Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Система збору й класифікації графічних даних
(тема)

Виконав:
студент II курсу, групи КСМм-19-1
Даценко Д. Г.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: проф. Завизіступ Ю. Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ ______ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________________
(підпис)
"____"_______________ 20__ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Даценку Данилу Гійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система збору й класифікації графічних даних

затверджена наказом по університету від " 30 " жовтня 2020 р. № 1487 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи
   1) документація мови програмування Python
   2) документація бібліотеки Keras, TensorFolow
   3)документація по OpenCV/Keras
   4)перелік використаних програмних засобів: Linux Ubuntu, Sublime Text 3
   5) перелік використаних пристроїв: 1 персональный комп'ютер

4. Перелік питань, що потрібно опрацювати в роботі
   1) Огляд предметної області
   2) Постановка завдання
   3) Використані технології
   4) Програмна реалізація
   5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (слайдів) Слайд презентація – 16 слайдів.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	03.11.20-09.11.20	
2	Формування переліку вимог до програми	10.11.20-17.11.20	
3	Вибор технології розробки та	18.11.20-23.11.20	
	інструментальних засобів		
4	Розробка алгоритмічного забезпечення	24.11.20-01.12.20	
5	Розробка та тестування програми	02.12.20-07.12.20	
6	Відлагодження програмних модулів	08.12.20-09.12.20	
7	Оформлення матеріалів атестаційної роботи	10.12.20-11.12.20	

Дата видачі завдання 02 листопада 2020 р.

Студент ____________________
(підпис)

Керівник роботи ____________________ проф. Завизіступ Ю. Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 72 с., 13 рис., 11 дод., 15 джерел.

СКРИПТ, СИСТЕМА ВИЯВЛЕННЯ ЗАХИСТНОЇ МАСКИ, COVID-19, PYTHON, TENSORFLOW, OPENCV, KERAS, ШТУЧНА НЕЙРОМЕРЕЖА, ЗГОРТКОВА НЕЙРОМЕРЕЖА.

Об'єкт розробки – скрипт для виявлення захисної маски для обличчя.

Мета роботи – розробити систему для виявлення захисної маски на обличчі.

В результаті виконання даної атестаційної роботи було розглянуто логічне і фізичне моделювання даних, спроектована і реалізована біблілотека малюнків. Програмне забезпечення було реалізоване за допомогою Python та бібліотеки Tensorflow, OpenCV, Keras.

ABSTRACT

Master’s thesis: 72 pages, 13 figures, 11 appendices, 15 sources.

SCRIPT, PROTECTIVE MASK DETECTOR, COVID-19, PYTHON, TENSORFLOW, OPENCV, KERAS, NEURAL NETWORK, CONVOLUTIONAL NEURAL NETWORK.

The object of development is a script to detect a protective face mask.

The purpose of the work is to develop a detector to detect a protective mask on the face.

As a result of this certification work, logical and physical data modeling was considered, and a drawing library was designed and implemented. The software was implemented using Python and the library Tensorflow, OpenCV, Keras.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Актуальність розробки та розгляд тематики атестаційної роботи 11
1.2 Проблеми і методи класифікації........ 11
1.3 Підхід, заснований на даних 12
1.4 Метод найближчого сусіда 12
1.6 Метод k- найближчих сусідів 14
1.7 Евклідова відстань 15
1.8 Гіперпараметри........ 15
1.9 Лінійна класифікація 17
1.10 Інтерпретація лінійного класифікатора 18
2 ПОСТАНОВКА ЗАВДАННЯ ПРОЕКТУВАННЯ........ 19
3 АНАЛІЗ МЕТОДІВ ОБРОБКИ........ 20
3.1 TenserFlow........ 20
3.1.1 Встановлення середовища розробки Python в систему........ 21
3.1.2 Створення віртуального середовища. 21
3.1.3 Встановлення пакету pip TensorFlow. 22
3.2 OpenCV........ 23
3.2.1 API Concepts 24
3.2.2 Автоматичне управління пам'яттю 24
3.2.3 Автоматичний розподіл вихідних даних........ 25
3.3 Двофазна система виявлення захисної маски для обличчя 27
3.3.1 Набір даних по виявленню захистної маски для обличчя 28
4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ........ 35
4.1 Структра поекту 35

4.2 Реалізація навчального скрипта системи виявлення маски. 35
4.3 Тренування системи виявлення маски для обличчя за допомогою Keras / TensorFlow 42
4.4 Впровадження нашої системи виявлення маски для обличчя для зображень з OpenCV 44
4.5 Виявлення маски для обличчя на зображеннях з OpenCV 48
4.6 Впровадження нашої системи виявлення маски для обличчя у потоках відео в реальному часі з OpenCV 52
4.7 Виявлення масок для обличчя за допомогою OpenCV у режимі реального часу. 58
ВИСНОВКИ... 61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62
ДОДАТОК А Графічний матеріал атестаційної роботи 64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВНЗ – вищий навчальний заклад

ГОСТ – «государственный стандарт» (міждержавний стандарт СНД)

ДСТУ – Державний стандарт України

ЕОМ – електронна обчислювальна машина

МОН – Міністерство освіти і науки України

МЗ – медичний заклад

НП – навчальний план

РГР – розрахунково-графічна робота

СЛС – структурно-логічна схема

СРС – самостійна робота студентів

Таблиці сутності – основні таблиці, в яких міститься інформація, що динамічно змінюється

ANSI – Американський національний інститут стандартів (англ., American National Standards Institute)

CNN – згорткова нейронна мережа (англ., Convolutional Neural Network)

ECTS – Європейська система переведення і накопичення кредитів (англ., European Credit Transfer and Accumulation System)

TF – платформа з відкритим вихідним кодом для машинного навчання (англ., TensorFlow)

UI – користувацький інтерфейс (англ., User Interface)

XML – розширювана мова розмітки (англ., eXtensible Markup Language)

ВСТУП

Комп'ютерний зір – це аналіз візуальних даних. Обсяг цих даних в сучасному світі стає дедалі більше. Близько 80% всього інтернет–трафіку становить відео – і це без урахування зображень і інших типів візуальної інформації. Тому важливо розробляти алгоритми, які зможуть розуміти і обробляти ці дані.

Іноді візуальну інформацію порівнюють з темною матерією за аналогією з фізикою. Темна матерія займає дуже велику частку маси у Всесвіті, і ми знаємо про це завдяки існуванню гравітаційного тяжіння на різних небесних тілах. Однак ми не можемо безпосередньо спостерігати її. З візуальними даними відбувається приблизно те ж саме: вони містять безліч біт, літаючих по просторах Інтернету. Але алгоритмам комп'ютерного зору дуже складно зрозуміти, з чого вони складаються насправді.

У ці важкі часи COVID-19, носіння масок в громадських місцях стає життєвою необхідністю. Проведені дослідження підтверджують, що маски зараз є найефективнішим захистом від COVID–19. Особливо важливо використовувати засоби захисту в громадських місцях. Фахівці нагадують, що маска затримує краплі, які виділяються під час розмови, кашлю або чхання. Адже переважно повітряно-крапельним шляхом і відбувається зараження. Маска повинна якомога ближче прилягати до обличчя, не слід торкатися до неї під час носіння. Перед використанням і після зняття обов'язково мити руки або обробляти їх антисептиком. Багаторазову маску потрібно прати при температурі не нижче 60 градусів, потім пропрасувати гарячою праскою. Ризик зараження знижується до 70%, якщо людина в громадських місцях знаходиться в масці, якщо здорова людина теж знаходиться в масці поруч з хворим то, ймовірність зараження знижується до 90%. Маску необхідно міняти кожні дві години.

Хіба не було б приємно зробити щось, пов'язане з цим? Я вирішив побудувати дуже просту Convolutional Neural Network (CNN) модель з використанням TensorFlow з Keras і OpenCV для виявлення, захисної маски на лиці, щоб захистити себе та оточуючих.

Для побудови цієї моделі, я буду використовувати набір даних лицьових масок, зібраних Prajna Bhandary. Він складається з близько 1376 зображень, з яких 690 зображень містять людей в масках і 686 зображень містять людей без масок.

Я збираюся використовувати ці зображення для створення моделі CNN за допомогою TensorFlow, щоб визначити, вділи ви маску чи ні, за допомогою веб-камери вашого ПК, або на зображеннях.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність розробки та розгляд тематики атестаційної роботи

Актуальність атестаційної роботи полягає саме у автоматизації процесу классифікації захистних масок на обличчі людини під, існує багато подібних програмних рішень але вони спеціалізовані під старі вимоги, або мають застарілі реалізації, що у більшості випадків не відповідають сучасним вимогам під час пандемії COVID-19 [7].

1.2 Проблеми і методи класифікації

Як же працює класифікація? Коли ваша система отримує вхідне зображення, їй вже відомий фіксований набір категорій або міток. Це можуть бути будь-які об'єкти: «кішка», «собака», «літак», «вантажівка» і так далі. Комп'ютер повинен подивитися на зображення і призначити йому одну з міток.

З боку завдання виглядає нескладною, оскільки велика частина нашої зорової системи запрограмована на розпізнавання об'єктів. Але для машини це не так-то просто.

Коли комп'ютер дивиться на зображення, то не бачить цілісне уявлення кішки або будь-якого іншого об'єкта. Він бачить лише гігантську сітку чисел. Наприклад, якщо розмір зображення 800 на 600 і кожен піксель представлений трьома числами для червоного, зеленого і синього каналів, то вийде сітка з 800х600х3 = 1,440,000 чисел. Дуже важко виділити з них якийсь конкретний предмет, представлений на фотографії.

Ця проблема отримала назву «семантичний розрив» – нерозуміння інформації, яка укладена в даних. Наприклад, якщо ми знімемо кішку з іншого ракурсу або при іншому освітленні, то вся сітка чисел буде виглядати

зовсім інакше. Крім цього, тварини можуть приймати безліч різних поз, або ж на фотографії може виявитися тільки частина кішки, наприклад, хвіст. Алгоритми розпізнавання повинні бути стійкі до таких змін.

Крім цих складнощів існує ще проблема внутріклассових варіацій, коли одне поняття охоплює безліч візуальних проявів. Наприклад, кішки можуть бути різних порід, різного віку і розмірів. І методи розпізнавання повинні обробляти всі можливі варіанти.

1.3 Підхід, заснований на даних

Замість того, щоб вручну намагатися створити набір правил, можна відкрити інтернет і зібрати великий набір даних з фотографіями кішок, літаків, оленів та інших речей. Для цього підійде пошук картинок Google або вже готовий датасета. Потім необхідно навчити класифікатор, відправивши в нього всі зібрані зображення. На виході ми отримаємо модель, узагальнюючу знання про розпізнавання різних об'єктів. Після цього вона зможе працювати на нових зображеннях і відрізняти кішок від собак.

Отже, замість однієї функції, яка просто розпізнає об'єкт на вхідному зображенні, ми отримали дві: перша називається «навчання» – це процес обробки зображень і створення моделі. Друга функція – «прогноз» – розпізнає нові фотографії. Разом вони складають основу для свёрточних нейронних мереж і глибокого навчання в цілому.

1.4 Метод найближчого сусіда

Для початку розглянемо найпростіший класифікатор, який носить назву «метод найближчого сусіда». У процесі навчання він запам'ятовує всі вихідні дані, а потім на етапі прогнозування намагається знайти серед них найбільш схожі на нове зображення.

Візьмемо набір даних CIFAR-10, що містить 10 різних класів і близько 50000 навчальних і 10000 тестових зразків.

Так на ньому працює класифікатор найближчого сусіда.

Тестові зображення найбільш схожі на них навчальні зразки. Можна помітити, що вони не завжди виявляються правильними. Взявши найбільш близького сусіда і його мітку, алгоритм класифікує тестове зображення.

1.5 Реалізація методу найближчого сусіда

Як же порівняти два зображення? Насправді для цього є багато різних способів. В наведеному вище прикладі використовувалося відстань L1, також відоме як Манхеттенський. Воно просто порівнює між собою пікселі фотографій. Припустимо, що у нас є тестовий зразок розміром 4x4 пікселя. Візьмемо одне з навчальних зображень і обчислимо абсолютну різницю між квітами пікселів навчає і тестового зразків, а потім підсумуємо отримані значення.

Реалізація методу найближчого сусіда на Python:

```
import numpy as np

class NearestNeighbor:
def __init__(self):
pass

def train(self, X, y):
# X - матриця прикладів NxD, Y - одновимірний масив розмірності N
# Класифікатор найближчого сусіда просто запам'ятовує навчальні дані
self.Xtr = X
self.ytr = y

def predict(self, X):
# X - матриця NxD, де кожен рядок - приклад, для якого
# необхідно спрогнозувати мітку
num_test = X.shape[0]
# Переконаємося, що тип вихідних значень збігається з типом вихідних значень
Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

# Цикл по всім тестовим рядкам
for i in xrange(num_test):
# Знаходимо найближчого сусіда для i-го тестового зображення
# за допомогою відстані L1 (суми абсолютних різниць)
distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
```

```
    # Отримуємо індекс об'єкта з мінімальним відстанню
    min_index = np.argmin(distances)
    # Прогнозуємо мітку
    Ypred[i] = self.ytr(min_index)
    return Ypred
```

Приклад 1.1 – Реалізація методу

Якщо в навчальному наборі міститься N прикладів, то навчання завжди буде виконуватися за постійний час O (1), а прогнозування – за лінійний час O (N), оскільки тестове зображення порівнюється з кожним навчальним зразком. «O» тут означає тимчасову складність алгоритму.

Лінійне час прогнозування – не дуже хороший результат. На практиці потрібно, щоб класифікатори навчалися повільно, а тестувалися швидко. Пізніше ми побачимо, що більш просунуті алгоритми працюють саме так: їх можна довго навчати в дата–центрі або на хмарних серверах, а потім запускати на мобільних телефонах.

Що саме робить алгоритм найближчого сусіда? На малюнку нижче показані так звані області прийняття рішень класифікатора.

Точки – це об'єкти навчальної вибірки, а кольори – категорії або мітки класів. Для кожного зразка визначається найбільш схожий приклад, і відповідно до нього точка закрашивается кольором знайденого класу.

На зображенні можна побачити кілька проблем, на які наражається класифікатор. Наприклад, зелена область містить одну що не вписується помаранчеву точку, а на кордоні синього і червоного класів присутні незрозумілі відгалуження. Ці дані можуть бути зашумлений або некоректними. Щоб виявити їх, існує більш загальний метод.

1.6 Метод k- найближчих сусідів

Замість того, щоб шукати одне схоже зображення, ми будемо шукати k найближчих сусідів відповідно до метрикою відстані. Після цього проведемо голосування по кожному сусідові і спрогнозуємо мітку відповідно до

більшості голосів. Тут показаний той же набір точок з числом сусідів k = 1, k = 3 і k = 5.

Можна побачити, що жовта точка всередині зеленого кластера більше не створює область навколо себе, а межа між червоним і синім класами стає більш гладкою. Як правило, для класифікації використовується значення k> 1. Білі регіони тут означають області, в яких не знайдено жодного найближчого сусіда.

1.7 Евклідова відстань

Відстань L2 (евклидово) теж часто використовується в задачах класифікації. Воно виглядає як корінь з суми квадратів різниці між квітами пікселів:

Метрики відстані роблять різні припущення про очікувану геометрії або топології простору. На малюнку видно, що L1 формує квадратну область, тоді як L2 створює коло. При повороті системи координат евклідова відстань не зміниться, а Манхеттенський видасть вже інший результат. Цей просторовий ефект важливо враховувати і вибирати метрику відповідно до початкової завданням.

Ви можете самостійно протестувати класифікатор найближчого сусіда в веб-демонстрації. Спробуйте пограти з метриками, числом сусідів, класів і точок.

1.8 Гіперпараметри

Процес вибору впливають на роботу будь-якого методу значень, таких як число сусідів k і метрика відстані, називається настроюванням гіперпараметров. Гіперпараметри не можуть бути явно витягнуті з навчальних даних і залежать тільки від самого алгоритму, тому не існує чітких рекомендацій по їх вибору. Найчастіше (і більшість людей робить

саме так) доводиться шукати значення методом проб і помилок, з'ясовуючи, які працюють найкраще.

Розберемося, що таке «найкраще».

Ідея 1. Підібрати такі параметри, які дадуть найвищу точність на навчальних даних.

І це дуже погана ідея. Ніколи не робіть так. У випадку з методом найближчих сусідів при k = 1 досягається майже ідеальна точність під час навчання, але з тестовими даними алгоритм справляється вкрай погано. Це явище називається «перенавчання».

Ідея 2. Розділити набір на навчальну і тестову вибірки і знайти гіперпараметри, які змусять алгоритм краще працювати на тестових зразках.

Ця стратегія виглядає більш розумною, але насправді вона теж дуже погана. Основна ідея машинного навчання в тому, що ми хочемо знати, як буде працювати метод. І якщо ми підбираємо параметри, з якими досягаються хороші результати на невідомих зображеннях, то немає ніякої гарантії, що вони будуть досягнуті на інших невідомих зображеннях.

Ідея 3. Розділити набір даних на навчальну, тестову і оцінну вибірки; підібрати гіперпараметри для оціночних даних і перевірити на тестових.

І це вже хороша ідея. Спочатку класифікатор навчається з різними варіантами параметрів. Потім вибираються ті значення, які найкраще працюють на оціночних даних. Після цього модель обробляє тестову вибірку тільки один раз. Точність, яка досягається при цьому, показує справжню ефективність класифікатора.

Ідея 4. Розділити навчальні дані на безліч невеликих підвибірок, потім зробити перехресну перевірку, використовуючи різні підвибірки в якості оціночних даних, і усереднити результати.

Метод називається «крос-валідація» і добре працює на невеликих датасета, але вимагає дуже багато обчислювальних ресурсів для величезних наборів даних. З цієї причини такий підхід нечасто використовується в глибокому навчанні.

Після перехресної перевірки ви отримаєте графік, який виглядає приблизно так:

На осі X показано значення k (число сусідів), а на осі Y – точність класифікатора. За графіком можна оцінити ефективність моделі в залежності від гіперпараметров, В цьому випадку найбільша точність досягається при k≈7.

1.9 Лінійна класифікація

Алгоритми лінійної класифікації досить прості. Проте, з їх допомогою створюються повноцінні нейронні мережі. Це схоже на складання Lego: можна скласти один з одним різні компоненти і побудувати «вежу» свёрточной нейромережі.

Лінійний класифікатор – один з основних будівельних блоків, що використовуються в різних додатках глибокого навчання. Спробуємо розібратися, як він працює.

Повернемося до набору даних CIFAR-10. Щоб класифікувати зображення, ми створимо просту параметричну модель з двох компонент. Перша – вхідні дані, які зазвичай позначаються як X, і друга – набір параметрів або терезів W. Тепер запишемо деяку функцію, яка приймає дані X і параметри W, а потім видає 10 чисел, що описують оцінки для кожної з 10 категорій в CIFAR-10.

Правильно працююча модель в прикладі вище видасть найбільшу оцінку для класу кішки. У цьому параметричному підході, на відміну від методу найближчого сусіда, ми узагальнюємо знання про навчальні даних і використовуємо їх при створенні параметрів. Тому під час тестування нам не потрібно кожного разу звертатися до навчальної вибірки.

Як же отримати функцію, яка об'єднує в собі ваги і дані? Найпростіший спосіб – просто перемножити їх. Це і буде лінійний класифікатор: fx, W = Wx.

Іноді в функцію додається елемент зсуву b – це лінійний вектор, розмір якого дорівнює числу класів (в нашому випадку 10). Він не взаємодіє з навчальними даними і видає незалежні припущення для класів. Наприклад, якщо в наборі набагато більше зображень з кішками, ніж з собаками, то відповідні класу кішок елементи вектора зміщення будуть вище за інших. Функція класифікатора зі зміщенням має вигляд: fx, W = Wx + b.

1.10 Інтерпретація лінійного класифікатора

Розглянемо просте зображення 2x2. Лінійний класифікатор розтягує його в вектор-стовпець з чотирма елементами. Тут ми використовуємо три класи: «кішка», «собака» і «корабель», тому вагова матриця буде мати розмірність 4x3. Вектор зміщення з трьох елементів створює незалежні умови для кожної категорії:

Оцінка для будь-якого з класів складається з твору між пікселями зображення і відповідними рядками вагової матриці, до якого додається зсув. Це схоже на порівняння зі зразком: кожен рядок відповідає деякому шаблонного зображенню, а оцінка вказує на його схожість з вихідною фотографією. Якщо спробувати згорнути рядки ваговій матриці назад в зображення, ми дійсно отримаємо шаблони, зібрані з навчальних даних:

Лінійний класифікатор використовує тільки один шаблон для кожного класу. Створюючи складніші функції, що зв'язують дані і параметри, ми зможемо вивчати більше шаблонів і досягнемо набагато кращої точності.

В області прийняття рішень кожне з зображень представлено у вигляді точки в багатовимірному просторі, яку лінійний класифікатор намагається помістити в межі лінійного рішення. Інакше кажучи, він буде розділяти категорії між собою прямими лініями.

У цьому розділі ми розглянули основні підходи до класифікації графічних даних які ми можем використати для проектування нашої системи виявлення захисних масок на обличчі людини.

2 ПОСТАНОВКА ЗАВДАННЯ ПРОЕКТУВАННЯ

Метою атестаційної роботи є створення системи виявлення захисної маски для обличчя, детально описавши, як будемо реалізовувати нейромережу комп'ютерного зору / глибокого навчання.

Ми розглянемо набір даних, який будемо використовувати для навчання нашого системи виявлення захисної маски для обличчя.

Потім реалізуємо скрипт Python для навчання системи виявлення для захисної маски обличчя в нашому наборі даних за допомогою Keras і TensorFlow.

Ми будемо використовувати цей скрипт Python для навчання системи виявлення і перегляду результатів.

З огляду на навчений система виявлення, ми приступимо до реалізації ще двох додаткових скриптів Python, які будемо використовувати для:

Виявлення захисних масок для обличчя на зображеннях

Виявлення захисних масок для обличчя в відеопотоках в реальному часі

Таким чином ми автоматизуемо контроль за виконнаням умов носіння захисних масок для обличчя у громадських місцях під час пандемії COVID-19. Ми завершимо роботу, подивившись на результати застосування системи виявлення.

Вимоги до реалізації проекту:

- мова програмування – Python;
- допоміжні бібліотеки – TensorFlow, Keras, OpenCV;
- середовище розробки – Sublime Text 3.

У цьому розділі ми поставили мету нашої роботи, та позначили основні вимоги для успішної реалізації проекту.

3 АНАЛІЗ МЕТОДІВ ОБРОБКИ

3.1 TenserFlow

Доступні пакети TensorFlow 2 [3,4]:

- tensorflow – останній стабільний випуск з підтримкою CPU і GPU (Ubuntu і Windows);
- tf-nightly nightly – попередня збірка (нестабільна). Ubuntu і Windows включають підтримку графічного процесора;
- більш ранні версії TensorFlow.

Для TensorFlow 1.x пакети CPU і GPU розділені:

- tensorflow 1.15 – випуск тільки для ЦП;
- tensorflow-gpu 1.15 – випуск з підтримкою графічного процесора (Ubuntu і Windows).

Системні вимоги:

- python 3.5-3.8;
- для підтримки Python 3.8 потрібно TensorFlow 2.2 або новіше;
- pip 19.0 або новіше (потрібна підтримка manylinux2010);
- ubuntu 16.04 або новіше (64-біт);
- macOS 10.12.6 (Sierra) або новішим (64-розрядної версії) (без підтримки графічного процесора);
- windows 7 або новіше (64–біт);
- розповсюджуваний компонент Microsoft Visual C ++ для Visual Studio 2015, 2017 і 2019;
- raspbian 9.0 або новіше;
- для підтримки графічного процесора потрібно карта з підтримкою CUDA® (Ubuntu і Windows);
- примітка: для установки TensorFlow 2 потрібно більш нова версія pip.

- починаючи з TensorFlow 1.6, виконавчі файли використовують інструкції AVX, які можуть не працювати на старих процесорах.

3.1.1 Встановлення середовища розробки Python в систему.

Переконайтеся, що ваша середа Python вже налаштована:

- потрібно Python 3.5-3.8, pip і venv> = 19.0;
- python3 – version;
- pip3 – version;

Якщо ці пакети вже встановлені, переходите до наступного кроку.

В іншому випадку встановіть Python, диспетчер пакетів pip і venv:

- ubuntu;
- macOS;
- windows;
- raspberry Pi;
- інші.

Якщо не у віртуальному середовищі, використовуйте python3 – m pip для команд нижче. Це гарантує, що ви відновите і будете використовувати Python pip замість системного pip.

3.1.2 Створення віртуального середовища.

Віртуальні середовища Python використовуються для ізоляції установки пакета від системи [1,2]:

- Ubuntu / macOS;
- Windows.

Створимо нову віртуальну середу, вибравши інтерпретатор Python і venv каталог для його зберігання:

- python3 -m venv –system-site-packages venv.

Активуйте віртуальне середовище за допомогою спеціальної команди оболонки:

```
source venv/bin/activate sh, bash, or zsh
venv/bin/activate.fish
source venv/bin/activate.csh
```

Приклад 3.1 – Активація віртуального середовища

Коли віртуальне середовище активна, запрошення оболонки має префікс (venv).

Встановлюйте пакети в віртуальному середовищі, не впливаючи на настройку хост–системи. Почніть з поновлення pip:

```
pip install -upgrade pip
pip list show packages installed within the virtual environment
```

Приклад 3.2 – Встановлення пакетів в віртуальному середовищі

3.1.3 Встановлення пакету pip TensorFlow.

Виберіть один з наступних пакетів TensorFlow для установки з PyPI:

- tensorflow – останній стабільний випуск з підтримкою CPU і GPU (Ubuntu і Windows);
- tf nightly nightly – попередня збірка (нестабільна). Ubuntu і Windows включають підтримку графічного процесора;
- tensorflow 1.15 версія TensorFlow 1.x.

Залежно пакетів встановлюються автоматично. Вони перераховані у файлі setup.py розділі REQUIRED_PACKAGES:

- установка віртуального середовища;
- установка системи.

3.2 OpenCV

OpenCV – це бібліотека з відкритим кодом, що включає кілька сотень алгоритмів комп'ютерного зору. Документ описує так званий API OpenCV 2.x, який, по суті, є API C ++, на відміну від API OpenCV 1.x на основі C (C API застарілий і не тестується за допомогою компілятора "C" з моменту випуску OpenCV 2.4) [14].

OpenCV має модульну структуру, що означає, що пакет включає кілька спільних або статичних бібліотек. Доступні такі модулі.

Основна функціональність (core) – компактний модуль, що визначає основні структури даних, включаючи щільний багатовимірний масив Mat і основні функції, що використовуються всіма іншими модулями.

Обробка зображень (imgproc) – модуль обробки зображень, що включає лінійну та нелінійну фільтрацію зображень, геометричні перетворення зображень (зміна розміру, аффінне та перспективне перекошування, загальне переназначення на основі таблиці), перетворення кольорового простору, гістограми тощо.

Відеоаналіз (відео) – модуль відеоаналізу, що включає оцінку руху, віднімання фону та алгоритми відстеження об’єктів.

Калібрування камери та 3D–реконструкція (calib3d) – основні алгоритми геометрії декількох видів, калібрування одинарної та стереокамери, оцінка пози об'єкта, алгоритми стереовідповідності та елементи 3D-реконструкції [15].

2D Features Framework (features2d) – основні система виявленняи, дескриптори та збіги дескрипторів.

Виявлення об’єктів (objdetect) – виявлення об’єктів та екземплярів заздалегідь визначених класів (наприклад, обличчя, очі, кружки, людей, машини тощо).

Графічний інтерфейс високого рівня (highgui) – простий у використанні інтерфейс для простих можливостей інтерфейсу.

Video I / O (videoio) – простий у використанні інтерфейс для зйомки відео та відеокодеків. Деякі інші допоміжні модулі, такі як тестові обгортки FLANN та Google, прив'язки Python та інші.

Наступні розділи документа описують функціональність кожного модуля. Але спочатку обов’язково ознайомтесь із загальними поняттями API, які ретельно використовуються в бібліотеці.

3.2.1 API Concepts

Усі класи та функції OpenCV розміщуються у просторі Namespace cv. Тому, щоб отримати доступ до цієї функціональності з вашого коду, використовуйте специфікатор cv :: або використовуючи Namespace cv; директива:

```
#include "opencv2/core.hpp"
...
cv::Mat H = cv::findHomography(points1, points2, cv::RANSAC, 5);
...

#include "opencv2/core.hpp"
using namespace cv;
...
Mat H = findHomography(points1, points2, RANSAC, 5 );
g...
```

Приклад 3.3 – Класи та функції OpenCV

3.2.2 Автоматичне управління пам'яттю

OpenCV обробляє всю пам'ять автоматично. Перш за все, std :: vector, cv :: Mat та інші структури даних, що використовуються функціями та методами, мають деструктори, які при необхідності звільняють базові буфери пам'яті. Це означає, що деструктори не завжди звільняють буфери, як у випадку з Mat. Вони враховують можливий обмін даними. Деструктор зменшує лічильник посилань, пов'язаний з буфером даних матриці. Буфер вивільняється тоді і тільки тоді, коли лічильник посилань досягає нуля, тобто

коли жодна інша структура не посилається на той самий буфер. Подібним чином, коли копіюється екземпляр Mat, фактично не копіюються фактичні дані. Натомість лічильник посилань збільшується, щоб запам'ятати, що існує інший власник тих самих даних. Існує також метод Mat :: clone, який створює повну копію даних матриці.

Дивіться приклад нижче:

```
// create a big 8Mb matrix
Mat A(1000, 1000, CV_64F);
// create another header for the same matrix;
// this is an instant operation, regardless of the matrix size.
Mat B = A;
// create another header for the 3-rd row of A; no data is copied either
Mat C = B.row(3);
// now create a separate copy of the matrix
Mat D = B.clone();
// copy the 5-th row of B to C, that is, copy the 5-th row of A
// to the 3-rd row of A.
B.row(5).copyTo(C);
// now let A and D share the data; after that the modified version
// of A is still referenced by B and C.
A = D;
// now make B an empty matrix (which references no memory buffers),
// but the modified version of A will still be referenced by C,
// despite that C is just a single row of the original A
B.release();
// finally, make a full copy of C. As a result, the big modified
// matrix will be deallocated, since it is not referenced by anyone
C = C.clone();
```

Приклад 3.4 – Методи OpenCV

3.2.3 Автоматичний розподіл вихідних даних

OpenCV автоматично звільняє пам'ять, а також автоматично виділяє пам'ять для вихідних параметрів функції більшу частину часу. Отже, якщо функція має один або кілька вхідних масивів (екземпляри cv :: Mat) та деякі вихідні масиви, вихідні масиви автоматично розподіляються або перерозподіляються. Розмір і тип вихідних масивів визначаються на основі розміру та типу вхідних масивів. Якщо потрібно, функції беруть додаткові параметри, які допомагають з'ясувати властивості вихідного масиву.

Приклад:

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
int main(int, char**)
{
VideoCapture cap(0);
if(!cap.isOpened()) return -1;
Mat frame, edges;
namedWindow("edges", WINDOW_AUTOSIZE);
for(;;)
{
cap >> frame;
cvtColor(frame, edges, COLOR_BGR2GRAY);
GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
Canny(edges, edges, 0, 30, 3);
imshow("edges", edges);
if(waitKey(30) >= 0) break;
}
return 0;
}
```

Приклад 3.5 – Автоматичний розподіл вихідних даних

Кадр масиву автоматично розподіляється оператором >>, оскільки роздільна здатність відеокадро і глибина бітів відомі модулю відеозйомки. Ребра масиву автоматично виділяються функцією cvtColor. Він має такий же розмір і глибину бітів, що і вхідний масив. Кількість каналів дорівнює 1, оскільки передано код перетворення кольору cv :: COLOR_BGR2GREY, що означає перетворення кольору у відтінки сірого. Зверніть увагу, що кадр і краї виділяються лише один раз під час першого виконання тіла циклу, оскільки всі наступні відеокадри мають однакову роздільну здатність. Якщо ви якимось чином змінили роздільну здатність відео, масиви автоматично перерозподіляються.

Ключовим компонентом цієї технології є метод Mat :: create. Він приймає бажаний розмір і тип масиву. Якщо масив уже має вказаний розмір і тип, метод нічого не робить. В іншому випадку він звільняє раніше виділені дані, якщо такі є (ця частина передбачає зменшення лічильника посилань і порівняння його з нулем), а потім виділяє новий буфер необхідного розміру.

Більшість функцій викликають метод Mat :: create для кожного вихідного масиву, і тому реалізовано автоматичне розподіл вихідних даних.

Деякі помітні винятки з цієї схеми – це cv mixChannels, cv RNG fill та кілька інших функцій та методів. Вони не можуть розподілити вихідний масив, тому вам доведеться зробити це заздалегідь.

3.3 Двофазна система виявлення захисної маски для обличчя

Щоб навчити система виявлення маски обличчя, нам потрібно розбити наш проект на дві окремі фази (рисунок 3.1), кожна зі своїми власними відповідними підетапами.

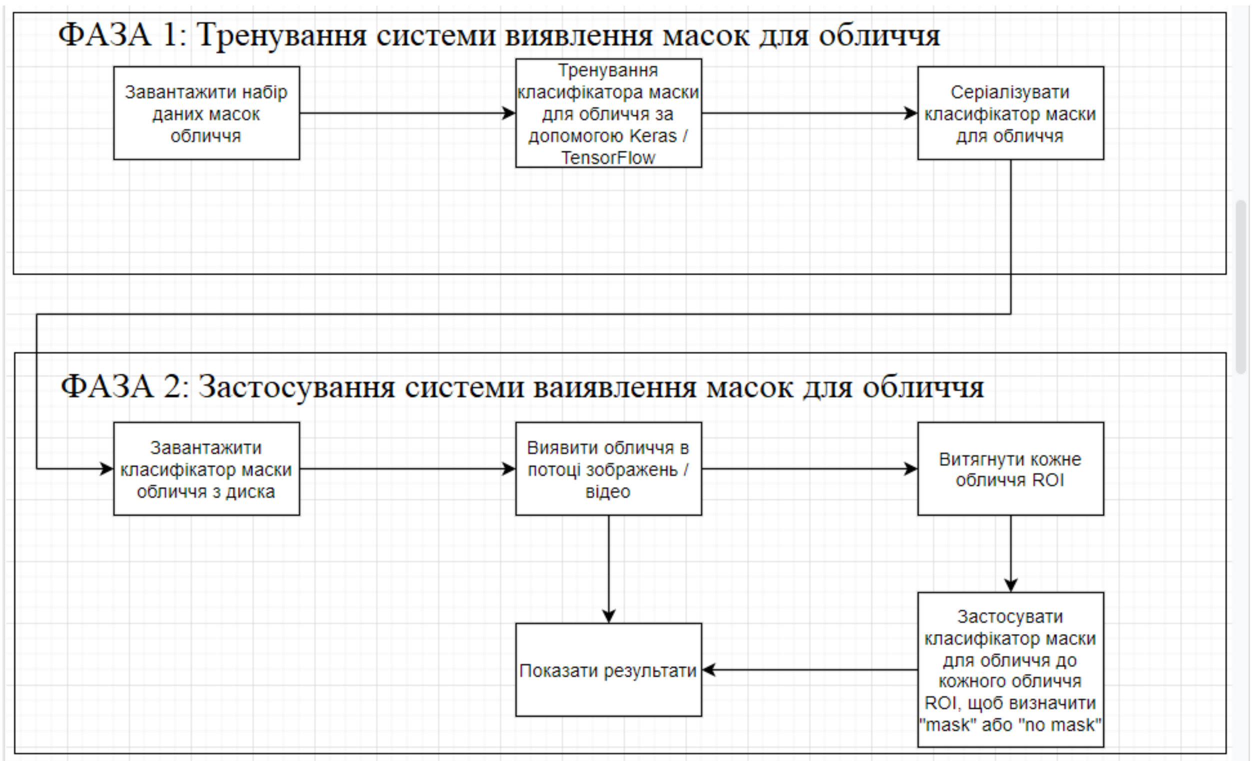

Рисунок 3.1 – Двофазна система виявлення захисної маски для обличчя

Навчання: тут ми зосередимося на завантаженні нашого набору даних виявлення маски особи з диска, навчанні моделі (з використанням Keras / TensorFlow) на цьому наборі даних, а потім сериализации системи виявлення маски особи на диск.

Розгортання: після того, як система виявлення маски особи навчений, ми можемо перейти до завантаження системи виявлення маски, виконання виявлення особи, а потім класифікації кожної особи як with_mask чи without_mask.

3.3.1 Набір даних по виявленню захистної маски для обличчя

Набір даних виявлення маски особи складається з зображень «with_mask» і «without_mask» (рисунок 3.2).

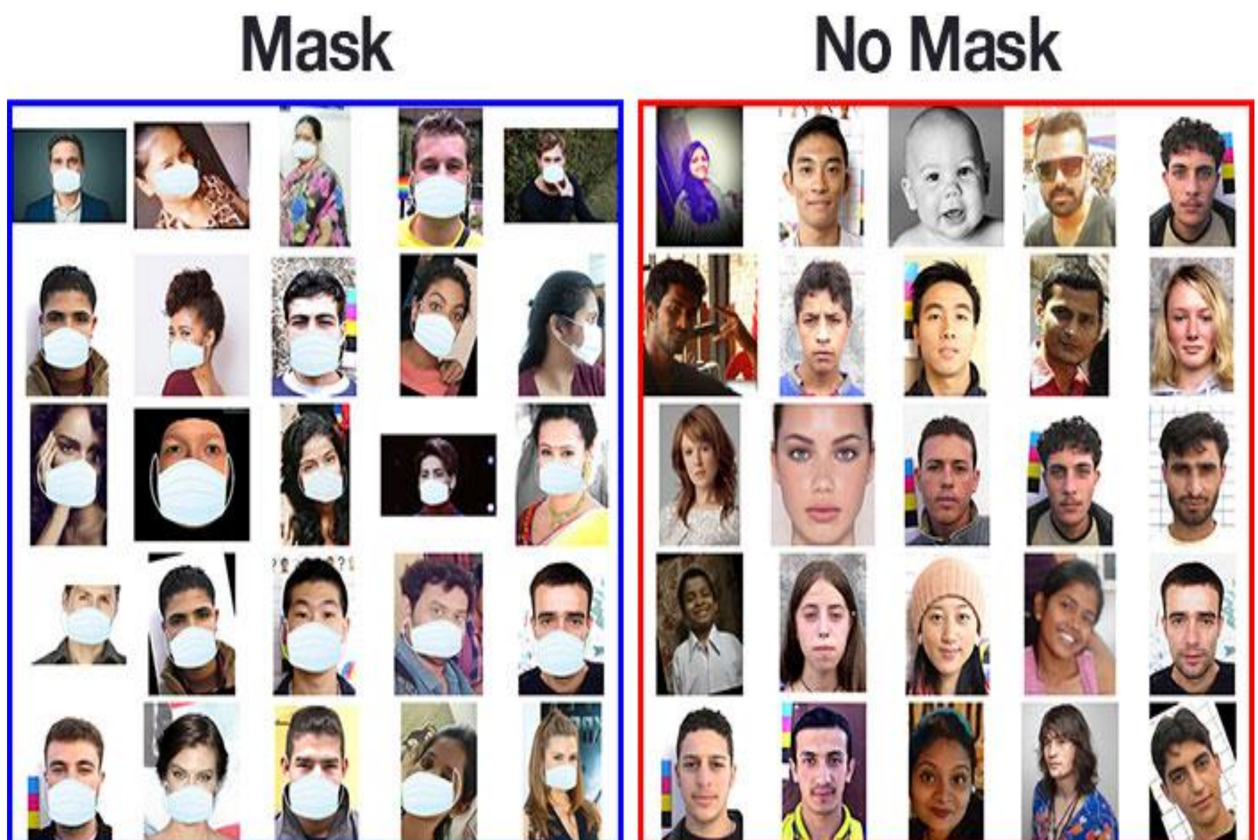

Рисунок 3.2 – Набір даних по виявленню захистної маски для обличчя

Ми будемо використовувати цей набір даних для створення системи виявлення маски особи COVID-19 з комп'ютерним зором і глибоким навчанням з використанням Python, OpenCV і TensorFlow /Keras. Ми будемо використовувати набір даних створений PyImageSearch від Prajna Bhandary.

Цей набір даних складається з 1376 зображень, що відносяться до двох класів:

- with_mask: 690 зображень;
- without_mask: 686 зображень.

Наша мета – навчити індивідуальну модель глибокого навчання визначати, чи носить людина маску чи ні.

Щоб не відчувати пригніченість щодо стану світу – щодня гинуть тисячі людей, ми можемо зробити дуже мало (якщо взагалі щось). Я вирішив розробити цей проект.

У кращому випадку – використовувати свій проект, щоб допомагати іншим.

Найгірший сценарій – це просто відволіктись від пандемії.

У будь-якому випадку це безпрограшний варіант!

Для створення цього набору даних у Prajna було геніальне рішення:

- приймаючи нормальні зображення з граней;
- потім створюємо власний скрипт Python для комп'ютерного зору, щоб додати до них маски для обличчя, тим самим створивши штучний (але все ще можна застосовувати в реальному світі) набір даних.

Цей метод насправді набагато простіше, ніж здається, якщо застосувати до проблеми лицьові орієнтири.

Орієнтири на обличчі дозволяють нам автоматично визначати розташування структур особи, в тому числі:

- очі;
- брови;
- ніс;
- рот;
- лінія підборіддя.

Щоб використовувати лицьові орієнтири для побудови набору даних осіб в масках, нам потрібно спочатку почати з зображення людини без маски для обличчя (рисунок 3.3).

Рисунок 3.3 – Зображення людини без маски

Щоб створити набір даних масок для обличчя для пандемії, ми спочатку почнемо з фотографії людини без маски.

Звідси ми застосовуємо виявлення осіб, щоб обчислити положення обмеження рамки особи на зображенні (рисунок 3.4).

Рисунок 3.4 – положення обмеження рамки особи на зображенні

Наступний крок – застосувати визначання обличчя. Тут ми використовували метод глибокого навчання для виявлення осіб з допомогою OpenCV.

Як тільки ми дізнаємося, де знаходиться особа на зображенні, ми можемо витягти потрібну область обличчя (ROI) (рисунок 3.5).

Рисунок 3.5 – Виявлення особи на зображенні

Наступним кроком є отримання області інтересу особи за допомогою нарізки OpenCV і NumPy. І звідти ми наносимо лицьові орієнтири, що дозволяють нам визначити очі, ніс, рот і т. Д. (рисунок 3.6).

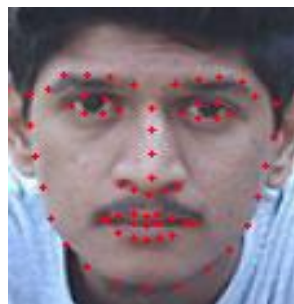

Рисунок 3.6 – Наносимо лицьові орієнтири

Потім ми виявляємо лицьові орієнтири за допомогою dlib, щоб ми знали, де розмістити маску на обличчі.

Далі нам знадобиться зображення маски (з прозорим фоном), як показано нижче (рисунок 3.7).

Рисунок 3.7 – Зображення захисної маски

Ця маска для обличчя буде автоматично накладена на вихідну область інтересу особи, оскільки нам відомі місця розташування орієнтирів на обличчі.

Ця маска буде автоматично застосовуватися до особи за допомогою лицьових орієнтирів (а саме точок уздовж підборіддя і носа) для обчислення, де маска буде розміщена.

Потім маска змінюється і повертається, поміщаючи її на обличчя (рисунок 3.8).

Рисунок 3.8 – Автоматично накладена маска для обличчя

На цьому малюнку лицьова маска поміщена на обличчя людини в вихідної рамці. З першого погляду складно сказати, що маска була застосована з комп'ютерним зором за допомогою OpenCV і dlib– орієнтирів особи.

Потім ми можемо повторити цей процес для всіх наших вхідних зображень, тим самим створивши набір даних штучної маски для обличчя (рисунок 3.9):

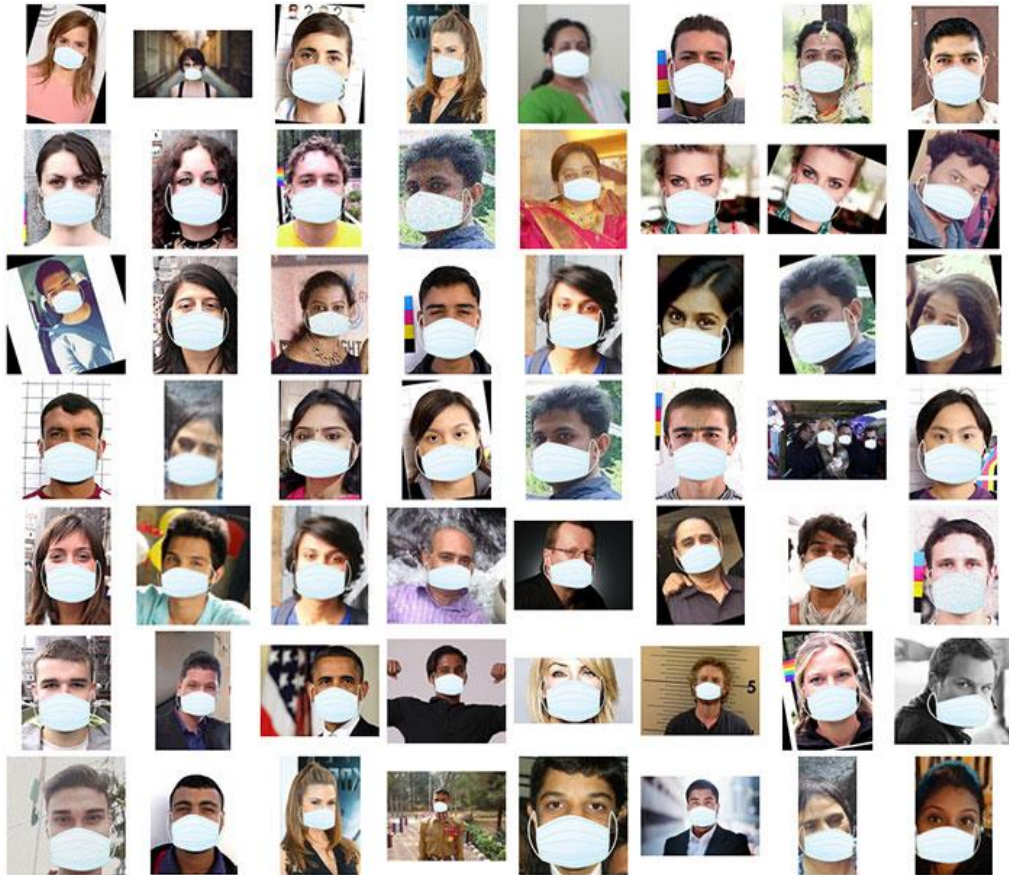

Рисунок 3.9 – Штучний набір зображень лицьових масок

Показаний штучний набір зображень лицьових масок. Цей набір буде частиною нашого набору даних «з маскою» / «без маски» для виявлення маски особи за допомогою комп'ютерного зору і глибокого навчання з використанням Python, OpenCV і TensorFlow / Keras.

Однак є одне застереження, про яку слід пам'ятати при використанні цього методу для штучного створення набору даних!

Якщо ви використовуєте набір зображень для створення штучного набору даних людей в масках, ви не можете «повторно використовувати» зображення без масок в своєму навчальному наборі – вам все одно потрібно зібрати зображення без масок, які не використовувалися в штучних масках. процес генерації!

Якщо ви включите вихідні зображення, використовувані для створення зразків масок особи, як зразки масок без масок, ваша модель стане сильно зміщеною і не зможе добре узагальнювати. Уникайте цього за всяку ціну, знаходячи час, щоб зібрати нові зразки осіб без масок.

У цьому розділі ми розгорнуто аналізували метод классифікації графічних даниих, за допомогою якого наша система з легкістю може виявляти захистні маски на обличчі, проаналізували методи навчання CNN.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Структра поекту

Надано дерево зображень (приклад 4.1), щоб ми могли протестувати система виявлення маски для статичного зображення.

Ми розглянемо три скрипта на Python:

- train_mask_detector.py: приймає наш вхідний набір даних та доопрацьовує MobileNetV2, щоб створити наш mask_detector.model. Також створюється історія тренувань plot.png.;
- detect_mask_image.py: виконує виявлення маски обличчя на статичних зображеннях;
- detect_mask_video.py: за допомогою веб-камери цей сценарій застосовує виявлення маски обличчя до кожного кадру в потоці.

```
$ tree --dirsfirst --filelimit 10
. ├── dataset
│ ├── with_mask [690 entries]
│ └── without_mask [686 entries]
├── examples
│ ├── example_01.png
│ ├── example_02.png
│ └── example_03.png
├── face_detector
│ ├── deploy.prototxt
│ └── res10_300x300_ssd_iter_140000.caffemodel
├── detect_mask_image.py
├── detect_mask_video.py
├── mask_detector.model
├── plot.png
└── train_mask_detector.py
```

Приклад 4.1 – Дерево зображень

4.2 Реалізація навчального скрипта системи виявлення маски.

Тепер, коли ми переглянули наш набір даних масок для обличчя, давайте дізнаємося, як ми можемо використовувати Keras та TensorFlow для

підготовки класифікатора для автоматичного виявлення, чи носить людина маску чи ні. Для виконання цього завдання ми допрацюємо архітектуру MobileNet V2, високоефективну архітектуру, яку можна застосувати до вбудованих пристроїв з обмеженою обчислювальною ємністю (наприклад, Raspberry Pi, Google Coral, NVIDIA Jetson Nano тощо).

Розгортання нашого системи виявлення маски для обличчя на вбудованих пристроях може зменшити витрати на виготовлення таких систем виявлення масок для обличчя, отже, чому ми вирішили використовувати цю архітектуру.

```
train_mask_detector.py
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os
```

Приклад 4.2 – Набір імпорту tensorflow.keras

Набір імпорту tensorflow.keras має багато функцій:

- збільшення даних;
- завантаження класифікатора MobilNetV2;
- створення нової повністю з'єднаної (FC) голови;
- попередня обробка;
- завантаження даних зображення.

Ми будемо використовувати scikit–learn для бінаризації міток класів, сегментування нашого набору даних та друку звіту про класифікацію.

Реалізація шляхів допоможе нам знаходити та перераховувати зображення в наборі даних. І ми використаємо matplotlib для побудови наших навчальних кривих.

Давайте розберемо кілька аргументів командного рядка, необхідних для запуску нашого скрипта з терміналу (приклад 4.2):

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
default="mask_detector.model",
help="path to output face mask detector model")
args = vars(ap.parse_args())
```

Приклад 4.2 – Запуск скрипта з терміналу

Наші аргументи командного рядка включають:

- dataset: шлях до вхідного набору даних граней та граней з масками;

- plot: шлях до вихідного сюжету історії навчання, який буде сформовано за допомогою matplotlib;

- model: шлях до отриманої серіалізованої моделі класифікації масок для обличчя.

Визначемо свої гіперпараметри глибокого навчання в одному місці (приклад 4.3):

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
```

Приклад 4.3 – Гіперпараметри глибокого навчання

Тут вказані константи гіперпараметрів, включаючи початкову швидкість навчання, кількість навчальних епох та розмір партії. Пізніше ми застосуємо графік занепаду швидкості навчання, саме тому ми назвали змінну швидкості навчання INIT_LR.

На даний момент ми готові завантажити та попередньо обробити наші навчальні дані (приклад 4.4).

```
# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
# extract the class label from the filename
label = imagePath.split(os.path.sep)[-2]
# load the input image (224x224) and preprocess it
image = load_img(imagePath, target_size=(224, 224))
image = img_to_array(image)
image = preprocess_input(image)
# update the data and labels lists, respectively
data.append(image)
labels.append(label)
# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
```

Приклад 4.4 – Обробка навчальних данних

У цьому блоці виконаемо:

- захоплення всіх imagePaths у наборі даних;
- ініціалізацію списків даних та міток;
- зациклювання шляхів зображення та завантаження та попередня обробка зображень.

Етапи попередньої обробки включають зменшення розміру до 224 × 224 пікселів, перетворення у формат масиву та масштабування інтенсивності пікселів у вхідному зображенні до діапазону [–1, 1] (за допомогою функції зручності preprocess_input):

- додавання попередньо обробленого зображення та відповідного ярлика до списків даних та ярликів відповідно;

- перевірка, що наші навчальні дані мають формат масиву NumPy.

Вищезазначені рядки коду передбачають, що весь ваш набір даних досить малий, щоб вміститися в пам'яті. Якщо ваш набір даних перевищує доступну пам'ять треба використовувати HDF5.

Далі ми закодуємо наші мітки, розділимо набір даних і підготуємось до збільшення даних (приклад 4.5).

```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.20, stratify=labels, random_state=42)
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")
```

Приклад 4.5 – Розділимо набір даних

Одноразово кодуемо мітки наших класів, що означають, що наші дані будуть у такому форматі (приклад 4.6).

```
$ python train_mask_detector.py --dataset dataset
[INFO] loading images...
-> (trainX, testX, trainY, testY) = train_test_split(data, labels,
(Pdb) labels[500:]
array([[1., 0.],
[1., 0.],
[1., 0.],
...,
[0., 1.],
[0., 1.],
[0., 1.]], dtype=float32)
(Pdb)
```

Приклад 4.6 – Кодування міток

Кожен елемент нашого масиву складається з масиву, в якому лише один індекс є «гарячим» (тобто 1).

Використовуючи зручний метод scikit–learn, сегментуемо наші дані на 80% тренувань, а решта 20% на тестування.

Під час навчання ми будемо застосовувати мутації на льоту на наших знімках, намагаючись покращити узагальнення. Це називається збільшенням даних, де параметри випадкового обертання, масштабування, зсуву та перекидання. Ми використовуватимемо об’єкт aug під час навчання.

Але спочатку нам потрібно підготувати MobileNetV2 до тонкої настройки (приклад 4.7).

```
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
layer.trainable = False
```

Приклад 4.7 – MobileNetV2

Точне налаштування – це триетапний процес:

- завантажемо MobileNet з попередньо навченими ImageNet, відключаючи голову мережі;

- сконструемо нову головну модель ФК та прикріпемо її до основи замість старої голови;

- заморозимо базові шари мережі. Вага цих базових шарів не будуть оновлюватися в процесі зворотного розмноження, тоді як ваги головного шару будуть налаштовані.

Точне налаштування – це стратегія, яку я майже завжди рекомендуе створити базову модель, заощадивши значний час (приклад 4.7).

Завдяки підготовленим даним та архітектурі моделей для точної настройки, ми готові до компіляції та навчання нашої мережі система виявленняів масок для обличчя.

```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])
# train the head of the network
print("[INFO] training head...")
H = model.fit(
aug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)
```

Приклад 4.7 – Точне налаштування

Компілюемо нашу модель за допомогою оптимізатора, графіка спаду темпів навчання та двійкової крос–ентропії. Якщо ви будуєте на основі цього навчального сценарію більше 2 класів, використовуйте категоричну перехресну ентропію.

Тренування маски для обличчя розпочато. Наш об’єкт збільшення даних (aug) надаватиме пакети даних із мутованими зображеннями.

Після завершення навчання ми оцінимо отриману модель на наборі тестів (приклад 4.8).

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)
```

```
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")
```

Приклад 4.8 – Об’єкт збільшення даних

Робимо прогнози на тестовому наборі, захоплюючи індекси найвищих класів ймовірності. Потім виводимо звіт про класифікацію в терміналі для перевірки.

Серіалізуємо модель класифікації масок для обличчя на диск.

Останній крок – побудувати наші криві точності та втрат.

```
# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Приклад 4.9 – Криві точності та втрат

Після того, як наш графік буде готовий, зберігаємо рисунок на диск, використовуючи шлях до файла - plot.

4.3 Тренування системи виявлення маски для обличчя за допомогою Keras / TensorFlow

Тепер ми готові тренувати наш система виявлення маски для обличчя за допомогою Keras, TensorFlow та Deep Learning.

У терміналі виконаемо таку команду (приклад 4.9).

```
$ python train_mask_detector.py --dataset dataset
[INFO] loading images...
[INFO] compiling model...
[INFO] training head...
Train for 34 steps, validate on 276 samples
Epoch 1/20
34/34 [==============================] - 30s 885ms/step - loss: 0.6431 - accuracy: 0.6676 - val_loss: 0.3696 - val_accuracy: 0.8242
Epoch 2/20
34/34 [==============================] - 29s 853ms/step - loss: 0.3507 - accuracy: 0.8567 - val_loss: 0.1964 - val_accuracy: 0.9375
Epoch 3/20
34/34 [==============================] - 27s 800ms/step - loss: 0.2792 - accuracy: 0.8820 - val_loss: 0.1383 - val_accuracy: 0.9531
Epoch 4/20
34/34 [==============================] - 28s 814ms/step - loss: 0.2196 - accuracy: 0.9148 - val_loss: 0.1306 - val_accuracy: 0.9492
Epoch 5/20
34/34 [==============================] - 27s 792ms/step - loss: 0.2006 - accuracy: 0.9213 - val_loss: 0.0863 - val_accuracy: 0.9688
...
Epoch 16/20
34/34 [==============================] - 27s 801ms/step - loss: 0.0767 - accuracy: 0.9766 - val_loss: 0.0291 - val_accuracy: 0.9922
Epoch 17/20
34/34 [==============================] - 27s 795ms/step - loss: 0.1042 - accuracy: 0.9616 - val_loss: 0.0243 - val_accuracy: 1.0000
Epoch 18/20
34/34 [==============================] - 27s 796ms/step - loss: 0.0804 - accuracy: 0.9672 - val_loss: 0.0244 - val_accuracy: 0.9961
Epoch 19/20
34/34 [==============================] - 27s 793ms/step - loss: 0.0836 - accuracy: 0.9710 - val_loss: 0.0440 - val_accuracy: 0.9883
Epoch 20/20
34/34 [==============================] - 28s 838ms/step - loss: 0.0717 - accuracy: 0.9710 - val_loss: 0.0270 - val_accuracy: 0.9922
[INFO] evaluating network...
precision recall f1-score support
with_mask 0.99 1.00 0.99 138
without_mask 1.00 0.99 0.99 138
accuracy 0.99 276
macro avg 0.99 0.99 0.99 276
weighted avg 0.99 0.99 0.99 276
```

Приклад 4.9 – Тренування системи виявлення маски

Криві точності / втрати тренажера системи виявлення маски для обличчя демонструють високу точність та незначні ознаки переобладнання даних (рисунок 4.1).

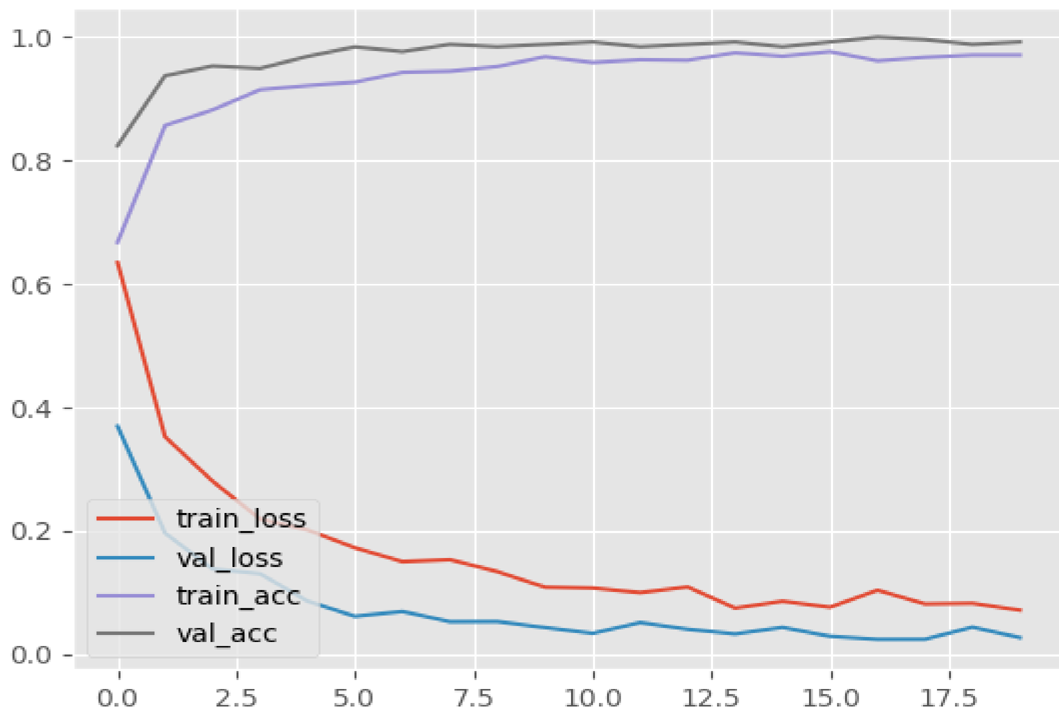

Рисунок 4.1 – Криві точності

Тепер ми готові застосувати наші знання з комп'ютерного зору та глибокого навчання за допомогою Python, OpenCV та TensorFlow / Keras для виявлення маски обличчя. Ми отримуємо ~ 99% точності на нашому тестовому наборі. Дивлячись на рисунок 4.1, ми можемо побачити, що є невеликі ознаки переобладнання, при цьому втрати від перевірки нижчі, ніж втрати під час навчання. Враховуючи ці результати, ми сподіваємось, що наша модель буде добре узагальнена на зображення поза нашим навчальним та тестовим набором.

4.4 Впровадження нашої системи виявлення маски для обличчя для зображень з OpenCV

Тепер, коли наш система виявлення маски для обличчя навчений, ми можемо:

- завантажити вхідне зображення з диска;
- виявити обличчя на зображенні;

- застосувати наш система виявлення маски для обличчя, щоб класифікувати обличчя як with_mask, так і without_mask.

Відкриемо файл detect_mask_image.py у структурі каталогів (приклад 4.10).

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import cv2
import os
```

Приклад 4.10 – Імпорти

Наш скрипт драйвера вимагає трьох імпортів TensorFlow / Keras для завантаження нашої моделі MaskNet та попередньої обробки зображення.

OpenCV необхідний для відображення та обробки зображень.

Наступним кроком є аналіз аргументів (приклад 4.11):

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
help="path to input image")
ap.add_argument("-f", "--face", type=str,
default="face_detector",
help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
default="mask_detector.model",
help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Приклад 4.11 – Аналіз аргументів командного рядка

Наші чотири аргументи командного рядка включають:

- image: Шлях до вхідного зображення, що містить грані для умови воду;

- face: Шлях до каталогу моделі системи виявлення облич (нам потрібно локалізувати обличчя перед їх класифікацією);

- model: Шлях до моделі системи виявлення маски для обличчя, яку ми навчили раніше;

- впевненість: Необов'язковий поріг ймовірності може бути встановлений для перевизначення 50% для фільтрації слабких виявлення обличчя.

Далі ми завантажимо наші моделі системи виявлення обличчя та класифікатора масок (приклад 4.12).

```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
"res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNet(prototxtPath, weightsPath)
# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
model = load_model(args["model"])
```

Приклад 4.12 – Моделі системи виявлення

Завдяки нашим моделям глибокого навчання, які вже є в пам'яті, наступним кроком є завантаження та попередня обробка вхідного зображення (приклад 4.13).

```
# load the input image from disk, clone it, and grab the image spatial
# dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(h, w) = image.shape[:2]
# construct a blob from the image
blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
(104.0, 177.0, 123.0))
# pass the blob through the network and obtain the face detections
print("[INFO] computing face detections...")
net.setInput(blob)
detections = net.forward()
```

Приклад 4.13 – Завантаження та попередня обробка вхідного зображення

Завантажуючи наше – зображення з диска, ми робимо копію та захоплюємо розміри кадру для подальшого масштабування та відображення. Попередня обробка виконується функцією blobFromImage OpenCV.

Як показано в параметрах, ми зменшуємо розмір до 300 × 300 пікселів і виконуємо середнє віднімання.

Потім виконуемо виявлення обличчя, щоб локалізувати, де на зображенні знаходяться всі обличчя. Після того, як ми дізнаємось, де передбачається перебування кожного обличчя, ми переконаємось, що вони відповідають пороговому значенню confidence, перш ніж витягувати ROI:

```
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > args["confidence"]:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
# ensure the bounding boxes fall within the dimensions of
# the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Приклад 4.14 – Виявлення обличчя

Тут ми перебираємо наші виявлення та отримуємо впевненість, щоб виміряти поріг -confidence. Потім ми обчислюємо значення обмежувального поля для певної грані та переконуємось, що поле потрапляє в межі зображення. Далі ми проведемо рентабельність інвестицій за допомогою нашої моделі MaskNet:

```
# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
face = image[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
face = np.expand_dims(face, axis=0)
# pass the face through the model to determine if the face
# has a mask or not
(mask, withoutMask) = model.predict(face)[0]
```

Приклад 4.15 – Обчислюємо значення обмежувального поля

У цьому блоці ми:

- витягнемо ROI обличчя за допомогою NumPy;
- попередньо обробіть ROI так само, як ми це робили під час навчання;
- виконаемо виявлення маски, щоб передбачити with_mask або without_mask.

Звідси ми будемо анотувати та відображати результат (приклад 4.16).

```
# determine the class label and color we'll use to draw
# the bounding box and text
label = "Mask" if mask > withoutMask else "No Mask"
color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
# include the probability in the label
label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
# display the label and bounding box rectangle on the output
# frame
cv2.putText(image, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)


# show the output image
cv2.imshow("Output", image)
cv2.waitKey(0)
```

Приклад 4.16 – Відображення результату

Спочатку ми визначаємо мітку класу на основі ймовірностей, які повертає модель системи виявлення маски, і призначаємо пов'язаний колір для анотації. Колір буде «зеленим» для with_mask та «червоним» для without_mask.

Потім ми малюємо текст мітки, а також обмежувальний прямокутник для обличчя, використовуючи функції малювання OpenCV. Після того, як всі виявлення будуть оброблені, буде відображено вихідне зображення.

4.5 Виявлення маски для обличчя на зображеннях з OpenCV

Давайте застосуємо наш система виявлення маски для обличчя. У терміналі виконаємо таку команду:

```
$ python detect_mask_image.py --image examples/example_01.png
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] computing face detections...
```

Приклад 4.17 – Застосуємо система виявлення маски

Чи носить цей чоловік публічно маску для обличчя? Так, він є, і наш комп’ютерний зір та метод глибокого навчання за допомогою Python, OpenCV та TensorFlow / Keras дозволили автоматично виявити наявність маски (рисунок 4.2).

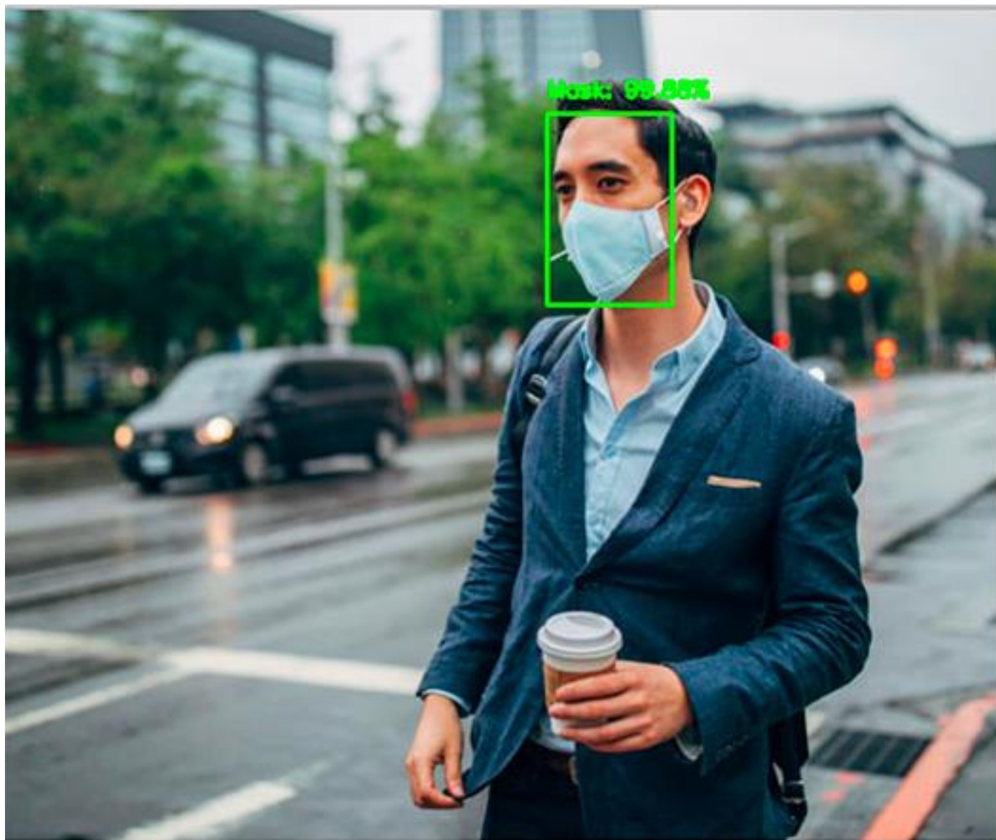

Рисунок 4.2 – Чоловік у масці

Як бачите, наш система виявлення маски для обличчя правильно позначив це зображення як маска.

Давайте спробуємо інше зображення, це зображення людини, яка не носить маску для обличчя (приклад 4.18).

```
$ python detect_mask_image.py --image examples/example_02.png
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] computing face detections...
```

Приклад 4.18 – Спробуємо інше зображення

На цьому знімку (рисунок 4.3) людина не має маски для обличчя. Використовуючи Python, OpenCV та TensorFlow / Keras, наша система правильно виявила “no mask” для обличчя.

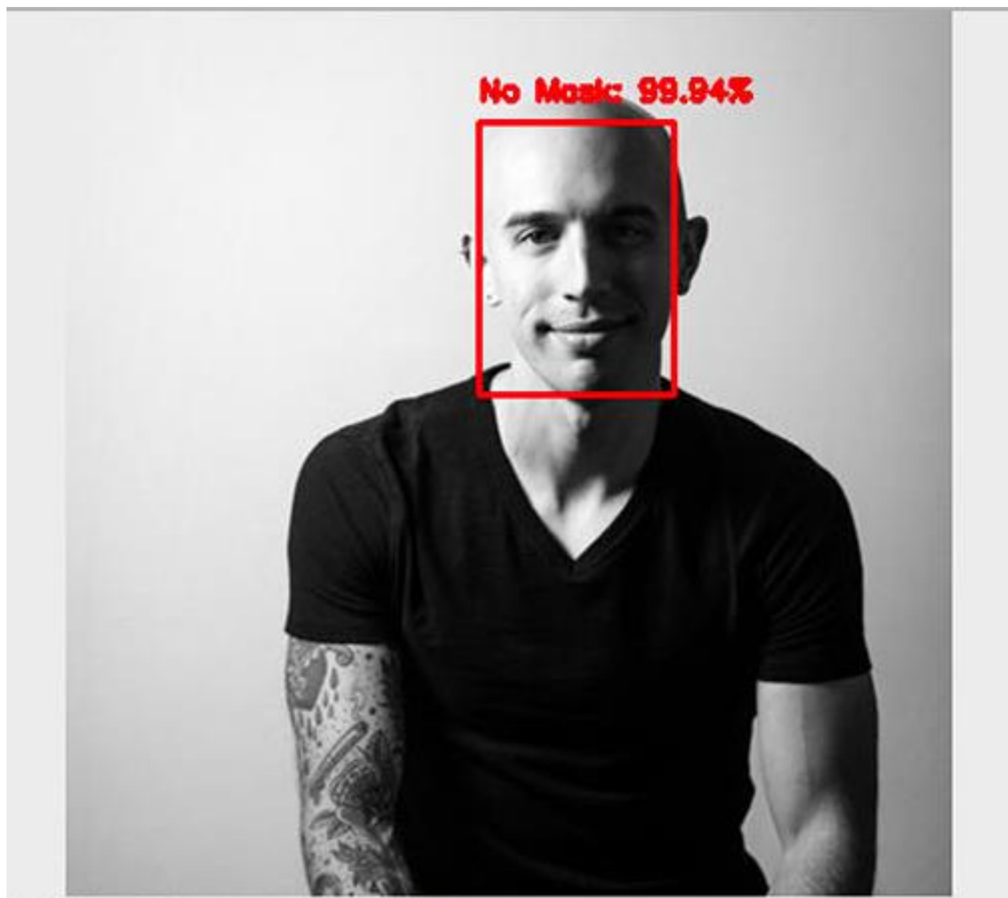

Рисунок 4.3 – Чоловік без маски

Наш система виявлення маски для обличчя правильно передбачив відсутність маски.

Давайте спробуємо одне остаточне зображення (приклад 4.19).

```
$ python detect_mask_image.py --image examples/example_03.png
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] computing face detections...
```

Приклад 4.19 – Спробуємо остаточне зображення

Що тут сталося? Чому ми змогли розпізнати обличчя двох панів у фоновому режимі та правильно класифікували маску / маску для них, але ми не змогли виявити жінку на передньому плані (рисунок 4.4)?

Рисунок 4.4 – Некоректна робота системи виявлення

Що відбувається в цьому результаті? Чому даму на передньому плані не виявляють, що вона носить маску для обличчя? Чи не вдався нам нашсистема виявлення маски для обличчя, побудований за допомогою комп'ютерного зору та глибокого навчання за допомогою Python, OpenCV та TensorFlow / Keras?

Майте на увазі, що для того, щоб класифікувати, чи одягнена людина в маску, спочатку потрібно виконати виявлення обличчя – якщо обличчя не

знайдено (що трапилось на цьому зображенні), тоді система виявлення маски застосовувати не можна!

Причина, по якій ми не можемо розпізнати обличчя на передньому плані, полягає в тому, що:

- це занадто затемнено маскою;
- набір даних, що використовується для тренування системи виявлення обличчя, не містив прикладів зображень людей, які носять маски для обличчя;
- тому, якщо велика частина обличчя закрита, наш система виявлення обличчя, швидше за все, не зможе виявити обличчя.

4.6 Впровадження нашої системи виявлення маски для обличчя у потоках відео в реальному часі з OpenCV

На даний момент ми знаємо, що можемо застосовувати розпізнавання маски обличчя до статичних зображень – але як щодо потоків відео в реальному часі?

Відкрийте файл detect_mask_video.py у вашій структурі каталогів:

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2
import os
```

Приклад 4.20 – detect_mask_video.py

Алгоритм цього сценарію однаковий, але він складений таким чином, щоб забезпечити обробку кожного кадру потоку веб-камери.

Таким чином, єдина різниця, що стосується імпорту, полягає в тому, що нам потрібні клас і час VideoStream. І те, і інше допоможе нам працювати з потоком. Ми також скористаємось перевагами imutils за відомими аспектами спосіб зміни розміру.

Наша логіка розпізнавання обличчя / передбачення маски для цього сценарію знаходиться у функції detect_and_predict_mask:

```
def detect_and_predict_mask(frame, faceNet, maskNet):
# grab the dimensions of the frame and then construct a blob
# from it
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
(104.0, 177.0, 123.0))
# pass the blob through the network and obtain the face detections
faceNet.setInput(blob)
detections = faceNet.forward()
# initialize our list of faces, their corresponding locations,
# and the list of predictions from our face mask network
faces = []
locs = []
preds = []
```

Приклад 4.21 – detect_and_predict_mask

Визначивши тут цю зручну функцію, наш цикл обробки кадру буде трохи легшим для читання пізніше. Ця функція визначає обличчя, а потім застосовує наш класифікатор маски до кожного рентабельності інвестицій обличчя. Така функція консолідує наш код – її навіть можна перенести в окремий файл Python, якщо ви так вирішите.

Наша функція detect_and_predict_mask приймає три параметри:

- frame: кадр з нашого потоку;
- faceNet: модель, яка використовується для виявлення місця на зображенні;
- maskNet: наша модель класифікатора маски для обличчя.

Всередині ми створюємо крапку, виявляємо обличчя та ініціалізуємо списки, два з яких функція повертається. Ці списки включають наші обличчя (тобто ROI), locs (розташування облич) та preds (список масок / відсутність прогнозів маски).

Звідси ми перейдемо до виявлення обличчя (приклад 4.22).

```
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > args["confidence"]:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
# ensure the bounding boxes fall within the dimensions of
# the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Приклад 4.22 – Виявлення обличчя

Усередині петлі ми фільтруємо слабкі виявлення і витягуємо обмежувальні рамки, забезпечуючи при цьому, щоб координати обмежувальної рамки не виходили за межі зображення (приклад 4.23).

Далі ми додамо ROI в два відповідні списки.

```
# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
face = frame[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
# add the face and bounding boxes to their respective
# lists
faces.append(face)
locs.append((startX, startY, endX, endY))
```

Приклад 4.23 – Додамо ROI в два відповідні

Після вилучення ROI на обличчя та попередньої обробки ми додаємо рентабельність інвестицій на обличчя та обмежувальні поля до відповідних списків. (приклад 4.24)

Тепер ми готові пропустити наші обличчя через наш прогноз маски.

```
# only make a predictions if at least one face was detected
if len(faces) > 0:
# for faster inference we'll make batch predictions on *all*
# faces at the same time rather than one-by-one predictions
# in the above `for` loop
faces = np.array(faces, dtype="float32")
preds = maskNet.predict(faces, batch_size=32)
# return a 2-tuple of the face locations and their corresponding
# locations
return (locs, preds)
```

Приклад 4.24 – Прогноз маски

Логіка тут побудована на швидкості. Спочатку ми переконуємось, що принаймні одне обличчя було виявлено – якщо ні, ми повернемо порожні преси.

Поєднання цих двох змін тепер виправляє помилку, яка перешкоджала поверненню кількох preds з виводу. Завдяки виправленню, кілька граней на одному зображенні правильно розпізнаються як такі, що мають маску, або не мають маски. По-друге, ми робимо умовивід для всієї нашої партії граней у кадрі, щоб наш конвеєр був швидшим. Немає сенсу писати інший цикл, щоб робити прогнози на кожному обличчі окремо через накладні витрати (особливо якщо ви використовуєте графічний процесор, який вимагає багато накладних зв’язків на системній шині). Більш ефективно виконувати прогнозування в пакетному режимі. Повертає місце розташування вікон, що обмежують обличчя, та відповідні прогнози маски / не маски абоненту.

Далі ми визначимо наші аргументи командного рядка (приклад 4.25).

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
default="face_detector",
help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
default="mask_detector.model",
help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Приклад 4.25 – Аргументи командного рядка

Наші аргументи командного рядка включають:

- face: шлях до каталогу системи виявлення облич
- model: шлях до нашого навченого класифікатора масок для обличчя
- впевненість: мінімальний поріг ймовірності для фільтрації виявлення слабких граней.

З нашими імпортами, функцією зручності та аргументами командного рядка готовими до роботи, нам залишається лише декілька ініціалізацій, які потрібно обробити, перш ніж перебирати кадри.

```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
"res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])
# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

Приклад 4.26 – Ініціалізація

Тут ми ініціалізували наш:

- система виявлення обличчя;
- система виявлення захисної маски для обличчя;
- відеопотік веб-камери.

Давайте продовжимо циклічний перегляд кадрів у потоці.

```
# loop over the frames from the video stream
while True:
# grab the frame from the threaded video stream and resize it
# to have a maximum width of 400 pixels
frame = vs.read()
frame = imutils.resize(frame, width=400)
# detect faces in the frame and determine if they are wearing a
# face mask or not
(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
```

Приклад 4.27 – Циклічний перегляд кадрів

Ми починаємо циклічний перегляд кадрів. Усередині ми беремо кадр із потоку та змінюємо його розмір.

Звідти ми використовуємо нашу зручну утиліту, визначаємо та передбачає, чи носять люди свої маски чи ні.

Давайте оброблятимемо результати виявлення маски для обличчя.

```
# loop over the detected face locations and their corresponding
# locations
for (box, pred) in zip(locs, preds):
# unpack the bounding box and predictions
(startX, startY, endX, endY) = box
(mask, withoutMask) = pred
# determine the class label and color we'll use to draw
# the bounding box and text
label = "Mask" if mask > withoutMask else "No Mask"
color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
# include the probability in the label
label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
# display the label and bounding box rectangle on the output
# frame
cv2.putText(frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
```

Приклад 4.28 – Результати виявлення маски для обличчя

Усередині нашого циклу над результатами прогнозування ми:

- розпакуйте рамку для обмеження обличчя mask/not mask;
- визначимо label та color;
- анотуйте ярлик та рамку для обмеження граней;

Нарешті, ми відображаємо результати та проводимо очищення.

```
# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
# if the `q` key was pressed, break from the loop
if key == ord("q"):
break
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Приклад 4.29 – Фіксуємо натискання клавіш

Після відображення кадру ми фіксуємо натискання клавіш. Якщо користувач натискає q (вийти), ми вириваємося з циклу. Чудова робота із впровадженням системи виявлення маски для обличчя в режимі реального часу за допомогою Python, OpenCV та глибокого навчання за допомогою TensorFlow / Keras!

4.7 Виявлення масок для обличчя за допомогою OpenCV у режимі реального часу.

Щоб побачити наш система виявлення маски для обличчя у реальному часі в дії, для завантаження вихідного коду та попередньо навченої моделі системи виявлення маски для обличчя.

Запуститимо система виявлення маски у потоках відео в режимі реального часу, використовуючи таку команду (приклад 4.30).

```
$ python detect_mask_video.py
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] starting video stream...
```

Приклад 4.30 – Система виявлення маски у потоках відео в режимі реального часу

Пропозиції щодо вдосконалення

Як ви можете бачити з розділів результатів вище, наш система виявлення маски для обличчя працює досить добре, незважаючи на:

- маючи обмежені дані про навчання;
- клас with_mask, який створюється штучно;
- для подальшого вдосконалення нашої моделі виявлення маски для обличчя слід збирати фактичні зображення людей, які носять маски.

Незважаючи на те, що наш штучний набір даних у цьому випадку добре працював, реальної речі не можна замінити.

По-друге, також слід зібрати зображення облич, які можуть "заплутати" наш класифікатор, думаючи, що людина носить маску, а насправді вона не є – потенційні приклади включають сорочки, обмотані навколо обличчя, бандану над ротом тощо.

Все це приклади того, що наш система виявлення маски може сплутати як маску для обличчя.

Нарешті, вам слід подумати про навчання спеціального двокласного системи виявлення об’єктів, а не простого класифікатора зображень.

Наш сучасний метод виявлення, чи носить людина маску чи ні, є двоступеневим процесом:

- виконуемо виявлення обличчя;
- наносимо наш система виявлення маски на кожне обличчя.

Проблема такого підходу полягає в тому, що маска для обличчя за визначенням затемнює частину обличчя. Якщо достатньо обличчя закрито, його неможливо виявити, а отже, система виявлення маски для обличчя застосовуватись не буде.

Щоб обійти цю проблему, слід навчити двокласовий система виявлення об’єктів, який складається з класу with_mask та without_mask.

Поєднання системи виявлення об’єктів із виділеним класом with_mask дозволить вдосконалити модель у двох відношеннях.

По-перше, система виявлення об’єктів зможе природним чином виявляти людей у масках, які в іншому випадку виявити обличчя система виявленняом було б неможливо через занадто велику кількість обличчя, яке було затемнено.

По-друге, цей підхід зводить наш трубопровід комп'ютерного зору до одного кроку – замість того, щоб застосовувати розпізнавання обличчя, а потім нашу модель системи виявлення маски обличчя, все, що нам потрібно зробити, це застосувати система виявлення об’єктів, щоб дати нам обмежувальні рамки для людей як with_mask, так і without_mask в один прямий прохід мережі.

Такий метод не тільки більш обчислювально ефективний, але і більш та наскрізний.

Ми створили система виявлення маски для обличчя за допомогою OpenCV, Keras / TensorFlow та Deep Learning.

Для створення нашого системи виявлення маски для обличчя ми навчили двокласну модель людей, які носять маски та людей, які не носять масок.

Ми точно налаштували MobileNetV2 на наборі даних mask/no mask та отримали класифікатор, який є точним на ~ 99%.

Потім ми взяли цей класифікатор маски для обличчя та застосували його як до зображень, так і до відеопотоків у реальному часі, виконавши:

- виявлення облич на зображеннях / відео;
- витягування кожного окремого обличчя;
- застосовуючи наш класифікатор масок для обличчя;

Наш система виявлення маски для обличчя є точним, і оскільки ми використовували архітектуру MobileNetV2, він також ефективно обчислювальний, що спрощує розгортання моделі на вбудованих системах (Raspberry Pi, Google Coral, Jetosn, Nano тощо).

У цьому розділі ми успішно розробили та протестували нашу систему виявлення захисних масок на обличчі людини.

ВИСНОВКИ

В кінці цієї роботи ми побудували Convolutional Neural Network (CNN) модель з використанням TensorFlow з Keras і OpenCV для виявлення, захисної маски на обличчі, щоб захистити себе та оточуючих від COVID-19 та інших респіраторних захворювань. Це може використовуватися в багатьох додатках. В найближчому майбутньому, з огляду на кризу COVID-19 цей метод визначення того, чи носить людина маску для обличчя, може стати в нагоді.

Ця тема є дуже актуальною в даний період, обов’язкові правила маски для обличчя стають все більш поширеними у загальнодоступних закладах світу. Зростають наукові докази, що підтверджують ефективність носіння масок для обличчя щодо зменшення поширення вірусу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Жерон, Орельен. Ж61 Прикладное машинное обучение с помощью Scikit–Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. Пер. с англ. – СпБ.: ООО "Альфа–книга': 2018. – 688 с.: ил. – Парал. тит. англ. ISBN 978–5–9500296–2–2
2. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. – СПб.: Питер, 2019. – 368 с.: ил. – (Серия «Бестселлеры O’Reilly»). ISBN 978–5–4461–1153–4
3. Шолле Франсуа Ш78 Глубокое обучение на Python. – СПб.: Питер, 2018. – 400 с.: ил. – (Серия «Библиотека программиста»). ISBN 978-5-4461-0770-4
4. Рашид Т. Создаем нейронную сеть –СПб.: Альфа-книга, 2017. – 274 с. – ISBN 9785990944572.
5. Christopher M. Bishop Pattern Recognition and Machine Learning. – Cambridge CB3 0FB, U.K ISBN 978-0387-31073-2
6. Andrew W. Trask. grokking Deep Learning – Manning Publications, ISBN 9781617293702
7. Даценко Д. Г., Система виявлення захисної маски для обличчя – Восьма міжнародна науково-технічна конференція “Проблеми інформатизації”, 2020.
8. Vahrenkamp R, Mattfeld D (2007) Logistiknetzwerke. Gabler, Wiesbaden
9. Fleischmann B, Gnutzmann S, Sandvoß E (2004) Dynamic vehicle routing based on online traffic information. Transp Sci 38(4):420–433
10. Jaillet P, Wagner MR (2006) Online routing problems: value of advanced information as improved competitive ratios. Transp Sci 40(2):200–210
11. Hiller B, Krumke SO, Rambau J (2006) Reoptimization gaps versus

model errors in online-dispatching of service units for adac. Discrete Appl Math 154:1897–1907

12. Savelsbergh M, Sol M (1998) Drive: dynamic routing of independent vehicles. Oper Res 46:474–490

13. Bengforth, Bilbro, Ojeda. Applied Text Analysis with Python. Enabling Language – Aware Data Products with Machine Learning. 2019. ISBN 978-5-4461-1153-4

14. Howse, OpenCV Computer Vision with Python, 2013 , ISBN 978-1-78216-382-3

15. Prateek Joshi., OpenCV with Python By Example, 2015, ISBN 9781785289873