

Завантаження зображень людей без каски.

Тепер, коли у нас є багато зображень людей у касках, потрібно скопіювати блок коду та завантажити зразок набору людей без касок. Продублюємо існуючий блок коду та внесемо такі зміни:

- Змінимо «hardhat» на «people» скрізь, де воно використовується, включаючи назви змінних та текстові рядки.
- Змінимо ідентифікатор API з n03492922 на n07942152.

Код тепер має виглядати так:

```
peopleLoc = 'http://www.image-
net.org/api/text/imagenet.synset.geturls?wnid=n07942152'
peopleLoc = 'http://www.image-
net.org/api/text/imagenet.synset.geturls?wnid=n07942152'

peopleImages = req.get(peopleLoc).text
noOfImages = 0

if not os.path.exists('people'):
    os.makedirs('people')

for i in peopleImages.split('\n'):
    try:
        r = req.get(i, timeout=0.5)
        file = i.split("/")[-1].split('\r')[0]
        if 'image/jpeg' in r.headers['Content-Type']:
            if len(r.content) > 8192:
                with open('people\\' + file, 'wb') as outfile:
                    outfile.write(r.content)
                    noOfImages += 1
                    print('Success: ' + file)
            else:
                print('Failed: ' + file + ' -- Image too small')
        else:
            print('Failed: ' + file + ' -- Not an image')

    except Exception as e:
        print('Failed: ' + file + ' -- Error')

print('********** Download Finished **********')
```

Коли ми запустимо цей блок коду, він має працювати так само, як і раніше. Але цього разу, якщо ми перевіримо папку, де збережено наш блокнот Jupyter, там має бути папка "people", що містить загальні зображення груп людей без касок (рис. 3.2).

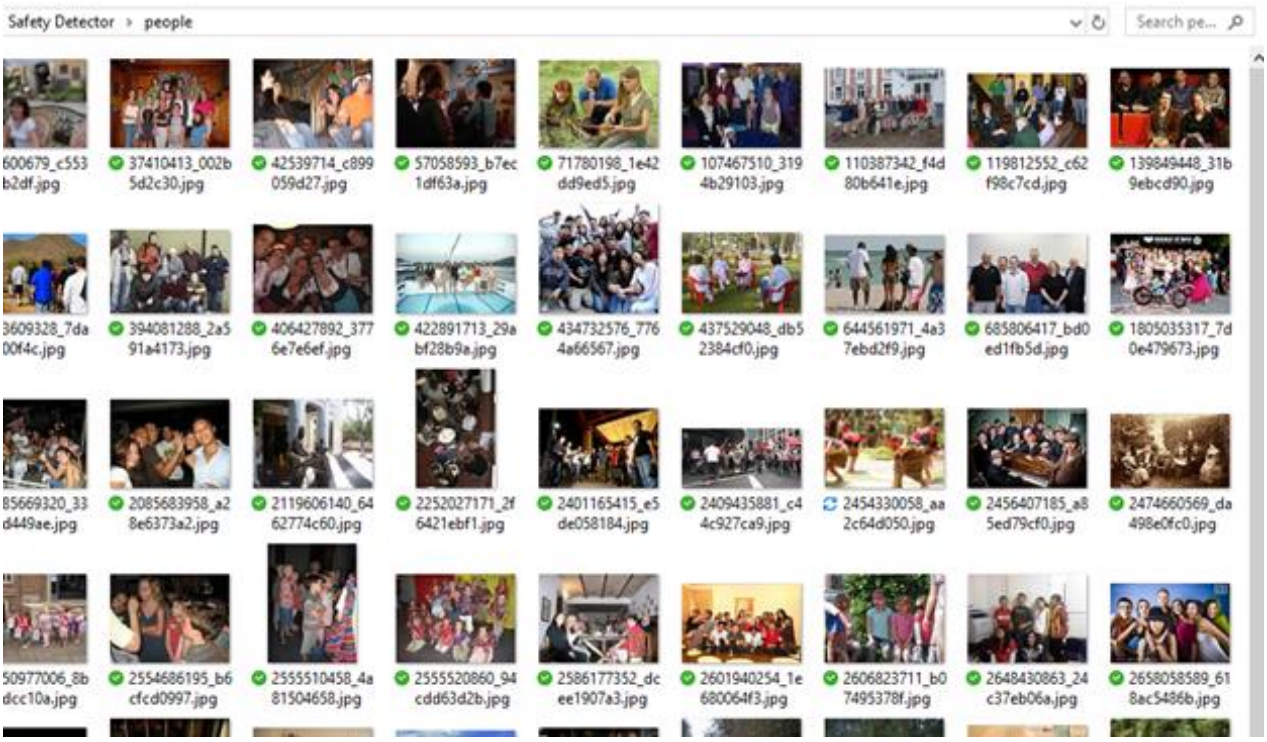


Рисунок 3.2 – Приклади отримання зображень людей без каски

Зображень людей без касок візьмемо 713. Разом у наборі даних у нас вийде 1400 зображень (687 у касках та 713 без касок). Цей набір 1400 зображень розділимо на навчальний набір 90% і валідаційний набір 10% (1260 навчальних зображень + 140 валідаційних зображень). А для тестування повністю навченої моделі можна потім взяти довільну кількість тестових зображень людей як у касках, так і без касок.

Таким чином, маємо набір даних, який можна використовувати для навчання та тестування розроблюваного програмного модуля для виявлення людей у касках. Далі розглянемо результати, які можна отримати, використовуючи отримані зображення з попередньо навченою моделлю згорткової нейромережі YOLO.

3.3 Програмна реалізація модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3

Перш ніж використовувати розроблений програмний модуль для контролю носіння касок на підприємстві (покращення безпеки на робочому місці), нам потрібно встановити необхідні інструменти: OpenCV та ImageAI.

OpenCV — це бібліотека комп'ютерного зору з відкритим кодом з інтерфейсами C++, Python, Java та MATLAB. ImageAI — це бібліотека машинного навчання, яка спрощує навчання ШІ та виявлення об'єктів на зображеннях. Ці дві бібліотеки надзвичайно спрощують вирішення низки проблем виявлення об'єктів на зображеннях та відео. Ми одразу перейдемо до нашого рішення, налаштувавши ці бібліотеки за допомогою Python у Jupyter Notebook (у Windows).

Налаштування середовища розробки.

Птрібно встановити платформу для обробки даних Anaconda Python та налаштувати її з використанням основних налаштувань за замовчуванням.

Щоб створити Jupyter Notebook для написання програми, потрібно встановити певні версії OpenCV, Tensorflow, Keras та ImageAI за допомогою Anaconda. Знайдемо та запустимо командний рядок Anaconda з меню «Пуск» і введемо таку команду:

```
conda create -n ImageAI -c anaconda keras=2.3.1 tensorflow=1.15.0 tensorflow-gpu=1.15.0 jupyter
```

Далі ми перейдемо до середовища ImageAI та використаємо pip для встановлення OpenCV та ImageAI за допомогою таких команд:

```
conda activate ImageAI
pip install opencv-python==4.1.2.30 imageai
```

Ми використовуємо останню версію ImageAI, 2.1.5. Потрібно встановити ще один елемент — бібліотеку requests — щоб ми могли використовувати деякі специфічні методи HTML.

Тепер створимо новий блокнот у Jupyter під назвою "Hard-Hat-Control" — і додамо наступний блок коду для ініціалізації бібліотек:

```
import cv2 as cv
from imageai.Detection import ObjectDetection as od

import numpy as np
import requests as req
import os as os
```

Два ключові імпорти тут – це OpenCV (у змінній cv) та компонент детекції ImageAI (у змінній od). Інші три бібліотеки – це загальні бібліотеки, специфічні для Python: numpy використовується для великих масивів і матриць; requests дозволяє працювати з HTTP-запитами, а os використовується для роботи з функціями, специфічними для операційної системи.

Далі можна потестувати OpenCV, щоб перевірити отримання даних. Наприклад, отримання зображення з hiclipart.com та збереження його як файлу в каталозі Jupyter Notebook. При цьому використовується два методи з бібліотеки requests:

.get(url) – отримує веб-контент за певною URL-адресою

.content – надає доступ до необробленого контенту, отриманого з URL-адреси

Ми використовуємо подібний процес щоб отримати навчальні дані для нашої моделі виявлення. Після того як отримано набір навчальних даних (див. п. 3.2), можна приступати до навчання та тестування розроблюваного програмного модуля для виявлення людей у касках.

Розглянемо деякі попередньо навчені моделі, які ми можемо використовувати в ImageAI для виявлення людей на зображеннях.

ImageAI надає низку дуже зручних методів для виконання виявлення об'єктів на зображеннях та відео, використовуючи комбінацію Keras, TensorFlow, OpenCV та навчених моделей.

Вибір попередньо навченої моделі.

У репозиторії ImageAI GitHub зберігається низка попередньо навчених моделей для розпізнавання зображень та виявлення об'єктів, включаючи:

- ResNet – згорткова нейронна мережа, створена для високої продуктивності та точності, але з довшим часом виявлення.
- YOLOv3 – реалізація алгоритму You Only Look Once (YOLO), створена для помірної продуктивності, точності та часу виявлення.
- TinyYOLOv3 – ще одна реалізація YOLO, але створена для швидкого виявлення та продуктивності з помірною точністю.

Оскільки ми сподіваємося створити досить точну програму виявлення, яка може потребувати роботи з відео, то виберемо YOLOv3. Почнемо новий блок коду та введемо наступне:

```
modelRetinaNet =
'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_
v2.0.1.h5'
modelYOLOv3 =
'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo.h5'
modelTinyYOLOv3 =
'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo-tiny.h5'

if not os.path.exists('yolo.h5'):
    r = req.get(modelYOLOv3, timeout=0.5)
    with open('yolo.h5', 'wb') as outfile:
        outfile.write(r.content)
```

Усі три посилання включено в налаштування цього блоку коду, щоб їх можна було змінити за потреби. Використовуючи метод, подібний до попереднього, цей блок коду завантажить відповідну модель і збереже її в базовому каталозі проекту, готову до використання.

Виявлення людей.

Після завантаження моделі нам потрібно завантажити її в наш детектор. Це потрібно зробити лише один раз, оскільки завантаження моделі може зайняти деякий час, тому створимо блок коду лише для завантаження моделі:

```
detector = ObjectDetection()
```

```
detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath('yolo.h5')
detector.loadModel()
```

Цей блок коду завантажує модель ЗНМ у змінну детектора за допомогою:

- setModelTypeAsYOLOv3() – встановлює модель, яку ми використовуємо для виявлення об’єктів, як YOLOv3; інші опції включають setModelTypeAsRetinaNet або setModelTypeAsTinyYOLOv3,
- setModelPath() – надає місце розташування моделі,
- loadModel() – завантажує модель у пам’ять.

Коли модель активна та готова до роботи, можна випадковим чином витягнути зображення та почати виявляти людей. Створимо новий блок коду та додамо наступне:

```
import random
peopleImages = os.listdir("people")
randomFile = peopleImages[random.randint(0, len(peopleImages) - 1)]

detectedImage,  detections  =  detector.detectObjectsFromImage(output_type="array",
input_image="people/{0}".format(randomFile), minimum_percentage_probability=30)
convertedImage = cv.cvtColor(detectedImage, cv.COLOR_RGB2BGR)
showImage(convertedImage)

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ",
eachObject["box_points"] )
    print("--------------------------------")
```

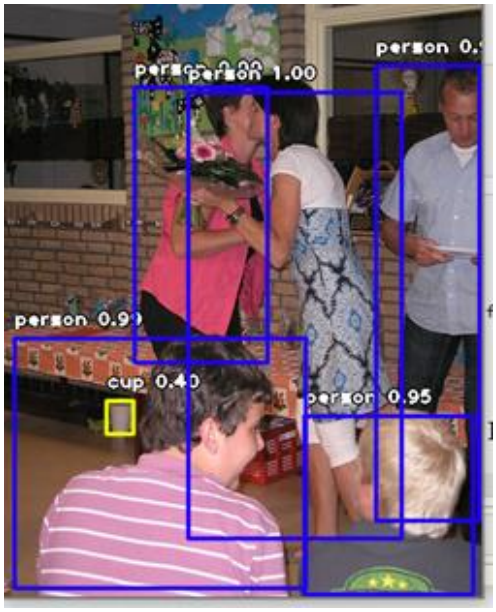
Цей блок коду розділено на три розділи. Перший розділ просто вибирає випадковий файл з нашого каталогу "people" для виконання виявлення. Друга частина використовує модель для виконання виявлення шляхом:

– Використання методу detectObjectsFromImage, який може повертати масив NumPy (у першу змінну, detectedImage) або виводити дані у файл зображення:
  - Параметр input_image вказує на файл, на якому потрібно виконувати виявлення,
  - Параметр minimum_percentage_probability вказує, наскільки впевнена модель у тому, що об'єкт відповідає навченому типу,
  - Метод також виводить список виявлень у змінну detections.
– Взяття виявленого масиву зображень NumPy та завантаження його в зображення OpenCV. OpenCV використовує масив BGR для зберігання даних, тому нам потрібно конвертувати масив RGB у BGR.
– Використання методу showImage в OpenCV для відображення зображення.

Після відображення зображення останній блок коду виводить різні виявлення та обмежувальні рамки, які їх покривають. Під час першого запуску виявлень помічаємо, що для відображення зображення потрібно багато часу. Це тому, що модель «розігрівається». Однак, якщо повторно запустити цей блок коду, зображення повинні відображатися досить швидко. Також, залежно від зображення, можна побачити, що під час запуску моделі виявляються не лише люди (рис. 3.3). Це тому, що надана модель YOLOv3 була навчена приблизно на 80 різних типах об'єктів.

Усі об’єкти також позначені тегами. Якщо натиснути будь-яку клавішу, щоб закрити зображення, і подивитися на виведений текст, ви побачите три значення:

Назва об’єкта. У цьому випадку більшість об’єктів мають бути людиною, але також можна знайти чашку, краватку, сумочку тощо.




```
tie  ---  42.86769926548004  ---  [78, 146, 87, 173]
handbag  ---  49.53037202358246  ---  [22, 178, 44, 245]
person  ---  31.346148252487183  ---  [279, 113, 331, 200]
person  ---  36.62037551403046  ---  [91, 113, 143, 201]
person  ---  44.622498750686646  ---  [278, 114, 332, 307]
person  ---  74.34216737747192  ---  [79, 117, 141, 328]
person  ---  80.34021258354187  ---  [376, 108, 436, 314]
person  ---  90.66885113716125  ---  [171, 96, 243, 173]
person  ---  99.65248703956604  ---  [32, 104, 99, 332]
person  ---  99.71648454666138  ---  [308, 115, 386, 329]
person  ---  99.73611831665039  ---  [390, 138, 470, 327]
person  ---  99.80701804161072  ---  [171, 135, 250, 329]
person  ---  99.82082843780518  ---  [231, 127, 312, 332]
person  ---  99.85131621360779  ---  [112, 137, 180, 332]
```

Рисунок 3.3 – Приклад виявлення об’єктів попередньо натренованою моделлю YOLOv3

Рівень впевненості моделі в тому, що виявлення правильне, можливо, 98% або 44%, або будь-що інше. Мінімальна ймовірність (minimum_percentage_probability) встановлюється вибором порогу (зазвичай поріг =0,5).

Обмежувальна рамка, яка охоплює об’єкт у піксельних розмірах від верхнього лівого кута до нижнього правого кута, наприклад, від X,Y до Z,W.

Виявлення лише людей (без каски).

Все це добре, але ми хочемо виявляти лише людей у нашому зображенні. ImageAI надає метод для фільтрації об’єктів: detectCustomObjectsFromImage. Змінимо другий розділ нашого попереднього блоку коду, щоб побачити, як це працює:

```
peopleOnly = detector.CustomObjects(person=True)
detectedImage, detections =
detector.detectCustomObjectsFromImage(custom_objects=peopleOnly,
output_type="array", input_image="people/{0}".format(randomFile),
minimum_percentage_probability=30)
convertedImage = cv.cvtColor(detectedImage, cv.COLOR_RGB2BGR)
showImage(convertedImage)
```

Якщо запустити цей блок коду кілька разів, то побачимо, що тепер на всіх зображеннях виявляються лише люди. А тепер, коли у нас є прийнятне налаштування детектора, ми створимо власну модель детектора для пошуку людей у касках.

Тепер, коли у нас є набір зображень і налаштований детектор, навчимо нашу власну модель виявляти, чи люди носять каски. Щоб отримати найкращі результати від нашої моделі, нам потрібно переконатися, що дані, на яких ми її навчаємо, є точними. Нам також потрібно анотувати дані та залишити деякі з них для валідації нашої моделі після її навчання.

Очищення даних

Завжди якісний результат дає ручний перегляд набору даних, навіть якщо це швидкий перегляд, щоб переконатися, що дані, які використовуються, чисті. Однак, оскільки у нас налаштований детектор, який працює досить надійно, ми дозволимо детектору очистити наші дані. Очистимо блок коду, який містить код для відображення випадкових зображень, і замінимо його наступним:

```
hardhatImages = os.listdir("hardhat")
peopleOnly = detector.CustomObjects(person=True)

for i in hardhatImages:
    imageFile = "hardhat/{0}".format(i)
    detectedImage, detections =
detector.detectCustomObjectsFromImage(custom_objects=peopleOnly,
output_type="array", input_image=imageFile, minimum_percentage_probability=30)
    if len(detections) < 1:
        os.remove(imageFile)
```

Цей блок коду отримує список завантажених нами зображень людей у касках та визначає детектор, щоб він виявляв лише людей. Оскільки ми знаємо, що всі зображення стосуються касок, і оскільки ми хочемо навчати нашу модель лише на людях у касках, ми можемо використовувати цей детектор, щоб переконатися, що наші дані здебільшого правильні. Потім код перебирає кожне зображення та намагається виявити людей на ньому. Якщо це не вдається, зображення видаляється.

Цей процес відбувається досить швидко, і ми повинні отримати близько 560 зображень.

Розділення даних.

Наступний крок – розділити наш набір даних на два. Один набір буде призначений для навчання моделі, інший – для валідації нашої навченої моделі. Ми можемо або вручну розділити дані, або написати код для їх розділення. Щоб спробувати максимально автоматизувати ці процеси, розділимо дані програмно. Почнімо новий блок коду та додамо наступне:

```
if not os.path.exists('hardhat/train/images'):
    os.makedirs('hardhat/train/images')
if not os.path.exists('hardhat/validation/images'):
    os.makedirs('hardhat/validation/images')

hardhatImages = os.listdir("hardhat")
hardhatTrainNums = round(len(hardhatImages) * 0.90)

for i in range(0, hardhatTrainNums):
    file = "hardhat/" + hardhatImages[i]
    if os.path.isfile(file):
        os.rename(file, "hardhat/train/images/" + hardhatImages[i])

hardhatImages = os.listdir("hardhat")

for i in hardhatImages:
    if i.is_file():
        file = "hardhat/" + i
        os.rename(file, "hardhat/validation/images/" + i)
```

Цей блок коду використовує низку методів з бібліотеки "os" для розділення файлів на навчальні дані та дані для валідації. Загальний порядок дій такий:

– Створення папок для навчання та валідації (os.makedirs),

– Перелік зображень у вихідному коді та позначення 90% з них (os.listdir),

– Переміщення 90% файлів до папки train/images (os.rename)

– Позначення решти зображень у вихідному коді (знову os.listdir)

– Переміщення решти зображень до папки validation/images (знову os.rename)

Після завершення цього процесу у нас буде два набори відсортованих, очищених даних, які ми можемо використовувати для навчання моделі виявленню людей у касках, а потім для перевірки ефективності цієї моделі.

Анотування зображень.

Після отримання навчальних даних нам потрібно анотувати зображення. Іноді можна знайти зображення, які вже мають анотації, але здебільшого це потрібно зробити, визначивши обмежувальні рамки навколо об'єктів, які потрібно виявити. Це можна робити за допомогою PASCAL VOC, XML-формату, який стандартизує набори даних зображень для розпізнавання об'єктів. На щастя, існує зручна утиліта під назвою LabelImg, яка трохи спрощує цей процес.

Чим більше зображень анотувати, тим краще буде навчена модель. Це не робить процес менш трудомістким, але він точно вартий зусиль. Після того, як закінчимо з папкою «train», нам також потрібно буде додати анотації до папки «validation». Для спрощення, можна завантажити готові анотовані зображення з інтернету. Всі анотації накопичуються у папці у вигляді XML-файлів.

Ми дізналися, що потрібно для очищення та підготовки набору даних зображень, щоб підготувати його до навчання моделі ШІ.

Цей етап видається дещо трудомістким. Зрештою, ми прагнемо навчати та використовувати ШІ, а не вручну просіювати дані. Але при роботі з ШІ ми стикаємося з відомим правилом, яке зустрічається в багатьох галузях обчислень: сміття на вході - сміття на виході. Якщо ви хочете, щоб ваша модель генерувала хороші результати, вам потрібно навчати її на якісних даних.

Таким чином, ми очистили наші дані та розділили їх на навчальні та валідаційні набори даних.

Тепер розпочнемо процес створення власної моделі виявлення об'єктів. Загальні кроки навчання власної моделі виявлення:

– Навчання моделі.

- Валідація моделі; якщо валідація погана, налаштування моделі та повторне навчання.
- Візуальне тестування моделі та результатів.
- Розгортання моделі.

Навчання моделі.

Створимо новий блок коду та введемо наступне:

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="hardhat")

trainer.setTrainConfig(object_names_array=["person     hardhat"],     batch_size=4,
num_experiments=20,
                       train_from_pretrained_model="yolo.h5")

trainer.trainModel()
```

Цей блок коду використовує новий метод у класі виявлення ImageAI, DetectionModelTrainer. Процес навчання будь-якої моделі такий:

- Визначити новий метод DetectionModelTrainer().
- Встановити тип моделі як YOLOv3.
- Встановити каталог, який містить дані. Зверніть увагу, що ця папка повинна містити одну папку з назвою "train" та одну з назвою "validation". Ці папки повинні містити одну папку з назвою "images" та ще одну з назвою "annotations".
- Встановити конфігурацію тренера наступним чином:
  - Вкажіть назви анотацій, що використовуються в зображеннях. У нашому випадку ми використовуємо лише "person hardhat".
  - Визначте розмір пакету, у нашому випадку чотири. Це визначає, скільки зображень модель буде навчати в кожному пакеті. Чим більший розмір пакету, тим краще можна навчити модель, але тим потужніший графічний процесор буде потрібен.

- Вкажіть, скільки ітерацій моделювання потрібно виконати за допомогою num_experiments. Чим більше ітерацій, тим кращий кінцевий результат, але тим більше часу це займе.
- За бажанням, вкажіть попередньо навчену модель для перенесення навчання, щоб швидше отримати кращий результат.

– Запустити процес навчання моделі за допомогою trainModel().

Модель розпочне навчання та виведе статус для кожного циклу (епохи) навчання. Для кожної з цих епох повідомляється про помилку (втрати), яка визначає, чи модель краща, ніж у попередній епосі. Якщо так, то модель буде збережена, тому переконайтеся, що є достатньо вільного місця на диску!

Валідація (перевірка) моделі.

Навчання моделі може тривати дуже довго. Навчання цієї моделі протягом 20 епох зайняло трохи більше чотирьох годин. Деякі рекомендації щодо навчання моделей пропонують понад 200 годин. Перш ніж лишати наш комп'ютер на кілька днів для навчання моделі, давайте подивимося, що створюється після того, як наша модель пройшла навчання протягом 20 епох (рис. 3.4).

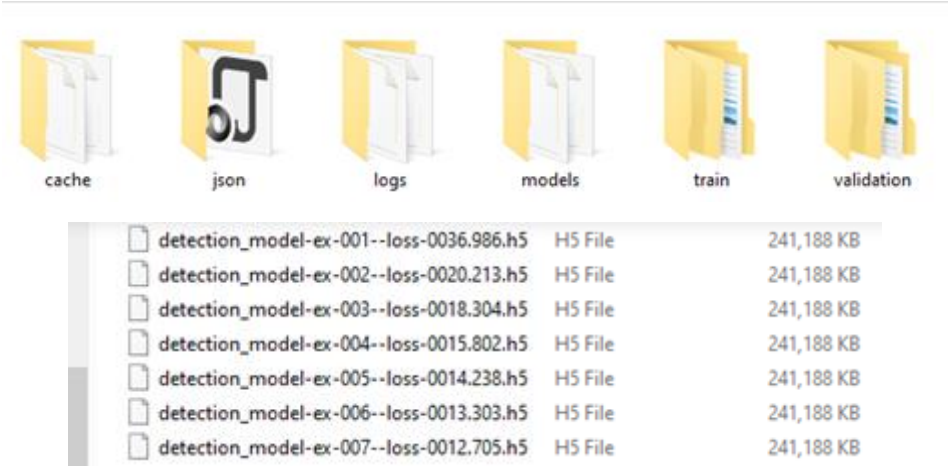


Рисунок 3.4 – Результати навчання моделі протягом 20 епох

У каталозі "hardhat" буде створено кілька додаткових каталогів: "cache", "json", "logs" та "models" (рис. 3.4). Дві важливі директорії тут - "json" та "models". Директорія "json" містить файл конфігурації JSON, необхідний для

використання моделі. Директорія "model" містить кілька досить великих файлів моделей з порядковими номерами. Кожен з цих файлів є результатом епохи навчання моделі, яка була кращою за попередню.

Отже, маємо кілька моделей, які теоретично стають все кращими й кращими, залежно від номера епохи. Перевіримо їх, запустивши валідацію. Почнемо новий блок коду та введемо наступне:

```
trainer.evaluateModel(model_path="hardhat\models\detection_model-ex-020--loss-
0008.462.h5",
                      json_path="hardhat\json\detection_config.json",
iou_threshold=0.5,
                      object_threshold=0.3, nms_threshold=0.5)
```

Єдина зміна, яку потрібно буде внести, це шлях до моделі з рядком: hardhat\models\detection_model-ex-020--loss-0008.462.h5, оскільки кожен навчальний запуск буде різним. Цей метод приймає такі параметри:

- model_path – вказує модель, для якої ви хочете провести перевірку,
- json_path – вказує файл конфігурації для навчання моделі,
- iou_threshold – представляє співвідношення перетину та об'єднання передбачуваної та фактичної обмежувальної рамки,
- object_threshold – рівень достовірності виявлень, які потрібно видалити,
- nms_threshold – рівень достовірності, коли виявлено кілька обмежувальних рамок.

Коли ми запускаємо цю перевірку для 20-ітераційної моделі, ми отримуємо середню точність 0,84464, або приблизно 84,5%, що непогано. Але як це порівнюється з деякими іншими? Давайте розширимо наш блок коду до наступного:

```
model05 = trainer.evaluateModel(model_path="hardhat\models\detection_model-ex-005--
loss-0014.238.h5",
                      json_path="hardhat\json\detection_config.json",
iou_threshold=0.5,
                      object_threshold=0.3, nms_threshold=0.5)
model10 = trainer.evaluateModel(model_path="hardhat\models\detection_model-ex-010--
loss-0011.053.h5",
                      json_path="hardhat\json\detection_config.json",
iou_threshold=0.5,
                      object_threshold=0.3, nms_threshold=0.5)
model15 = trainer.evaluateModel(model_path="hardhat\models\detection_model-ex-015--
loss-0009.620.h5",
                      json_path="hardhat\json\detection_config.json",
iou_threshold=0.5,
                      object_threshold=0.3, nms_threshold=0.5)
model20 = trainer.evaluateModel(model_path="hardhat\models\detection_model-ex-020--
loss-0008.462.h5",
                      json_path="hardhat\json\detection_config.json",
iou_threshold=0.5,
                      object_threshold=0.3, nms_threshold=0.5)

print('------------------------------------------------------')
print('Iteration 05:', model05[0]['average_precision']['person hardhat'])
print('Iteration 10:', model10[0]['average_precision']['person hardhat'])
print('Iteration 15:', model15[0]['average_precision']['person hardhat'])
print('Iteration 20:', model20[0]['average_precision']['person hardhat'])
print('------------------------------------------------------')
```

Результат виконання цього коду подано на рисунку 3.5

```
------------------------------------
Iteration 05: 0.7164811838244572
Iteration 10: 0.786450228152799
Iteration 15: 0.8354553845774733
Iteration 20: 0.8446486610895216
------------------------------------
```

Рисунок 3.5 – Результати валідації моделі на різних епохах

Виконання цього блоку коду займе деякий час, оскільки йому потрібно завантажити 4 різні моделі, перевірити їх та зберегти результати. Коли цей блок коду завершиться, останні кілька рядків нададуть нам результати:

- 5 ітерацій – 71%,
- 10 ітерацій – 78%,
- 15 ітерацій – 83%,
- 20 ітерацій – 84%,

Отже, з цих результатів бачимо, що чим більше ми виконуємо епох, тим кращою стає наша модель. Однак, у певний момент, віддача зменшується, тому нам також потрібно враховувати це під час навчання моделі.

Отже, ми отримали навчену модель для розпізнавання людей у касках. Як використовувати цю модель для виявлення, чи носять люди каски, розглянемо в наступному розділі 4.

3.4 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек ImageAI та OpenCV для програмної реалізації модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3. Проведено опис процеесу отримання набору даних зображень на основі ImageAI для навчання нейронної мережі YOLOv3. Проаналізовано структуру набору даних, принципи формування анотацій для обмежувальних прямокутників. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3, що складається із завантаження та формування навчального та валідаційного наборів даних, завантаження фреймворку попередньо навченої нейронної мережі YOLOv3, її налаштування, навчання нейронної мережі YOLOv3 та використання навченої нейронної мережі YOLOv3 для розпізнавання людей у касці.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі

Розглянемо як працює розроблений нами програмний модуль.

В основі модуля є готова до використання власна модель ЗНМ. Тепер будемо її використовувати. Для цього визначимо кілька нових блоків коду, подібних до тих, що ми використовували раніше. Спочатку, щоб визначити нашу власну модель виявлення, створимо наступний блок коду:

```
from imageai.Detection.Custom import CustomObjectDetection

detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("hardhat\models\detection_model-ex-020--loss-0008.462.h5")
detector.setJsonPath("hardhat\json\detection_config.json")
detector.loadModel()
```

Цей блок коду працює подібно до розглянутого у п.3.3, попередньо навченого детектора моделей, за винятком того, що ми визначаємо шлях до моделі та шлях JSON для конфігурації. Це дозволяє ImageAI імпортувати імена об'єктів з файлу конфігурації. Далі копіюємо наш попередній блок коду, який виявляв би людей на зображеннях, у наш каталог "people" та трохи змінюємо код, щоб виявляти їх на наших завчасно підготовлених валідаційних зображеннях:

```
import random
peopleImages = os.listdir("people")
randomFile = peopleImages[random.randint(0, len(peopleImages) - 1)]

detectedImage, detections = detector.detectObjectsFromImage(output_type="array",
input_image="people/{0}".format(randomFile), minimum_percentage_probability=30)
convertedImage = cv.cvtColor(detectedImage, cv.COLOR_RGB2BGR)
showImage(convertedImage)

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ",
eachObject["box_points"] )
    print("--------------------------------")
```

Якщо запустити ці два блоки коду та подивитися на результуюче зображення, то побачимо, що наша модель виконує досить непогану роботу (рис. 4.1) навіть після лише 20 ітерацій навчання. Якщо кілька разів повторно запустити блок коду для випадкового зображення, то побачимо, що майже всі зображення з людьми в касках виявляються.

Рисунок 4.1 –Результат роботи програмного модуля

Таким чином, скориставшись перевагами двох основних бібліотек, OpenCV та ImageAI, ми змогли використовувати попередньо навчену модель виявлення об'єктів та розробити власну модель для виявлення людей у касках.

Тепер можна поєднати ці дві моделі та аналізувати зображення, щоб переконатися, що всі люди на зображенні носять каски, а якщо хтось не носить каску в робочому середовищі, то попередити таких людей,. Ось як це зробити:

```
person_detector = ObjectDetection()
person_detector.setModelTypeAsYOLOv3()
person_detector.setModelPath('yolo.h5')
person_detector.loadModel()

hard_hat_detector = CustomObjectDetection()
hard_hat_detector.setModelTypeAsYOLOv3()
hard_hat_detector.setModelPath("hardhat\models\detection_model-ex-020--loss-
0008.462.h5")
hard_hat_detector.setJsonPath("hardhat\json\detection_config.json")
hard_hat_detector.loadModel()
```

Тут ми створили два детектори: один для виявлення всіх людей, а інший для виявлення людей у касках. Далі завантажимо випадкове зображення та пропустимо його через обидва наші детектори:

```
peopleImages = os.listdir("people")
randomFile = peopleImages[random.randint(0, len(peopleImages) - 1)]

peopleOnly = person_detector.CustomObjects(person=True)

_, person_detections =
detector.detectCustomObjectsFromImage(custom_objects=peopleOnly,
output_type="array", input_image="people/{0}".format(randomFile),
minimum_percentage_probability=30)

_, hard_hat_detections = detector.detectObjectsFromImage(output_type="array",
input_image="people/{0}".format(randomFile), minimum_percentage_probability=30)
```

Зрештою, ми порівняємо кількість виявлень на кожному зображенні, щоб побачити, чи помітили ми когось без каски (якщо кількість людей більше кількості людей у касках):

```
if len(person_detections) > len(hard_hatdetections):
  print("hardhat non-compliance detected! Police have been dispatched!")
  # or perhaps just e-mail the boss to inform them of the violation
```

Таким чином, програмний модуль для виявлення людей у касках було створено успішно.

Є ряд речей, які можна було б зробити, щоб покращити нашу програму для виявлення касок. Наступним кроком, ймовірно, буде виконання більшої кількості епох для навчання моделі. На даний момент модель досить добре виявляє людей у касках, але цифри рівня довіри здебільшого коливаються в межах 50%-80% (рис. 4.2).

```
person hardhat  :  81.50655627250671  :  [137, 74, 194, 119]
--------------------------------
person hardhat  :  64.44770097732544  :  [290, 89, 328, 123]
--------------------------------
person hardhat  :  70.66482901573181  :  [252, 90, 290, 130]
--------------------------------
person hardhat  :  54.37401533126831  :  [358, 93, 385, 131]
--------------------------------
person hardhat  :  59.80772376060486  :  [125, 100, 145, 136]
--------------------------------
person hardhat  :  45.671346783638  :  [222, 101, 249, 130]
--------------------------------
person hardhat  :  77.1479070186615  :  [314, 99, 361, 142]
--------------------------------
```

Рисунок 4.2 –Значення рівня довіри при виявленні людей у касках

Витрачаючи більше часу на навчання моделі, ми могли б збільшити ці цифри, зробивши нашу модель точнішою.

Якщо є графічний процесор NVIDIA, то можна значно пришвидшити навчання, встановивши збірку Tensorflow з підтримкою графічного процесора.

Якщо немає сумісного графічного процесора, то можна отримати доступ до нього недорого в хмарі.

Деякі інші опції, які надає ImageAI, також можуть покращити зручність використання моделі:

- Метод detectObjectsFromImage() може приймати параметр під назвою extract_detected_objects=True. Можна використовувати цей параметр, щоб витягти всіх людей із зображення, перш ніж перебирати їх і перевіряти, чи носять вони каску. Якщо ні, їхнє індивідуальне зображення можна надіслати менеджеру.
- ImageAI також може підтримувати виявлення відеокамер або прямих трансляцій за допомогою класу VideoObjectDetection. Цей клас працює подібно до класу ImageObjectDetection і може бути використаний для підтримки перевірки відео майже в реальному часі.

Таким чином, в результаті тестування було доведено повну працездатність програми і відповідність поставленому завданню.

4.2 Аналіз результатів роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі

Оцінка якості роботи програми відіграє велику роль під час експериментів і тестування нових програмних продуктів.

Для оцінки продуктивності різних моделей розпізнавання об’єктів на зображеннях (зокрема, людей у касці) застосовується така широко використовувана метрика як середня точність (mAP – mean average precision) та середня точність для класу (AP - average). AP полягає у обчисленні площі під кривою «Точність × Повнота» (precision x recall), тоді як mAP – це AP, усереднена за кількома класами. Точність та повнота обчислюються для кожного класу та визначаються наступним чином:

$$\text{Precision} = TP/(TP + FP)$$
$$\text{Recall} = TP/(TP + FN) \quad (4.1)$$

де TP визначається як кількість правильних виявлень із взаємодією об'єднання (IoU) ≥ 0,5. FP – кількість неправильних виявлень, тоді як FN – кількість невиявлених дійсних результатів.

У п.1.3 обгрунтовано, що як перший аналог було обрано «Детектор випадків невикористання каски на основі Faster R-CNN» [25], а як другий аналог було обрано «Автоматичний детектор носіння касок на основі SSD» [26]. У даній роботі розроблено програмний модуль розпізнавання людей у касці на основі згорткової нейромережі YOLOv3. Порівняння параметрів розробленого модуля та аналогів наведено у таблиці 4.1.

Таблиця 4.1 – Порівняння параметрів розробленого програмного модуля розпізнавання людей у касці та аналогів

№ п/п	Метрика	Розмір входу нейромережі	Обсяг моделі, Мбайт	Середня точність mAP, %
1	Аналог 1: «Детектор випадків невикористання каски на основі Faster R-CNN» [25],	300x500	37,2	80,75
2	Аналог 2: «Автоматичний детектор носіння касок на основі SSD» [26].	512x512	36,4	83,0
	Розроблений програмний модуль (на основі YOLOv3)	416x416	34,7	84,5

Із табл. 4.1 видно, що розроблений програмний модуль (на основі нейромережі YOLOv3) переважає аналог 1 (на основі нейромережі Faster R-CNN) за середньою точністю mAP на 3,75% (84,5 % проти 80,75%), а аналог 2 (на основі нейромережі SSD) на 1,5% (84,5 % проти (84,0 %). Крім цього, розроблений програмний модуль розпізнавання людей у касці займає на диску менший об'єм пам'яті (34,7 Мбайт), ніж обидва аналоги (відповідно 37,2 і 36.4 Мбайт).

Таким чином, можна зробити висновок, що розроблений програмний модуль на основі згорткової нейронної мережі YOLOv3 має порівняно з кращим аналогом збільшену на 1,5% середню точність розпізнавання. Тобто мета роботи досягнута – точність розпізнавання людей у касці підвищена.

4.3 Висновок до розділу 4

В результаті тестування програмного модуля розпізнавання людей у касці було доведено його повну працездатність та відповідність поставленому завданню. Модуль обводить рамкою кожну людину у касці та виводить координати рамки та відсоток рівня довіри. Для оцінки точності роботи модуля використовувалась така метрика як середня точність mAP. Розроблений програмний модуль на основі згорткової нейронної мережі YOLOv3 має порівняно з кращим аналогом збільшену на 1,5% середню точність розпізнавання. Тобто мета роботи досягнута – точність розпізнавання людей у касці підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу розпізнавання людей у касці на зображенні. Було описано детальну постановку задачі розпізнавання людей у касці. Також розглядаються різні методи розв'язання задачі розпізнавання об’єктів на зображенні і оцінюються їх переваги і недоліки. Серед таких методів як традиційні методи; двоступеневі методи на основі глибокого навчання; одноступеневі методи на основі глибокого навчання як найбільш перспективний було обрано одноступеневі методи на основі глибокого навчання. Вони будуються на таких нейронних мережах як YOLO, SSD, RetinaNet, CornerNet, CenterNet, DETR та інши. На основі недоліків відомих методів виявлення об’єктів на зображеннях було сформульовано мету роботи – підвищення точності розпізнавання людей у касці. Крім цього, було обгрунтовано для визначення переваг розробленого модуля порівнювати його з двома аналогами: перший на основі згорткової нейронної мережі Faster R-CNN, а другий аналог на основі згорткової нейронної мережі SSD.

Також було обґрунтувано вибір типу згорткової нейронної мережі для модуля розпізнавання людей у касці. Із таких типів згорткових нейронних мереж як R-CNN, Fast R-CNN, Faster R-CNN, SSD, YOLO для практичної реалізації програмного модуля було обрано нейромережу YOLOv3 як найбільш перспективну за сполученням параметрів точність-швидкодія. Також було проаналізовано архітектуру моделі YOLOv3 та принципи її функціонування.. крім цього, було розроблено алгоритм роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3.

У практичній частині було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек ImageAI та OpenCV для програмної реалізації модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3. Проведено опис процеесу отримання набору даних зображень на основі ImageAI для навчання нейронної мережі YOLOv3. Проаналізовано структуру набору даних, принципи формування анотацій для обмежувальних

прямокутників. Описано основні етапи програмної реалізація та функціонування модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3, що складається із завантаження та формування навчального та валідаційного наборів даних, завантаження фреймворку попередньо навченої нейронної мережі YOLOv3, її налаштування, навчання нейронної мережі YOLOv3 та використання навченої нейронної мережі YOLOv3 для розпізнавання людей у касці.

В результаті тестування програмного модуля розпізнавання людей у касці було доведено його повну працездатність та відповідність поставленому завданню. Модуль обводить рамкою кожну людину у касці та виводить координати рамки та відсоток рівня довіри. Для оцінки точності роботи модуля використовувалась така метрика як середня точність mAP. Розроблений програмний модуль на основі згорткової нейронної мережі YOLOv3 має порівняно з кращим аналогом збільшену на 1,5% середню точність розпізнавання. Тобто мета роботи досягнута – точність розпізнавання людей у касці підвищена.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 “Hypercolumns for object segmentation and finegrained localization,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 447–456.

2 P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in CVPR, vol. 1. IEEE, 2001, pp. I–I.

3 Орел А. О. Аналіз методу Віоли-Джонса для розпізнавання облич / А. О. Орел // Кібербезпека в сучасному світі : матеріали II Всеукраїнської науково-практичної конференції (м. Одеса, 20 листопада 2020 р.) / за ред. О. В. Дикого ; уклад.: Н. І. Логінова, В. Д. Бойко, М. О. Флюнт. – Одеса : Видавничий дім «Гельветика», 2020. - С. 195-198..

4 N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in CVPR, vol. 1. IEEE, 2005, pp. 886–893.

5 P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in CVPR. IEEE, 2008, pp. 1–8.

6 A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

7 R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in CVPR, 2014, pp. 580–587.

8 О. Д. Сизоненко, Л. М. Божуха Методи локалізації об'єктів на основі зображень із використанням комбінації алгоритмів та багатопоточної зв'язки Faster R-CNN / Актуальні проблеми автоматизації та інформаційних технологій, Том 27 (2023). DOI: http://dx.doi.org/10.15421/432316

9 J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, “Selective search for object recognition,” International journal of computer vision, vol. 104, no. 2, pp. 154–171, 2013.

10 K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in ECCV. Springer, 2014, pp. 346–361.

11 R. Girshick, “Fast R-CNN,” in ICCV, 2015, pp. 1440– 1448.

12 S. Ren, K. He, R. Girshick, and J. Sun, “Faster RCNN: Towards real-time object detection with region proposal networks,” in Advances in neural information processing systems, 2015, pp. 91–99.

13 Литвин В.В. Методи та засоби інженерії даних та знань. Львів: Магнолія, 2012. 241 с.

14 T.-Y. Lin, P. Dollar, R. B. Girshick, K. He, B. Hariharan, ´ and S. J. Belongie, “Feature pyramid networks for object detection.” in CVPR, vol. 1, no. 2, 2017, p. 4.

15 J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in CVPR, 2016, pp. 779–788.

16 W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single shot multibox detector,” in ECCV. Springer, 2016, pp. 21–37.

17 T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal loss for dense object detection,” IEEE transactions on pattern analysis and machine intelligence, 2018.

18 H. Law and J. Deng, “Cornernet: Detecting objects as paired keypoints,” in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 734–750.

19 X. Zhou, D. Wang, and P. Krahenb ¨ uhl, “Objects as ¨ points,” arXiv preprint arXiv:1904.07850, 2019.

20 N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” in European Conference on Computer Vision. Springer, 2020, pp. 213–229.

21 X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, “Deformable detr: Deformable transformers for end-toend object detection,” arXiv preprint arXiv:2010.04159, 2020.

22 H. Zhang, X. Yan, H. Li, R. Jin, H. Fu, Real-time alarming, monitoring, and locating for non-hard-hat use in construction, J. Constr. Eng. Manag. 145 (2019) 4019006, https://doi.org/10.1061/(asce)co.1943-7862.0001629.

23 О.К.Колесницкий, Самра Муавия Хассан Хамо Метод распознавания многомерных временных рядов при помощи импульсных нейронных сетей// Інформаційні технології та комп‘ютерна інженерія, 2006, №2(6), С. 86-93.

24 B.E. Mneymneh, M. Abbas, H. Khoury, Vision-based framework for intelligent monitoring of hardhat wearing on construction sites, J. Comput. Civ. Eng. 33 (2018) 04018066, , https://doi.org/10.1061/(asce)cp.1943-5487.0000813.

25 Q. Fang, H. Li, X. Luo, L. Ding, H. Luo, T.M. Rose, Detecting non-hardhat-use by a deep learning method from far-field surveillance videos, Autom. Constr. 85 (2018) 1–9, https://doi.org/10.1016/j.autcon.2017.09.018.

26 Jixiu Wu, Nian Cai, Wenjie Chen, Huiheng Wang, Guotian Wang Automatic detection of hardhats worn by construction personnel: A deep learning approach and benchmark dataset, Automation in Construction 106 (2019), https://doi.org/10.1016/j.autcon.2019.102894

27 Model Zoo. Detection [Електронний ресурс] – Режим доступу: https://cv.gluon.ai/model_zoo/detection.html

28 Welcome to Python [Електронний ресурс] – Режим доступу: https://www.python.org

29 OpenCV is the world's biggest computer vision library. [Електронний ресурс] – Режим доступу: https://opencv.org/

30 ImageAI [Електронний ресурс] – Режим доступу: https://github.com/OlafenwaMoses/ImageAI/

31 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп'ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль розпізнавання людей у касці на основі згорткової нейронної мережі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 21,62 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колесницький О.К.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ящук Т.Є.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Hard Hat Control\n",
    "\n",
    "### Initialize the Libraries"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "Using TensorFlow backend.\n"
     ]
    }
   ],
   "source": [
    "import cv2 as cv\n",
    "from imageai.Detection import ObjectDetection\n",
    "\n",
    "import numpy as np\n",
    "import requests as req\n",
    "import os as os"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Show Window Function"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [],
   "source": [
    "def showImage(img):\n",
    "    window_name = 'image'\n",
    "    cv.imshow(window_name, img)\n",
    "    cv.waitKey(0)\n",
    "    cv.destroyAllWindows()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Download Data\n",
    "\n",
    "### Download Images of Hard Hats"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "hardhatLoc = 'http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n03492922'\n",
    "\n",
```

```
  "hardhatImages = req.get(hardhatLoc).text\n",
  "noOfImages = 0\n",
  "\n",
  "if not os.path.exists('hardhat'):\n",
  "    os.makedirs('hardhat')\n",
  "\n",
  "for i in hardhatImages.split('\\n'):\n",
  "    try:\n",
  "        r = req.get(i, timeout=0.5)\n",
  "        file = i.split(\"/\")[-1].split('\\r')[0]\n",
  "        if 'image/jpeg' in r.headers['Content-Type']:\n",
  "            if len(r.content) > 8192:\n",
  "                with open('hardhat\\\\' + file, 'wb') as outfile:\n",
  "                    outfile.write(r.content)\n",
  "                    noOfImages += 1\n",
  "                    print('Success: ' + file)\n",
  "            else:\n",
  "                print('Failed: ' + file + ' -- Image too small')\n",
  "        else:\n",
  "            print('Failed: ' + file + ' -- Not an image')\n",
  "\n",
  "    except Exception as e:\n",
  "        print('Failed: ' + file + ' -- Error')\n",
  "        \n",
  "print('*********** Download Finished **************')"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Download Images of People"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "peopleLoc = 'http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n07942152'\n",
  "\n",
  "peopleImages = req.get(peopleLoc).text\n",
  "noOfImages = 0\n",
  "\n",
  "if not os.path.exists('people'):\n",
  "    os.makedirs('people')\n",
  "\n",
  "for i in peopleImages.split('\\n'):\n",
  "    try:\n",
  "        r = req.get(i, timeout=0.5)\n",
  "        file = i.split(\"/\")[-1].split('\\r')[0]\n",
  "        if 'image/jpeg' in r.headers['Content-Type']:\n",
  "            if len(r.content) > 8192:\n",
  "                with open('people\\\\' + file, 'wb') as outfile:\n",
  "                    outfile.write(r.content)\n",
  "                    noOfImages += 1\n",
  "                    print('Success: ' + file)\n",
  "            else:\n",
  "                print('Failed: ' + file + ' -- Image too small')\n",
  "        else:\n",
  "            print('Failed: ' + file + ' -- Not an image')\n",
  "\n",
  "    except Exception as e:\n",
  "        print('Failed: ' + file + ' -- Error')\n",
  "        \n",
  "print('*********** Download Finished **************')"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Download Pre-Train Models"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
```

```
   "metadata": {},
   "outputs": [],
   "source": [
    "modelRetinaNet =
'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_v2.0.1.h5'\n",
    "modelYOLOv3 = 'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo.h5'\n",
    "modelTinyYOLOv3 = 'https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo-
tiny.h5'\n",
    "\n",
    "if not os.path.exists('yolo.h5'):\n",
    "    r = req.get(modelYOLOv3, timeout=0.5)\n",
    "    with open('yolo.h5', 'wb') as outfile:\n",
    "        outfile.write(r.content)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Train the Model\n",
    "\n",
    "### Define the Detector"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "WARNING:tensorflow:From E:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\tensorflow_core\\python\\ops\\resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is
deprecated and will be removed in a future version.\n",
      "Instructions for updating:\n",
      "If using Keras pass *_constraint arguments to layers.\n",
      "WARNING:tensorflow:From E:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\tensorflow_core\\python\\ops\\array_ops.py:1475: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.\n",
      "Instructions for updating:\n",
      "Use tf.where in 2.0, which has the same broadcast rule as np.where\n"
     ]
    }
   ],
   "source": [
    "detector = ObjectDetection()\n",
    "detector.setModelTypeAsYOLOv3()\n",
    "detector.setModelPath('yolo.h5')\n",
    "detector.loadModel()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Clean the Data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [
    {
     "ename": "ValueError",
     "evalue": "Ensure you specified correct input image, input type, output type and/or output
image path ",
     "output_type": "error",
     "traceback": [
      "\u001b[1;31m---------------------------------------------------------------------------
\u001b[0m",
      "\u001b[1;31mPermissionError\u001b[0m                           Traceback (most recent call
last)",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\imageai\\Detection\\__init__.py\u001b[0m in
\u001b[0;36mdetectCustomObjectsFromImage\u001b[1;34m(self, custom_objects, input_image,
```

```
output_image_path, input_type, output_type, extract_detected_objects,
minimum_percentage_probability, display_percentage_probability, display_object_name,
thread_safe)\u001b[0m\n\u001b[0;32m    795\u001b[0m                     \u001b[1;32mif\u001b[0m
\u001b[1;33m(\u001b[0m\u001b[0minput_type\u001b[0m \u001b[1;33m==\u001b[0m
\u001b[1;34m\"file\"\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b
[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m--> 796\u001b[1;33m
\u001b[0mimage\u001b[0m \u001b[1;33m=\u001b[0m
\u001b[0mImage\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mopen\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0min
put_image\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[
0m\u001b[0;32m    797\u001b[0m                         \u001b[0minput_image\u001b[0m
\u001b[1;33m=\u001b[0m
\u001b[0mread_image_bgr\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0minput_image\u001b[0m\u001b[1;33m)\u00
1b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\PIL\\Image.py\u001b[0m in \u001b[0;36mopen\u001b[1;34m(fp, mode)\u001b[0m\n\u001b[0;32m
2842\u001b[0m     \u001b[1;32mif\u001b[0m
\u001b[0mfilename\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\
n\u001b[1;32m-> 2843\u001b[1;33m         \u001b[0mfp\u001b[0m \u001b[1;33m=\u001b[0m
\u001b[0mbuiltins\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mopen\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0
mfilename\u001b[0m\u001b[1;33m,\u001b[0m
\u001b[1;34m\"rb\"\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m
\n\u001b[0m\u001b[0;32m   2844\u001b[0m         \u001b[0mexclusive_fp\u001b[0m
\u001b[1;33m=\u001b[0m
\u001b[1;32mTrue\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;31mPermissionError\u001b[0m: [Errno 13] Permission denied: 'hardhat/test'",
      "\nDuring handling of the above exception, another exception occurred:\n",
      "\u001b[1;31mValueError\u001b[0m                                Traceback (most recent call
last)",
      "\u001b[1;32m<ipython-input-4-6d8480a8c66a>\u001b[0m in
\u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m      6\u001b[0m     detectedImage,
detections = detector.detectCustomObjectsFromImage(custom_objects=peopleOnly,
output_type=\"array\",\n\u001b[0;32m      7\u001b[0m
\u001b[0minput_image\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0m\u001b[0mimageFile\u001b[0m\u001b[1;33m,\u001b[0m
\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m----> 8\u001b[1;33m
minimum_percentage_probability=30)\n\u001b[0m\u001b[0;32m      9\u001b[0m
\u001b[1;32mif\u001b[0m
\u001b[0mlen\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mdetections\u001b[0m\u001b[1;33m)\u001b[0m
\u001b[1;33m>\u001b[0m
\u001b[1;36m0\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u0
01b[0;32m     10\u001b[0m
\u001b[0mos\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mrename\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mima
geFile\u001b[0m\u001b[1;33m,\u001b[0m \u001b[1;34m\"hardhat-
clean/{0}\"\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mformat\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mi\u
001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[
0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\imageai\\Detection\\__init__.py\u001b[0m in
\u001b[0;36mdetectCustomObjectsFromImage\u001b[1;34m(self, custom_objects, input_image,
output_image_path, input_type, output_type, extract_detected_objects,
minimum_percentage_probability, display_percentage_probability, display_object_name,
thread_safe)\u001b[0m\n\u001b[0;32m    917\u001b[0m
\u001b[1;32mexcept\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m
\n\u001b[0;32m    918\u001b[0m                 raise ValueError(\n\u001b[1;32m--> 919\u001b[1;33m
\"Ensure you specified correct input image, input type, output type and/or output image path
\")\n\u001b[0m\u001b[0;32m    920\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m
921\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;31mValueError\u001b[0m: Ensure you specified correct input image, input type, output
type and/or output image path "
     ]
    }
   ],
   "source": [
    "hardhatImages = os.listdir(\"hardhat\")\n",
    "peopleOnly = detector.CustomObjects(person=True)\n",
    "\n",
    "for i in hardhatImages:\n",
    "    imageFile = \"hardhat/{0}\".format(i)\n",
    "    detectedImage, detections =
detector.detectCustomObjectsFromImage(custom_objects=peopleOnly, output_type=\"array\",\n",
    "                                                                        input_image=imageFile,
\n",
    "
minimum_percentage_probability=30)\n",
    "    if len(detections) < 0:\n",
    "        os.remove(imageFile)"
   ]
  },
  {
   "cell_type": "markdown",
```

```
   "metadata": {},
   "source": [
    "### Split the Data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "metadata": {},
   "outputs": [],
   "source": [
    "if not os.path.exists('hardhat/train/images'):\n",
    "    os.makedirs('hardhat/train/images')\n",
    "if not os.path.exists('hardhat/validation/images'):\n",
    "    os.makedirs('hardhat/validation/images')\n",
    "\n",
    "hardhatImages = os.listdir(\"hardhat\")\n",
    "hardhatTrainNums = round(len(hardhatImages) * 0.90)\n",
    "\n",
    "for i in range(0, hardhatTrainNums):\n",
    "    file = \"hardhat/\" + hardhatImages[i]\n",
    "    if os.path.isfile(file):\n",
    "        os.rename(file, \"hardhat/train/images/\" + hardhatImages[i])\n",
    "    \n",
    "hardhatImages = os.listdir(\"hardhat\")\n",
    "\n",
    "for i in hardhatImages:\n",
    "    file = \"hardhat/\" + i\n",
    "    if os.path.isfile(file):\n",
    "        os.rename(file, \"hardhat/validation/images/\" + i)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Train the Model"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 21,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Generating anchor boxes for training images and annotation...\n",
      "Average IOU for 9 anchors: 0.75\n",
      "Anchor Boxes generated.\n",
      "Detection configuration saved in  hardhat\\json\\detection_config.json\n",
      "Training on: \t['person hardhat']\n",
      "Training with Batch Size:  4\n",
      "Number of Experiments:  200\n",
      "Training with transfer learning from pretrained Model\n"
     ]
    },
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "E:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\callbacks\\callbacks.py:998: UserWarning: `epsilon` argument is deprecated and will
be removed, use `min_delta` instead.\n",
      "  warnings.warn('`epsilon` argument is deprecated and '\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Epoch 1/200\n",
      "944/944 [==============================] - 856s 907ms/step - loss: 36.9858 -
yolo_layer_16_loss: 7.0103 - yolo_layer_17_loss: 10.9458 - yolo_layer_18_loss: 19.0297 - val_loss:
22.8127 - val_yolo_layer_16_loss: 5.6805 - val_yolo_layer_17_loss: 6.6321 - val_yolo_layer_18_loss:
12.3109\n",
```

```
      "WARNING:tensorflow:From E:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\callbacks\\tensorboard_v1.py:343: The name tf.Summary is deprecated. Please use
tf.compat.v1.Summary instead.\n",
      "\n",
      "Epoch 2/200\n",
      "944/944 [==============================] - 774s 820ms/step - loss: 20.2127 -
yolo_layer_16_loss: 4.6473 - yolo_layer_17_loss: 5.6500 - yolo_layer_18_loss: 9.9154 - val_loss:
12.3263 - val_yolo_layer_16_loss: 5.5798 - val_yolo_layer_17_loss: 6.8109 - val_yolo_layer_18_loss:
11.0726\n",
      "Epoch 3/200\n",
      "944/944 [==============================] - 776s 822ms/step - loss: 18.3042 -
yolo_layer_16_loss: 4.7082 - yolo_layer_17_loss: 4.9417 - yolo_layer_18_loss: 8.6543 - val_loss:
30.1679 - val_yolo_layer_16_loss: 3.5330 - val_yolo_layer_17_loss: 5.7483 - val_yolo_layer_18_loss:
10.3959\n",
      "Epoch 4/200\n",
      "944/944 [==============================] - 772s 817ms/step - loss: 15.8019 -
yolo_layer_16_loss: 3.5019 - yolo_layer_17_loss: 4.2786 - yolo_layer_18_loss: 8.0214 - val_loss:
25.0318 - val_yolo_layer_16_loss: 3.3268 - val_yolo_layer_17_loss: 4.8476 - val_yolo_layer_18_loss:
8.9261\n",
      "Epoch 5/200\n",
      "944/944 [==============================] - 775s 821ms/step - loss: 14.2383 -
yolo_layer_16_loss: 2.8789 - yolo_layer_17_loss: 3.9230 - yolo_layer_18_loss: 7.4365 - val_loss:
20.3871 - val_yolo_layer_16_loss: 2.7601 - val_yolo_layer_17_loss: 4.3188 - val_yolo_layer_18_loss:
8.2737\n",
      "Epoch 6/200\n",
      "944/944 [==============================] - 774s 820ms/step - loss: 13.3032 -
yolo_layer_16_loss: 2.6941 - yolo_layer_17_loss: 3.6415 - yolo_layer_18_loss: 6.9677 - val_loss:
12.4483 - val_yolo_layer_16_loss: 2.4334 - val_yolo_layer_17_loss: 4.4343 - val_yolo_layer_18_loss:
7.3737\n",
      "Epoch 7/200\n",
      "944/944 [==============================] - 773s 819ms/step - loss: 12.7051 -
yolo_layer_16_loss: 2.4235 - yolo_layer_17_loss: 3.3868 - yolo_layer_18_loss: 6.8948 - val_loss:
13.0070 - val_yolo_layer_16_loss: 2.1589 - val_yolo_layer_17_loss: 4.2864 - val_yolo_layer_18_loss:
7.7522\n",
      "Epoch 8/200\n",
      "944/944 [==============================] - 774s 820ms/step - loss: 12.0266 -
yolo_layer_16_loss: 2.3517 - yolo_layer_17_loss: 3.1444 - yolo_layer_18_loss: 6.5304 - val_loss:
10.2641 - val_yolo_layer_16_loss: 2.4435 - val_yolo_layer_17_loss: 3.8413 - val_yolo_layer_18_loss:
7.1583\n",
      "Epoch 9/200\n",
      "944/944 [==============================] - 774s 819ms/step - loss: 11.5796 -
yolo_layer_16_loss: 2.0791 - yolo_layer_17_loss: 3.0983 - yolo_layer_18_loss: 6.4023 - val_loss:
16.9550 - val_yolo_layer_16_loss: 1.9408 - val_yolo_layer_17_loss: 3.7394 - val_yolo_layer_18_loss:
8.5159\n",
      "Epoch 10/200\n",
      "944/944 [==============================] - 773s 819ms/step - loss: 11.0530 -
yolo_layer_16_loss: 2.0455 - yolo_layer_17_loss: 2.9342 - yolo_layer_18_loss: 6.0733 - val_loss:
11.9410 - val_yolo_layer_16_loss: 2.4114 - val_yolo_layer_17_loss: 3.6174 - val_yolo_layer_18_loss:
7.4209\n",
      "Epoch 11/200\n",
      "944/944 [==============================] - 776s 822ms/step - loss: 10.6817 -
yolo_layer_16_loss: 1.9981 - yolo_layer_17_loss: 2.8343 - yolo_layer_18_loss: 5.8493 - val_loss:
7.0664 - val_yolo_layer_16_loss: 1.9985 - val_yolo_layer_17_loss: 3.6278 - val_yolo_layer_18_loss:
6.6847\n",
      "Epoch 12/200\n",
      "944/944 [==============================] - 774s 820ms/step - loss: 10.4226 -
yolo_layer_16_loss: 1.9022 - yolo_layer_17_loss: 2.7187 - yolo_layer_18_loss: 5.8017 - val_loss:
13.9414 - val_yolo_layer_16_loss: 2.1432 - val_yolo_layer_17_loss: 3.8370 - val_yolo_layer_18_loss:
6.8831\n",
      "Epoch 13/200\n",
      "944/944 [==============================] - 775s 821ms/step - loss: 10.0310 -
yolo_layer_16_loss: 1.8245 - yolo_layer_17_loss: 2.6859 - yolo_layer_18_loss: 5.5206 - val_loss:
11.2866 - val_yolo_layer_16_loss: 1.8409 - val_yolo_layer_17_loss: 3.3783 - val_yolo_layer_18_loss:
6.5706\n",
      "Epoch 14/200\n",
      "944/944 [==============================] - 775s 821ms/step - loss: 9.7983 -
yolo_layer_16_loss: 1.7165 - yolo_layer_17_loss: 2.5344 - yolo_layer_18_loss: 5.5474 - val_loss:
15.4132 - val_yolo_layer_16_loss: 1.7203 - val_yolo_layer_17_loss: 3.8468 - val_yolo_layer_18_loss:
6.7124\n",
      "Epoch 15/200\n",
      "944/944 [==============================] - 774s 820ms/step - loss: 9.6199 -
yolo_layer_16_loss: 1.6960 - yolo_layer_17_loss: 2.5321 - yolo_layer_18_loss: 5.3918 - val_loss:
11.7558 - val_yolo_layer_16_loss: 1.6791 - val_yolo_layer_17_loss: 3.3249 - val_yolo_layer_18_loss:
6.7528\n",
      "Epoch 16/200\n",
      "944/944 [==============================] - 778s 824ms/step - loss: 9.2273 -
yolo_layer_16_loss: 1.6217 - yolo_layer_17_loss: 2.3432 - yolo_layer_18_loss: 5.2624 - val_loss:
17.8903 - val_yolo_layer_16_loss: 1.8734 - val_yolo_layer_17_loss: 3.7065 - val_yolo_layer_18_loss:
6.7896\n",
```

```
      "Epoch 17/200\n",
      "944/944 [==============================] - 778s 825ms/step - loss: 9.0276 -
yolo_layer_16_loss: 1.5986 - yolo_layer_17_loss: 2.3144 - yolo_layer_18_loss: 5.1147 - val_loss:
8.6253 - val_yolo_layer_16_loss: 1.6680 - val_yolo_layer_17_loss: 3.7442 - val_yolo_layer_18_loss:
5.6655\n",
      "Epoch 18/200\n",
      "944/944 [==============================] - 780s 827ms/step - loss: 8.8141 -
yolo_layer_16_loss: 1.5611 - yolo_layer_17_loss: 2.3312 - yolo_layer_18_loss: 4.9219 - val_loss:
6.8751 - val_yolo_layer_16_loss: 1.4509 - val_yolo_layer_17_loss: 3.2254 - val_yolo_layer_18_loss:
6.5645\n",
      "Epoch 19/200\n",
      "944/944 [==============================] - 779s 825ms/step - loss: 8.6509 -
yolo_layer_16_loss: 1.5625 - yolo_layer_17_loss: 2.2129 - yolo_layer_18_loss: 4.8754 - val_loss:
4.2779 - val_yolo_layer_16_loss: 1.4135 - val_yolo_layer_17_loss: 3.0607 - val_yolo_layer_18_loss:
6.1202\n",
      "Epoch 20/200\n",
      "944/944 [==============================] - 812s 860ms/step - loss: 8.4625 -
yolo_layer_16_loss: 1.4308 - yolo_layer_17_loss: 2.1983 - yolo_layer_18_loss: 4.8334 - val_loss:
19.4190 - val_yolo_layer_16_loss: 1.4224 - val_yolo_layer_17_loss: 3.5843 - val_yolo_layer_18_loss:
7.1598\n",
      "Epoch 21/200\n",
      "944/944 [==============================] - 824s 872ms/step - loss: 8.3525 -
yolo_layer_16_loss: 1.4480 - yolo_layer_17_loss: 2.2054 - yolo_layer_18_loss: 4.6992 - val_loss:
14.1312 - val_yolo_layer_16_loss: 1.6272 - val_yolo_layer_17_loss: 2.7905 - val_yolo_layer_18_loss:
5.9421\n",
      "Epoch 22/200\n",
      "682/944 [====================>.........] - ETA: 3:37 - loss: 8.1493 - yolo_layer_16_loss:
1.3431 - yolo_layer_17_loss: 2.0674 - yolo_layer_18_loss: 4.7388"
     ]
    },
    {
     "ename": "KeyboardInterrupt",
     "evalue": "",
     "output_type": "error",
     "traceback": [
      "\u001b[1;31m---------------------------------------------------------------------------
\u001b[0m",
      "\u001b[1;31mKeyboardInterrupt\u001b[0m                         Traceback (most recent call
last)",
      "\u001b[1;32m<ipython-input-21-cdca61b8cf02>\u001b[0m in
\u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m      8\u001b[0m
train_from_pretrained_model=\"yolo.h5\")\n\u001b[0;32m      9\u001b[0m
\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m---> 10\u001b[1;33m
\u001b[0mtrainer\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mtrainModel\u001b[0m\u001b[1;33m(\u001b[0m\u0
01b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0m",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\imageai\\Detection\\Custom\\__init__.py\u001b[0m in
\u001b[0;36mtrainModel\u001b[1;34m(self)\u001b[0m\n\u001b[0;32m    289\u001b[0m
\u001b[0mcallbacks\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mcallbacks\u001b[0m\u001b[1;33m,\u001b[0m\u
001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m    290\u001b[0m
\u001b[0mworkers\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;36m4\u001b[0m\u001b[1;33m,\u001b[0m\u001b[1;
33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m--> 291\u001b[1;33m
\u001b[0mmax_queue_size\u001b[0m\u001b[1;33m=\u001b[0m\u001b[1;36m8\u001b[0m\u001b[1;33m\u001b[0m\u0
01b[1;33m\u001b[0m\u001b[0m\n\u001b[0m\n\u001b[0m\u001b[0;32m    292\u001b[0m         )\n\u001b[0;32m
293\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\legacy\\interfaces.py\u001b[0m in \u001b[0;36mwrapper\u001b[1;34m(*args,
**kwargs)\u001b[0m\n\n\u001b[0;32m     89\u001b[0m                 warnings.warn('Update your `' +
object_name + '` call to the ' +\n\u001b[0;32m     90\u001b[0m                               'Keras
2 API: ' + signature, stacklevel=2)\n\u001b[1;32m---> 91\u001b[1;33m
\u001b[1;32mreturn\u001b[0m
\u001b[0mfunc\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m*\u001b[0m\u001b[0margs\u001b[0m\u001b[1;33m
,\u001b[0m
\u001b[1;33m**\u001b[0m\u001b[0mkwargs\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;3
3m\u001b[0m\u001b[0m\n\n\u001b[0m\u001b[0;32m     92\u001b[0m
\u001b[0mwrapper\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0m_original_function\u001b[0m
\u001b[1;33m=\u001b[0m
\u001b[0mfunc\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\n\u001b[0;32m
93\u001b[0m         \u001b[1;32mreturn\u001b[0m
\u001b[0mwrapper\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\engine\\training.py\u001b[0m in \u001b[0;36mfit_generator\u001b[1;34m(self,
generator, steps_per_epoch, epochs, verbose, callbacks, validation_data, validation_steps,
validation_freq, class_weight, max_queue_size, workers, use_multiprocessing, shuffle,
initial_epoch)\u001b[0m\n\n\u001b[0;32m   1730\u001b[0m
\u001b[0muse_multiprocessing\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0muse_multiprocessing\u001b[0m\u00
1b[1;33m,\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m   1731\u001b[0m
\u001b[0mshuffle\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mshuffle\u001b[0m\u001b[1;33m,\u001b[0m\u001b
[1;33m\u001b[0m\u001b[1;33m\u001b[0m\n\u001b[1;32m-> 1732\u001b[1;33m
```

```
initial_epoch=initial_epoch)\n\u001b[0m\u001b[0;32m   1733\u001b[0m
\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m   1734\u001b[0m
\u001b[1;33m@\u001b[0m\u001b[0minterfaces\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mlegacy_generator_me
thods_support\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\engine\\training_generator.py\u001b[0m in
\u001b[0;36mfit_generator\u001b[1;34m(model, generator, steps_per_epoch, epochs, verbose, callbacks,
validation_data, validation_steps, validation_freq, class_weight, max_queue_size, workers,
use_multiprocessing, shuffle, initial_epoch)\u001b[0m\n\u001b[0;32m    218\u001b[0m
\u001b[0msample_weight\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0msample_weight\u001b[0m\u001b[1;33m,\u0
01b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m    219\u001b[0m
\u001b[0mclass_weight\u001b[0m\u001b[1;33m=\u001b[0m\u001b[0mclass_weight\u001b[0m\u001b[1;33m,\u001
b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m--> 220\u001b[1;33m
reset_metrics=False)\n\u001b[0m\u001b[0;32m    221\u001b[0m
\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m    222\u001b[0m                \u001b[0mouts\u001b[0m
\u001b[1;33m=\u001b[0m
\u001b[0mto_list\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mouts\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;
33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\keras\\engine\\training.py\u001b[0m in \u001b[0;36mtrain_on_batch\u001b[1;34m(self, x, y,
sample_weight, class_weight, reset_metrics)\u001b[0m\n\u001b[0;32m   1512\u001b[0m
\u001b[0mins\u001b[0m \u001b[1;33m=\u001b[0m \u001b[0mx\u001b[0m \u001b[1;33m+\u001b[0m
\u001b[0my\u001b[0m \u001b[1;33m+\u001b[0m
\u001b[0msample_weights\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m
1513\u001b[0m
\u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0m_make_train_function\u001b[0m\u001b[1;33m(\u001
b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[1;32m->
1514\u001b[1;33m         \u001b[0moutputs\u001b[0m \u001b[1;33m=\u001b[0m
\u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mtrain_function\u001b[0m\u001b[1;33m(\u001b[0m\u
001b[0mins\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b
[0m\u001b[0;32m   1515\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m   1516\u001b[0m
\u001b[1;32mif\u001b[0m
\u001b[0mreset_metrics\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001
b[0m\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\tensorflow_core\\python\\keras\\backend.py\u001b[0m in
\u001b[0;36m__call__\u001b[1;34m(self, inputs)\u001b[0m\n\u001b[0;32m   3474\u001b[0m
\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m   3475\u001b[0m     fetched =
self._callable_fn(*array_vals,\n\u001b[1;32m-> 3476\u001b[1;33m
run_metadata=self.run_metadata)\n\u001b[0m\u001b[0;32m   3477\u001b[0m
\u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0m_call_fetch_callbacks\u001b[0m\u001b[1;33m(\u00
1b[0m\u001b[0mfetched\u001b[0m\u001b[1;33m[\u001b[0m\u001b[1;33m-
\u001b[0m\u001b[0mlen\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u00
1b[0m_fetches\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m]\u001b[0m\u001b[1;33m
)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u001b[0;32m   3478\u001b[0m
output_structure = nest.pack_sequence_as(\n",
      "\u001b[1;32mE:\\Development\\Anaconda\\envs\\ImageAI\\lib\\site-
packages\\tensorflow_core\\python\\client\\session.py\u001b[0m in
\u001b[0;36m__call__\u001b[1;34m(self, *args, **kwargs)\u001b[0m\n\u001b[0;32m   1470\u001b[0m
ret = tf_session.TF_SessionRunCallable(self._session._session,\n\u001b[0;32m   1471\u001b[0m
\u001b[0mself\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0m_handle\u001b[0m\u001b[1;33m,\u001b[0m
\u001b[0margs\u001b[0m\u001b[1;33m,\u001b[0m\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b[0m\n\u0
01b[1;32m-> 1472\u001b[1;33m
run_metadata_ptr)\n\u001b[0m\u001b[0;32m   1473\u001b[0m         \u001b[1;32mif\u001b[0m
\u001b[0mrun_metadata\u001b[0m\u001b[1;33m:\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b
[0m\n\u001b[0;32m   1474\u001b[0m           \u001b[0mproto_data\u001b[0m \u001b[1;33m=\u001b[0m
\u001b[0mtf_session\u001b[0m\u001b[1;33m.\u001b[0m\u001b[0mTF_GetBuffer\u001b[0m\u001b[1;33m(\u001b[
0m\u001b[0mrun_metadata_ptr\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m
\u001b[0m\n",
      "\u001b[1;31mKeyboardInterrupt\u001b[0m: "
     ]
    }
   ],
   "source": [
    "from imageai.Detection.Custom import DetectionModelTrainer\n",
    "\n",
    "trainer = DetectionModelTrainer()\n",
    "trainer.setModelTypeAsYOLOv3()\n",
    "trainer.setDataDirectory(data_directory=\"hardhat\")\n",
    "\n",
    "trainer.setTrainConfig(object_names_array=[\"person hardhat\"], batch_size=4,
num_experiments=200, \n",
    "                       train_from_pretrained_model=\"yolo.h5\")\n",
    "\n",
    "trainer.trainModel()"
   ]
  },
  {
   "cell_type": "markdown",
```

```
   "metadata": {},
   "source": [
    "### Evaluate the Model"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 34,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Starting Model evaluation....\n",
      "Model File:  hardhat\\models\\detection_model-ex-005--loss-0014.238.h5 \n",
      "\n",
      "Using IoU :  0.5\n",
      "Using Object Threshold :  0.3\n",
      "Using Non-Maximum Suppression :  0.5\n",
      "person hardhat: 0.7165\n",
      "mAP: 0.7165\n",
      "==============================\n",
      "Starting Model evaluation....\n",
      "Model File:  hardhat\\models\\detection_model-ex-010--loss-0011.053.h5 \n",
      "\n",
      "Using IoU :  0.5\n",
      "Using Object Threshold :  0.3\n",
      "Using Non-Maximum Suppression :  0.5\n",
      "person hardhat: 0.7865\n",
      "mAP: 0.7865\n",
      "==============================\n",
      "Starting Model evaluation....\n",
      "Model File:  hardhat\\models\\detection_model-ex-015--loss-0009.620.h5 \n",
      "\n",
      "Using IoU :  0.5\n",
      "Using Object Threshold :  0.3\n",
      "Using Non-Maximum Suppression :  0.5\n",
      "person hardhat: 0.8355\n",
      "mAP: 0.8355\n",
      "==============================\n",
      "Starting Model evaluation....\n",
      "Model File:  hardhat\\models\\detection_model-ex-020--loss-0008.462.h5 \n",
      "\n",
      "Using IoU :  0.5\n",
      "Using Object Threshold :  0.3\n",
      "Using Non-Maximum Suppression :  0.5\n",
      "person hardhat: 0.8446\n",
      "mAP: 0.8446\n",
      "==============================\n",
      "------------------------------------------------------------\n",
      "Iteration 05: 0.7164811838244572 Iteration 10: 0.786450228152799 Iteration 15:
0.8354553845774733 Iteration 20: 0.8446486610895216\n",
      "------------------------------------------------------------\n"
     ]
    }
   ],
   "source": [
    "model05 = trainer.evaluateModel(model_path=\"hardhat\\\\models\\\\detection_model-ex-005--loss-
0014.238.h5\", \n",
    "                      json_path=\"hardhat\\\\json\\\\detection_config.json\", iou_threshold=0.5,
\n",
    "                      object_threshold=0.3, nms_threshold=0.5)\n",
    "model10 = trainer.evaluateModel(model_path=\"hardhat\\\\models\\\\detection_model-ex-010--loss-
0011.053.h5\", \n",
    "                      json_path=\"hardhat\\\\json\\\\detection_config.json\", iou_threshold=0.5,
\n",
    "                      object_threshold=0.3, nms_threshold=0.5)\n",
    "model15 = trainer.evaluateModel(model_path=\"hardhat\\\\models\\\\detection_model-ex-015--loss-
0009.620.h5\", \n",
    "                      json_path=\"hardhat\\\\json\\\\detection_config.json\", iou_threshold=0.5,
\n",
    "                      object_threshold=0.3, nms_threshold=0.5)\n",
    "model20 = trainer.evaluateModel(model_path=\"hardhat\\\\models\\\\detection_model-ex-020--loss-
0008.462.h5\", \n",
    "                      json_path=\"hardhat\\\\json\\\\detection_config.json\", iou_threshold=0.5,
\n",
    "                      object_threshold=0.3, nms_threshold=0.5)\n",
    "\n",
```

```
    "print('-----------------------------------------------------------')\n",
    "print('Iteration 05:', model05[0]['average_precision']['person hardhat'],\n",
    "      'Iteration 10:', model10[0]['average_precision']['person hardhat'],\n",
    "      'Iteration 15:', model15[0]['average_precision']['person hardhat'],\n",
    "      'Iteration 20:', model20[0]['average_precision']['person hardhat'])\n",
    "print('-----------------------------------------------------------')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Random Hard Hat Test"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 36,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "person hardhat  :  48.9349365234375  :  [198, 16, 253, 127]\n",
      "--------------------------------\n",
      "person hardhat  :  92.02308058738708  :  [84, 71, 127, 114]\n",
      "--------------------------------\n"
     ]
    }
   ],
   "source": [
    "from imageai.Detection.Custom import CustomObjectDetection\n",
    "\n",
    "detector = CustomObjectDetection()\n",
    "detector.setModelTypeAsYOLOv3()\n",
    "detector.setModelPath(\"hardhat\\\\models\\\\detection_model-ex-020--loss-0008.462.h5\")\n",
    "detector.setJsonPath(\"hardhat\\\\json\\\\detection_config.json\")\n",
    "detector.loadModel()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 40,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "person hardhat  :  81.50655627250671  :  [137, 74, 194, 119]\n",
      "--------------------------------\n",
      "person hardhat  :  64.44770097732544  :  [290, 89, 328, 123]\n",
      "--------------------------------\n",
      "person hardhat  :  70.66482901573181  :  [252, 90, 290, 130]\n",
      "--------------------------------\n",
      "person hardhat  :  54.37401533126831  :  [358, 93, 385, 131]\n",
      "--------------------------------\n",
      "person hardhat  :  59.80772376060486  :  [125, 100, 145, 136]\n",
      "--------------------------------\n",
      "person hardhat  :  45.671346783638  :  [222, 101, 249, 130]\n",
      "--------------------------------\n",
      "person hardhat  :  77.1479070186615  :  [314, 99, 361, 142]\n",
      "--------------------------------\n"
     ]
    }
   ],
   "source": [
    "import random\n",
    "\n",
    "testImages = os.listdir(\"hardhat/validation/images\")\n",
    "randomFile = testImages[random.randint(0, len(testImages) - 1)]\n",
    "\n",
    "detectedImage, detections = detector.detectObjectsFromImage(output_type=\"array\", \n",
    "
input_image=\"hardhat/validation/images/{0}\".format(randomFile), \n",
    "
minimum_percentage_probability=30)\n",
    "showImage(detectedImage)\n",
```

```
    "\n",
    "for eachObject in detections:\n",
    "    print(eachObject[\"name\"] , \" : \", eachObject[\"percentage_probability\"], \" : \",
eachObject[\"box_points\"] )\n",
    "    print(\"--------------------------------\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.7.1"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ ЛЮДЕЙ У КАСЦІ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ»

Виконала: студентка 4 курсу,групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ящук Т. Є.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«04» червня 2025 р.

Вінниця ВНТУ - 2025 рік

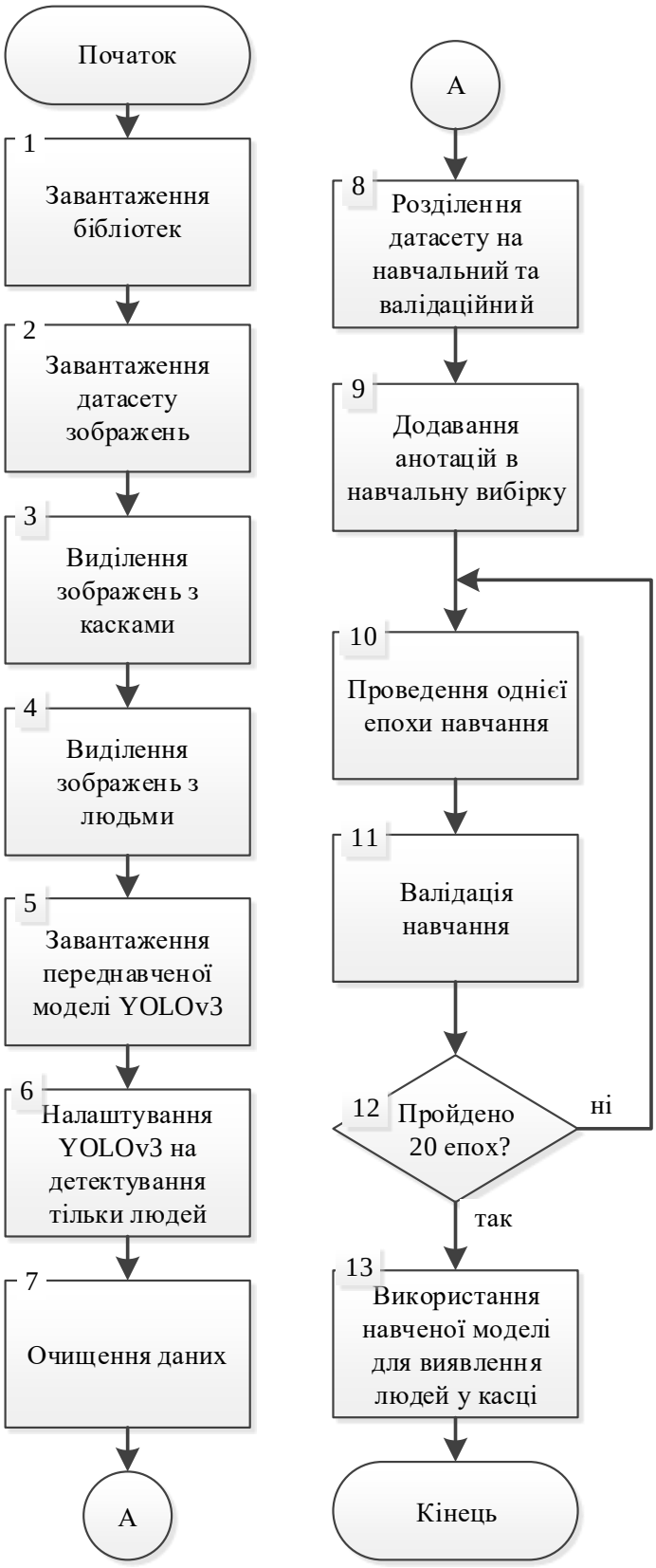


Рисунок В.1 – Схема алгоритму роботи програмного модуля розпізнавання людей у касці на основі згорткової нейронної мережі YOLOv3

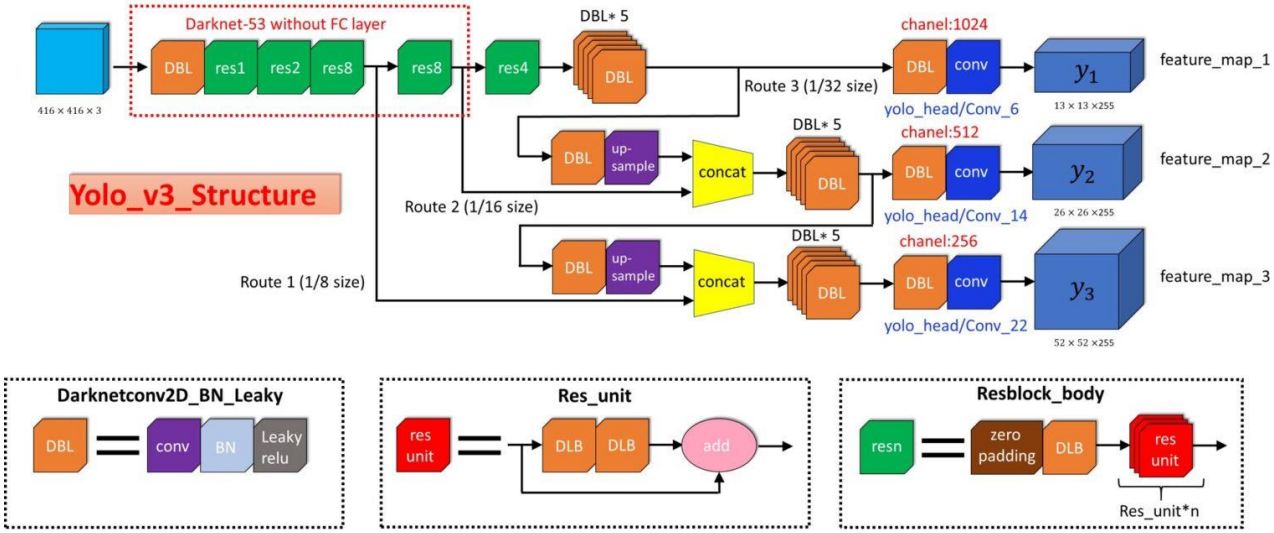


Рисунок В.2 – Структура згорткової нейронної мережі YOLOv3

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

Рисунок В.3 – Деталізація шарів згорткової нейронної мережі YOLOv3

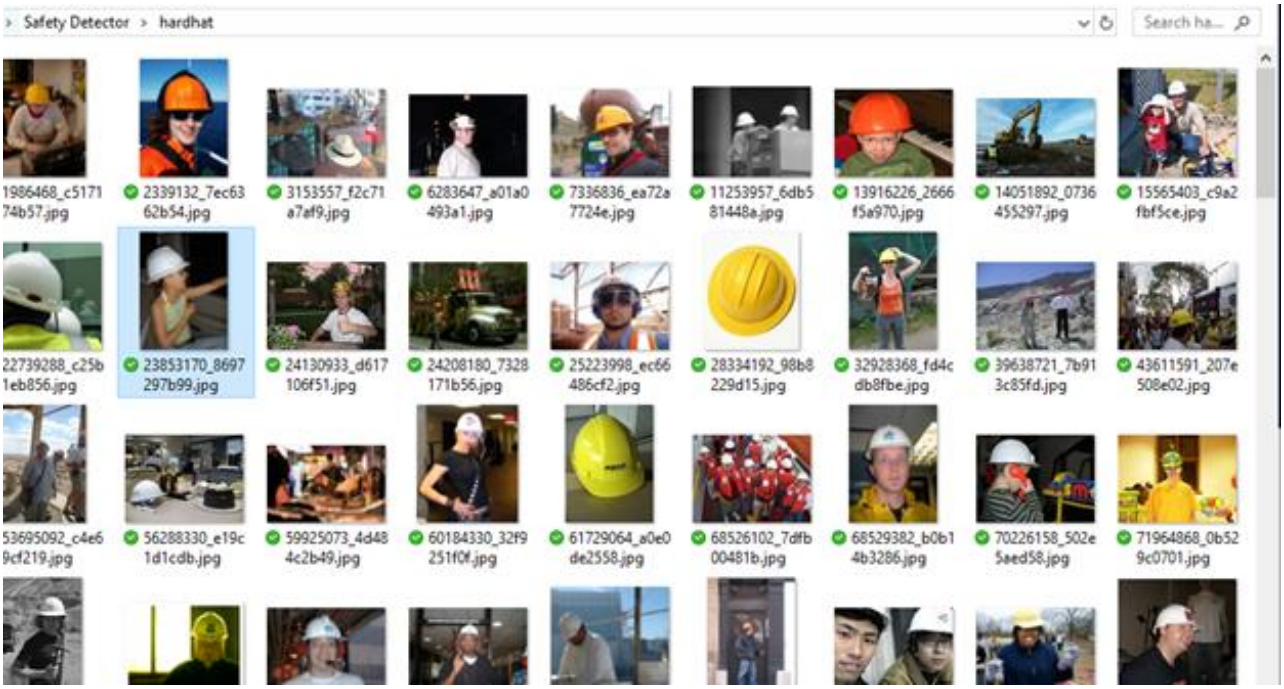


Рисунок В.4 – Приклади отримання зображень людей з каскою

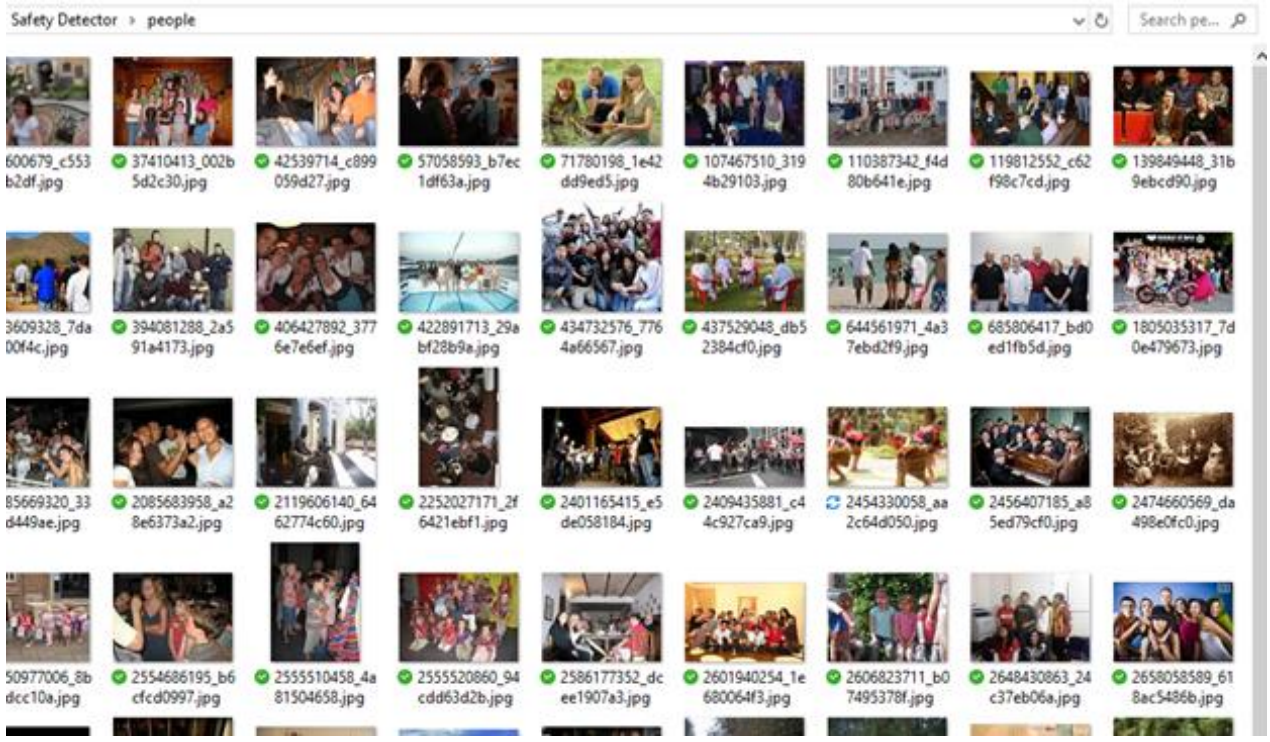
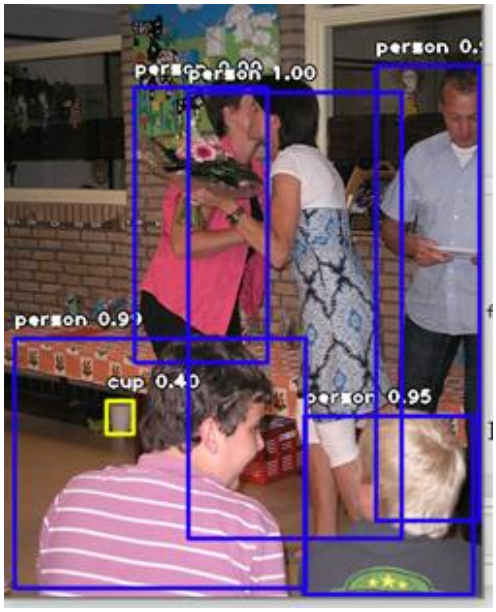


Рисунок В.5 – Приклади отримання зображень людей без каски




```
tie  ---  42.86769926548004  ---  [78, 146, 87, 173]
handbag  ---  49.53037202358246  ---  [22, 178, 44, 245]
person  ---  31.346148252487183  ---  [279, 113, 331, 200]
person  ---  36.62037551403046  ---  [91, 113, 143, 201]
person  ---  44.622498750686646  ---  [278, 114, 332, 307]
person  ---  74.34216737747192  ---  [79, 117, 141, 328]
person  ---  80.34021258354187  ---  [376, 108, 436, 314]
person  ---  90.66885113716125  ---  [171, 96, 243, 173]
person  ---  99.65248703956604  ---  [32, 104, 99, 332]
person  ---  99.71648454666138  ---  [308, 115, 386, 329]
person  ---  99.73611831665039  ---  [390, 138, 470, 327]
person  ---  99.80701804161072  ---  [171, 135, 250, 329]
person  ---  99.82082843780518  ---  [231, 127, 312, 332]
person  ---  99.85131621360779  ---  [112, 137, 180, 332]
```

Рисунок В.6 – Приклад виявлення об’єктів попередньо натренованою моделлю YOLOv3

Рисунок В.7 –Результат роботи програмного модуля

№ п/п	Метрика	Розмір входу нейромережі	Обсяг моделі, Мбайт	Середня точність mAP, %
1	Аналог 1: «Детектор випадків невикористання каски на основі Faster R-CNN» [с17],	300x500	37,2	80,75
2	Аналог 2: «Автоматичний детектор носіння касок на основі SSD» [с].	512x512	36,4	83,0
	Розроблений програмний модуль (на основі YOLOv3)	416x416	34,7	84,5

Рисунок В.8 – Порівняння параметрів розробленого програмного модуля розпізнавання людей у касці та аналогів