КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

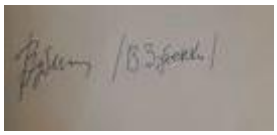
на тему:

РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ ОБЛИЧЧЯ

Виконав студент 4-го курсу

Микитчин Сергій Дмитрович

Науковий керівник:
доцент, кандидат фіз.-мат. наук



Зубенко Віталій Володимирович

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
« 01 » червня 2022 р.
Протокол № 10
Завідувач кафедри
Нікітченко М. С.

Київ – 2022

Реферат

Обсяг роботи: 69 сторінок, 35 ілюстрацій, 70 джерел посилань.

ВИЯВЛЕННЯ ОБЛИЧЧЯ, АЛГОРИТМ, БУСТІНГ, КАСКАДНА МОДЕЛЬ, ADABOOST, ОЗНАКИ ХААРА, МЕТОД ВІОЛИ-ДЖОНСА

Об’єктом роботи є програмне забеспечення для виявлення обличчя людини. Використано підхід глибинного навчання для візуального виявлення об’єктів, який характеризується високою швидкістю та показниками виявлення.

Предметом роботи є алгоритм виявлення обличчя з використанням метода Віоли-Джонса, як такий, що найбільш ефективний серед існуючих методів.

Метою роботи є розробка та реалізація системи виявлення обличчя найоптимальнішим методом, шляхом аналізу вже існуючих підходів для розв’язання даної задачі.

Методи розроблення та дослідження: використано алгоритмічний підхід до програмування системи виявлення обличчя, використані геометричні ознаки обличчя та нейронні мережі прямого поширення.

Інструменти розроблення: імперативна мова програмування Python.

Результати роботи: виконано загальний огляд проблеми виявлення обличчя, основних методів та алгоритмів для вирішення даної задачі, розроблено програмне забеспечення, яке можна застосовувати для задачі виявлення обличчя у багатьох галузях.

ЗМІСТ

ВСТУП

Задача виявлення обличчя — це проблема з розділу кібернетики, який називається розпізнавання образів. В цю проблему входить задача автоматичного знаходження облич на фотографіях та відеопотоках. Дана проблема стимулювала розвиток теорії виявлення об’єктів вцілому. Вже на ранніх стадіях розвитку комп'ютерного бачення ця проблема була дуже актуальною, тож багато дослідників намагалися розробити ефективний та швидкий алгоритм, який би відповідав всім необхідним потребам.

Виявлення обличчя розглядається як підзадача ідентифікації людини та набуває все більшої популярності в наукових колах. Описано та розроблено дуже багато як систем виявлення так і систем розпізнавання облич. Але не зважаючи на це, до сих пір наукові журнали публікують нові статті на цю тему.

Системи виявлення та ідентифікації облич надзвичайно актуальні та застосовуються в багатьох галузях таких як: цивільна, охоронна, військова, медична. В основі програмного забеспечення яке виконує потреби кожної окремої галузі стоїть алгоритм виявлення обличчя. Тому дуже важливо і необхідно обрати найефективніший алгоритм, який буде виконувати поставлену задачу. Для цього потрібно проаналізувати всі існуючи методи виявлення, порівняти їх та зробити висновок про те, який з них найкраще підходить для вирішення цієї проблеми.

В даній роботі після огляду всіх існуючих методів виявлення облич, буде обраний алгоритм Віоли-Джонса як такий, що показує найкращі результати. Цей алгоритм характеризується високою швидкістю та показниками виявлення, не зважаючи на те, що навчається він доволі довго. Система виявлення обличчя, яка буде реалізована в даній роботі буде мати вкрай низьку ймовірність помилкового виявлення. Такий результат буде досягатися саме тому, що був обраний алгоритм виявлення Віоли-Джонса.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ВИЯВЛЕННЯ

1.1 Методи виявлення облич

Методи виявлення в літературі важко строго класифікувати, оскільки більшість алгоритмів є комбінацією методів виявлення облич для підвищення точності. В основному, виявлення можна розділити на дві групи: методи на основі знань і методи на основі зображень. Методи виявлення наведені на рис. 1.

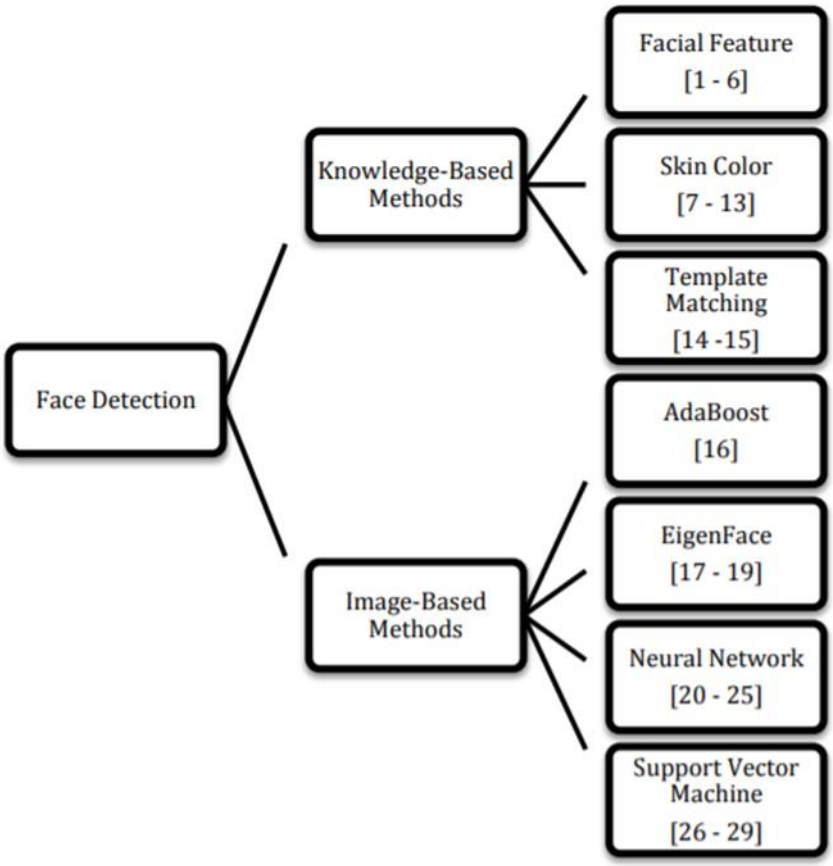


Рисунок 1. Методи виявлення обличчя

1.2 Методи виявлення засновані на знаннях

Методи, засновані на знаннях, використовують інформацію про риси обличчя, колір шкіри або відповідність шаблону. Риси обличчя використовуються, щоб знайти очі, рот, ніс або інші риси обличчя для виявлення людських облич. Колір шкіри відрізняється від інших кольорів і унікальний, і його характеристики не змінюються щодо зміни пози або оклюзії. Колір шкіри моделюється в кожному колірному просторі, як-от RGB, YCbCr, HSV, YUV, та в статистичних моделях. Обличчя мають унікальний патерн, щоб

відрізняти їх від інших об’єктів, і, отже, можна створити шаблон для сканування та виявлення таких облич.

Риси обличчя є важливою інформацією, і на основі цієї інформації можна створити стандартні зображення. У літературі є багато алгоритмів виявлення, заснованих на рисах обличчя [1 - 6].

Чжи-фан та ін. [1] виявляють риси обличчя шляхом виділення області, схожої на шкіру, потім в цій області в колірному просторі YCbCr виявляються краї. Потім в області де розташовані виявлені краї, знаходяться очі за допомогою методу головних компонент. Нарешті, рот знаходиться на основі геометричної інформації. Інший підхід виділяє область, схожу на шкіру, за нормалізованим колірним простором RGB, а обличчя перевіряється шляхом зіставлення шаблону. Щоб знайти очі, брови та рот, кольорові змійки наносяться на перевірене зображення обличчя [2]. Руан та Інь [3] сегментують ділянки шкіри в колірному просторі YCbCr, а обличчя перевіряються за допомогою методу опорних векторів. Для остаточної перевірки обличчя, очі та рот знаходяться з огляду на різницю Cb і Cr. Для області очей значення Cb більше, ніж значення Cr, а для області рота значення Cr більше, ніж значення Cb. Інший підхід використовує сегментацію шкірних областей з допомогою статистичної моделі. Статистична модель складається із значень кольору шкіри в каналах Cb і Cr у колірному просторі YCbCr. Потім кандидати на обличчя вибираються відповідно до прямокутного співвідношення сегментованої області. Нарешті, кандидати перевіряються за допомогою шаблона очей та рта [4]. Крім того, колірний простір RGB можна використовувати для сегментації шкірних областей, які потім витягуються, щоб бути кандидатами на обличчя. Кандидат перевіряється шляхом знаходження рис обличчя. Очі та рот можна знайти на основі властивості рівнобедреного трикутника. Два ока та рот утворюють рівнобедрений трикутник, також відстань між двома очима і відстань від середини очей до рота рівні. Після того, як очі та рот знайдені, нейронна мережа прямого поширення використовується для остаточної перевірки кандидата на обличчя [5]. Бебар та ін. [6] сегментують шкірну область за колірним простором YCbCr, а очі та рот знаходять як комбінацію сегментованого зображення та зображення з виявленими краями. Для остаточної перевірки використовуються горизонтальні та вертикальні профілі зображень для перевірки положення очей та рта. Усі методи використовують сегментацію шкіри для видалення об’єктів, які не є обличчями на зображеннях, це зменшує час обчислень.

1.3 Колір шкіри

Колір шкіри - одна з найважливіших рис людського обличчя. Колір шкіри можна моделювати параметризованими або непараметризованими методами. Область кольору шкіри може бути ідентифікована з точки зору порогової області, еліптичного моделювання, статистичного моделювання (тобто гауссового моделювання) або нейронної мережі. Колір шкіри описується у всіх колірних просторах, таких як RGB, YCbCr і HSV. RGB чутливий до змін світла, але YCbCr і HSV не чутливі до змін світла. Причина в тому, що ці два колірні простори мають окрему інтенсивність і колірний канал. У літературі є багато алгоритмів, заснованих на кольорі шкіри [7 - 13]. Керчауі та Уасін [7] моделювали колір шкіри, використовуючи Гауссівський розподіл з каналами Cb і Cr у колірному просторі YCbCr. Потім шкірна область вибирається як кандидат на обличчя з огляду на коефіцієнт обмежувальної рамки регіону, і кандидати перевіряються за допомогою відповідності шаблону. Інший метод на першому кроці попередньо обробляє дане зображення, щоб видалити фонову частину. Це робиться шляхом виявлення країв за компонентою Y колірного простору YCbCr. Потім закрита область заповнюється, щоб прийняти її як частину переднього плану. Після цього проводиться сегментація шкіри на колірному просторі YCrCb. Сегментовані частини приймаються як кандидати на обличчя, а перевірка здійснюється шляхом обчислення ентропії зображення кандидата, також використовується порогове значення для перевірки кандидата на обличчя [8]. Цян-жун і Хуа-лан [9] застосовували корекцію балансу білого перед виявленням облич. Значення кольору важливе для сегментації, тому під час отримання зображення кольори можуть відображати хибний колір. Щоб подолати це, спочатку слід виконати корекцію балансу білого. Потім кольори шкіри сегментують за допомогою еліптичної моделі в YCbCr. Після того, як ділянки шкіри знайдені, вони поєднуються з зображеннями на яких виявлені краї, щоб утворити зображення у відтінках сірого. Нарешті, об’єднані області перевіряються на наявність обличчя, перевіряється коефіцієнт обмежувальної рамки та площа всередині обмежувальної рамки. Інший підхід робить сегментацію шкірної області з пороговими значеннями в Cb, Cr та нормалізованими значеннями r і g. Потім кандидат на обличчя вибирається з огляду на коефіцієнт обмежувальної рамки, відношення площі всередині та площі обмежувальної рамки, а також мінімальної площі області. Після того, як кандидати знайдені, для пошуку кандидатів на обличчя застосовується метод AdaBoosting. Перевірка виконується шляхом об’єднання результатів з етапу сегментації шкірної області та AdaBoosting [10]. Також колір шкіри можна

моделювати в еліптичній області в каналах Cb і Cr колірного простору YCbCr. Шкірна область сегментується, якщо значення кольору знаходиться всередині еліптичної області, а регіони-кандидати перевіряються за допомогою відповідності шаблону [11]. Піе та ін. [12] виявляють обличчя, використовуючи лише сегментацію шкіри в колірному просторі YCbCr, також дослідники створюють колір шкіри в колірному просторі RGB. Інший підхід до моделювання кольору шкіри здійснюється за допомогою самоорганізаційній карті Кохонена. Після застосування сегментації шкіри кожен сегмент приймається як кандидат і перевіряється, чи може він помістится в еліптичну область чи ні [13].

1.4 Патерн людського обличчя

Іншою важливою інформацією про виявлення людського обличчя є патерн людського обличчя. Зіставлення шаблону можна застосувати до техніки скануючого вікна або сегментованої області. Техніка скануючого вікна застосовується до вікон невеликого розміру, наприклад, до вікон 20x20 або 30x30 пікселів. В цьому підході сканується все вхідне зображення, потім після певної ітерації це зображення зменшується, потім повторно сканується. Зменшення розміру важливо, щоб знайти обличчя великого або середнього розміру. Однак це вимагає додаткового обчислювального часу, щоб знайти розташування облич. Зіставлення шаблону в сегментованій області вимагає набагато менше обчислювального часу, ніж сканування, оскільки враховується лише збіг сегментованої частини. У літературі доступно багато застосувань із використанням відповідності шаблонів, наприклад [14], [15]. Чен та ін. [14], використовують шаблон половини обличчя замість шаблону всього обличчя. Такий підхід зменшує час обчислень. І це напівобличчя може бути адаптовано для ситуацій в яких відбувається зміна орієнтації обличчя. Інший підхід використовує абстрактні шаблони, які не схожі на зображення, але складаються з деяких параметрів (наприклад, розміру, форми, кольору та положення). Шкірна область сегментується в колірному просторі YCbCr. Потім до сегментованої області застосовуються абстрактні шаблони очей та пари очей. Перші шаблони визначають область розташування очей, а другі шаблони — розташування кожного ока. Другий шаблон також визначає орієнтацію очей.

1.5 Методи виявлення засновані на зображеннях

Методи, засновані на зображеннях, використовують методи навчання, що порівнюють зображення з обличчями та без них. Для цих методів береться велика кількість зображень обличчя і не обличчя, щоб підвищити точність

системи. Ці методи зазвичай використовують такі алгоритми як AdaBoost [16], EigenFace [17-19], нейронні мережі [20-25] та метод опорних векторів [26-29]. Зображення облич та не облич описуються в термінах вейвлет-функцій в методі AdaBoost. В методі EigenFace використовується метод головних компонент для створення вектора ознак обличчя та не обличчя. Також метод головних компонент використовується для стиснення заданого інформаційного вектора. В методі опорних векторів створюється ядрова функція для опису зображень обличчя та не обличчя. В нейронних мережах зображення облич та не облич класифікуються за структурою штучних нейронів.

AdaBoost — це алгоритм, який створює сильний класифікатор із слабких класифікаторів. Кандидати на обличчя знаходяться за допомогою алгоритму AdaBoost. Потім проводиться перевірка за допомогою каскадного класифікатора.

Як згадувалося вище, метод головних компонент використовується для генерування вектора ознак обличчя та не обличчя у методі EigenFace. Для цього можна використовувати техніку скануючого вікна та сегментацію. Застосування методу EigenFace для виявлення обличчя є в літературі [17 - 19]. Ван і Янг [17] застосовують зіставлення шаблонів, щоб знайти кандидатів на обличчя. Потім двовимірний метод головних компонент використовується для вилучення вектора ознак. Матриця зображення надається безпосередньо до двовимірного метода головних компонент замість вектора. Це зменшує час обчислень для розрахунку коваріаційної матриці. Після застосування метода головних компонент, класифікатор мінімальної відстані використовується для класифікації даних як для обличчя, так і для випадків без обличчя. Інший підхід використовує метод головних компонент та нейронну мережу. Метод головних компонент застосовується до зображення, щоб спочатку витягти кандидатів на обличчя. Потім зображення-кандидати класифікуються нейронною мережею, щоб виключити зображення без облич. Останні кандидати на обличчя перевіряються за геометричним розподілом країв у областях де ймовірно знаходяться обличчя [18]. Також метод головних компонент та AdaBoost використовуються з технікою скануючого вікна. Метод головних компонент генерує вектор ознак, який потім подається на вхід методу AdaBoost. Потім з тих векторів ознак, що подаються на вхід методу AdaBoost, генерується сильний класифікатор [19].

Нейронні мережі добре справляються з проблемою розпізнавання образів та класифікації, тоді як зображення обличчя та не обличчя можна розглядати як два окремих класи. Знову ж таки, для класифікації зображень можна

використовувати техніку скануючого вікна або сегментацію. Багато типів нейронних мереж використовуються для задач виявлення, а саме: Перцептрон, Самоорганізаційна карта Кохонена, Нейронна мережа зворотного поширення помилки, Нейронна мережа радіальних базисних функцій та Ймовірнісна нейронна мережа [20 - 25]. Аніжантіс та ін. [20] застосовують попередню обробку зображень, заповнюючи нейронну мережу, а потім використовують двошаровий персептрон, щоб визначити, чи є вхідне зображення обличчям чи не обличчям. Для виявлення облич використовується техніка скануючого вікна. Анагностопулос та ін. [21] сегментують шкірну область за нечіткою логікою в колірному просторі RGB, а сегментовані області відправляються в Ймовірнісну нейронну мережу для перевірки області на наявність облич. В іншому підході витягуються ділянки, подібні до шкіри, у колірному просторі YUV, а щоб зменшити час обчислень використовується техніка скануючого вікна. Потім сегментовані області надсилаються до Нейронної мережі зворотного поширення помилки, щоб згенерувати кандидатів на обличчя. Нарешті, кандидати на обличчя перевіряються за допомогою баєсового висновування [22]. Використовується сегментація шкірних ділянок в колірному просторі YCbCr. Потім вектор ознак генерується на сегментованій області за вейвлет-інваріантним моментом. Потім вектор ознак класифікується за допомогою самоорганізаційної карти Кохонена. Цей алгоритм може виявляти фронтальні та профільні обличчя на відео зі швидкістю до 20 кадрів в секунду [23]. Крім того, для сегментації шкірної області можна використовувати колірний простір HSV, а вектор ознак сегмента обчислюється за допомогою двовимірного дискретного косинусного перетворення. Нарешті, Нейронна мережа зворотного поширення помилки використовується для класифікації векторів ознак, які належать до шкірних областей [24]. З іншого боку, тільки використовуючи Нейронну мережу радіальних базисних функцій, можна виявляти обличчя за технікою скануючого вікна [25].

Останнім методом на основі зображень є Метод опорних векторів. Метод опорних векторів навчається із зображеннями обличчя та не обличчя, щоб побудувати ядрову функцію, яка класифікує зображення обличчя та не обличчя. Деякі реалізації методу опорних векторів опубліковані в літературі [26 - 29]. Інший підхід використовується для пошуку кандидатів на обличчя в [26]. Обличчя-кандидат знаходиться за узагальненою симетрією на основі розташування очей. Нарешті, кандидат на обличчя перевіряється та класифікується за допомогою метода опорних векторів. Також метод опорних векторів можна застосовувати для виявлення облич за допомогою техніки скануючого вікна [27]. Джі та ін. [28] застосовують сегментацію шкірних

областей в колірному просторі YCbCr, а око-кандидат знаходиться за інформацією про края всередині області. Потім очі перевіряються за допомогою метода опорних векторів. Після перевірки кандидат на обличчя витягується щодо положення очей. В якості остаточної перевірки кандидат на обличчя надсилається до методу опорних векторів для перевірки. Інший підхід до використання методу опорних векторів — це використання методу опорних векторів на основі одного класу. Замість того, щоб генерувати два класи: обличчя та не обличчя, генерується лише клас обличчя. Оскільки зображення не обличчя важко моделювати [29].

В кожному алгоритмі досліджується продуктивність виявлення обличчя, але вони використовують або власні бібліотеки зображень, або зображення з баз даних. Через це порівняння продуктивності між алгоритмами є складним, тому не можна визначити найкращий алгоритм.

1.6 Методи розпізнавання облич

Для системи розпізнавання облич іншою частиною є частина розпізнавання. Розпізнавання можна досягти за допомогою двовимірних або тривимірних даних зображень, вони обидва мають переваги і недоліки. Двовимірні дані зображень можна отримати легше і швидше ніж тривимірні. З іншого боку, двовимірні зображення чутливі до змін світла, а тривимірні зображення — ні. У тривимірних зображеннях поверхню обличчя можна легко моделювати, а двовимірні зображення не містять даних про глибину. Крім того, алгоритми розпізнавання обличчя виконуються над бібліотеками облич. Ці бібліотеки створені зі стандартними зображеннями облич, тому система розпізнавання обличчя повинна впоратися з цією проблемою. Методи розпізнавання обличчя наведені на рис. 2.

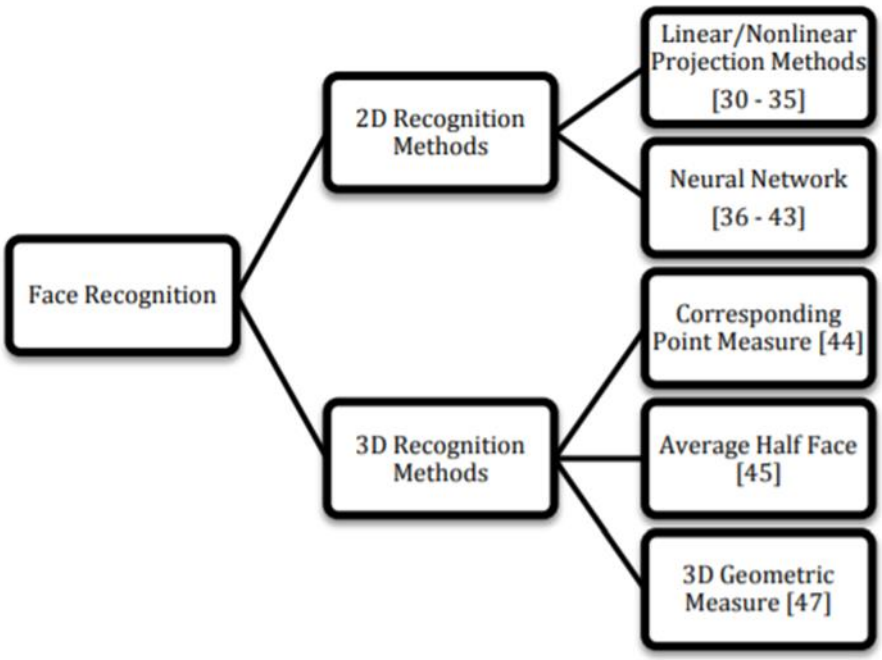


Рисунок 2. Методи розпізнавання облич

1.7 Розпізнавання облич

Розпізнавання облич – це проблема розпізнавання образів, тому для порівняння облич слід використовувати алгоритм навчання. Для розпізнавання у двовимірному просторі використовуються методи лінійної, нелінійної проекції та нейронні мережі. Методами лінійної, нелінійної проекції є Метод головних компонент, Лінійний дискримінантний аналіз, Вейвлет Габора та Аналіз спектральних ознак [30 - 35]. Крім того, для розпізнавання у двовимірному просторі використовуються такі типи нейронних мереж: Нейронна мережа прямого поширення, Нейронна мережа зворотного поширення помилки, Нейронна мережа радіальних базисних функцій, Вейвлет нейронна мережа та Нейронна мережа багатошарового кластера [36 - 43]. Для розпізнавання у тривимірному просторі використовуються такі техніки як: відповідна точкова міра, середня половина обличчя та тривимірна геометрична міра [44 - 47]. Методи двовимірної лінійної та нелінійної проекції генерують вектор ознак для кожної людини, а потім класифікують вхідну особу за базою даних. Генерування вектора ознак також має значення для зменшення розмірності вхідних зображень. Один з підходів застосовує покращення

зображення для придушення поганого освітлення перед процесом розпізнавання. Покращення зображення відомі як логарифмове перетворення та нормалізація. Потім вилучення ознак виконується за допомогою Вейвлета Габора. Нарешті, за допомогою метода Фішера, вхідне обличчя класифікується [30]. Сон та ін. [31] застосовують інший підхід до попередньої обробки та покращення зображення. Для попередньої обробки, перед вилученням озак, обчислюється різниця освітлення між правою та лівою частинами обличчя. Якщо різниця велика, то береться відображення середньої освітленої частини. Після попередньої обробки зображення, з обличчя витягуються ознаки за допомогою метода головних компонент. Класифікація вектора ознак виконується за Евклідовою відстанню. Інша реалізація використовує багатошаровий лінійний дискримінантний аналіз для класифікації облич, а перевага багатошарового лінійного дискримінантного аналізу полягає у використанні бази даних малого розміру вибірки [32]. Крім того, використання Аналіза спектральних ознак може зменшити розмір вибірки в базі даних [33]. Розширення вилучення ознак може підвищити продуктивність системи, наприклад, застосування вейвлет Габора, Метода головних компонент, а потім Аналіза незалежних компонент до зображення обличчя. Після виділення ознак, для розпізнавання використовується косинус подібності та правило класифікації найближчого сусіда [34]. Інший підхід до вхідного вектора використовує дистанцію до обличчя. Зменшення вимірного розміру виконується за допомогою метода головних компонент, а класифікація відбувається за допомогою нейронної мережі [35].

Розпізнавання облич є різновидом проблеми розпізнавання образів, нейронні мережі добре розпізнають образи та можуть бути здатні розрізняти обличчя навіть для великої кількості облич у базі даних. Обличчя можна надсилати до нейронної мережі як векторно-матричне зображення або зображення з витягнутими ознаками. Алгоритм із виділенням ознак може мати більшу продуктивність завдяки використанню вектора малої розмірності. У літературі використовуються такі типи нейронних мереж: Вейвлет нейронна мережа, Нейронна мережа радіальних базисних функцій, Нейронна мережа багатошарового кластера, Нейронна мережа прямого поширення та Нейронна мережа зворотного поширення помилки. Вейвлет нейронна мережа використовує передатну функцію, яка складається з ряду вейвлет-функцій, цей ряд дозволяє уникнути сліпоти в конструкції структури Нейронної мережі зворотного поширення помилки [36]. З іншого боку, замість використання зображень у відтінках сірого, кольорові зображення можна використовувати як вхідні дані для Нейронної мережі зворотного поширення помилки. R, G, B

канали є разом вхідними та мережевими каналами з інформацією про колір для легкого розрізнення [37]. Дослідники також працюють над впливом використання різних типів мереж та функцій витягнення. В якості мережі використовувалися Нейронні мережі зворотного поширення помилки, Нейронні мережі радіальних базисних функцій та Нейронні мережі багатошарового кластера, а як функції витягнення — Дискретне вейвлет-перетворення, Дискретне перетворення Радона, Дискретне косинусне перетворення та Метод головних компонент. Повідомляється, що найкраща продуктивність досягається при комбінації Нейронної мережі зворотного поширення помилки з Дискретним косинусним перетворенням [38]. Ріда і доктор БукеліфАуед [39] реалізують Нейронну мережу прямого поширення з мережею передатної функції Логарифм-Сигмоїд для класифікації облич. Швидкий Аналіз незалежних компонент використовується для витягнення ознак, а Нейронна мережа радіальних базисних функцій використовується як класифікатор [40], і тільки ця мережа використовується для класифікації вхідних зображень [41]. Крім того, Сингулярний розклад матриці використовується для витягнення ознак у Нейронній мережі зворотного поширення помилки. Інший підхід для покращення зображення — це використання фільтра Лапласа Гаусса, а потім застосування Сингулярного розкладу матриці до відфільтрованого зображення, щоб отримати вектор ознак. Нарешті, зображення обличчя класифікується за допомогою Нейронної мережі прямого поширення [43].

1.8 Тривимірні методи

Тривимірні методи розпізнавання облич використовують тривимірні дані обличчя, які складаються з хмари точок. Є реалізація, що використовує ітераційну найближчу точку для вирівнювання обличчя. Потім, у міру покращення зображення, шуми зменшуються, а спайки видаляються. Виявляється кінчик носа та обрізається сфера, де розташований кінчика носа. Ця сфера є рисою обличчя. Потім за допомогою вимірювання відповідного напрямку обличчя класифікується [44]. Деякі підходи використовують половину обличчя замість повного. Середня половина обличчя створюється за допомогою Сингулярного розкладу матриці, що зберігає симетрію. Потім виділення ознак виконується за допомогою Метода головних компонент, а класифікація відбувається за допомогою Метода найближчого сусіда [45]. Ельян і Угайл [46] використовували профілі обличчя як вхідні дані. Центральний профіль симетрії та профіль щоки витягуються для облич, а коефіцієнти Фур'є знаходяться для виділення ознак. Нарешті, для класифікації облич застосовується Коефіцієнт подібності. Геометрична інформація про риси

обличчя використовується в літературі [47]. Знаходяться ніс, очі та рот і застосовуються тривимірні геометричні вимірювання. Вимірювання в даному випадку — це Евклідова відстань по прямій лінії, відстань кривизни, площа, кут та об’єм. Класифікація заснована на Коефіцієнті подібності.

1.9 Висновки

Вибір найкращого алгоритму неможливо зробити через нестандартні бібліотеки облич і неуніверсальні бази даних про обличчя. Також на метод виявлення та розпізнавання можуть впливати зміни освітлення, зміни пози, оклюзії та вираз обличчя. Усі методи не можуть обробляти всі ефекти в одному алгоритмі. Отже, важко розробити алгоритм, який обробляв би всі ефекти.

Наша проблема полягає в розробці системи виявлення обличчя, яка знаходить обличчя на зображеннях і витягує їх. У літературі є деякі методи та комбінація методів, які вирішують цю проблему [48 - 54]. Лі та Лян [48] спершу знаходять очі та губи за допомогою методу AdaBoost. Потім вектор ознак створюється на основі значень пікселів області очей, губ і носа та відстані між рисами обличчя. Вимір вектора ознак зменшується за допомогою Метода головних компонент та Лінійного дискримінантного аналізу. Класифікатор створюється за допомогою Нейронної мережі радіальних базисних функцій. Прихована марковська модель використовується як для пошуку, так і для класифікації облич [49]. В іншій роботі виявлення та класифікація здійснюється за допомогою нейронної мережі, а виділення ознак виконується за допомогою швидкого перетворення Фур’є з виявлених зображень [50]. Також поліноміальна нейронна мережа використовується для виявлення облич, а псевдо двовимірна прихована марковська модель використовується для класифікації облич [51]. Інша реалізація сегментує шкірну область в колірному просторі RGB, а потім сегментовані частини перевіряються за допомогою відповідності шаблону. Виявлені обличчя витягуються за допомогою Методу головних компонент, а класифікація здійснюється за Коефіцієнтом подібності [52]. Чжан [53] сегментує шкірну область, а кандидати на обличчя знаходяться за допомогою перевірки симетрії обличчя. Потім за допомогою шаблону ока визначають поворот обличчя. Потім за допомогою Вейвлета Хаара та спрямованої характеристики на основі градієнта генерується вектор ознак. Нарешті, класифікація вектора ознак відбувається за допомогою нейронної мережі.

РОЗДІЛ 2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Постановка задачі

Основна проблема, яку необхідно вирішити, полягає в реалізації алгоритму виявлення облич на зображенні. На перший погляд завдання виявлення обличчя може здатися не таким складним, особливо якщо врахувати, наскільки легко його вирішує людина. Однак існує різкий контраст із тим, наскільки складно насправді змусити комп'ютер успішно вирішувати це завдання.

Щоб полегшити завдання, алгоритм Віоли-Джонса обмежується повним оглядом фронтальних вертикальних зображень облич. Тобто, щоб бути виявленим, все обличчя має бути спрямоване на камеру, і воно не повинно бути нахилено ні в яку сторону. Це може дещо поставити під загрозу вимогу щодо необмеженості, але враховуючи, що алгоритм виявлення найчастіше замінюється алгоритмом розпізнавання, ці вимоги здаються цілком розумними.

Типове вхідне зображення для алгоритму виявлення обличчя показано на рис. 3. Це зображення має відносно низький контраст, містить багато різних видів текстур і, нарешті, містить багато облич. Оскільки більш-менш усі обличчя вертикальні, є надія, що вони будуть виявлені за допомогою цього алгоритму.

Рисунок 3. Типове вхідне зображення

2.2 Короткий опис роботи алгоритму Віоли-Джонса

Основним принципом роботи алгоритму Віоли-Джонса є сканування підвікна, здатного виявляти обличчя на заданому вхідному зображенні. Стандартний підхід до обробки зображень полягав би у зміні масштабу вхідного зображення до різних розмірів, а потім запуску детектора фіксованого розміру через ці зображення. Цей підхід виявляється досить трудомістким через розрахунок зображень різного розміру.

Всупереч стандартному підходу Віола та Джонс змінюють масштаб детектора замість вхідного зображення і пропускають детектор багато разів через зображення – кожен раз з іншим розміром. Спочатку можна було запідозрити, що обидва підходи забирають однаково багато часу, але Віола та Джонс розробили детектор з інваріантним масштабом, який вимагає однакової кількості обчислень незалежно від розміру. Цей детектор побудований з використанням так званого інтегрального представлення зображення та деяких простих прямокутних ознак, що нагадують вейвлети Хаара.

2.3 Ознаки та інтегральне представлення зображення

Процедура виявлення обличчя класифікує зображення на основі значень простих ознак. Існує багато мотивів використання ознак, а не безпосередньо пікселів. Найпоширеніша причина полягає в тому, що ознаки можуть кодувати спеціальні знання предметної області, які важко вивчити, використовуючи кінцеву кількість навчальних даних. Для цієї системи є також друга критична мотивація використання ознак: система на основі ознак працює набагато швидше, ніж система на основі пікселів.

Використані прості ознаки нагадують базисні функції Хаара, які використовували Папагеоргіу та ін. [55]. Точніше, ми використовуємо три види ознак. Значення ознаки двох прямокутників — це різниця між сумою пікселів у двох прямокутних областях. Області мають однакові розміри і форму, розташовані горизонтально або вертикально поруч (див. рис. 4). Ознака трьох прямокутників обчислює суму в двох зовнішніх прямокутниках, що віднімається від суми в центральному прямокутнику. Нарешті, ознака чотирьох прямокутників обчислює різницю між діагональними парами прямокутників.

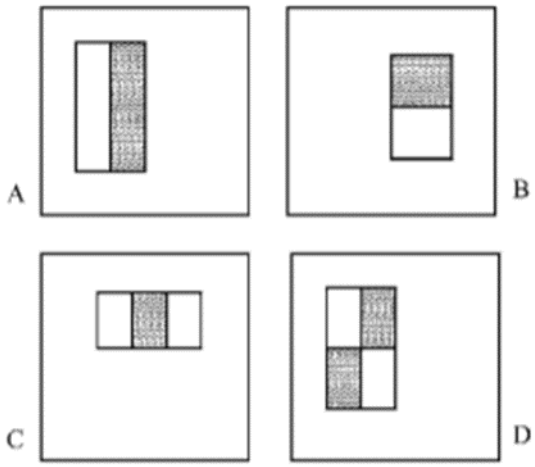
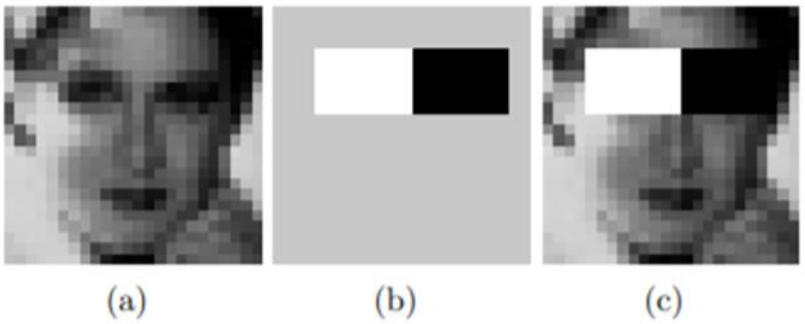


Рисунок 4. Приклади прямокутних ознак, показаних відносно обмежувального вікна виявлення. Сума пікселів, які лежать у білих прямокутниках, віднімається від суми пікселів у сірих прямокутниках. Ознаки двох прямокутників показані на рисунках (A) і (B). На рисунку (C) показано ознаку трьох прямокутників, а на рисунку (D) ознаку чотирьох прямокутників

Враховуючи, що базова роздільна здатність детектора становить 24 × 24, вичерпний набір ознак прямокутників досить великий, 160.000. Зауважте, що на відміну від базису Хаара, набір ознак прямокутників є надмірно повним.

Алгоритм Віоли-Джонса використовує ознаки Хаара, тобто скалярний добуток між зображенням та деякі шаблони Хаара. Точніше, нехай I та P позначають зображення та паттерн, обидва однакового розміру N × N (див. рис. 5). Характеристика, пов'язана з паттерном P зображення I, визначається за допомогою

$$\sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j)\text{білий}} - \sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j)\text{чорний}}$$



Рисинок 5. Ознаки Хаара. Тут, а також нижче, фон шаблону, наприклад (b), пофарбований у сірий колір, щоб виділити підтримку шаблону. Під час обчислення відповідної характеристики використовуються лише ті пікселі, позначені чорним або білим

Щоб компенсувати вплив різних умов освітлення, усі зображення мають бути попередньо середніми, а дисперсія нормалізована. Зображення з дисперсією меншою за одиницю, які в першу чергу мають мало цікавої інформації, залишаються поза увагою.

На практиці розглядається п'ять ознак (див. рис. 6 і Алгоритм 1). Вважається, що похідні ознаки містять всю інформацію, необхідну для характеристики обличчя. Оскільки обличчя за своєю природою загалом правильні, використання ознаків Хаара, здається виправданим. Однак існує ще один важливий елемент, який дає перевагу цьому набору ознак: інтегральне представлення зображення, яке дозволяє розрахувати їх за дуже низьких обчислювальних витрат.

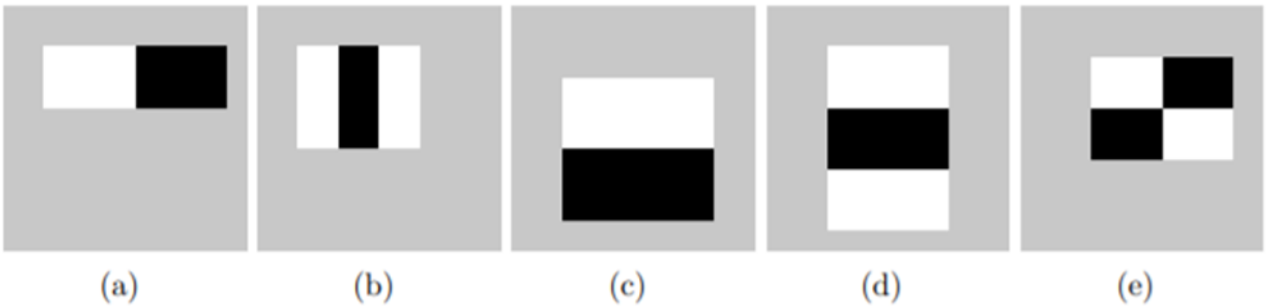


Рисунок 6. П'ять ознак Хаара. Розмір і положення паттернів можуть змінюватися за умови, що його чорно-білі прямокутники мають однаковий розмір, межують один з одним і зберігають взаємне розташування. Завдяки цьому обмеженню, кількість ознак, які можна отримати із зображення, є дещо керованим: зображення розміром 24 × 24, наприклад, має 43200, 27600, 43200,

27600 і 20736 ознак категорії (a), (b), (c), (d) та (e) відповідно, отже, всього 162336 ознак.

Алгоритм 1. Обчислення вектора ознак Хаара для зображення розміром 24 × 24
Вхідні дані: зображення розміром 24 × 24 з нульовим середнім значенням і одиницею дисперсії
Вихідні дані: скалярний вектор $d \times 1$ з індексом ознак f від 1 до d
Встановлюємо індекс функції $f \leftarrow 0$
Обчислюємо тип ознаки (a)
для всіх (i, j) таких, що $1 \le i \le 24$ і $1 \le j \le 24$ робимо
    для всіх (w, h) таких, що $i + h - 1 \le 24$ і $j + 2w - 1 \le 24$ робимо
        обчислюємо суму s_1 пікселів у $[i, i + h - 1] \times [j, j + w - 1]$
        обчислюємо суму s_2 пікселів у $[i, i + h - 1] \times [j + w, j + 2w - 1]$
        записуємо цю ознаку, параметризовану $(1, i, j, w, h)$: $s_1 - s_2$
        $f \leftarrow f + 1$
    кінець цикла
кінець цикла
Обчислюємо тип ознаки (b)
для всіх (i, j) таких, що $1 \le i \le 24$ і $1 \le j \le 24$ робимо
    для всіх (w, h) таких, що $i + h - 1 \le 24$ і $j + 3w - 1 \le 24$ робимо
        обчислюємо суму s_1 пікселів у $[i, i + h - 1] \times [j, j + w - 1]$
        обчислюємо суму s_2 пікселів у $[i, i + h - 1] \times [j + w, j + 2w - 1]$
        обчислюємо суму s_3 пікселів у $[i, i + h - 1] \times [j + 2w, j + 3w - 1]$
        записуємо цю ознаку, параметризовану $(2, i, j, w, h)$: $s_1 - s_2 + s_3$
        $f \leftarrow f + 1$

    кінець цикла
кінець цикла
Обчислюємо тип ознаки (c)
для всіх (i, j) таких, що $1 \le i \le 24$ і $1 \le j \le 24$ робимо
    для всіх (w, h) таких, що $i + 2h - 1 \le 24$ і $j + w - 1 \le 24$ робимо
        обчислюємо суму s_1 пікселів у $[i, i + h - 1] \times [j, j + w - 1]$
        обчислюємо суму s_2 пікселів у $[i + h, i + 2h - 1] \times [j, j + w - 1]$
        записуємо цю ознаку, параметризовану $(3, i, j, w, h)$: $s_1 - s_2$
        $f \leftarrow f + 1$
    кінець цикла
кінець цикла
Обчислюємо тип ознаки (d)
для всіх (i, j) таких, що $1 \le i \le 24$ і $1 \le j \le 24$ робимо

для всіх (w, h) таких, що $i + 3h - 1 \leq 24$ і $j + w - 1 \leq 24$ робимо
обчислюємо суму S_1 пікселів у $[i, i + h - 1] \times [j, j + w - 1]$
обчислюємо суму S_2 пікселів у $[i + h, i + 2h - 1] \times [j, j + w - 1]$
обчислюємо суму S_3 пікселів у $[i + 2h, i + 3h - 1] \times [j, j + w - 1]$
записуємо цю ознаку, параметризовану $(4, i, j, w, h)$: $S_1 - S_2 + S_3$
$f \leftarrow f + 1$
кінець цикла
кінець цикла
Обчислюємо тип ознаки (e)
для всіх (i, j) таких, що $1 \leq i \leq 24$ і $1 \leq j \leq 24$ робимо
для всіх (w, h) таких, що $i + 2h - 1 \leq 24$ і $j + 2w - 1 \leq 24$ робимо
обчислюємо суму S_1 пікселів у $[i, i + h - 1] \times [j, j + w - 1]$
обчислюємо суму S_2 пікселів у $[i + h, i + 2h - 1] \times [j, j + w - 1]$
обчислюємо суму S_3 пікселів у $[i, i + h - 1] \times [j + w, j + 2w - 1]$
обчислюємо суму S_4 пікселів у $[i + h, i + 2h - 1] \times [j + w, j + 2w - 1]$
записуємо цю ознаку, параметризовану $(5, i, j, w, h)$: $S_1 - S_2 - S_3 + S_4$
$f \leftarrow f + 1$
кінець цикла
кінець цикла

Використовуючи інтегральне представлення зображення, будь-яку прямокутну суму можна обчислити використовуючи чотири посилання на масив. Очевидно, що різницю між двома прямокутними сумами можна обчислити за вісьмома посиланнями. Оскільки ознаки з двома прямокутниками, які визначені вище, включають суміжні прямокутні суми, їх можна обчислити за шістьма посиланнями на масив, вісім посилань у випадку ознак із трьома прямокутниками та дев'ятьма посиланнями для ознак із чотирма прямокутниками.

Замість того, щоб підсумовувати всі пікселі всередині прямокутного вікна, цей метод відображає використання кумулятивних функцій розподілу. Інтегральне представлення зображення II

$$II(i,j) \coloneqq \begin{cases} \sum_{1 \leq s \leq i} \sum_{1 \leq t \leq j} I(s,t), 1 \leq i \leq N \text{та} 1 \leq j \leq N \\ 0, \text{інакше} \end{cases}$$

так визначено, що

$$\sum_{N_i \leq i \leq N_2} \sum_{N_3 \leq j \leq N_4} I(i,j) = II(N_2, N_4) - II(N_2, N_3 - 1) - II(N_1 - 1, N_4) + II(N_1 - 1, N_3 - 1) (1)$$

виконується для всіх $N_1 \le N_2$ і $N_3 \le N_4$. У результаті обчислення прямокутної локальної суми зображення вимагає не більше чотирьох елементарних операцій з урахуванням його інтегрального представлення. Більше того, отримання самого інтегрального зображення можна здійснити за лінійний час: встановивши $N_1 = N_2$ і $N_3 = N_4$ в (1), знаходимо

$$I(N_1, N_3) = II(N_1, N_3) - II(N_1, N_3 - 1) - II$$

Звідси випливає рекурсивне відношення, яке веде до Алгоритму 2.

Алгоритм 2. Інтегральне представлення зображення
Вхідні дані: зображення *I* розміром $N \times M$
Вихідні дані: його інтегральне представлення *II* того ж розміру
Встановлюємо $II(1, 1) = I(1, 1)$
для $i = 1$ до N робимо
    для $j = 1$ до M робимо
        $II(i, j) = I(i, j) + II(i, j - 1) + II(i - 1, j) - II(i - 1, j - 1)$ та *II* визначається як нуль, коли його аргумент (i, j) виходить за межі області *I*
    кінець цикла
кінець цикла

В якості додаткової примітки зазначимо, що після того, як корисні ознаки були вибрані алгоритмом бустінга, потрібно відповідним чином їх масштабувати, маючи справу з більшим вікном (див. Алгоритм 3). Але менші вікна дивитися не будуть.

Алгоритм 3. Масштабування ознак
Вхідні дані: зображення e × e з нульовим середнім і одиницею дисперсії ($e \ge 24$)
Параметр: тип ознак Хаара та його параметр (i, j, w, h), як визначено в алг. 1
Вихідні дані: значення ознаки
якщо тип ознаки (a), тоді
    встановлюємо вхідний розмір вектора ознак a ← 2wh
    i ← Jie/24K, j ← Jje/24K, h ← Jhe/24K, де JzK визначає найближче ціле число до z ∈ R+
    w ← max{κ ∈ N : κ ≤ J1 + 2we/24K/2, 2κ ≤ e − j + 1}
    обчислюємо суму S_1 пікселів у [i, i + h − 1] × [j, j + w − 1]
    обчислюємо суму S_2 пікселів у [i, i + h − 1] × [j + w, j + 2w − 1]
    повертаємо масштабовану ознаку $\frac{(S_1 - S_2)a}{2wh}$
кінець блоку якщо
якщо тип ознаки (b), тоді

встановлюємо вхідний розмір вектора ознак a ← 3wh
i ← Jie/24K, j ← Jje/24K, h ← Jhe/24K
w ← max{κ ∈ N : κ ≤ J1 + 3we/24K/3, 3κ ≤ e − j + 1}
обчислюємо суму S_1 пікселів у [i, i + h − 1] × [j, j + w − 1]
обчислюємо суму S_2 пікселів у [i, i + h − 1] × [j + w, j + 2w − 1]
обчислюємо суму S_3 пікселів у [i + h, i + 2h − 1] × [j, j + w − 1]
повертаємо масштабовану ознаку $\frac{(S_1-S_2+S_3)a}{2wh}$

кінець блоку якщо

якщо тип ознаки (c), тоді
встановлюємо вхідний розмір вектора ознак a ← 2wh
i ← Jie/24K, j ← Jje/24K, w ← Jwe/24K
h ← max{κ ∈ N : κ ≤ J1 + 2he/24K/2, 2κ ≤ e − i + 1}
обчислюємо суму S_1 пікселів у [i, i + h − 1] × [j, j + w − 1]
обчислюємо суму S_2 пікселів у [i + h, i + 2h − 1] × [j, j + w − 1]
повертаємо масштабовану ознаку $\frac{(S_1-S_2)a}{2wh}$

кінець блоку якщо

якщо тип ознаки (d), тоді
встановлюємо вхідний розмір вектора ознак a ← 3wh
i ← Jie/24K, j ← Jje/24K, w ← Jwe/24K
h ← max{κ ∈ N : κ ≤ J1 + 3he/24K/3, 3κ ≤ e − i + 1}
обчислюємо суму S_1 пікселів у [i, i + h − 1] × [j, j + w − 1]
обчислюємо суму S_2 пікселів у [i + h, i + 2h − 1] × [j, j + w − 1]
обчислюємо суму S_3 пікселів у [i + 2h, i + 3h − 1] × [j, j + w − 1]
повертаємо масштабовану ознаку $\frac{(S_1-S_2+S_3)a}{2wh}$

кінець блоку якщо

якщо тип ознаки (e), тоді
встановлюємо вхідний розмір вектора ознак a ← 4wh
i ← Jie/24K, j ← Jje/24K
w ← max{κ ∈ N : κ ≤ J1 + 2we/24K/2, 2κ ≤ e − j + 1}
h ← max{κ ∈ N : κ ≤ J1 + 2he/24K/2, 2κ ≤ e − i + 1}
обчислюємо суму S_1 пікселів у [i, i + h − 1] × [j, j + w − 1]
обчислюємо суму S_2 пікселів у [i + h, i + 2h − 1] × [j, j + w − 1]
обчислюємо суму S_3 пікселів у [i, i + h − 1] × [j + w, j + 2w − 1]
обчислюємо суму S_4 пікселів у [i + h, i + 2h − 1] × [j + w, j + 2w − 1]
повертаємо масштабовану ознаку $\frac{(S_1-S_2-S_3+S_4)a}{4wh}$

кінець блоку якщо

Одна альтернативна мотивація для інтегрального представлення зображення походить з роботи Сімард та ін. [56]. Автори зазначають, що у випадку лінійних операцій (наприклад, $f \cdot g$) будь-яку оборотну лінійну операцію можна застосувати до f або g, якщо до результату застосувати її обернену. Наприклад, у випадку згортки, якщо похідний оператор застосовується як до зображення, так і до ядра, то результат повинен бути подвійно інтегрованим:

$$f * g = \iint (f' * g')$$

Далі автори показують, що згортку можна значно прискорити, якщо похідні f та g розріджені (або їх можна зробити такими). Схоже розуміння полягає в тому, що до f можна застосувати оборотну лінійну операцію, якщо її обернення застосовується до g:

$$(f'') * \left(\iint g\right) = f * g$$

Розглянуте в цій системі, обчислення прямокутної суми можна виразити у вигляді скалярного добутку, $i \cdot r$, де i — зображення, а r — це значення 1 всередині прямокутника, що представляє інтерес, та 0 — поза ним. Цю операцію можна переписати:

$$i \cdot r = \left(\iint i\right) \cdot r''$$

Інтегральне представлення зображення фактично є подвійним інтегралом зображення (спочатку вздовж рядків, а потім уздовж стовпців). Друга похідна прямокутника (спочатку в рядку, а потім у стовпці) дає чотири дельта-функції в кутах прямокутника. Оцінка другого скалярного добутку виконується за допомогою чотирьох звернень до масиву.

Прямокутні ознаки є дещо примітивними в порівнянні з альтернативами, такими як керовані фільтри [57, 58]. Керовані фільтри та їх родичі відмінно підходять для детального аналізу меж, стиснення зображення та аналізу текстур. Хоча прямокутні ознаки також чутливі до наявності країв, смуг та інших простих структур зображення, вони досить грубі. На відміну від керованих фільтрів, доступні лише вертикальні, горизонтальні та діагональні орієнтації. Оскільки ортогональність не є центральною для цього набору ознак, то ми вирішили створити дуже великий та різноманітний набір прямокутних ознак. Цей надмірно повний набір, надає ознакам довільного співвідношення сторін і точного розташування. Емпірично здається, ніби набір прямокутних ознак забезпечує багате представлення зображення, яке підтримує ефективне

навчання. Надзвичайна обчислювальна ефективність прямокутних ознак забезпечує достатню компенсацію їхніх обмежень.

Щоб оцінити обчислювальну перевагу техніки інтегрального представлення зображення, розглянемо більш звичайний підхід, у якому обчислюється піраміда зображень. Як і більшість систем виявлення обличчя, наш детектор сканує вхідне зображення у багатьох масштабах; починаючи з базового масштабу, в якому обличчя виявляються з розміром 24 × 24 пікселі, зображення розміром 384 на 288 пікселів сканується в 12 масштабах, кожний в 1.25 раза більше, ніж останній. Звичайний підхід полягає в обчисленні піраміди з 12 зображень, кожне в 1.25 рази менше за попереднє зображення. Потім кожне з цих зображень сканується детектором фіксованого масштабу. Обчислення піраміди, хоча й просте, вимагає значного часу. Ефективно реалізований на звичайному обладнанні (з використанням білінійної інтерполяції для масштабування кожного рівня піраміди), обчислення 12-рівневої піраміди такого розміру займає приблизно 0.05 секунди (на процесорі Intel PIII 700 МГц). На відміну від цього, ми визначили значущий набір прямокутних ознак, які мають властивість, що одна ознака може бути оцінена в будь-якому масштабі та в будь-якому місці за кілька операцій. Ми покажемо, що ефективні детектори облич можна створити з двома прямокутними ознаками. Враховуючи обчислювальну ефективність цих ознак, процес виявлення обличчя можна завершити для всього зображення в кожному масштабі зі швидкістю 15 кадрів в секунду, приблизно за той самий час, який потрібно для оцінки лише 12-рівневої піраміди зображення. Будь-яка процедура, яка вимагає піраміди цього типу, обов’язково буде виконуватися повільніше, ніж наш детектор.

2.4 Вибір ознак за допомогою Adaboost

Враховуючи набір ознак і навчальний набір позитивних і негативних зображень, для вивчення класифікаційних функцій можна використовувати будь-яку кількість підходів машинного навчання. Сунг і Поджіо використовують суміш гауссової моделі [59]. Роулі та ін. [60] використовують невеликий набір простих ознак зображення та нейронну мережу. Осуна та ін. [61] використовували метод опорних векторів. Зовсім недавно Рот та ін. [62] запропонували нове і незвичайне представлення зображень і використали процедуру навчання Winnow.

Нагадаємо, що з кожним підвікном зображення пов’язано 160.000 прямокутників, число яких набагато більше, ніж кількість пікселів. Незважаючи на те, що кожну ознаку можна обчислити дуже ефективно, обчислення повного

набору є надзвичайно дорогим. Наша гіпотеза, підтверджена експериментом, полягає в тому, що дуже невелику кількість цих ознак можна об'єднати, щоб сформувати ефективний класифікатор. Головне завдання – знайти ці ознаки. У нашій системі варіант AdaBoost використовується як для вибору ознак, так і для навчання класифікатора [63]. У своїй початковій формі алгоритм навчання AdaBoost використовується для підвищення ефективності класифікації простого алгоритму навчання (наприклад, його можна використовувати для підвищення продуктивності простого персептрона). Він робить це шляхом об'єднання набору слабких функцій класифікації для формування більш сильного класифікатора. На мові бустінгу, простий алгоритм навчання називається слабким учнем. Так, наприклад, алгоритм навчання персептрона шукає в наборі можливих персептронів і повертає персептрон з найменшою помилкою класифікації. Учня називають слабким, тому що ми не очікуємо, що навіть найкраща функція класифікації добре класифікує навчальні дані (тобто для даної задачі найкращий персептрон може правильно класифікувати навчальні дані лише в 51% випадків). Для того, щоб слабкий учень отримав бустінг, він вирішує послідовність навчальних завдань. Після першого раунду навчання приклади повторно зважуються, щоб підкреслити ті, які були неправильно класифіковані попереднім слабким класифікатором. Остаточний сильний класифікатор має форму персептрона, зваженої комбінації слабких класифікаторів, за якими слідує порогове значення.

Формальні гарантії, надані процедурою навчання AdaBoost, досить сильні. Фройнд і Шапір довели, що похибка навчання сильного класифікатора наближається до нуля експоненціально за кількістю раундів. Що більш важливо, згодом було доведено низку результатів щодо ефективності узагальнення [64]. Ключове розуміння полягає в тому, що продуктивність узагальнення пов'язана з запасом прикладів, і що AdaBoost швидко досягає великих запасів.

Звичайну процедуру AdaBoost можна легко інтерпретувати як жадібний процес вибору ознак. Розглянемо загальну проблему бустінга, в якій великий набір класифікаційних функцій об'єднується за допомогою зваженої більшості голосів. Завдання полягає в тому, щоб пов'язати велику вагу з кожною хорошою класифікаційною функцією і меншу вагу з поганими функціями. AdaBoost — це агресивний механізм для вибору невеликого набору хороших класифікаційних функцій, які, тим не менш, мають значну різноманітність. Проводячи аналогію між слабкими класифікаторами та ознаками, AdaBoost є ефективною процедурою для пошуку невеликої кількості хороших «ознак», які, однак,

мають значну різноманітність. Один практичний метод завершення цієї аналогії — обмежити слабкого учня набором класифікаційних функцій, кожна з яких залежить від однієї ознаки. На підтримку цієї мети, розроблено алгоритм слабкого навчання для вибору єдиної прямокутної ознаки, яка найкраще розділяє позитивні та негативні приклади (це схоже на підхід Тіє та Віоли [65] у сфері пошуку бази даних зображень). Для кожної ознаки слабкий учень визначає оптимальну порогову класифікаційну функцію, так що мінімальна кількість прикладів буде неправильно класифікована. Таким чином, слабкий класифікатор ($h(x, f, p, \theta)$) складається з ознаки (f), порогу (θ) і полярності (p), що вказує на напрямок нерівності:

$$h(x,f,p,\theta) = \begin{cases} 1 \text{якщо} pf(x) < p\theta \\ 0 \text{інакше} \end{cases}$$

Тут x — підвікно зображення розміром 24 × 24 пікселя.

На практиці жодна ознака не може виконати завдання класифікації з низькою похибкою. Ознаки, вибрані на початку процесу, дають коефіцієнт помилок від 0.1 до 0.3. Ознаки, вибрані в наступних раундах, оскільки завдання стає складнішим, дають коефіцієнт помилок від 0.4 до 0.5 (див. Алгоритм навчання 4).

Алгоритм 4. Кожна з гіпотез T будується з використанням однієї ознаки. Остаточна гіпотеза являє собою зважену лінійну комбінацію гіпотез T, де ваги обернено пропорційні помилкам навчання.

- Дані приклади зображень $(x_1, y_1), \dots, (x_n, y_n)$ де $y_i = 0{,}1$ для негативних і позитивних прикладів відповідно.
- Ініціалізуємо ваги $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ для $y_i = 0{,}1$ відповідно, де m і l – кількість негативних і позитивних прикладів відповідно.
- Для $t = 1, \dots, T$:
  1. Нормалізуємо ваги, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,i}}$
  2. Вибираємо найкращий слабкий класифікатор щодо зваженої похибки
  $$\epsilon_t = min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|.$$
  Див. Розділ 3.1 для обговорення ефективної реалізації.
  3. Визначаємо $h_t(x) = h(x, f_t, p_t, \theta_t)$ де f_t, p_t та θ_t – значення при яких функція ϵ_t приймає мінімальне значення.
  4. Оновлюємо ваги:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

де $e_i = 0$ якщо приклад x_i класифікований правильно, $e_i = 1$ інакше, та $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

- Остаточний сильний класифікатор:

$$C(x) = \begin{cases} 1 \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \text{інакше} \end{cases}$$

де $\alpha_t = log \frac{1}{\beta_t}$

Слабкі класифікатори, які ми використовуємо (порогові одиночні ознаки), можна розглядати як дерева рішень з одним вузлом. Такі структури в літературі машинного навчання називають стампами рішень. В оригінальній роботі Фройнда і Шапіра [63] також проводилися експерименти з бустінгом стампів рішень.

Класифікатор відображає спостереження на мітку, що оцінюється в кінцевому наборі. Для виявлення обличчя він приймає форму $f: R^d \to \{-1,1\}$, де 1 означає, що це є обличчя, а -1 навпаки (див. рис. 7), а d — кількість ознак Хаара, витягнуті із зображення. Враховуючи ймовірнісні вагові коефіцієнти $w \epsilon R_+$, призначені навчальному набору, що складається з n пар спостережень-мітка (x_i, y_i), Adaboost прагне ітеративно зменшити верхню межу емпіричної втрати:

$$\sum_{i=1}^{n} w_i 1_{y_i \neq f(x_i)}$$

за м'яких технічних умов. Примітно, що правило прийняття рішень, створене Adaboost, залишається досить простим, тому воно не схильне до переобладнання, а це означає, що емпірично вивчене правило часто добре узагальнює. Докладніше про цей метод можна знайти в [66, 67]. Незважаючи на його новаторський успіх, слід сказати, що Adaboost не вивчає, як має виглядати обличчя, сам по собі, оскільки саме люди, а не алгоритм, виконують маркування та перший раунд вибору ознак, як описано в попередньому розділі.

(a) (b)

Рисунок 7. Деякі приклади: (a) позитивні приклади (b) негативні приклади. Усі вони є зображеннями 24 × 24 у відтінках сірого

Будівельний блок детектора обличчя Віоли-Джонса — це стамп рішення або дерево рішень на одну глибину, параметризоване ознакою f ∈ {1, · · · , d}, порогом t ∈ R і перемикачем T ∈ {− 1, 1}. Враховуючи спостереження $x \in R^d$, стамп рішення h прогнозує свою мітку, використовуючи наступне правило

$$h(x) = \left(1_{\pi_f x \geq t} - 1_{\pi_f x < t}\right) T = \left(1_{\pi_f x \geq t} - 1_{\pi_f x < t}\right) 1_{T=1} + \left(1_{\pi_f x < t} - 1_{\pi_f x \geq t}\right) 1_{T=-1} \in \{-1,1\}$$

де $\pi f x$ – це вектор ознак f-тої координати. Далі кілька коментарів:

1. Будь-який додатковий паттерн, створений шляхом перестановки чорно-білих прямокутників у існуючий паттерн (див. рис. 7), є зайвим. Оскільки така ознака є просто протилежністю існуючої ознаки, потрібна лише зміна знака для t і T, щоб мати однакове правило класифікації.
2. Якщо навчальні приклади відсортовані в порядку зростання даної ознаки f, вичерпний пошук за лінійним часом на порозі та перемикач може знайти стамп рішення з використанням цієї ознаки, який досягає найменших емпіричних втрат

$$\sum_{i=1}^{n} w_i 1_{y_i \neq h(x_i)}$$

на навчальній множині (див. Алгоритм 5). Уявіть поріг, розміщений десь на реальній лінії, якщо перемикач встановлено на 1, отримане правило оголошуватиме приклад x позитивним, якщо $\pi_f x$ більше за поріг, і негативним в іншому випадку. Це дозволяє нам оцінити

емпіричну помилку правила, таким чином вибравши перемикач, який краще відповідає набору даних (рядки 8–16 Алгоритму 5).
Оскільки межа

$$\min_{i:y_i=-1}|\pi_f x_i - t| + \min_{i:y_i=1}|\pi_f x_i - t|$$

і ризик, або очікування емпіричного збитку, тісно пов'язані, двох стампів рішень, які мають однаковий емпіричний ризик, перевага надається одному з більшим запасом (рядок 14 алгоритму 4). Таким чином, за відсутності дублікатів існує n + 1 можливих порогів, і слід вибрати той який з найменшими емпіричними втратами. Однак можна мати однакові значення ознак з різних прикладів, і потрібно бути особливо уважним, щоб правильно обробити цей випадок (рядки 27–32 Алгоритму 5).

Алгоритм 5. Стамп рішення шляхом вичерпного пошуку
Вхідні дані: n навчальних прикладів, упорядкованих у порядку зростання ознаки $\pi_f x_i : \pi_f x_1 \leq \pi_f x_2 \leq \cdots \leq \pi_f x_n$, імовірнісні ваги прикладу $(w_k)_{1\leq k\leq n}$.
Вихідні дані: поріг рішення τ , перемикач T , помилка E і межа M.
Ініціалізація: $\tau \leftarrow \min_{1\leq i\leq n}\pi_f x_i - 1$, M ← 0 і E ← 2 (довільна верхня межа емпіричної втрати).
Підсумовуємо ваги позитивних (відповідно негативних) прикладів, f-ті ознаки яких більші за поточний поріг: $W_1^{+\leftarrow\sum_{i=1}^n w_i 1_{y_i=1}}$ (відп. $W_{-1}^{+\leftarrow\sum_{i=1}^n w_i 1_{y_i=-1}}$)
Підсумовуємо ваги позитивних (відповідно негативних) прикладів, f-ті ознаки яких менша за поточний поріг: $W_1^{-\leftarrow 0}$ (відп. $W_{-1}^{-\leftarrow 0}$)
Встановлюємо ітератор j ← 0, $\hat{t} \leftarrow \tau$ і $\hat{M} \leftarrow$ M.
поки істина робимо

Вибираємо перемикач, щоб мінімізувати зважену помилку:
помилка$_{+\leftarrow W_1^- + W_{-1}^+}$ та помилка$_{-\leftarrow W_1^+ + W_{-1}^-}$
якщо помилка$_+$ < помилка$_-$ тоді
$\hat{E} \leftarrow$ помилка$_+$ та $\hat{T} \leftarrow 1$
інакше
$\hat{E} \leftarrow$ помилка$_-$ та $\hat{T} \leftarrow -1$
кінець блока якщо
якщо $\hat{E} < E$ або $\hat{E} = E$ & $\hat{M} > M$ тоді
$E \leftarrow \hat{E}, \tau \leftarrow \hat{t}, M \leftarrow \hat{M}$ та $T \leftarrow \hat{T}$
кінець блока якщо

якщо j = n тоді
Закінчити
кінець блока якщо

j ← j + 1
поки істина робимо
якщо $y_{i_j} = -1$ тоді
$W_{-1}^{-\leftarrow W_{-1}^{-+w_{i_j}\text{та}W_{-1}^{+\leftarrow W_{-1}^{+-w_{i_j}}}}}$
інакше
$W_{1}^{-\leftarrow W_{1}^{-+w_{i_j}\text{та}W_{1}^{+\leftarrow W_{1}^{+-w_{i_j}}}}}$
кінець блока якщо
Щоб знайти новий дійсний поріг, нам потрібно обробити повторювані ознаки.
якщо j = n або $\pi_f x_{i_j} \neq \pi_f x_{i_{j+1}}$поки
Закінчити
інакше
j ← j + 1
кінець блока якщо
кінець цикла
якщо j = n тоді
$\hat{t} \leftarrow max_{1\le i\le n}\pi_f x_i + 1 \text{та} \hat{M} \leftarrow 0$
інакше
$\hat{t} \leftarrow \left(\pi_f x_{i_j} + \pi_f x_{i_{j+1}}\right)/2 \text{та} \hat{M} \leftarrow \pi_f x_{i_{j+1}} - \pi_f x_{i_j}$
кінець блока якщо
кінець цикла

Коригуючи вагові показники окремих прикладів (Алгоритм 7, рядок 10), Adaboost докладає більше зусиль для вивчення складніших прикладів і додає більше стампів для прийняття рішень (див. Алгоритм 6) у процесі. Інтуїтивно, під час остаточного голосування стамп h_t з меншою емпіричною втратою винагороджується більшим словом (вищим α_t, див. Алгоритм 7, рядок 9), коли Т-члени (класифікатор на основі голосування) призначає приклад відповідно до

$$f^T(\quad) = sign\left[\sum_{t=1}^{T} \alpha_t h_t(\quad)\right]$$

Алгоритм 6. Найкращий стамп
Вхідні дані: n навчальних прикладів, їхні ймовірнісні ваги $(w_i)_{1\le i\le n}$, кількість ознак d
Вихідні дані: поріг найкращого рішення, перемикач, помилка та межа

Встановіть помилку стампа найкращого рішення в 2.
для f = 1 до d робимо
    Обчислюємо стамп рішення, пов'язаний з ознакою f, за допомогою алг. 5.
    якщо цей стамп рішення має меншу зважену похибку (3), ніж найкращий стамп, або ширший запас, якщо зважена помилка однакова тоді
        ставимо це рішення як найкраще
    кінець блока якщо
кінець цикла

Алгоритм 7. Adaboost
Вхідні дані: n прикладів навчання $(x_i, y_i) \in R^d \times \{-1, 1\}$, 1 ≤ i ≤ n, кількість тренувань T
Параметр: початкові ймовірнісні коефіцієнти $w_i(1)$ для 1 ≤ i ≤ n
Вихідні дані: сильний учень
для t = 1 до T робимо
    Запускаємо Алгоритм 6, щоб навчити стамп рішення h_t за допомогою ваг w(t) і отримати його зважену помилку ϵ_t

$$\epsilon_t = \sum_{i=1}^{n} w_i(t) 1_{h_t(x_i) \neq y_i}$$

    якщо ϵ_t = 0 та t = 1 тоді
        навчання закінчується і повертається $h_1(\ \ )$
    інакше
        встановлюємо $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
        оновлюємо ваги

$$\forall i, w_i(t+1) = \frac{w_i(t)}{2}\left(\frac{1}{\epsilon_t} 1_{h_t(x_i) \neq y_i} + \frac{1}{1-\epsilon_t} 1_{h_t(x_i) = y_i}\right)$$

    кінець блока якщо
кінець цикла

На рис. 8 показано випадок, коли Adaboost одночасно зменшує хибнопозитивні та хибнонегативні показники, оскільки додаються все більше і більше стампів. Для простоти позначення позначимо емпіричну втрату через

$$\sum_{i=1}^{n} w_i(1) 1_{y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \leq 0} \coloneqq P(f^T(X) \neq Y)$$

де (X, Y) — випадкова пара, розподілена відповідно до ймовірності P, визначеної ваговими коефіцієнтами $w_i(1)$, 1 ≤ i ≤ n, встановленими на момент початку навчання. Оскільки емпірична втрата з T збігається до нуля,

збільшуються як хибнопозитивні $P(f^T(X) = 1 \vee Y = -1)$ так і помилково негативні показники $P(f^T(X) = -1 \vee Y = 1)$ внаслідок

$$P(f^T(X) \neq Y) = P(Y = 1)P(f^T(X) = -1|Y = 1) + P(Y = -1)P(f^T(X) = 1 \vee Y = -1)$$

Таким чином рівень виявлення

$$P(f^T(X) = 1|Y = 1) = 1 - P(f^T(X) = -1 \vee Y = 1)$$

повинен прагнути до 1.

Таким чином, розмір T підготовленого комітету залежить від цільових хибнопозитивних і хибнонегативних показників. Крім того, зазначимо, що, враховуючи n_- негативних і n_+ позитивних прикладів у навчальному пулі, прийнято давати негативному (відповідно позитивному) прикладу початкову вагу, що дорівнює $0.5/n_-$ (відповідно $0.5/n_+$) так що Adaboost не надає перевагу жодній категорії на початку.

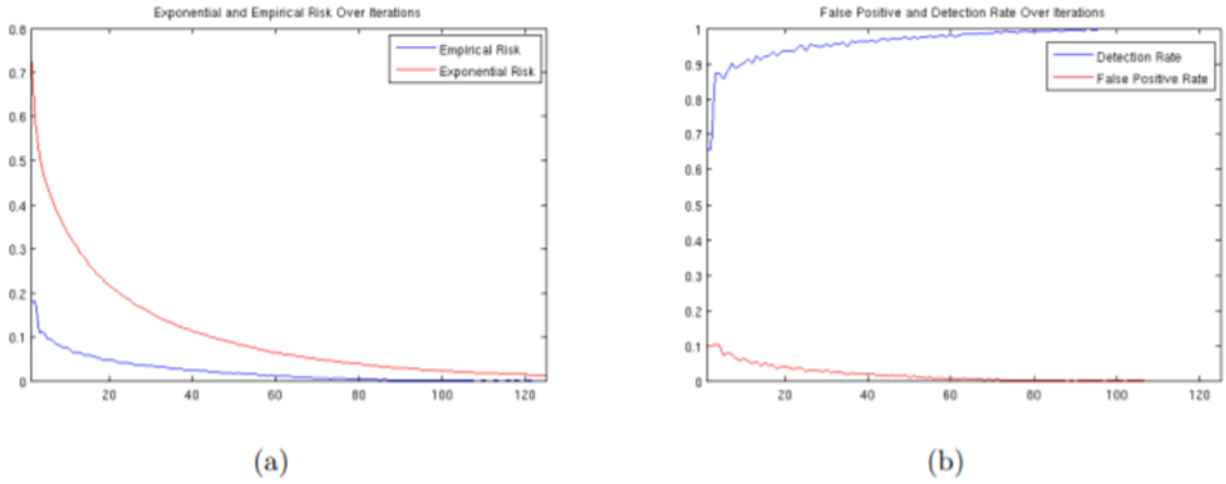


Рисунок 8. Алгоритм 7 працював з однаково зваженими 2500 позитивними і 2500 негативними прикладами. Рисунок (a) показує, що емпіричний ризик і його верхня межа, інтерпретовані як експоненційна втрата, неухильно зменшуються протягом ітерацій. Це означає, що хибнопозитивні та хибнонегативні показники також повинні зменшуватися, як це спостерігається в (b).

Алгоритм 4 використовується для вибору ключових слабких класифікаторів із набору можливих слабких класифікаторів. Хоча процес AdaBoost досить ефективний, набір слабких класифікаторів надзвичайно великий. Оскільки існує один слабкий класифікатор для кожної окремої комбінації ознака/порогове значення, то фактично існують KN слабких класифікаторів, де K – кількість ознак, а N – кількість прикладів. Щоб оцінити

залежність від N, припустимо, що приклади відсортовані за заданим значенням ознаки. Стосовно процесу навчання будь-які два пороги, які лежать між однією парою відсортованих прикладів, є еквівалентними. Тому загальна кількість різних порогів дорівнює N. Для задачі з $N = 20000$ і $K = 160000$ існує 3.2 мільярда різних бінарних слабких класифікаторів.

Метод обгортки також можна використовувати для навчання персептрона, який використовує M слабких класифікаторів [67]. Метод обгортки також діє поступово, додаючи один слабкий класифікатор до персептрона в кожному раунді. Доданий слабкий класифікатор — це той, який при додаванні до поточного набору дає персептрон з найменшою помилкою. Кожен раунд займає щонайменше $O(NKN)$ (або 60 трильйонів операцій); час для перерахування всіх бінарних ознак і оцінки кожного прикладу за допомогою цієї ознаки. Ця техніка нехтує часом для навчання ваг персептрона. Незважаючи на це, навчання класифікатора з 200 ознак буде щось на кшталт $O(MNKN)$, тобто 10^{16} операцій.

Ключовою перевагою AdaBoost як механізму вибору ознак перед конкурентами, такими як метод обгортки, є швидкість навчання. За допомогою AdaBoost можна навчити класифікатор 200 ознак за $O(MNK)$ або за близько 10^{11} операцій. Однією з ключових переваг є те, що в кожному раунді вся залежність від раніше вибраних ознак ефективно та компактно кодується за допомогою прикладів ваг. Ці ваги можуть бути використані для оцінки даного слабкого класифікатора за постійний час.

Алгоритм вибору слабкого класифікатора працює наступним чином. Для кожної ознаки приклади сортуються за значенням ознаки. Оптимальний поріг AdaBoost для цієї ознаки потім може бути обчислений за один прохід над цим відсортованим списком. Для кожного елемента в відсортованому списку зберігаються та оцінюються чотири суми: загальна сума ваг позитивних прикладів T^+, загальна сума ваг негативних прикладів T^-, сума додатних ваг нижче поточного прикладу S^+ і сума негативних ваг нижче поточного прикладу S^-. Помилка для порогу, який розділяє діапазон між поточним і попереднім прикладом у відсортованому списку:

$$e = min$$

Ці суми легко оновлюються в міру пошуку.

Було запропоновано багато загальних процедур вибору ознак [69]. Наша остаточна програма вимагала дуже агресивного процесу, який відкидав би переважну більшість ознак. Для подібної проблеми розпізнавання [55] запропонували схему вибору ознак на основі дисперсії ознак. Вони продемонстрували хороші результати, вибравши 37 ознак із загальної кількості

1734. Хоча це значне зменшення, кількість ознак, оцінених для кожного підвікна зображення, все ще досить велика.

Рот та ін. [62] пропонують процес вибору ознак на основі правила навчання експоненціального персептрона Winnow. Ці автори використовують дуже великий і незвичайний набір ознак, де кожен піксель відображається у двійковий вектор з d вимірами (коли конкретний піксель набуває значення x, у діапазоні $[0, d-1]$, x-й вимір встановлюється в 1, а інші розміри в 0). Двійкові вектори для кожного пікселя об'єднуються, щоб утворити один двійковий вектор з nd вимірами (n – кількість пікселів). Правило класифікації — це персептрон, який призначає одну вагу кожному виміру вхідного вектора. Процес навчання Winnow сходить до рішення, де багато з цих ваг дорівнюють нулю. Тим не менш, дуже велика кількість ознак зберігається (можливо, кілька сотень або тисяч).

2.5 Каскад уваги

У цьому розділі описується алгоритм побудови каскаду класифікаторів, який забезпечує підвищення ефективності виявлення при радикальному скороченні часу обчислень. Ключове розуміння полягає в тому, що можна створити менші, а отже, більш ефективні посилені класифікатори, які відкидають багато негативних підвікон, виявляючи майже всі позитивні екземпляри. Простіші класифікатори використовуються, щоб відхилити більшість підвікон, перш ніж використовувати складніші класифікатори для досягнення низького хибнопозитивного рівня.

Етапи в каскаді будуються навчальними класифікаторами за допомогою AdaBoost. Починаючи з сильного класифікатора з двома ознаками, ефективний фільтр обличчя можна отримати, налаштувавши поріг сильного класифікатора, щоб мінімізувати хибнонегативний рівень. Початковий поріг AdaBoost, $\frac{1}{2}\sum_{t=1}^{T}\alpha_t$ розроблено, щоб забезпечити низький рівень помилок на навчальних даних. Нижчий поріг дає вищі показники виявлення та вищий хибнопозитивний рівень. На основі ефективності, виміряної за допомогою навчального набору для перевірки, класифікатор із двома ознаками можна налаштувати, щоб виявити 100% облич із 50% хибнопозитивним рівнем.

Продуктивність виявлення класифікатора з двома ознаками далека від прийнятної як системи виявлення обличчя. Тим не менш, класифікатор може значно зменшити кількість підвікон, які потребують подальшої обробки, за допомогою дуже малої кількості операцій:

1. Оцінка прямокутних ознак (потребує від 6 до 9 посилань на массив для кожної ознаки)
2. Обчислення слабкого класифікатора для кожної ознаки (потребує одну операцію порогового значення для кожної ознаки)
3. Об’єднання слабких класифікаторів (потребує одне множення на об’єкт, додавання та поріг)

Класифікатор з двома ознаками складає близько 60 інструкцій мікропроцесора. Здається, важко уявити, що будь-який простіший фільтр міг би досягти більшого рівня відхилення. Для порівняння, для сканування простого шаблону зображення потрібно принаймні в 20 разів більше операцій на одне підвікно.

Загальна форма процесу виявлення — це вироджене дерево рішень, яке ми називаємо «каскадом» [70] (див. рис. 9). Позитивний результат першого класифікатора ініціює оцінку другого класифікатора, який також був скоригований для досягнення дуже високих показників виявлення. Позитивний результат другого класифікатора запускає третій класифікатор і так далі. Негативний результат у будь-який момент призводить до негайної відмови від підвікна.

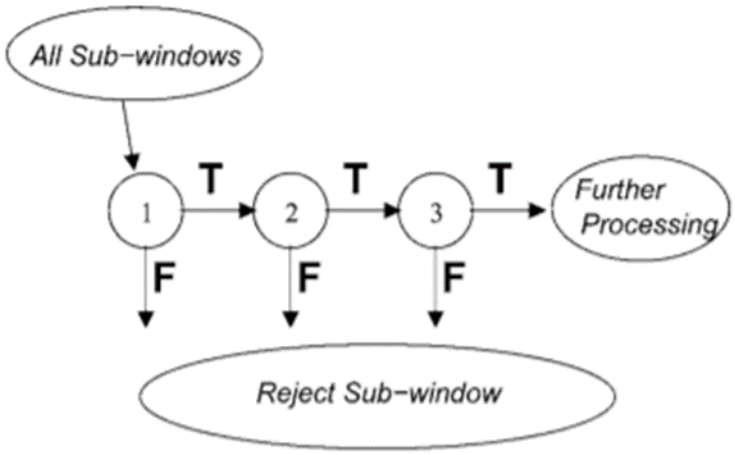


Рисунок 9. Схематичне зображення каскаду виявлення. Серія класифікаторів застосовується до кожного підвікна. Початковий класифікатор усуває велику кількість негативних прикладів з дуже малою обробкою. Наступні шари усувають додаткові негативи, але вимагають додаткових обчислень. Після кількох етапів обробки, кількість підвікон докорінно скорочується. Подальша обробка може приймати будь-яку форму, наприклад

додаткові етапи каскаду (як у нашій системі виявлення) або альтернативну систему виявлення.

Структура каскаду відображає той факт, що в будь-якому окремому зображенні переважна більшість підвікон є негативними. Таким чином, каскад намагається відкинути якомога більше негативів на якомога ранньому етапі. Хоча позитивний екземпляр ініціює оцінку кожного класифікатора в каскаді, це надзвичайно рідкісна подія.

Подібно до дерева рішень, наступні класифікатори навчаються за допомогою тих прикладів, які проходять усі попередні етапи. У результаті перед другим класифікатором постає складніше завдання, ніж перед першим. Приклади, які проходять перший етап, «важчі», ніж типові приклади. Складніші приклади, з якими стикаються більш глибокі класифікатори, штовхають всю криву робочої характеристики приймача (ROC) вниз. При заданій швидкості виявлення, більш глибокі класифікатори мають відповідно вищі показники хибнопозитивних результатів.

Теоретично, Adaboost може створити єдиний комітет стампів рішень, які добре узагальнюють. Однак, щоб досягти цього, на самому початку потрібен величезний негативний навчальний набір, щоб зібрати всі можливі негативні моделі. Крім того, єдиний комітет передбачає, що всі вікна всередині зображення мають пройти один і той же тривалий процес прийняття рішень. Має бути інший більш рентабельний спосіб.

Попередня ймовірність появи обличчя на зображенні не має жодного відношення до представленої конструкції класифікатора, оскільки вимагає, щоб як емпіричний хибнонегативний, так і хибнопозитивний показники наближалися до нуля. Однак власний досвід підказує, що на зображенні досить обмежена кількість підвікон заслуговує на більшу увагу, ніж інші. Це справедливо навіть для групових фотографій із інтенсивним використанням обличчя. Звідси виникла ідея багаторівневого каскаду уваги, який втілює принцип, подібний до кодування Шеннона: алгоритм повинен використовувати більше ресурсів для роботи з тими вікнами, які, швидше за все, містять обличчя, при цьому витрачаючи якомога менше зусиль на решту.

Очікується, що кожен рівень у каскаді уваги відповідатиме цілі навчання, вираженій у хибнопозитивних і хибнонегативних показниках: серед n негативних прикладів, оголошених позитивними всіма його попередніми шарами, шар l повинен розпізнавати щонайменше $(1 - \gamma_l)$n як негативний а тим

часом намагатися не жертвувати його продуктивністю на позитивних результатах: швидкість виявлення має підтримуватися вище $1 - \beta_l$.

Зрештою, враховується лише помилка узагальнення, яку, на жаль, можна оцінити лише за допомогою деяких прикладів перевірки, які Adaboost не може бачити на етапі навчання. Отже, в алгоритмі 11 у рядку 10 робиться консервативний вибір щодо того, як оцінювати коефіцієнт помилок: вищий рівень помилково позитивних результатів, отриманий під час навчання та перевірки, використовується для оцінки того, наскільки добре алгоритм навчився відрізняти обличчя від необлич. Таким же чином оцінюється хибнонегативний показник.

Слід мати на увазі, що Adaboost сам по собі не сприяє жодному з показників помилок: він прагне зменшити обидві одночасно, а не одне за рахунок іншого. Для забезпечення гнучкості введено один додатковий елемент керування s ∈ [−1, 1] для зсуву класифікатора:

$$f_s^T(\quad) = sign\left[\sum_{t=1}^{T} \alpha_t (h_t(\quad) + s)\right]$$

так що строго додатне s робить класифікатор більш схильним передбачати обличчя і навпаки.

Щоб забезпечити ефективний розподіл ресурсів, розмір комітету повинен бути невеликим на перших кількох рівнях, а потім поступово зростати, щоб велику кількість легких негативних моделей можна було усунути з невеликими обчислювальними зусиллями.

Додавання шару до каскаду означає, що алгоритм навчився відкидати кілька нових негативних шаблонів, які раніше вважалися складними, при цьому зберігаючи більш-менш той самий позитивний навчальний пул. Таким чином, щоб побудувати наступний рівень, потрібно більше негативних прикладів, щоб процес навчання був змістовним. Щоб замінити виявлені негативи, ми запускаємо каскад на великому наборі сірих зображень без людського обличчя та збираємо їхні помилкові позитивні вікна. Така ж процедура використовується для побудови та поповнення набору перевірки (див. Алгоритм 8). Оскільки на етапі навчання можна використовувати лише приклади розміром 24×24, ці більші хибні спрацьовування зменшуються (див. Алгоритм 9) і переробляються за допомогою Алгоритму 10.

Припустимо, що на рівні l формується комітет слабких класифікаторів T_l разом із зміною s, щоб можна було виміряти продуктивність класифікатора на

навчальній та валідаційній сукупності. Позначимо досягнутий показник хибнопозитивних і хибнонегативних показників через $\hat{\gamma}_l$ і $\hat{\beta}_l$. Залежно від їх співвідношення з цілями γ_l і β_l представлено чотири випадки:

1. Якщо ціль навчання рівня виконана (Алгоритм 11, рядок 11: $\hat{\gamma}_l \leq \gamma_l$ і $\hat{\beta}_l \leq \beta_l$), за потреби алгоритм переходить до навчання наступного рівня.
2. Якщо є можливість покращити швидкість виявлення (Алгоритм 11, рядок 13: $\hat{\gamma}_l \leq \gamma_l$ і $\hat{\beta}_l > \beta_l$), s збільшується на u, одиницю з префіксом.
3. Якщо є можливість покращити частоту помилкових позитивних результатів (Алгоритм 11, рядок 20:$\hat{\gamma}_l > \gamma_l$ і $\hat{\beta}_l \leq \beta_l$), s зменшується на u.
4. Якщо обидва коефіцієнти помилок не досягають цільового значення (Алгоритм 11, рядок 27: $\hat{\gamma}_l > \gamma_l$ і $\hat{\beta}_l > \beta_l$), алгоритм, якщо поточний комітет не перевищує встановлений ліміт розміру певного рівня, навчає ще одного члена для додавання до комітету.

Тут слід звернути особливу увагу: алгоритм міг чергувати випадок 2 і 3 і створювати мертвий цикл. Рішення полягає в тому, щоб кожен раз, коли це відбувається, зменшувати одиницю u вдвічі, поки u не стане меншим за 10^{-5}. Коли це станеться, рекомендується ще один раунд тренування на цьому рівні.

Як зазначалося раніше, щоб запобігти розростанню комітету занадто великого розміру, алгоритм припиняє уточнення пов’язаного з ним шару після порушення обмеження розміру, що залежить від рівня (Алгоритм 11, рядок 28). У цьому випадку зсув s встановлюється на найменше значення, яке задовольняє помилково негативній вимозі. Таким чином, більш складний випадок навчання переноситься на наступний рівень. Ця стратегія працює, оскільки нездатність Adaboost досягти цілі навчання часто можна пояснити тим фактом, що класифікатор, навчений на обмеженій кількості прикладів, може погано узагальнювати набір перевірки. Однак ці жорсткі негативні моделі в кінцевому підсумку повинні з’явитися і засвоїтися, якщо навчання буде продовжуватися, хоча і потроху.

Щоб проаналізувати, наскільки добре працює каскад, припустимо, що на рівні *l* Adaboost може надати класифікатор $f_{l,s_l}^{T_l}$ з хибнопозитивним рівнем γ_l і рівнем виявлення $1 - \beta_l$. У імовірнісному плані це означає:

$$P\left(f_{l,s_l}^{T_l}(X) = 1 \middle| Y = 1\right) \geq 1 - \beta_l \text{і} P\left(f_{l,s_l}^{T_l}(X) = 1 \middle| Y = -1\right) \leq \gamma_l$$

де через зловживання позначеннями зберігається P для позначення ймовірності на деякому просторі зображень. Якщо Алгоритм 11 зупиняється після L ітерацій, то правило таке:

$$f_{\text{каскад}}(X) = 2\left(\prod_{l=1}^{L} 1_{f_{l,s_l}^{T_l}(X)=1} - \frac{1}{2}\right)$$

Таким чином, вікно оголошується позитивним тоді й тільки тоді, коли всі його складові шари мають однакову думку:

$$P\left(f_{\text{каскад}}(X) = 1 \vee Y = -1\right)$$

$$P\left(l = 1 \middle| \; \left|L\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = -1\right.\right)$$

$$P\left(f_{l,s_l}^{T_l}(X) = 1 \vee l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \text{та} Y = -1\right.\right) P\left(l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = -1\right.\right)$$

$$\leq \gamma_l P\left(l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = -1\right.\right)$$

$$\leq \gamma_l^L$$

Аналогічно, загальний рівень виявлення можна оцінити наступним чином

$$P\left(f_{\text{каскад}}(X) = 1 \vee Y = 1\right)$$

$$P\left(l = 1 \middle| \; \left|L\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = -1\right.\right)$$

$$P\left(f_{l,s_l}^{T_l}(X) = 1 \vee l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \text{та} Y = 1\right.\right) P\left(l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = 1\right.\right)$$

$$\geq (1 - \beta_l) P\left(l = 1 \middle| \; \left|L - 1\left\{f_{l,s_l}^{T_l}(X) = 1\right\} \vee Y = 1\right.\right)$$

$$\geq (1 - \beta_l)^L$$

Іншими словами, якщо емпірично отримані показники є будь-якими свідченнями, безлике вікно матиме ймовірність, що перевищує $1 - \gamma_l$, то воно буде позначене таким на кожному шарі, ефективно спрямовуючи увагу каскадного класифікатора на тих, хто, швидше за все, має обличчя.

Алгоритм 8. Виявлення облич за допомогою тренованого каскадного класифікатора Adaboost
Вхідні дані: зображення у відтінках сірого M × N і каскад L-шарів зміщених класифікаторів, навчених за допомогою Алгоритму 10
Параметр: множник масштабу вікна c
Вихідні дані: P, набір вікон, оголошених додатними за допомогою каскаду
Встановлюємо P = {[i, i + e − 1] × [j, j + e − 1] ⊂ I : e = J24c^kK, k ∈ N}
для l = 1 до L робимо

для кожного вікна з P робимо

Видаляємо середнє значення віконного зображення та обчислюємо його стандартне відхилення.
якщо стандартне відхилення більше ніж 1 тоді
розділюємо зображення на це стандартне відхилення та обчислюємо його характеристики, необхідні для зміщеного класифікатора на шарі l, за допомогою Алгоритму 3
якщо l-й шар каскаду прогнозує негативний результат тоді
відкидаємо це вікно з P
кінець блока якщо
інакше
відкидаємо це вікно з P
кінець бока якщо
кінець цикла
кінець цикла
Повертаємо P

Алгоритм 9. Зменшення дискретизації квадратного зображення
Вхідні дані: зображення I e × e (e > 24)
Вихідні дані: зображення O зі зниженою дискретизацією розміром 24 × 24
Розмивання I за допомогою ядра Гаусса зі стандартним відхиленням $\sigma = 0.6\sqrt{\left(\left(\frac{e}{24}\right)^2 - 1\right)}$

Виділяємо матрицю O розмірністю 24 × 24
для i = 0 до 23 робимо
для j = 0 до 23 робимо
Обчислюємо масштабовані координати $\text{ъ} \leftarrow \frac{e-1}{25}(i+1), \text{ы} \leftarrow \frac{e-1}{25}(j+1)$
Встановлюємо
$\text{ъ}_{max} \leftarrow min(Jъ K + 1, e-1), \text{ъ}_{max} \leftarrow max(0, Jъ K), \text{ы}_{max} \leftarrow min(Jъ K + 1, e-1), \text{ы}_{min} \leftarrow max(0, Jъ K)$
Встановлюємо
$$O(i,j) = \frac{1}{4}[I(\text{ъ}_{max}, \text{ы}_{max}) + I(\text{ъ}_{min}, \text{ы}_{max}) + I(\text{ъ}_{min}, \text{ы}_{min}) + I(\text{ъ}_{max}, \text{ы}_{min})]$$
кінець цикла
кінець цикла
Повертаємо O

Алгоритм 10. Збір хибнопозитивних прикладів для навчання каскадного (L + 1)-го шару
Вхідні дані: набір зображень у відтінках сірого без людських облич і L-шаровий каскад зміщених класифікаторів
Параметр: множник масштабу вікна c

Вихідні дані: набір хибнопозитивних прикладів V
для кожного зображення у відтінках сірого робимо
Запускаємо Алгоритм 7, щоб отримати всі його помилкові результати Q
для кожного віконного зображення в Q робимо
якщо розмір вікна більше 24 × 24, тоді
зменшуємо вибірку цього підзображення за допомогою Алгоритму 8 і запускаємо на ньому Алгоритм 7
якщо зображення зі зниженою дискретизацією залишається позитивним тоді
приймаємо цей хибний позитив для V
кінець блока якщо
інакше
приймаємо цей хибний позитив для V
кінець блока якщо
кінець цикла
кінець цикла
Повертаємо V

Алгоритм 10. Каскад уваги
Вхідні дані: n позитивних тренінгів, m позитивних результатів перевірки, два набори сірих зображень без людських облич для малювання негативів для навчання та перевірки, бажана загальна хибнопозитивна частота γ_o, а також хибнопозитивний результат цільового шару та швидкість виявлення γ_l і $1 - \beta_l$.
Параметр: максимальний розмір комітету на рівні l: $N_l = \min(10l + 10, 200)$
Вихідні дані: каскад комітетів
Встановлюємо досягнутий загальний показник хибнопозитивних результатів $\hat{\gamma}_o \leftarrow 1$ та кількість шарів $l \leftarrow 0$

Малюємо випадковим чином $10n$ негативних прикладів навчання та m негативних прикладів перевірки
поки $\hat{\gamma}_o > \gamma_0$ робимо
$u \leftarrow 10^{-2}, l \leftarrow l + 1, s_t \leftarrow 0, \text{і} T_l \leftarrow 1$
Виконуємо Алгоритм 6 на навчальному наборі, щоб створити класифікатор $f_l^{T_l} = sign\left[\sum_{t=1}^{T_l} \alpha_t h_t\right]$
Запускаємо класифікатор зі зміщенням s_l $f_{l,s_l}^{T_l} = sign\left[\sum_{t=1}^{T_t} \alpha_t (h_t + s_l)\right]$ як на наборі навчання, так і на валідації, щоб отримати емпіричний та узагальнений хибнопозитивний (відповідно хибнонегативний) показник γ_e і γ_g (відповідно β_e і β_g)
$\hat{\gamma}_l \leftarrow max(\gamma_e, \gamma_g)$ і $\hat{\beta}_l \leftarrow max(\beta_e, \beta_g)$

якщо $\hat{\gamma}_l \le \gamma_l$ і $1 - \hat{\beta}_l \ge 1 - \beta_l$ тоді

$\hat{\gamma}_o \leftarrow \hat{\gamma}_o \times \hat{\gamma}_l$

інакше якщо $\hat{\gamma}_l \le \gamma_l, 1 - \hat{\beta}_l < 1 - \beta_l$ і $u > 10^{-5}$ (є можливість покращити показник виявлення) тоді

$s_l \leftarrow s_l + u$

якщо траєкторія s_l не монотонна тоді

$u \leftarrow u/2$

$s_l \leftarrow s_l - u$

кінець блока якщо

Переходимо до рядка 9

інакше якщо $\hat{\gamma}_l > \gamma_l, 1 - \hat{\beta}_l \ge 1 - \beta_l$ і $u > 10^{-5}$ (є можливість покращити показник хибнопозитивних результатів) тоді

$s_l \leftarrow s_l - u$

якщо траєкторія s_l не монотонна тоді

$u \leftarrow u/2$

$s_l \leftarrow s_l + u$

кінець блока якщо

Переходимо до рядка 9

інакше

якщо $T_l > N_l$ тоді

$s_l \leftarrow -1$

поки $1 - \hat{\beta}_l < 0.99$ робимо

Виконуємо рядки 9 і 10

кінець цикла

$\hat{\gamma}_o \leftarrow \hat{\gamma}_o \times \hat{\gamma}_l$

інакше

$T_l \leftarrow T_l + 1$ (Навчаємо ще одного члена, щоб додати його до комітету.)

Переходимо до рядка 8

кінець блока якщо

кінець блока якщо

Видаляємо помилкові та справжні негативи, виявлені поточним каскадом

$$f_{\text{каскад}}(X) = 2\left(\prod_{p=1}^{l} 1_{f_{p,s_p}^{T_p}(x)=1} - \frac{1}{2}\right)$$

Використовуємо цей каскад з алгоритмом 9, щоб отримати кілька помилкових спрацьовувань, щоб у наступному раунді було n тренувальних негативів і m негативів підтвердження

кінець цикла

Повертаємо каскад

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Ознаки та інтегральне представлення зображення

Один з ключових внесків, які зробили Віола та Джонс, був набір простих ознак для виявлення обличчя. У більшості випадків, значення пікселів вводяться в алгоритм. Однак Віола та Джонс представили нові ознаки.

Сума пікселів у незаштрихованих прямокутниках віднімається від суми пікселів у заштрихованих прямокутниках. Легко помітити, що навіть для невеликих зображень є багато ознак (понад 160 000 для зображення розміром 24 x 24). Оскільки алгоритм вимагає ітерації за всіма ознаками, то їх обчислення необхідно проводити дуже ефективно. Для цього Віола і Джонс запровадили інтегральне представлення. Інтегральне представлення визначається наступним рекурсивним відношенням (див. рис. 10).

$$\mathrm{ii}(-1, y) = 0$$
$$\mathrm{s}(x, -1) = 0$$
$$\mathrm{s}(x, y) = \mathrm{s}(x, y-1) + \mathrm{i}(x, y)$$
$$\mathrm{ii}(x, y) = \mathrm{ii}(x-1, y) + \mathrm{s}(x, y)$$

Рисунок 10. Рекурсивне відношення інтегрального представлення зображення

s(x, y) — це сукупна сума рядка в точці (x, y), ii(x, y) — інтегральне значення зображення в тій же точці, а i(x, y) — значення пікселя в цій точці. Це співвідношення говорить про те, що цілісне зображення в точці (x, y) є сумою всіх пікселів вище і ліворуч від поточного пікселя. Це дозволяє легко обчислити суму пікселів у прямокутній області, як це показано нижче на рис. 11.

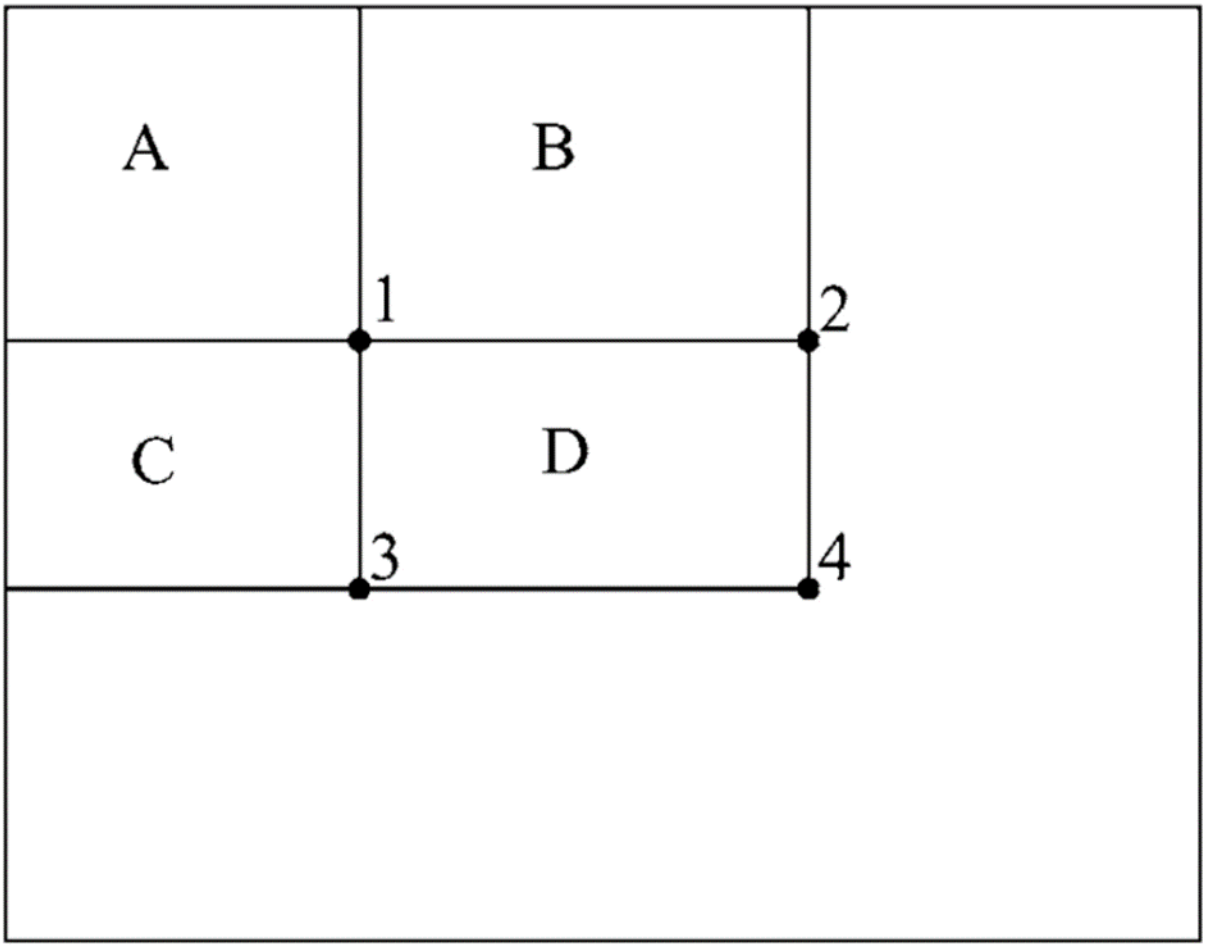


Рисунок 11. Інтегральне представлення зображення

Сума пікселів в області D дорівнює ii(4) + ii(1) – ii(2) – ii(3), що є лише чотирма посиланнями на масив. Розпочнемо реалізацію алгоритму зі створення допоміжної функції, яка обчислює інтегральне представлення зображення, по заданому зображенню (представлене у вигляді двовимірного масиву numpy) (див. рис. 12).

```python
import numpy as np

def integral_image(image):
    ii = np.zeros(image.shape)
    s = np.zeros(image.shape)
    for y in range(len(image)):
        for x in range(len(image[y])):
            s[y][x] = s[y-1][x] + image[y][x] if y-1 >= 0 else image[y][x]
            ii[y][x] = ii[y][x-1]+s[y][x] if x-1 >= 0 else s[y][x]
    return ii

```

Рисунок 12. Функція *integral_image*

Ця функція просто реалізує рекурсивне відношення, визначене вище. Оскільки s(x, -1) = 0, коли y - 1 < 0, s(x, y) є просто i(x, y), а також для ii(-1, y). Далі визначимо допоміжний клас для зберігання прямокутних областей, щоб потім було легко обчислювати значення ознак (див. рис. 13).

```python
class RectangleRegion:
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

    def compute_feature(self, ii):
        return ii[self.y + self.height][self.x + self.width] +
            ii[self.y][self.x] - (ii[self.y + self.height][self.x] +
            ii[self.y][self.x+self.width])
```

Рисунок 13. Клас *RectangleRegion*

3.2 Алгоритм Віоли-Джонса

Тепер, коли ми налаштували наші допоміжні класи та функції, ми можемо приступити до створення алгоритму Віоли-Джонса. Почнемо з визначення класу ViolaJones. Єдиний гіперпараметр, який буде використовувати наш алгоритм, - це кількість ознак (а також кількість слабких класифікаторів) (див. рис. 14).

```
23  class ViolaJones:
24      def __init__(self, T = 10):
25          self.T = T
26
```

Рисунок 14. Клас *ViolaJones*

Для навчання Віола-Джонс використовує варіант Adaboost. Загальна ідея посилення полягає в тому, щоб кожен наступний слабкий класифікатор виправляв помилки попереднього класифікатора. Для цього він призначає вагу кожному навчальному прикладу, навчає класифікатори, вибирає найкращий класифікатор і оновлює ваги відповідно до помилки класифікатора. Неправильно позначені приклади отримають більшу вагу, тому вони правильно класифікуються наступним вибраним класифікатором (див. рис. 15).

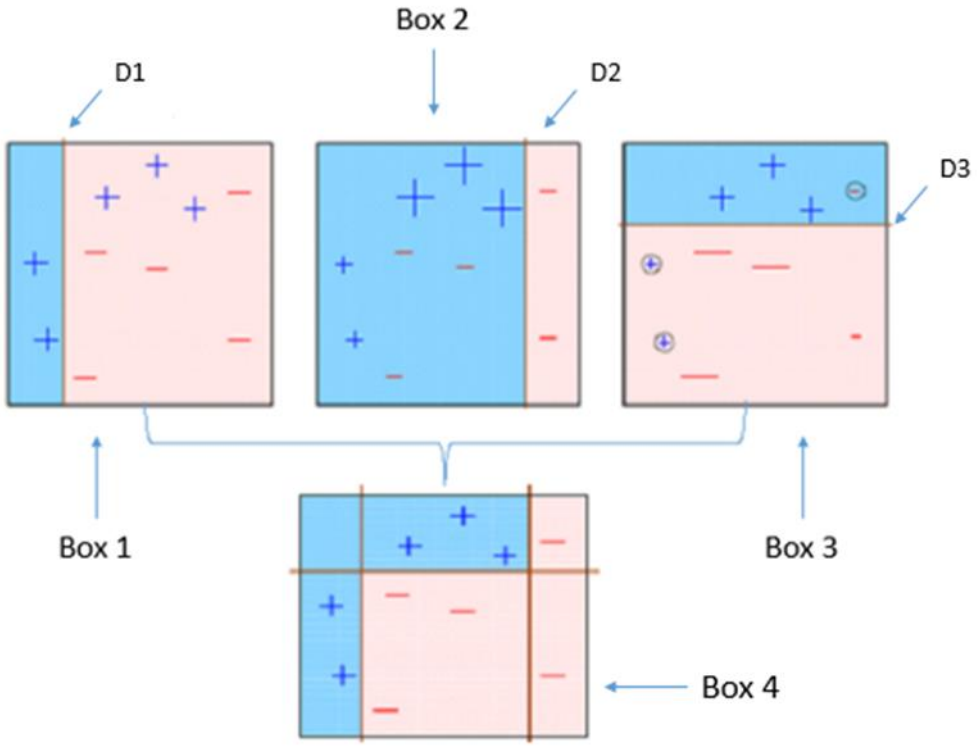


Рисунок 15. Візуальне представлення бустінга

Це дає наступний план алгоритму:

1. Ініціалізуємо ваги
2. Нормалізуємо ваги

3. Вибираємо найкращий слабкий класифікатор (на основі зваженої похибки)
4. Оновлюємо ваги на основі помилки обраних класифікаторів
5. Повторюємо пункти 2-4 T разів, де T – це бажана кількість слабких класифікаторів

Додаємо метод *train* до класу *ViolaJones*, де ми можемо реалізувати навчання (див. рис. 16).

```
27  def train(self, training):
28      training_data = []
29      for x in range(len(training)):
30      training_data.append((integral_image(training[x][0]),
31          training[x][1]))
```

Рисунок 16. Метод *train*

Наразі *train* прийматиме один параметр, *training_data* , який є масивом кортежів. Перший елемент у кортежі буде двовимірним масивом numpy, що представляє зображення, а другий елемент буде його класифікацією (1 для позитивного, 0 для негативного). Цикл перетворить усі зображення в їхнє інтегральне представлення. Ми додаємо інтегральні представлення зображень до нового масиву, щоб зберегти вхідний набір даних.

3.3 Ініціалізація ваг

На початку роботи алгоритму ми не маємо помилок, на основі яких базуватись вагові показники, тому кожен навчальний приклад одного класу має однакову вагу (тобто всі позитивні приклади однаково зважені, як і всі негативні). Це означає для і-го зображення:

$$w_i = \begin{cases} \frac{1}{2p} \text{якщо} x = 1 \\ \frac{1}{2n} \text{якщо} x = 0 \end{cases}$$

Де p – кількість позитивних прикладів, а n – кількість негативних прикладів. Припустимо, ми заздалегідь знаємо кількість позитивних і негативних прикладів, тому поїзд візьме їх як параметр (див. рис. 17).

```
33    def train(self, training, pos_num, neg_num):
34      weights = np.zeros(len(training))
35      training_data = []
36      for x in range(len(training)):
37        training_data.append((integral_image(training[x][0]),
38          training[x][1]))
39        if training[x][1] == 1:
40          weights[x] = 1.0 / (2 * pos_num)
41        else:
42          weights[x] = 1.0 / (2 * neg_num)
```

Рисунок 17. Метод *train*

3.4 Створення ознак

Основний цикл навчання вимагає вибору найкращого слабкого класифікатора, але для кожної можливої ознаки є один слабкий класифікатор. Через це ми повинні створити всі ознаки, перш ніж почати реалізовувати основний цикл навчання. Нагадаємо, що ознаки виглядають так (див. рис 18).

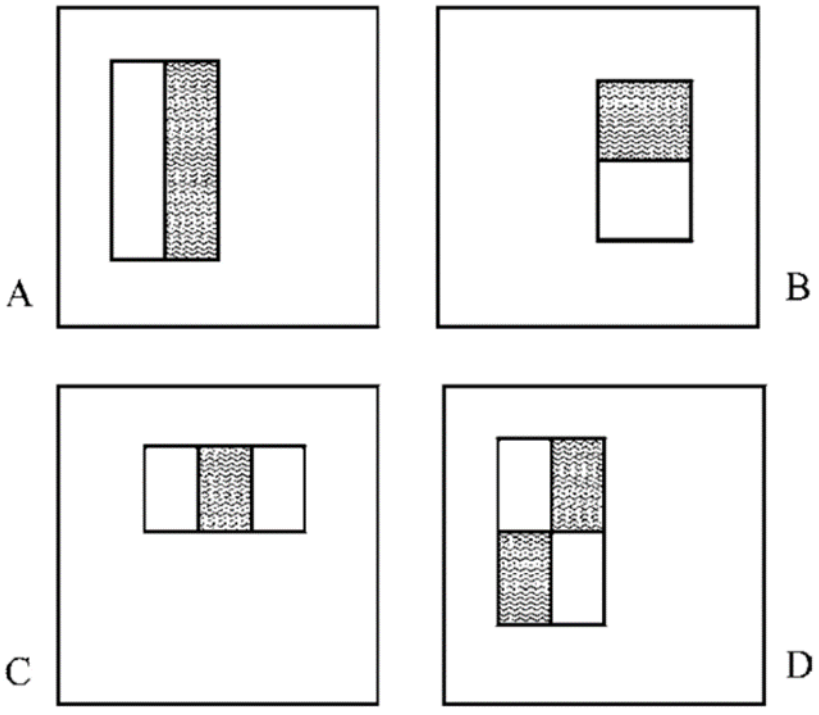


Рисунок 18. Ознаки Хаара

Реалізуємо *build_features* як частину класу *ViolaJones*, щоб повернути масив усіх ознак (див. рис. 19).

```
45  def build_features(self, image_shape):
46      height, width = image_shape
47      features = []
48      for w in range(1, width+1):
49          for h in range(1, height+1):
50              i = 0
51              while i + w < width:
52                  j = 0
53                  while j + h < height:
54                  #2 rectangle features
55                  immediate = RectangleRegion(i, j, w, h)
56                  right = RectangleRegion(i+w, j, w, h)
57                  if i + 2 * w < width: #Horizontally Adjacent
58                      features.append(([right], [immediate]))
59                  bottom = RectangleRegion(i, j+h, w, h)
60                  if j + 2 * h < height: #Vertically Adjacent
61                      features.append(([immediate], [bottom]))
62                  right_2 = RectangleRegion(i+2*w, j, w, h)
63                  #3 rectangle features
64                  if i + 3 * w < width: #Horizontally Adjacent
65                      features.append(([right], [right_2, immediate]))
66                  bottom_2 = RectangleRegion(i, j+2*h, w, h)
67                  if j + 3 * h < height: #Vertically Adjacent
68                      features.append(([bottom], [bottom_2, immediate]))
69                  #4 rectangle features
70                  bottom_right = RectangleRegion(i+w, j+h, w, h)
71                  if i + 2 * w < width and j + 2 * h < height:
72                      features.append(([right, bottom], [immediate, bottom_right]))
73              j += 1
74          i += 1
75      return features
```

Рисунок 19. Метод *build_features*

Параметр *image_shape* — це кортеж у формі (висота, ширина). Решта алгоритму обходить усі прямокутники на зображенні та перевіряє, чи можна створити з ним ознаку. Наше представлення ознаки — це кортеж, що містить два масиви. Першим масивом буде *RectangleRegions*, які позитивно впливають на ознаку, а другий масив — *RectangleRegions*, які негативно вносяться до ознаки. Це уявлення дозволить нам легко зберегти наш класифікатор пізніше.

3.5 Застосування ознак

Коли ми знаходимо оптимальні слабкі класифікатори для подальшого використання в алгоритмі, то буде потрібна оцінка кожної ознаки для кожного навчального прикладу. Щоб заощадити обчислення, ми зробимо це, перш ніж почнемо навчати класифікатори. Це ефективніше, оскільки кожен класифікатор потрібно перенавчати щоразу, коли ми вибираємо новий. Якби ми застосували ознаки до навчальних прикладів у циклі навчання, ми б повторили роботу, оскільки значення кожної ознаки для зображень ніколи не змінюється. Застосування ознак перед тренуванням також дозволить нам попередньо вибрати ознаки пізніше, щоб прискорити навчання. Збереження окремого масиву з міткою кожного навчального прикладу спростить наш код, тому ми також створимо цей масив під час цього кроку. Додаємо метод *apply_features* до класу *ViolaJones* (див. рис. 20).

```
77   def apply_features(self, features, training_data):
78     X = np.zeros((len(features), len(training_data)))
79     y = np.array(map(lambda data: data[1], training_data))
80     i = 0
81     for positive_regions, negative_regions in features:
82       feature = lambda ii: sum([pos.compute_feature(ii) for pos in
83           positive_regions]) - sum([neg.compute_feature(ii) for neg in
84           negative_regions])
85         X[i] = list(map(lambda data: feature(data[0]),
86           training_data))
87       i += 1
88     return X, y
```

Рисунок 20. Метод *apply_features*

Метод *train* тепер має виглядати так (див. рис. 21).

```
90   def train(self, training, pos_num, neg_num):
91     weights = np.zeros(len(training))
92     training_data = []
93     for x in range(len(training)):
94       training_data.append((integral_image(training[x][0]),
95         training[x][1]))
96       if training[x][1] == 1:
97         weights[x] = 1.0 / (2 * pos_num)
98       else:
99         weights[x] = 1.0 / (2 * neg_num)
100    features = self.build_features(training_data[0][0].shape)
101    X, y = self.apply_features(features, training_data)
```

Рисунок 21. Метод *train*

3.6 Слабкі класифікатори

Нарешті у нас є всі елементи, щоб почати створювати та навчати слабкі класифікатори. Нагадаємо, що Віола-Джонс використовує низку слабких класифікаторів і зважує їхні результати разом, щоб створити остаточну класифікацію. Кожен слабкий класифікатор є «слабким», оскільки сам по собі він не може точно виконати завдання класифікації. Кожен слабкий класифікатор розглядає одну ознаку (f). Він має як поріг (θ), так і полярність (p), щоб визначити класифікацію навчального прикладу.

$$h(x, f, p, \theta) = \begin{cases} 1 \text{якщо} pf(x) < p\theta \\ 0 \text{інакше} \end{cases}$$

Полярність може бути -1 або 1. Коли p = 1, слабкий класифікатор виводить позитивний результат, коли f(x) < θ, або значення ознаки менше порогового значення. Коли p = -1, слабкий класифікатор виводить позитивний результат, коли f(x) > θ. Тепер визначимо клас для інкапсуляції слабкої функціональності класифікатора (див. рис. 22).

```
103  class WeakClassifier:
104      def __init__(self, positive_regions, negative_regions, threshold, polarity):
105          self.positive_regions = positive_regions
106          self.negative_regions = negative_regions
107          self.threshold = threshold
108          self.polarity = polarity
109
110      def classify(self, x):
111          feature = lambda ii: sum([pos.compute_feature(ii) for pos in
112              self.positive_regions]) - sum([neg.compute_feature(ii) for neg in
113              self.negative_regions])
114          return 1 if self.polarity * feature(x) < self.polarity * self.threshold else 0
```

Рисунок 22. Клас *WeakClassifier*

Зауважте, що кожна ознака є сумою додатних областей прямокутника з сумою від'ємних областей прямокутника, відніманої від них. Нагадаємо, що наступним кроком в алгоритмі є вибір найкращого слабкого класифікатора. Для цього нам потрібно знайти оптимальний поріг і полярність для кожного з них.

3.7 Навчання слабких класифікаторів

Навчання слабких класифікаторів є найдорожчою частиною алгоритму, оскільки кожен раз, коли новий слабкий класифікатор вибирається як найкращий, усі вони повинні бути перенавчені, оскільки навчальні приклади мають різну вагу. Однак існує ефективний спосіб знайти оптимальний поріг і полярність для одного слабкого класифікатора за допомогою ваг. Спочатку відсортовуємо ваги відповідно до значення ознак, якому вони відповідають. Тепер перебираємо масив ваг і обчислюємо помилку, якщо порогове значення було обрано як ця ознака. Знаходимо поріг і полярність з мінімальною похибкою. Можливі значення порогу – це значення ознаки на кожному навчальному прикладі. Похибку можна виміряти за допомогою:

$$e = min$$

T являє собою загальну суму ваг, а S представляє суму ваг усіх прикладів, які ми бачили. Верхні індекси «+» і «-» позначають, до якого класу відноситься сума. Концептуально ця помилка порівнює, скільки прикладів буде неправильно класифіковано, якщо всі приклади нижче поточного розташування позначені як негативні, зі скількома прикладами буде неправильно класифіковано, якщо всі приклади нижче поточного розташування будуть позначені як позитивні (з урахуванням того, як кожен приклад зважений).

Таким чином, ми можемо оцінити похибку кожного можливого порогу в постійний час (O(1)) і помилку всіх порогів у лінійному часі (O(n)). Порогове

значення встановлюється на значення ознаки, при якому похибка є мінімальною. Полярність визначається тим, скільки позитивних і негативних прикладів лежать ліворуч (менше) і праворуч від порогу (більше). Якщо від порогу залишилося більше позитивних прикладів, p=1. Інакше p = -1. Ось це було описано у коді (частина класу *ViolaJones*). Цей метод тренує всі слабкі класифікатори та повертає їх у масиві (див. рис. 23).

```
116   def train_weak(self, X, y, features, weights):
117     total_pos, total_neg = 0, 0
118     for w, label in zip(weights, y):
119       if label == 1:
120         total_pos += w
121       else:
122         total_neg += w
123       classifiers = []
124       total_features = X.shape[0]
125       for index, feature in enumerate(X):
126         if len(classifiers) % 1000 == 0 and len(classifiers) != 0:
127           print("Trained %d classifiers out of %d" % (len(classifiers), total_features))
128         applied_feature = sorted(zip(weights, feature, y), key=lambda x: x[1])
129         pos_seen, neg_seen = 0, 0
130         pos_weights, neg_weights = 0, 0
131         min_error, best_feature, best_threshold, best_polarity = float('inf'), None, None, None
132         for w, f, label in applied_feature:
133           error = min(neg_weights + total_pos - pos_weights,
134             pos_weights + total_neg - neg_weights)
135           if error < min_error:
136             min_error = error
137             best_feature = features[index]
138             best_threshold = f
139             best_polarity = 1 if pos_seen > neg_seen else -1
140           if label == 1:
141             pos_seen += 1
142             pos_weights += w
143           else:
144             neg_seen += 1
145             neg_weights += w
146         clf = WeakClassifier(best_feature[0], best_feature[1],
147           best_threshold, best_polarity)
148         classifiers.append(clf)
149     return classifiers
```

Рисунок 23. Метод *train_weak*

3.8 Вибір найкращого слабкого класифікатора

Після того, як ми навчили всі слабкі класифікатори, тепер ми можемо знайти найкращий. Це так само просто, як ітерація всіх класифікаторів і обчислення середньозваженої помилки кожного з них. Додаємо метод *select_best* до класу *ViolaJones* (див. рис. 24).

```
151   def select_best(self, classifiers, weights, training_data):
152     best_clf, best_error, best_accuracy = None, float('inf'), None
153     for clf in classifiers:
154       error, accuracy = 0, []
155       for data, w in zip(training_data, weights):
156         correctness = abs(clf.classify(data[0]) - data[1])
157         accuracy.append(correctness)
158         error += w * correctness
159       error = error / len(training_data)
160       if error < best_error:
161         best_clf, best_error, best_accuracy = clf, error, accuracy
162     return best_clf, best_error, best_accuracy
```

Рисунок 24. Метод *select_best*

Звертаэмо увагу, що ми повернулися до нашого початкового представлення навчальних даних (масив кортежів). Це тому, що нам потрібно інтегральне зображення, щоб передати слабкому класифікатору, а не значення ознак. Також звертаємо увагу, що ми відстежуємо точність. Це вступить у дію під час оновлення ваг на наступному кроці алгоритму. Разом *train_weak* і *select_best* складають третій крок високорівневого алгоритму, який був описан раніше. Оновлюємо метод *train*, щоб використовувати ці два методи (див. рис. 25).

```
164   def train(self, training, pos_num, neg_num):
165     ...
166     for t in range(self.T):
167       weak_classifiers = self.train_weak(X, y, features, weights)
168       clf, error, accuracy = self.select_best(weak_classifiers, weights, training_data)
```

Рисунок 25. Метод *train*

Ці функції належать до циклу, тому що найкращий слабкий класифікатор залежить від ваги кожного навчального прикладу, і нагадуємо, що ваги для кожного навчального прикладу змінюються після кожного вибору нового слабкого класифікатора.

3.9 Оновлення ваг

Основна частина роботи у Віоли-Джонс йде на створення ознак, навчання класифікаторів і вибір найкращого слабкого класифікатора на кожній ітерації. Решта методів навчання відносно прості. Перш ніж вибрати найкращий класифікатор, слід нормалізувати ваги. Після вибору найкращого слабкого класифікатора ми повинні оновити ваги з похибкою вибраного слабкого класифікатора. Приклади навчання, які були класифіковані правильно,

отримають менші ваги, приклади, які класифіковані неправильно, залишаться незмінними (див. рис. 26).

$$\epsilon = min_{f,p,\theta} \sum_{i}^{N} w_i |h(x, f, p, \theta) - y_i|$$
$$\beta = \frac{\epsilon}{1-\epsilon}$$
$$\mathrm{w}_i = w_i \beta^{1-e_i}$$

Рисунок 26. Оновлення ваг

Епсилон представляє похибку найкращого класифікатора, w — вага i-го прикладу, а бета — дільник, чому потрібно змінити вагу. Експонента бети дорівнює 1-e, де e дорівнює 0, якщо навчальний приклад був класифікований правильно, і 1, якщо класифікований неправильно (саме тому ми повернули масив точності при виборі найкращого слабкого класифікатора) (див. рис. 27).

```
170   def train(self, training, pos_num, neg_num):
171     ...
172     for t in range(self.T):
173       weights = weights / np.linalg.norm(weights)
174       weak_classifiers = self.train_weak(X, y, features, weights)
175       clf, error, accuracy = self.select_best(weak_classifiers, weights, training_data)
176       beta = error / (1.0 - error)
177       for i in range(len(accuracy)):
178         weights[i] = weights[i] * (beta ** (1 - accuracy[i]))
```

Рисунок 27. Метод *train*

3.10 Сильний класифікатор

Нарешті, ми повинні зібрати сильний класифікатор із наших слабких класифікаторів. Сильний класифікатор визначається як на рис. 28.

$$\mathrm{C(x)} = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t, \\ 0 & \text{otherwise} \end{cases}$$
$$\alpha_t = ln(\frac{1}{\beta_t})$$

Рисунок 28. Визначення сильного класифікатора

Коефіцієнт альфа - це вага кожного слабкого класифікатора в остаточному рішенні, і він залежить від помилки, оскільки є природним логарифмом, оберненим бета. Зважену суму рішень слабких класифікаторів порівнюють з половиною суми альф, тому що це схоже на твердження «Принаймні половина слабких класифікаторів погоджується з позитивною класифікацією».

Нам знадобиться зберегти всі класифікатори та відповідні їм альфи, тому додаємо дві нові змінні екземпляра до методу *init* (див. рис. 29).

```
180    def __init__(self, T = 10):
181        self.T = T
182        self.alphas = []
183        self.clfs = []
```

Рисунок 29. Метод *init*

Далі оновлюємо цикл методу *train*, щоб обчислити альфи та зберегти їх, а також класифікатори (див. рис. 30).

```
185    def train(self, training, pos_num, neg_num):
186        ...
187        for t in range(self.T):
188            weights = weights / np.linalg.norm(weights)
189            weak_classifiers = self.train_weak(X, y, features, weights)
190            clf, error, accuracy = self.select_best(weak_classifiers,
191                weights, training_data)
192            beta = error / (1.0 - error)
193            for i in range(len(accuracy)):
194                weights[i] = weights[i] * (beta ** (1 - accuracy[i]))
195            alpha = math.log(1.0/beta)
196            self.alphas.append(alpha)
197            self.clfs.append(clf)
```

Рисунок 30. Метод *train*

Остаточний метод *train* має виглядати так як на рис. 31.

```
199   def train(self, training, pos_num, neg_num):
200     weights = np.zeros(len(training))
201     training_data = []
202     for x in range(len(training)):
203       training_data.append((integral_image(training[x][0]),
204         training[x][1]))
205       if training[x][1] == 1:
206         weights[x] = 1.0 / (2 * pos_num)
207       else:
208         weights[x] = 1.0 / (2 * neg_num)
209     features = self.build_features(training_data[0][0].shape)
210     X, y = self.apply_features(features, training_data)
211     for t in range(self.T):
212       weights = weights / np.linalg.norm(weights)
213       weak_classifiers = self.train_weak(X, y, features, weights)
214       clf, error, accuracy = self.select_best(weak_classifiers,
215         weights, training_data)
216       beta = error / (1.0 - error)
217       for i in range(len(accuracy)):
218         weights[i] = weights[i] * (beta ** (1 - accuracy[i]))
219       alpha = math.log(1.0/beta)
220       self.alphas.append(alpha)
221       self.clfs.append(clf)
```

Рисунок 31. Остаточний метод *train*

Обов’язково додаємо *import math* вгорі файлу. Нарешті, реалізуємо метод класифікації для сильного класифікатора (див. рис. 32).

```
223   def classify(self, image):
224     total = 0
225     ii = integral_image(image)
226     for alpha, clf in zip(self.alphas, self.clfs):
227       total += alpha * clf.classify(ii)
228     return 1 if total >= 0.5 * sum(self.alphas) else 0
```

Рисунок 32. Метод *classify*

3.11 Поліпшення

Кількість ознак зростає неймовірно швидко. Для зображення 24x24 існує понад 160 000 ознак, а для зображення 28x28 — понад 250 000. Більше того, багато з цих ознак не забезпечать великої потужності класифікації і в основному марні. Було б корисно видалити якомога більше ознак, щоб ми могли навчати менше *WeakClassifiers*. На щастя, пакет *SciKit-Learn* може допомогти нам звузити простір ознак за допомогою свого класу *SelectPercentile*. Щоб скористатися цим, додаємо наступне до методу *train* та імпортуємо необхідні класи (див. рис. 33).

```
230  from sklearn.feature_selection import SelectPercentile, f_classif
231  def train(self, training_data, pos_num, neg_num):
232    ...
233    X, y = self.apply_features(features, training_data)
234    indices = SelectPercentile(f_classif, percentile=10).fit(X.T, y).get_support(indices=True)
235    X = X[indices]
236    features = features[indices]
```

Рисунок 33. Поліпшення

Встановлення класу *SelectPercentile* дозволяє знайти найкращі k% ознак (вибрано 10% тут). Потім метод *get_support* повертає індекси цих ознак. Під час встановлення *SelectPercentile* звертаємо увагу на те, що передається X.T, а не просто X. Це відбувається тому, що *SelectPercentile* очікує, що кожен рядок буде одним навчальним прикладом, а кожен стовпець — значенням ознаки, але в масиві X це перемикається.

3.12 Збереження та завантаження

Оскільки навчання *ViolaJones* займає багато часу, було б дуже корисно мати можливість зберігати та завантажувати модель з файлу. Це легко зробити з модулем *Pickle*. Додаємо наступне до класу *ViolaJones* (див. рис. 34).

```
239  def save(self, filename):
240    with open(filename+".pkl", 'wb') as f:
241    pickle.dump(self, f)
242
243  @staticmethod
244  def load(filename):
245    with open(filename+".pkl", 'rb') as f:
246      return pickle.load(f)
```

Рисунок 34. Збереження та завантаження

3.13 Тестування

ViolaJones був створений для завдання виявлення обличчя, тож найкращий спосіб перевірити наш код на тому ж завданні. Хорошим набором даних для перевірки цього є база даних *CBCL*, створена Центром біологічного та обчислювального навчання Массачусетського технологічного інституту. Набір даних містить навчальний набір із понад 2000 зображень облич і понад 4500 зображень без облич. Кожне зображення має розмір 19x19 у відтінках сірого. Оригінальний набір даних містить кожне зображення у форматі .pgm, тому було створено два файли *Pickle*, один для навчання та один для тестування, що дозволить безпосередньо завантажити їх у клас *ViolaJones* (див. рис. 35)

```
248  def train(t):
249      with open("training.pkl", 'rb') as f:
250          training = pickle.load(f)
251      clf = ViolaJones(T=t)
252      clf.train(training, 2429, 4548)
253      evaluate(clf, training)
254      clf.save(str(t))
255
256  def test(filename):
257      with open("test.pkl", 'rb') as f:
258          test = pickle.load(f)
259      clf = ViolaJones.load(filename)
260      evaluate(clf, test)
261
262  def evaluate(clf, data):
263      correct = 0
264      for x, y in data:
265          correct += 1 if clf.classify(x) == y else 0
266      print("Classified %d out of %d test examples" % (correct, len(data)))
267
268  train(10)
269  test("10")
```

Рисунок 35. Тестування

При T = 10 алгоритм зайняв годину на навчання і отримав точність 85.5% на навчальному наборі з точністю 78% на тестовому наборі. При T=50 алгоритм зайняв п’ять годин на тренування і отримав 93% точність як на тренувальному, так і на тестовому наборі.

ВИСНОВКИ

У даній кваліфікаційній бакалаврській роботі було досліджено та реалізовано алгоритм виявлення обличчя. Був проведений аналіз існуючих методів і в результаті чього був обраний алгоритм Віоли-Джонса, як такий, що показує найоптимальніші результати для поставленої задачі.

Задача виявлення обличчя на фото або відео стала надзвичайно популярною в розділі кібернетики, який називається – розпізнавання образів.

Вирішуючи цю задачу дослідники навчилися краще аналізувати та розуміти зображення в цілому.

Дана робота описує підхід глибинного навчання для візуального розпізнавання об’єктів, який характеризується високою швидкістю та показниками виявлення облич. Цей метод відрізняється тим, що по-перше зображення представляється у «Інтегральному вигляді», а по-друге, що навчання відбувається за допомогою алгоритму бустінга, який з великої кількості ознак виділяє лише найважливіші, що сильно зменшує час виявлення.

Був досягнутий високий рівень показника виявлення обличчя. Реалізована система виявлення обличчя має швидкість в 15 раз вищу, аніж в будь-якого іншого метода, який існував до моменту створення даного алгоритму.

Розроблена система виявлення обличчя має широкий спектр застосування. Її можна використовувати в охоронних, військових, цивільних та медичних галузях.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. L. Zhi-fang, Y. Zhi-sheng, A.K. Jain and W. Yun-qiong, 2003, “Face Detection and Facial Feature Extraction in Color Image”, Proc. The Fifth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA’03), pp.126-130, Xi’an, China.
2. K. Seo, W. Kim, C. Oh and J. Lee, 2002, “Face Detection and Facial Feature Extraction Using Color Snake”, Proc. ISIE 2002 - 2002 IEEE International Symposium on Industrial Electronics, pp.457-462, L 'Aquila, Italy.
3. J. Ruan and J. Yin, 2009, “Face Detection Based On Facial Features and Linear Support Vector Machines”, Proc. 2009 International Conference on Communication Software and Networks, pp.371-375, Macau, China.
4. L. Zhao, X. Sun and X. Xu, 2006, “Face Detection Based On Facial Features”, Proc. ICSP2006, Guilin, China.
5. C. Lin, 2005, “Face Detection by Color and Multilayer Feedforward Neural Network”, Proc. 2005 IEEE International Conference on Information Acquisition, pp.518-523, Hong Kong and Macau, China.
6. M. A. Berbar, H. M. Kelash and A. A. Kandeel, 2006, “Faces and Facial Features Detection in Color Images”, Proc. Geometric Modeling and Imaging— New Trends (GMAI'06), pp.209-214, London, UK.
7. S. Kherchaoui and A. Houacine, 2010, “Face Detection Based On a Model of the Skin Color with Constraints and Template Matching”, Proc. 2010 International Conference on Machine and Web Intelligence, pp. 469 - 472, Algiers, Algeria.
8. D. Huang, T. Lin, C. Ho and W. Hu, 2010, “Face Detection Based On Feature Analysis and Edge Detection Against Skin Color-like Backgrounds”, Proc. 2010 Fourth International Conference on Genetic and Evolutionary Computing, pp.687-690, Shenzen, China.
9. J. Qiang-rong and L. Hua-lan, 2010, “Robust Human Face Detection in Complicated Color Images”, Proc. 2010 The 2nd IEEE International Conference 48 on Information Management and Engineering (ICIME), pp.218 – 221, Chengdu, China.
10. Z. Li, L. Xue and F. Tan, 2010, “Face Detection in Complex Background Based On Skin Color Features and Improved Adaboost Algorithms”, Proc. 2010 IEEE International Conference on Progress in Informatics and Computing (PIC), pp.723 – 727, Shanghai, China.
11. C. Aiping, P. Lian, T. Yaobin and N. Ning, 2010, “Face Detection Technology Based On Skin Color Segmentation and Template Matching”, Proc. 2010

Second International Workshop on Education Technology and Computer Science, pp.708-711, Wuhan, China.

12. P. Peer, J. Kovac and F. Solina, 2003, “Robust Human Face Detection in Complicated Color Images”, Proc. 2010 The 2nd IEEE International Conference on Information Management and Engineering (ICIME), pp. 218 – 221, Chengdu, China.
13. Y. Kun, Z. Hong and P. Ying-jie, 2006, “Human Face Detection Based On SOFM Neural Network”, Proc. 2006 IEEE International Conference on Information Acquisition, pp.1253-1257, Weihai, Shandong, China.
14. W. Chen, T. Sun, X. Yang and L. Wang, 2009, “Face Detection Based On Half Face-Template”, Proc. The Ninth International Conference on Electronic Measurement & Instruments ICEMI’2009, pp. 4-54-4-59, Beijing, China.
15. H. Guo, Y. Yu and Q. Jia, 2010, “Face Detection with Abstract Template”, Proc. 2010 3rd International Congress on Image and Signal Processing (CISP2010), pp.129-134, Yantai, China.
16. M. Ş. Bayhan and M. Gökmen, 2008, “Scale and Pose Invariant Real-Time Face Detection and Tracking”, Proc. 23rd International Symposium on Computer and Information Sciences ISCIS '08, pp.1-6, Istanbul, Turkey.
17. . J. Wang and H. Yang, 2008, “Face Detection Based On Template Matching and 2DPCA Algorithm”, Proc. 2008 Congress on Image and Signal Processing, pp.575-579, Hainan, China.
18. C.C. Tsai, W.C. Cheng, J.S. Taur and C.W. Tao, 2006, “Face Detection Using Eigenface and Neural Network”, Proc. 2006 IEEE International Conference on Systems, Man, and Cybernetics, pp.4343-4347, Taipei, Taiwan.
19. A. R. Mohan and N. Sudha, 2009, “Fast Face Detection Using Boosted Eigenfaces”, Proc. 2009 IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009), pp.102-1006, Kuala Lumpur, Malaysia.
20. D. Anijiantis, E. Dernzatas and G. Kukkinakis, 1999, “A Neural Network Method for Accurate Face Detection On Arbitrary Images”, Proc. The 6th IEEE International Conference on Electronics, Circuits and Systems, pp. 109 - 112 vol.1, Pafos, Cyprus.
21. C. Anagnostopoulos, I. Anagnostopoulos, D. Vergados, I. Papaleonidopoulos, E. Kayafas, V. Loumos and G. Stasinopoulos, 2002, “A Probabilistic Neural Network for Face Detection On Segmented Skin Areas Based On Fuzzy Rules”, Proc. IEEE MELECON 2002, pp.493-497, Cairo, Egypt.
22. X. Liu, G. Geng and X. Wang, 2010, “Automatically Face Detection Based On BP Neural Network and Bayesian Decision”, Proc. 2010 Sixth International

Conference on Natural Computation (ICNC 2010), pp.1590-1594, Shandong, China.
23. Ye-Zhengchun and Lin-Hongji, 2007, “Face Detection Based On SCNN and Wavelet Invariant Moment in Color Image”, Proc. 2007 International Conference on Wavelet Analysis and Pattern Recognition, pp. 783-787, Beijing, China.
24. M. Tayyab and M. F. Zafar, 2009, “Face Detection Using 2D-Discrete Cosine Transform and Back Propagation Neural Network”, Proc. 2009 International Conference on Emerging Technologies, pp.35-39, Islamabad, Pakistan.
25. K. A. A. Aziz and R. A. Ramlee, 2009, “Face Detection Using Radial Basis Function Neural Networks with Variance Spread Value”, Proc. 2009 International Conference of Soft Computing and Pattern Recognition, pp.399-403, Malacca, Malaysia.
26. W. Wang, Y. Gao, S. C. Hui and M. K. Leung, 2002, “A Fast and Robust Algorithm for Face Detection and Localization”, Proc. 9th International Conference on Neural Information Processing (ICONIP'O2), pp.2118-2121, Orchid Country Club, Singapore.
27. C. Shavers, R. Li and G. Lebby, 2006, “An SVM-Based Approach to Face Detection”, Proc. 38th Southeastern Symposium on System Theory, pp.362-366, Tennessee Technological University, Cookeville, TN, USA.
28. H. Jee, K. Lee and S. Pan, 2004, “Eye and Face Detection Using SVM”, Proc. ISSNIP 2004, pp.577-580, Melbourne, Australia.
29. H. Jin, Q. Liu and H. Lu, 2004, “Face detection using one-class-based support vectors”, Proc. Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FGR’04), pp.457 - 462, Seoul, Korea.
30. G. Wang and Z. Ou, 2006, “Face Recognition Based On Image Enhancement and Gabor Features”, Proc. 6th World Congress on Intelligent Control and Automation, pp.9761-9764, Dalian, China.
31. Y. Song, Y. Kim, U. Chang and H. B. Kwon, 2006, “Face Recognition Robust to Left-Right Shadows Facial Symmetry”, Pattern Recognition Vol.39 (2006), pp.1542–1545.
32. M. I. Razzak, M. K. Khan, K. Alghathbar and R. Yousaf, 2010, “Face Recognition Using Layered Linear Discriminant Analysis and Small Subspace”, Proc. 2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010), pp.1407-1412, West Yorkshire, UK.
33. F. Wang, J. Wang, C. Zhang and J, Kwok, 2007, “Face Recognition Using Spectral Features”, Pattern Recognition Vol.40 (2007), pp.2786–2797.

34. C. Liu and H. Wechsler, 2003, “Independent Component Analysis of Gabor Features for Face Recognition”, Proc. IEEE Transactions On Neural Networks, Vol. 14, pp.919-928.
35. A.H. Boualleg, Ch. Bencheriet and H. Tebbikh, 2006, “Automatic Face Recognition Using Neural Network-PCA”, Proc. 2nd Information and Communication Technologies ICTTA '06, pp. 1920 – 1925, Damascus, Syria.
36. J. Youyi and L. Xiao, 2010, “A Method for Face Recognition Based On Wavelet Neural Network”, Proc. 2010 Second WRI Global Congress on Intelligent Systems, pp.133-136, Wuhan, China.
37. K. Youssef and P. Woo, 2007, “A New Method for Face Recognition Based on Color Information and a Neural Network”, Proc. Third International Conference on Natural Computation (ICNC 2007), pp.585 – 589, Hainan, China.
38. M.R.M. Rizk and A. Taha, 2002, “Analysis of Neural Networks for Face Recognition Systems with Feature Extraction to Develop an Eye Localization Based Method”, Proc. 9th International Conference on Electronics, Circuits and Systems, pp. 847 - 850 vol.3, Dubrovnik, Croatia.
39. A. Rida and Dr BoukelifAoued, 2004, “Artificial Neural Network-Based Face Recognition”, Proc. First International Symposium on Control, Communications and Signal Processing, pp.439 – 442, Hammamet, Tunisia.
40. Z. Mu-chun, 2008, “Face Recognition Based On FastICA and RBF Neural Networks”, Proc. 2008 International Symposium on Information Science and Engineering, pp.588-592, Shanghai, China.
41. W. Wang, 2008, “Face Recognition Based On Radial Basis Function Neural Networks”, Proc. 2008 International Seminar on Future Information Technology and Management Engineering, pp.41-44, Leicestershire, UK.
42. T. S. M. Rasied, O. O. Khalifa and Y. B. Kamarudin, 2005, “Face Recognition Based On Singular Valued Decomposition and Back Propagation Neural Network”, Proc. 1st International Conference on Computers, Communications, & Signal Processing with Special Track on Biomedical Engineering CCSP 2005, pp.304 – 309, Kuala Lumpur, Malaysia.
43. D.N Pritha, L. Savitha and S.S. Shylaja, 2010, “Face Recognition by Feedforward Neural Network Using Laplacian of Gaussian Filter And Singular Value Decomposition”, Proc. 2010 First International Conference on Integrated Intelligent Computing, pp.56-61, Bangalore, India.
44. X. Wang, Q. Ruan and Y. Ming, 2010, “3D Face Recognition Using Corresponding Point Direction Measure and Depth Local Features”, Proc. ICSP 2010, pp.86-89, Beijing, China.

45. J. Harguess, S Gupta and J. K. Aggarwal, 2008, “3D Face Recognition with The Average-Half-Face”, Proc. 19th International Conference on Pattern Recognition ICPR 2008, pp.1-4, Florida, USA.
46. E. Elyan and H. Ugail, 2009, “Automatic 3D Face Recognition Using Fourier Descriptors”, Proc. 2009 International Conference on CyberWorlds, pp.246-252, Bradford, UK.
47. Y. Song, W. Wang and Y. Chen, 2009, “Research On 3D Face Recognition Algorithm”, Proc. 2009 First International Workshop on Education Technology and Computer Science, pp.47-50, Wuhan, China.
48. D. Lee and J. Liang, 2010, “A Face Detection and Recognition System Based On Rectangular Feature Orientation”, Proc. 2010 International Coriference on System Science and Engineering, pp.495-499, Taipei, Taiwan.
49. A. V. Nefian and M. H. Hayes III, 1999, “An Embedded HMM-Based Approach for Face Detection and Recognition”, Proc. 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '99, pp.3553 - 3556 vol.6, Arizona, USA.
50. H. M. El-Bakry, M. A. Abo-ElSoud and M. S. Kamel, 1999, “Automatic Face Recognition Using Neural Networks”, Proc. The Eleventh International Conference on Microelectronics ICM '99, pp.105 – 108, Kuwait.
51. R. Xu, B. Li and B. Wang, 2003, “Face Detection and Recognition Using Neural Network and Hidden Markov Models”, Proc. IEEE Int. Conf. Neural Networks & Signal Processing, pp.228-231, Nanjing, China.
52. L. H. Xuan and S. Nitsuwat, 2007, “Face Recognition in Video, A Combination of EigenFace and Adaptive Skin-Color Model”, Proc. International Conference on Intelligent and Advanced Systems 2007, pp.742-747, Kuala Lumpur, Malaysia.
53. P. Zhang, 2007, “A Video-based Face Detection and Recognition System using Cascade Face Verification Modules”, Proc. 37th IEEE Applied Imagery Pattern Recognition Workshop AIPR '08, pp.1-8, DC, USA.
54. T. Sawangsri, V. Patanavijit, and S. Jitapunkul, 2005, “Face Segmentation Based On Hue-Cr Components and Morphological Technique”, Proc. IEEE International Symposium on Circuits and Systems, ISCAS 2005, pp.5401 - 5404 Vol. 6, Kobe, Japan.
55. Papageorgiou, C., Oren, M., and Poggio, T. 1998. A general framework for object detection. In International Conference on Computer Vision.
56. Simard, P.Y., Bottou, L., Haffner, P., and LeCun, Y. (1999). Boxlets: A fast convolution algorithm for signal processing and neural networks. In M. Kearns,

S. Solla, and D. Cohn (Eds.), Advances in Neural Information Processing Systems, vol. 11, pp. 571– 577.
57. Freeman, W.T. and Adelson, E.H. 1991. The design and use of steerable filters. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(9):891–906.
58. Greenspan, H., Belongie, S., Gooodman, R., Perona, P., Rakshit, S., and Anderson, C. 1994. Overcomplete steerable pyramid filters and rotation invariance. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
59. Sung, K. and Poggio, T. 1998. Example-based learning for viewbased face detection. IEEE Patt. Anal. Mach. Intell., 20:39–51.
60. Rowley, H., Baluja, S., and Kanade, T. 1998. Neural network-based face detection. IEEE Patt. Anal. Mach. Intell., 20:22–38.
61. Osuna, E., Freund, R., and Girosi, F. 1997a. Training support vector machines: An application to face detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
62. Roth, D., Yang, M., and Ahuja, N. 2000. A snowbased face detector. In Neural Information Processing 12.
63. Freund, Y. and Schapire, R.E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt 95, Springer-Verlag, pp. 23–37.
64. Schapire, R.E., Freund, Y., Bartlett, P., and Lee, W.S. 1997. Boosting the margin: A new explanation for the effectiveness of voting methods. In Proceedings of the Fourteenth International Conference on Machine Learning.
65. Tieu, K. and Viola, P. 2000. Boosting image retrieval. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
66. Y. Freund, R. Schapire, and N. Abe, a short introduction to boosting, Journal of Japanese Society for Artificial Intelligence, 14 (1999), pp. 771–780.
67. J. Friedman, T. Hastie, and R. Tibshirani, The Elements of Statistical Learning, vol. 1, Springer Series in Statistics, 2001.
68. John, G., Kohavi, R., and Pfeger, K. 1994. Irrelevant features and the subset selection problem. In Machine Learning Conference Proceedings.
69. Webb, A. 1999. Statistical Pattern Recognition. Oxford University Press: New York.
70. Quinlan, J. 1986. Induction of decision trees. Machine Learning, 1:81–106.