Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: **«ГІБРИДИЗАЦІЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ТА ТРАДИЦІЙНИХ ДЕСКРИПТОРІВ ОЗНАК»**

Виконав: студент 2-го року навчання
освітньо-наукової програми
«Прикладна математика»,
спеціальності 113 Прикладна
математика

Миколайчик Ярослав Андрійович

Керівник: Швай Н. О.,
кандидат фіз.-мат. наук, доцент

Рецензент_____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2022

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Освітній ступінь магістр

Спеціальність 113 Прикладна математика
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
проф., д.ф.-м.н. Олійник Б. В.
“____” _____20___року

З А В Д А Н Н Я

ДЛЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Миколайчику Ярославу Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Гібридизація згорткових нейронних мереж та традиційних дескрипторів ознак
керівник роботи: Швай Надія Олександрівна, кандидат фіз.-мат. наук, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «__»_____20__ року №___
2. Строк подання студентом роботи _____
3. План роботи:

1) Наведення основних означень: нейронні та зготркові нейронні мережі, гібридизація;
2) Огляд традиційних дескрипторів ознак – SIFT та HOG;
3) Програмна реалізація наведених дескрипторів ознак;
4) Реалізація гібридів згорткових нейронних мереж з дескрипторами;
5) Порівняння та оцінка варіантів звичайних та гібридних моделей у роботі з наборами даних.

ГРАФІК ПІДГОТОВКИ МАГІСТЕРСЬКОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми та закріплення наукового керівника.	22.10.21			
2.	Отримання списку опорної літератури за темою роботи та індивідуального завдання	04.11.21			
3.	Базове ознайомлення з літературою, науковими статтями	04.11.21 – 25.11.21			
4.	Уточнення, обговорення і корекція запропонованих питань індивідуального завдання	25.11.21			
5.	Поглиблене вивчення літератури, наукових статей за пов'язаними темами, збір та узагальнення фактів, даних	*грудень - квітень*			
6.	Складання календарного графіку виконання, оформлення індивідуального завдання, написання розділів магістерської роботи	25.04.22 – 26.04.22			
7.	Написання магістерської роботи в цілому, ознайомлення з її першим варіантом наукового керівника				
	Розділ 1: Нейронні мережі та згорткові нейронні мережі. Гібридизація. Основні означення	*травень*			
	Розділ 2: Традиційні дескриптори ознак	*травень – червень*			
	Розділ 3: Реалізація та порівняння варіантів моделей і їх гібридів	*травень – червень*			
8.	Завершення написання магістерської роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	*червень*			
9.	Попередній захист магістерської роботи на засіданні кафедри	16.06.22			
10.	Подання роботи на зовнішню рецензію	26.06.22			
11.	Подання магістерської роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності і на кафедру з усіма супроводжувальними документами	30.06.22			
12.	Підготовка до захисту магістерської роботи на засіданні кафедри, написання доповіді	01.07.22 – 06.07.22			
13.	Публічний захист магістерської роботи перед екзаменаційною комісією	07.07.22			

Графік узгоджено «___» _____ 20___ р.
Науковий керівник Швай Н.О. (ПІБ)
Виконавець магістерської роботи Миколайчик Я.А. (ПІБ)

ЗМІСТ

АНОТАЦІЯ

В даній роботі було досліджено гібридизацію, а саме поєднання переваг, згорткових нейронних мереж з традиційними дескрипторами ознак SIFT та HOG. Для дескриптора HOG було запропоновано новий метод гібридизації зі згортковою нейронною мережею.

Було запрограмовано дві згорткові мережі, різні за складністю, і для кожної утворено гібриди HOG та SIFT, після чого створені моделі протестували на двох наборах даних зображень і порівняли результати між собою.

Під час експериментів гібрид SIFT показав найкращу точність в обох моделях, але був дуже повільним, особливо на більш складній архітектурі моделі. Гібрид HOG показав кращу точність у порівнянні зі звичайною моделлю у простішому варіанті архітектури, але призвів до погіршення результатів у більш складній архітектурі.

ВСТУП

Глибинне навчання за останнє десятиліття досягло неабияких висот і продовжує стрімко розвиватись. Однією з найпоширеніших архітектур моделей глибинного навчання, яка широко застосовується в таких задачах, як обробка природньої мови, текстових файлів, та, особливо, класифікація зображень і виявлення їх ознак, є архітектура згорткових нейронних мереж. Сучасні згорткові нейронні мережі можуть бути спроектовані з мільйонами параметрів і можуть бути навчені виконувати завдання все більшої складності. Також вони дуже добре здатні вивчати складні репрезентації ознак зображень. Проте навіть у них можуть бути певні обмеження. Не дивлячись на їхню потужність, вони все ще мають такі недоліки як не здатність узагальнюватись на роботу з зображеннями під новими кутами зору і нездатність вловити просторову залежність між ознаками низького рівня, для подолання яких було запропоновано створити гібридну модель згорткової нейронної мережі і дескриптора ознак SIFT [6]. Ідея гібридизації полягає у поєднанні високої здатності нейронних мереж до виявлення ознак з різноманітними перевагами дескрипторів ознак, такими, як, наприклад інваріантність до обертань, стійкість до афінних перетворень у зображенні, зміни кута зору, освітлення, шуму тощо.

Метою даної роботи є з’ясувати, чи можливо гібридизувати згорткові нейронні мережі з іншими традиційними дескрипторами ознак, окрім вже запропонованого варіанту SIFT, і визначити, наскільки така практика є доречною.

Виходячи з мети дослідження, постає наступне наукове завдання: зробити огляд традиційних дескрипторів ознак, таких як SIFT і HOG; програмно реалізувати розглянуті дескриптори ознак; реалізувати гібридні моделі згорткових нейронних мереж з дескрипторами, в тому числі з тим, який не було запропоновано раніше, а також запропонувати власний метод гібридизації; оцінити їх роботу та порівняти зі стандартними моделями.

Робота складається з трьох розділів.

Перший розділ присвячений розкриттю базових понять, пов’язаних з темою дослідження, визначенню самого поняття гібридизації, а також огляду існуючої літератури за цією темою.

У другому розділі буде зроблено огляд традиційних дескрипторів ознак, а саме SIFT та HOG, розглянуто раніше запропонований метод гібридизації SIFT, а також запропоновано власний метод гібридизації для нового дескриптора HOG.

Третій розділ присвячено програмній реалізації гібридів HOG та SIFT для двох варіантів архітектури моделей, проведено тестування цих моделей на двох наборах даних з подальшим аналізом та порівнянням продуктивності цих моделей, та інтерпретацією отриманих результатів.

РОЗДІЛ 1: Нейронні мережі та згорткові нейронні мережі. Гібридизація. Основні означення

1.1. Навчання з вчителем. Задача класифікації. Штучний нейрон

1.1.1. Навчання з вчителем

Навчання з вчителем [1] – клас задач машинного навчання, завданням якого є, маючи тренувальний набір даних, що складається з N пар вхід-вихід $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$, де x_j – вхідні значення, а кожне значення y_j згенеровано невідомою функцією $y = f(x)$, знайти функцію h, яка якомога краще апроксимує справжню функцію f. Апроксимуюча функція h називається гіпотезою, а x та y можуть бути будь-якими значеннями, зазвичай – скалярними числовими значеннями або числовими векторами.

Значення x часто називають ознаками (features), значення y – мітками (labels), а також вхідними та вихідними точками даних (data points) відповідно. Гіпотеза h є функцією-предиктором, яка залежить від певних тренованих параметрів, встановлює відповідність між вхідними і вихідними даними, і часто позначається як $\hat{y}(x, \theta)$ або $\hat{f}(x, \theta)$, де θ – параметри моделі цієї функції, що апроксимує справжню функцію $f(x)$. Її точність перевіряють використовуючи тестувальний набір (test set) даних, тобто даних, які не використовувались під час тренування моделі.

1.1.2. Задача класифікації

Класифікація – тип задач з класу навчання з вчителем, у якому вихідні значення y набувають одного з дискретних значень зі скінченного набору. Значення виходу y для заданого входу x позначає клас, якому належить x. Якщо класів два, така задача класифікації називається бінарною.

1.1.3. Штучний нейрон

Штучний нейрон [2] – основна складова частина нейронних мереж, обчислювальна одиниця, яка приймає на вхід вектор вхідних значень $\boldsymbol{x}$ розмірності k, і обчислює вихідне значення, що залежить від внутрішніх ваг (параметрів) нейрона $\boldsymbol{w}$ і його функції активації.

Ваги нейрона описуються вектором дійсних чисел довжиною $p = k + 1$, $\boldsymbol{w} = (w_0, w_1, \dots, w_p) \in \mathbb{R}^p$, вхідні значення – вектором $\boldsymbol{x} = (x_1, \dots, x_p)$.

Вихідне значення нейрона $g(\boldsymbol{w}, \boldsymbol{x})$ обчислюється за формулою:

$$g(\boldsymbol{w}, \boldsymbol{x}) = f(w_0 + w_1 x_1 + \cdots + w_k x_k),$$

де $w_0, w_1 \dots, w_k$ – ваги нейрона;

$x_1, x_2 \dots, x_k$ – вхідні значення;

де f – функція активації.

Параметр w_0 показує зміщення функції (bias), і його позначають як b. Якщо відділити b окремо від вектора ваг, $g(\boldsymbol{w}, \boldsymbol{x})$ можна переписати у наступному вигляді:

$$g(\boldsymbol{w}, \boldsymbol{x}) = f(\boldsymbol{w} * \boldsymbol{x} + b),$$

де $\boldsymbol{w}$ – вектор ваг;

$\boldsymbol{x}$ – вектор вхідних значень;

b – зміщення;

$\boldsymbol{w} * \boldsymbol{x}$ – скалярний добуток векторів $\boldsymbol{w}$ та $\boldsymbol{x}$**;**

f – функція активації.

1.2. Нейронна мережа у машинному навчанні

Нейронна мережа [3] – обчислювальна система, що має у своїй основі сукупність з'єднаних між собою вузлів – нейронів – що певною мірою моделює роботу нейронів біологічного мозку. Кожне з'єднання між нейронами, аналогічно синапсу, може передавати сигнали між ними. Штучний нейрон отримує сигнал, обробляє його і може передати іншим нейронам, з'єднаним з ним. Принцип роботи нейрона описано у пункті 1.1.3.

Нейронні мережі – типові моделі глибинного навчання. Ціллю нейронних мереж, як моделей машинного навчання, є апроксимація певної функції f, яка описує набір даних, з якими ми хочемо навчити працювати модель. У випадку задачі класифікації, $y = f(\boldsymbol{x})$ ставить вхідне значення $\boldsymbol{x}$ у відповідність певній категорії y. Нейронна мережа визначає відображення $\boldsymbol{y} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ і вивчає значення параметрів $\boldsymbol{\theta}$, що дають у результаті найкращу апроксимацію функції.

Дана робота була проведена з використанням звичайних нейронних мереж, які називаються нейронними мережами прямого поширення (feedforward neural networks). Це вид нейронних мереж, в яких інформація – сигнали нейронів – поширюється в одному напрямку, від вхідних значень до вихідних крізь проміжні обчислення, які визначаються заданням моделі. У таких мережах немає зворотних з'єднань, тобто таких, при яких проміжні значення на певному етапі моделі або вихідні значення спрямовуються назад у модель. Нейронні мережі зі зворотними з'єднаннями називаються рекурентними. Рекурентні нейронні мережі, як і інші спеціалізовані види нейронних мереж, як, наприклад, згорткові (підрозділ 1.2), що широко використовуються при роботі з зображеннями, будуються на основі звичайних нейронних мереж прямого поширення, шляхом певної їх модифікації.

Нейронні мережі прямого спрямування називаються так власне тому, що їхня структура утворюється шляхом об'єднання нейронів у шари і додавання послідовного зв'язку між шарами, об'єднання моделі їх у певний ланцюг, що може асоціюватись з направленим ациклічним графом. Наприклад, ми можемо

представити шари моделі у вигляді трьох функцій – $f^{(1)}$, $f^{(2)}$ і $f^{(3)}$ – об'єднаних у ланцюг, які сформують $\hat{f}(\boldsymbol{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right)$. В даному випадку, $f^{(1)}$ називатиметься першим або вхідним шаром мережі, $f^{(2)}$ називатиметься другим шаром, і так далі. Загальна довжина ланцюга – глибина моделі. Останній шар мережі прямого поширення називається вихідним шаром.

Під час навчання нейронної мережі, ми хочемо змусити $\hat{f}(\boldsymbol{x})$ збігтися з $f(\boldsymbol{x})$. Тренувальні дані дають приблизні приклади значень $f(\boldsymbol{x})$, визначених на різних тренувальних точках, які включають шум. Кожен приклад $\boldsymbol{x}$ супроводжується міткою $y \approx f(\boldsymbol{x})$. Тренувальні приклади напряму вказують, який результат має дати вихідний шар для кожної точки $\boldsymbol{x}$, а саме видати значення, близьке до y. Поведінка інших шарів не задається напряму тренувальними даними. Визначити поведінку шарів мережі, яка б у результаті якомога краще реалізувала апроксимацію f, без явного задання цієї поведінки тренувальними даними – завдання навчального алгоритму. Через те, що тренувальні дані не визначають напряму бажані вихідні значення кожного з цих шарів, вони називаються прихованими шарами.

Розмірність прихованих шарів визначає ширину моделі. Про шари мережі можна думати як про сукупність обчислювальних одиниць, що діють паралельно, і кожна з яких представляє собою функцію вектор-скаляр. Кожна одиниця отримує вхідні дані від багатьох інших одиниць і обчислює власне значення активації.

Нейронні мережі, а особливо сучасні глибинні нейронні мережі, є потужним інструментом дослідників для навчання з вчителем. Збільшення кількості шарів і нейронів в шарах дозволяє збільшувати складність функцій, які ці мережі можуть описати. Значна частина практичних задач, що полягають у встановленні відображення з вхідних даних на вихідні, можуть бути успішно виконані з допомогою методів глибинного навчання за умови використання моделей достатньої складності і наявності достатньо великої кількості розмічених даних.

1.3. Згорткова нейронна мережа

Згорткова нейронна мережа (Convolutional Neural Network, CNN) [3] – спеціалізований вид нейронних мереж, призначений для роботи з даними, що мають решітчасту структуру. До таких даних відносяться, наприклад, часові ряди, які можуть бути розглянуті як одновимірна решітка точок даних з певним інтервалом між ними, а також дані графічних зображень, що представляють собою двовимірну решітку з пікселів.

Такі мережі називаються «згортковими», оскільки у своїй архітектурі вони використовують математичну операцію згортки, якою в контексті згорткових нейронних мереж прийнято називати ту, яка насправді є операцією крос-кореляції. Шар нейронної мережі, який реалізує цю операцію, також називається згортковим, і нейронна мережа є згортковою, якщо вона використовує принаймні один такий шар замість звичайного щільного (dense) шару, в основі якого лежить просте матричне множення.

Типова архітектура згорткової нейронної мережі складається із кількох послідовних блоків [4]. Вхідні і вихідні дані для кожного з них є наборами матриць, що називаються мапами ознак (feature maps). Наприклад, якщо на вхід подається кольорове зображення, кожна feature map буде двовимірною матрицею числових значень, що відповідає за свій канал кольору. На виходах блоків отримуємо нові мапи ознак, обчислені з вхідних ознак. Зазвичай такі блоки обчислень поєднують у собі три послідовні шари: згортковий шар (пункт 1.3.1), шар нелінійних активацій (пункт 1.3.2) та pooling шар (пункт 1.3.3).

1.3.1. Згортка. Згортковий шар

Згорткою [3] двох функцій дійсних змінних x і w називають математичну операцію вигляду:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da.$$

За термінологією згорткових нейронних мереж, перший аргумент – функція x – називається входом (input), а другий аргумент – функція w – ядром (kernel). При роботі з нейронними мережами, дані є дискретними. Входи зазвичай представляють собою багатовимірні масиви даних, а ядра – багатовимірні масиви параметрів, тренованих моделлю. Тому на практиці операція згортки, враховуючи також те, що вона є комутативною, може бути представлена у вигляді операцій сум:

$$(K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n),$$

де I – вхідна матриця;

K – матриця ядра.

Хоча більшість бібліотек нейронних мереж реалізують не операцію згортки, а наближену до неї операцію крос-кореляції, яку прийнято називати згорткою у контексті згорткових нейронних мереж:

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Приклад згортки двох матриць наведений на рисунку нижче.

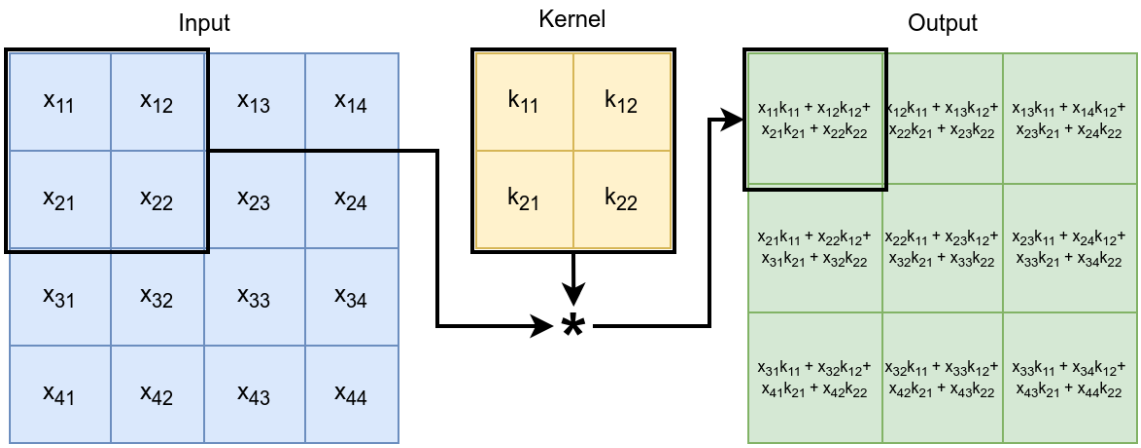

Рисунок – Приклад згортки матриці входів 4 × 4 (Input) і ядра 2 × 2 (Kernel)

Згортковий шар, у випадку мережі, призначеної для роботи із зображеннями, обчислює свої вихідні дані за наступною формулою:

$$y_j = b_j + \sum_i k_{ij} * x_i,$$

де x_i – двовимірна мапа ознак розміру $n_2 \times n_3$ з тривимірного набору n_1 вхідних мап ознак x розмірністю $n_1 \times n_2 \times n_3$;

k_{ij} – тренована матриця ваг, фільтр (filter, kernel) розміру $l_1 \times l_2$, що з'єднує вхідні мапи ознак x_i з вихідними y_j, з набору фільтрів, які утворюють конкретний згортковий шар, і кожен з яких виявляє конкретну ознаку для кожної позиції на зображенні;

$*$ – дискретний оператор згортки двовимірних матриць;

b_j – тренований параметр зміщення (bias);

y_j – двовимірна мапа ознак розміру $m_2 \times m_3$, що утворюватиме тривимірний набір m_1 вихідних мап ознак y розмірністю $m_1 \times m_2 \times m_3$.

1.3.2. Шар нелінійних активацій

Шар нелінійних активацій застосовує певну нелінійну функцію активації відносно кожного з елементів вхідних мап ознак, і на виході отримуємо нові значення мапи ознак відносно цієї функції.

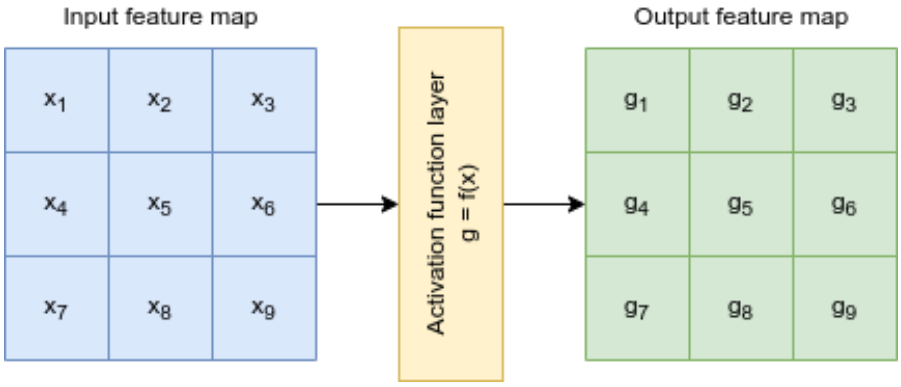

Рисунок – Схематичне зображення роботи шару активацій на вхідній мапі ознак 3×3

1.3.2. Pooling шар

Pooling [5] шар проходить з кроком заданої величини по кожній з мап ознак «вікном» певної розмірності, і обраховує функцію (наприклад, взяття максимуму або середнього значення) для області значень, розташованих у цьому вікні. Pooling є одним з ключових кроків у згорткових нейронних мережах, який зменшує розмірність мап ознак шляхом поєднання наборів значень у меншу кількість значень, намагаючись сконцентрувати цінну інформацію. Таким чином, ця операція допомагає зменшуючи обчислювальну складність на наступних шарах моделі за рахунок зменшення кількості необхідних математичних операцій на елементах мап ознак за рахунок зменшення самої кількості цих елементів, а також певною мірою вносить регуляризацію в модель і допомагає запобігти перенавчанню, відкидаючи надто специфічну інформацію про ознаки і залишаючи лише найбільш важливу.

Приклад роботи pooling з функцією максимуму (Max-Pooling) наведено на рисунку нижче.

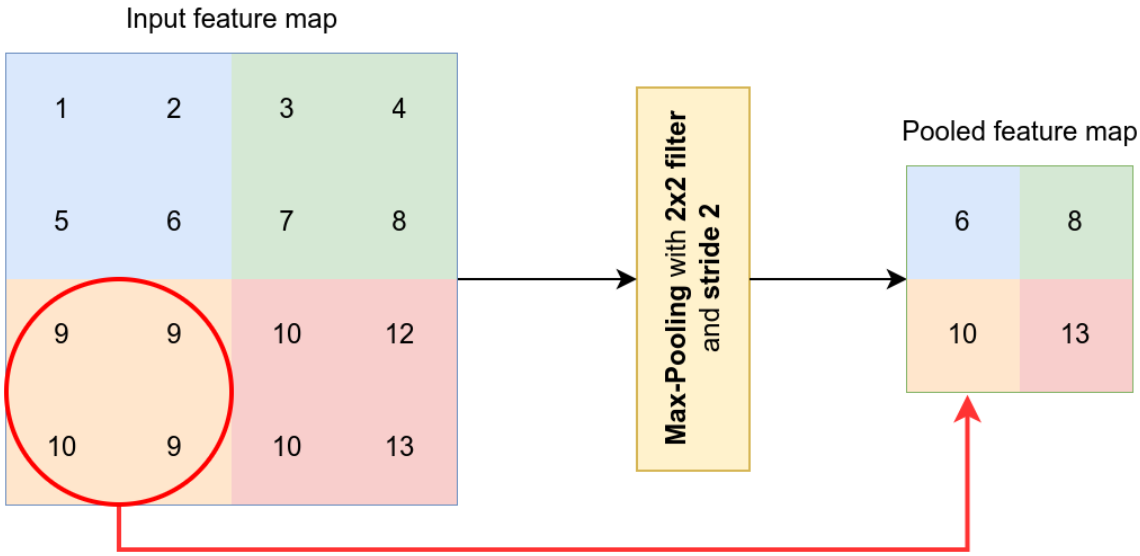

Рисунок – Приклад роботи операції Max-Pooling з розміром фільтра 2×2 і кроком (stride) 2 на вхідній мапі ознак 4×4

1.4. Гібридизація згорткових нейронних мереж з дескриптором. Огляд існуючого запропонованого методу гібридизації

Під поняттям гібридизації згорткових нейронних мереж з дескрипторами ознак в даній роботі матиметься на увазі їх поєднання шляхом певної зміни архітектури моделей згорткових мереж і включення в їх роботу дескрипторів з метою спроби одержання переваг від обох технологій в поєднаних моделях і потенційного отримання покращення метрик продуктивності моделей, головною з яких в більшості задач є точність.

У статті «Exploiting SIFT Descriptor for Rotation Invariant Convolutional Neural Network» [6] було проведено дослідження з гібридизації згорткових нейронних мереж з дескриптором ознак SIFT (Scale-Invariant Feature Transform) (пункт 2.1.1). За словами авторів, запропонована ними гібридна мережа поєднує у собі здатність виділення ознак (feature extraction) згорткової мережі і

інваріантність відносно обертання SIFT дескриптора. Автори стверджують, що за результатами експериментів з наборами даних MNIST та fashionMNIST (пункт 3.1.1) вони отримали певні покращення у порівнянні з загальноприйнятими методами, доступними в літературі. Метод гібридизації, який був застосований авторами статті, більш детально буде розглянутий у пункті 2.2.1.

РОЗДІЛ 2: Традиційні дескриптори ознак

2.1. Дескриптори ознак

Дескриптор ознак – це алгоритм, який обчислює чисельні вектори ознак (які також часто називають дескрипторами) із зображення або області зображення, і кодує важливу інформацію в цих векторах, яка може бути використана, щоб детально описати зображення або відрізнити одну ознаку зображення від іншої.

Основна мета таких алгоритмів – обчислити інформацію про ознаки, яка буде інваріантною відносно перетворень зображення, щоб ці ознаки могли бути повторно виявлені навіть коли зображення буде якимось чином трансформовано.

Дескриптори поділяються на два типи – локальний та глобальний. Локальні дескриптори обчислюють певне числове представлення околів точок, що найкраще описують зображення, які називають ключовими (keypoints), і також допомагають повторно виявляти такі точки після трансформації зображень. Глобальні дескриптори описують зображення в цілому, і представляють собою спрощене представлення зображення, що в ідеалі містить лише найбільш важливу інформацію про нього.

2.1.1. SIFT

SIFT (Scale-Invariant Feature Transform) [7] – алгоритм комп’ютерного зору, призначений для виявлення, опису і встановлення відповідності між локальними ознаками зображень. Алгоритм включає в себе чотири основні кроки, описані далі.

2.1.1.1. Крок 1. Scale-space extrema detection

Даний крок включає в себе пошук потенційних точок, інваріантних відносно масштабу та орієнтації, з використанням методу DoG (difference-of-Gaussians).

Scale space зображення – це функція $L(x, y, \sigma)$, що є результатом згортки ядра Гауса і різних масштабів вхідного зображення. Scale-space ділиться на октави, і кількість октав залежить від розміру оригінального зображення. Розмір зображення на кожній наступній октаві – половина від розміру на попередній. Всередині октав, зображення поступово розмиваються через використання ядра Гауса. Розмиті зображення отримуються за формулою:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

де G – оператор розмиття Гауса;

I – зображення;

x, y – координати зображення;

σ – масштаб.

Розмиття Гауса задається формулою:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Для подальшого виявлення потенційних ключових точок у scale space, обраховується значення функції $D(x, y, \sigma)$, що є різницею гаусіан (difference-of-Gaussians) згорнутою з зображенням, яка може бути обрахована з різниці двох сусідніх масштабів, розділених сталим коефіцієнтом k за формулою:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma).$$

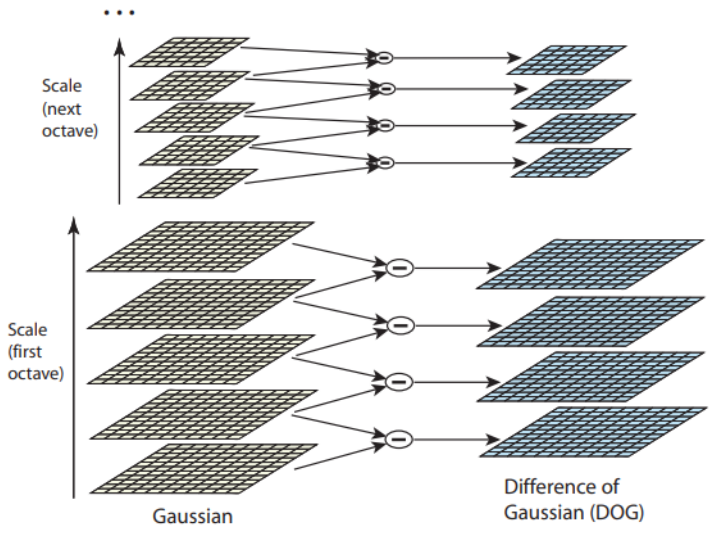

Рисунок – Схема обчислення різниці гаусіан (difference-of-Gaussians) [7]

Далі знаходяться локальні екстремуми отриманого $D(x, y, \sigma)$. Кожен піксель порівнюється з усіма сусідніми – вісьмома на поточному масштабі, і дев'ятьма на попередньому і наступному. Якщо точка є мінімумом або максимумом серед них, вона є потенційною ключовою точкою.

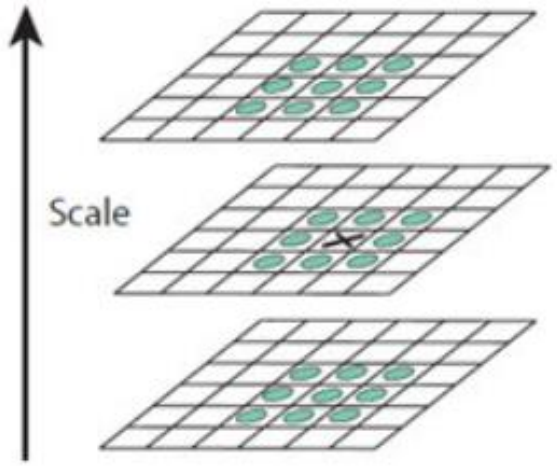

Рисунок – Схема порівняння точки (позначена хрестиком) з сусідніми [7]

2.1.1.2. Крок 2. Keypoint localization

На попередньому кроці було згенеровано багато потенційних ключових точок. Деякі з них лежать на краях або не мають достатнього контрасту. В обох випадках вони не будуть корисними як ознаки, тож на цьому кроці їх позбуваємось.

Щоб відкинути екстремуми з недостатнім контрастом, автор алгоритму використав розклад scale space в ряд Тейлора для знаходження більш точних розташувань екстремумів, і якщо інтенсивність в точці цих екстремумів менша, ніж певний поріг (в даному випадку $|D(\hat{x})| < 0.03$), точка відкидається.

Щоб позбутися крайових точок, автор скористався нерівністю

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r},$$

де **H –** матриця Гессіан розмірності 2x2 у потеційній ключовій точці;

$Tr(\boldsymbol{H})$ – слід матриці;

$Det(\boldsymbol{H})$ – визначник матриці;

r – порогове значення (було використане $r = 10$).

2.1.1.3. Крок 3. Orientation assignment

Для кожної точки на зображенні з масштабом, що відповідає масштабу ключової точки, $L(x, y)$, величина градієнта $m(x, y)$ і орієнтація $\theta(x, y)$ обраховуються, використовуючи різницю у значеннях пікселів:

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + \big(L(x,y+1) - L(x,y-1)\big)^2}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

На основі орієнтацій градієнтів точок з області навколо ключової точки формується гістограма орієнтацій, що має 36 проміжків, які покривають діапазон у 360 градусів. Кожне значення, що додається до гістограми, зважується на величину градієнта і на кругове вікно з Гаусовими вагами з σ у 1.5 разів більшим, ніж σ ключової точки. Пікові значення гістограми (найбільше і ті, що знаходяться в межах 80% від найбільшого) обираються для створення ключової точки з відповідною орієнтацією.

2.1.1.4. Крок 4. Keypoint descriptor

Останній крок – створення репрезентації дескриптора локальної області зображення, яка є дуже характерною і відмінною від інших, і є якомога більш інваріантною до таких змін в зображенні, як зміна в освітленні або точки спостереження. Наступне зображення ілюструє обчислення дескриптора ключової точки.

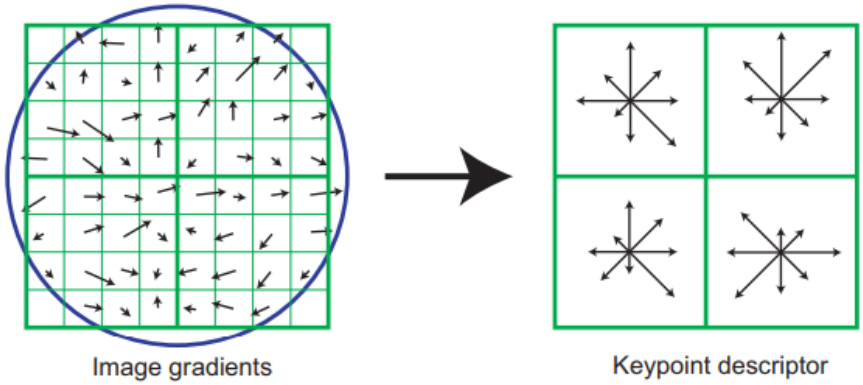

Рисунок – Ілюстрація обчислення дескриптора ключової точки [7]

Спочатку величини градієнтів зображення і їх орієнтації визначаються навколо ключової точки, використовуючи масштаб ключової точки для вибору величини розмиття Гауса.

Функція Гауса, що показана на зображенні у вигляді кругового вікна, з σ, рівним половині ширини вікна дескриптора, використовується для зважування величини градієнта кожної точки.

Дескриптор ключової точки показаний справа. На зображенні видно вісім напрямків для кожної орієнтаційної гістограми, де довжина кожної стрілки відповідає величині даного значення гістограми. Дескриптор утворюється з вектора, що містить всі значення орієнтаційних гістограм, що відповідають довжинам стрілок. У експериментах автор використав масив орієнтаційних гістограм розмірності 4×4 замість 2×2, який зображено на малюнку, з вісьмома орієнтаційними значеннями в кожній. Тож у такому випадку буде отримано $4 \times 4 \times 8 = 128$-елементний вектор ознак для кожної ключової точки.

Насамкінець, вектор ознак нормалізується до одиничної довжини, обмежується пороговим значенням 0.2 (визначене експериментально автором) і знову нормалізуєтсья до одиничної довжини для зменшення впливу зміни в освітленні.

2.1.2. HOG

HOG (Histogram of Oriented Gradients) [8] – дескриптор ознак, що використовується для виявлення об’єктів у задачах комп’ютерного зору. У основі методу лежить підрахунок напрямків градієнтів в локалізованих частинах зображення, чим він схожий на SIFT, але відрізняється тим, що обраховується на щільній сітці рівномірно розташованих клітинок і використовує перетини між ними для підвищення точності. Алгоритм складається з чотирьох кроків, описаних далі.

2.1.2.1. Крок 1. Normalize gamma & color

За наявності варто використовувати кольорові канали зображення, оскільки дескриптор HOG може призводити до менш якісних результатів при обмеженні зображення у відтінках сірого. Також варто використовувати гамма-компресію зі значенням $\gamma = \frac{1}{2}$ для можливого покращення результатів.

Окрім цього, для наступного кроку потрібно змінити розмір вхідного зображення таким чином, щоб ширина і висота ділились націло на вісім.

2.1.2.2. Крок 2. Compute gradients

На цьому кроці зображення розбивається на області розміру 8×8 пікселів, і проводиться обрахунок величини і напрямку градієнту (підпункт 2.1.1.3) для кожного пікселя.

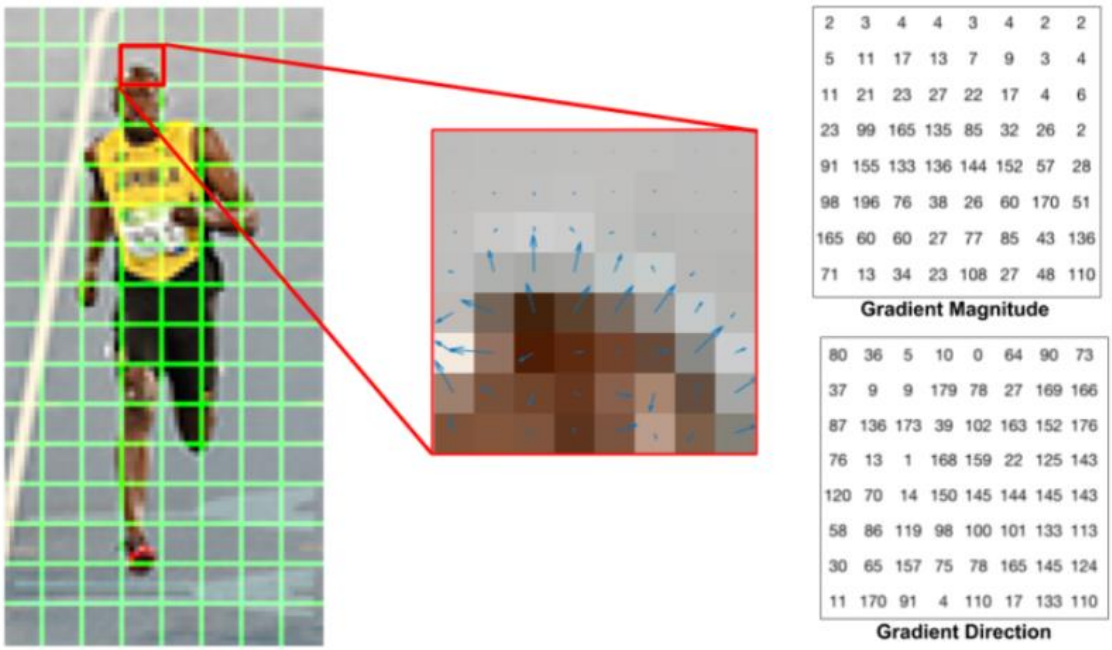

Рисунок – Ілюстрація розбиття зображення на області 8×8 і знаходження градієнтів і їх напрямів для кожного пікселя з них [9]

2.1.2.3. Крок 3. Spatial and Orientation Binning

Для кожної області, на яку було розбито зображення, створюємо гістограму орієнтацій градієнтів, яка рівномірно розбивається на дев’ять проміжків між 0 і 180 градусами. Для кожного пікселя рахується зважений «голос» для цих гістограм, а проміжок, у який додається це значення, залежить від орієнтації градієнта. Наступне зображення ілюструє цей процес.

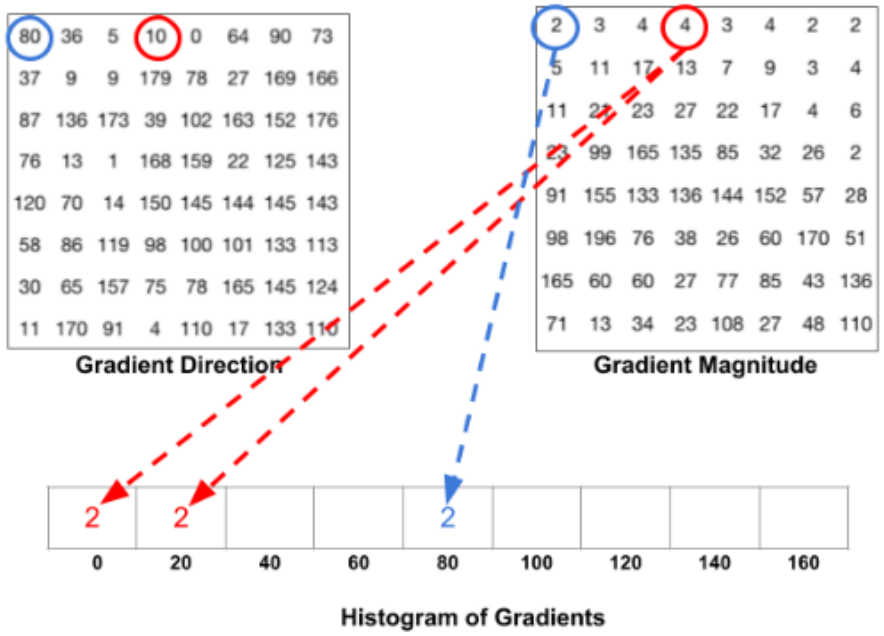

Рисунок – Побудова гістограм градієнтів областей зображення [9]

2.1.2.4. Крок 4. Normalization and Descriptor Blocks

Області об’єднуються у квадратні блоки по чотири, відповідні гістограми градієнтів конкатенуються, і утворені вектори значень гістограм нормалізуються L^2 нормою. Процедура проводиться послідовно для всіх блоків зображення, проходження квадратного вікна, що утворює блок, проілюстроване на рисунках нижче.

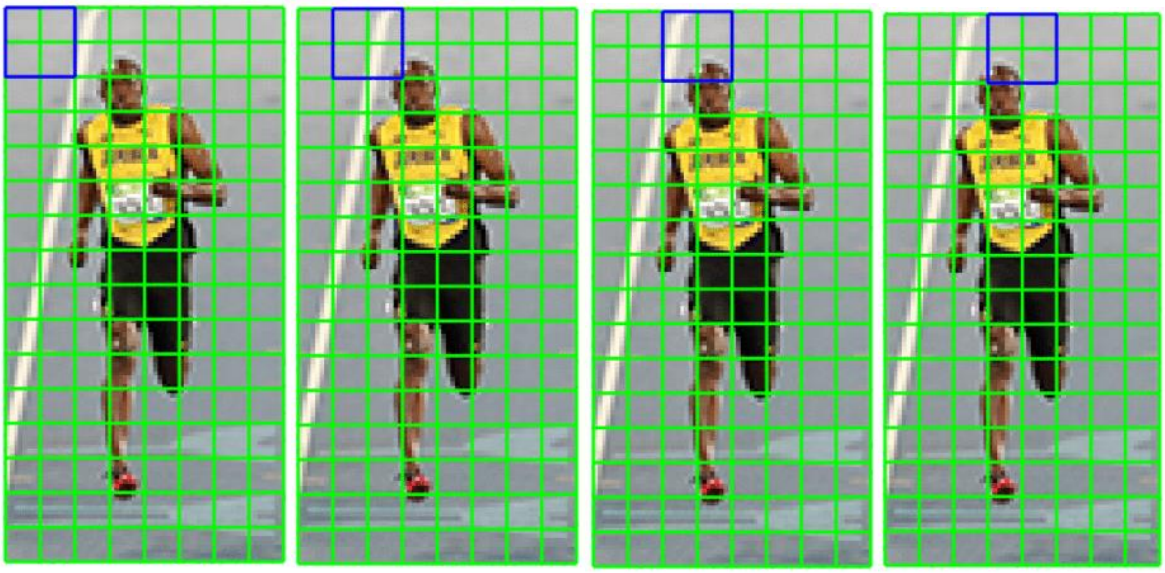

Рисунок – Проходження вікна обчислення блокових гістограм по зображенню [9]

В даному випадку, маючи розмір блоку 2×2 і гістограму з дев’яти елементів для кожної з областей, що складають блок, отримаємо на кожному кроці нормалізований вектор ознак розміру $2 \times 2 \times 9 = 36 \times 1$.

Для того, щоб отримати дескриптор всього зображення, вектори всіх блоків конкатенуються у єдиний вектор.

2.2. Методи гібридизації

2.2.1. Перший метод гібридизації для локальних дескрипторів

Перший метод гібридизації, який буде розглянуто, був запропонований у статті «Exploiting SIFT Descriptor for Rotation Invariant Convolutional Neural Network» [6]. У запропонованій автором архітектурі моделі, pooling шари заміняються SIFT дескриптором, розташованим перед щільним (dense) шаром моделі. Автор мотивує це тим, що застосування традиційних pooling шарів вносить певні обмеження і недоліки у ознаки, виявлені моделями на етапах згорток (наприклад, втрата просторових взаємозв’язків між ними, що заважає моделям узагальнюватись на зображення під новими точками зору), які він

пропонує подолати шляхом використання SIFT через його інваріантність відносно масштабування та обертання.

Даний метод гібридизації доречно застосовувати з локальними дескрипторами ознак, як SIFT, що працюють у околах ключових точок. Задачу знаходження ключових точок (ознак) у цьому випадку буде покладено на згорткову нейронну мережу, оскільки подібні моделі вдало її виконують. А задачу опису цих ознак (feature description) і, власне, створення векторів-дескрипторів, буде виконувати локальний дескрипторний алгоритм.

Для порівняння, на рисунках нижче наведена типова схема згорткової нейронної мережі і схема гібридної за першим методом.

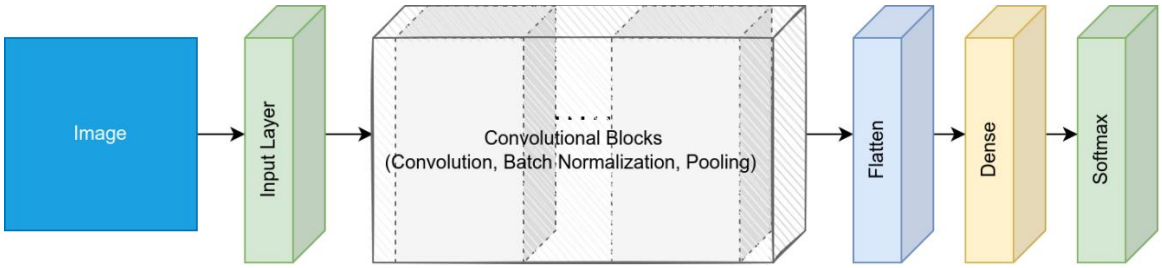

Рисунок – Типова схема згорткової нейронної мережі

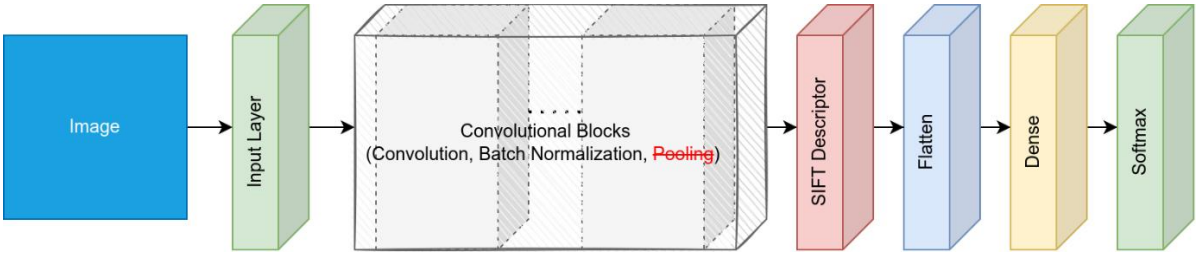

Рисунок – Схема гібридної (перший метод) згорткової мережі

В основі тренування моделей нейронних мереж лежить алгоритм градієнтного спуску, який використовує градієнти моделі, обчислені під час процесу, що називається backpropagation [10]. Тому для того, щоб реалізувати цей метод гібридизації, за яким дескриптори напряму послідовно включаються в архітектуру моделі, алгоритми дескрипторів повинні бути реалізовані засобами бібліотек машинного навчання, на основі яких будуються моделі, таких як Keras

[11] чи Tensorflow [12], оскільки для програмних математичних функцій цих бібліотек вже передбачений процес обчислення градієнтів у backpropagation.

2.2.2. Другий метод гібридизації для глобальних дескрипторів

Для тестування гібридних моделей з глобальними дескрипторами мною буде запропонований другий метод гібридизації, що полягає у використанні дескрипторів паралельно з нейронними мережами (на відміну від їх послідовного включення в архітектуру моделей згорткових мереж, як було описано у першому методі) з подальшим поєднанням векторів ознак, отриманих окремо від моделі і від дескриптора, і використанні об'єднаної сукупності ознак в щільному шарі на кінцевому етапі моделі. Це зумовлено тим, що глобальні дескриптори, на відміну від локальних, працюють на цілому зображенні, а не в околах ключових точок, і тому не повинні потребувати використання ознак, виявлених мережею. Структуру нейронної мережі при цьому, в тому числі наявність pooling шарів, залишатимемо майже без змін, окрім вхідного шару, через який надходять зображення, який потрібно приєднати до дескриптора, і щільного шару, до якого буде приєднаний дескриптор і в який надходитимуть, поряд з ознаками, виявленими згортковою мережею, вихідні вектори ознак глобального дескрипторного алгоритму.

Схему гібридної нейронної мережі за другим методом гібридизації показано на рисунку нижче.

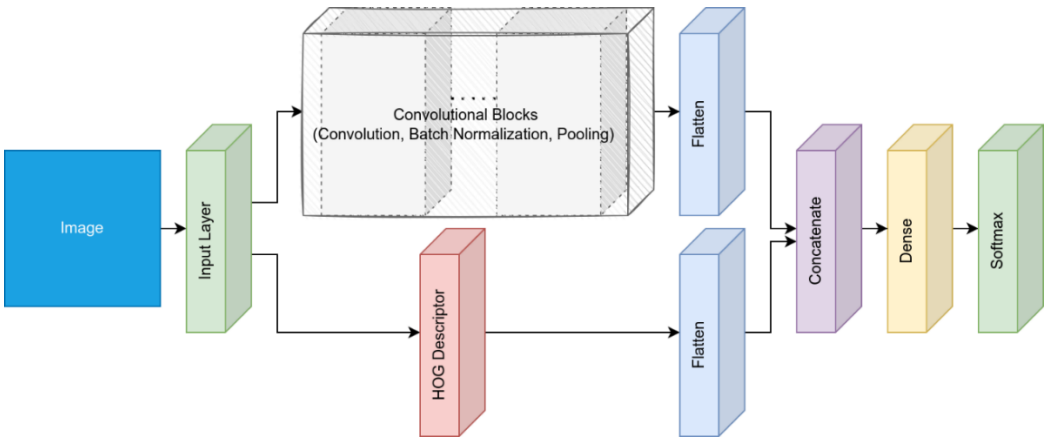

Рисунок – Схема гібридної (другий метод) згорткової мережі

РОЗДІЛ 3: Реалізація та порівняння варіантів моделей і їх гібридів

3.1. Набори тестувальних даних

3.1.1. Fashion-MNIST

Набір даних Fashion-MNIST [13, 14] – набір фронтальних зображень предметів одягу, розроблений як альтернатива набору MNIST (Modified National Institute of Standards and Technology [15]), що широко застосовується у дослідженнях в галузі машинного навчання, і разом з ним є одним зі стандартних наборів даних для здійснення швидкого проектування та порівняння якості роботи різних моделей. Fashion-MNIST зберігає ту саму простоту у застосуванні і можливість використання для швидкого тестування досліджуваних моделей, що й традиційний MNIST, але є порівняно більш складним у плані задачі класифікації зображень.

Цей набір даних складається з 60000 тренувальних і 10000 тестувальних зображень, що утворюють 10 класів предметів одягу. Кожне зображення є чорно-білим і має розмір 28x28 пікселів. Fashion-MNIST наявний для використання у бібліотеці Keras. Приклади зображень набору наведені на наступному рисунку.

Рисунок – Приклади зображень з набору Fashion-MNIST [16]

3.1.2. CIFAR-10

Другим набором даних, обраним для тестування моделей, є CIFAR-10 [17]. До цього входять з 60000 кольорових зображень розміром 32x32 пікселів, що належать до 10 класів, з 6000 зображеннями у кожному класі. Всього у наборі 50000 тренувальних зображень і 10000 тестувальних і він також представлений для використання у бібліотеці Keras. Зображення взаємно виключно належать єдиним конкретним класам і не перетинаються з іншими класами. Приклади зображень набору CIFAR-10 наведені на рисунку нижче.

Рисунок – Приклади зображень з набору CIFAR-10 [17]

3.2. Тестувальні нейронні мережі

3.2.1. Модель №1 для Fashion-MNIST

Дана модель призначена для роботи зі зображеннями набору даних Fashion-MNIST.

Кожен з варіантів моделі починається двома згортковими шарами з ядрами розміру 3×3, які на виході дають 32 і 64 мапи ознак відповідно. За згортковими шарами слідує шар batch нормалізації, після якого знаходиться max-pooling шар з ядром розмірністю 2×2, який у варіанті гібридної моделі SIFT замінений на цей алгоритм. Далі знаходиться шільний шар з 128 нейронами, dropout зі значенням коефіцієнту $rate = 0.5$ і класифікатор softmax. У варіанті гібриду HOG, паралельно з ознаками моделі обчислюються ознаки цього дескриптора і також надходять у шільний шар. Функції активації, застосовані у всіх згорткових і щільних шарах, – ReLU, а значення padding – “valid”.

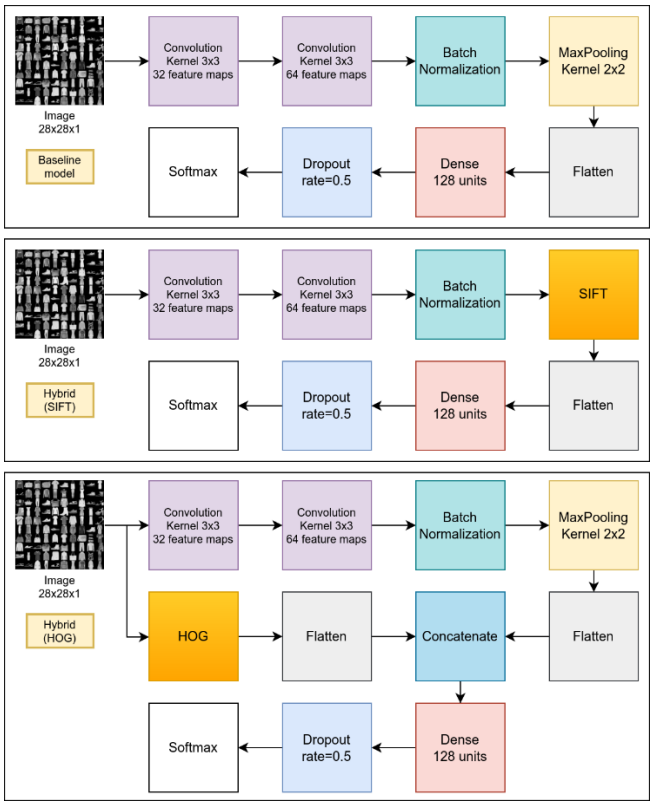

Рисунок – Схеми варіантів моделі №1: базова (baseline), гібрид SIFT (Hybrid SIFT) і гібрид HOG (hybrid HOG)

3.2.2. Модель №2 для CIFAR-10

Дана модель призначена для роботи зі зображеннями набору даних CIFAR.

Кожен з її варіантів починається з двох блоків згорткових шарів, які складаються з трьох згорток з ядрами розміру 3×3, що обчислюють 16, 32 і 64 мапи ознак. Кожен з блоків замикаються batch нормалізацією та max-pooling шарами 2×2 (окрім гібриду SIFT, в якому pooling не використовується). Після них слідує щільний шар з 256 нейронами і dropout зі значенням коефіцієнту $rate = 0.5$, а у гібриді HOG до нього приєднаний сконкатенований вектор ознак згорткової мережі і дескриптора. Замикають модель два щільні шари з 128 та 64 нейронами і softmax класифікатор. Функції активації, застосовані у всіх згорткових і щільних шарах, – ReLU, а значення padding – “same”.

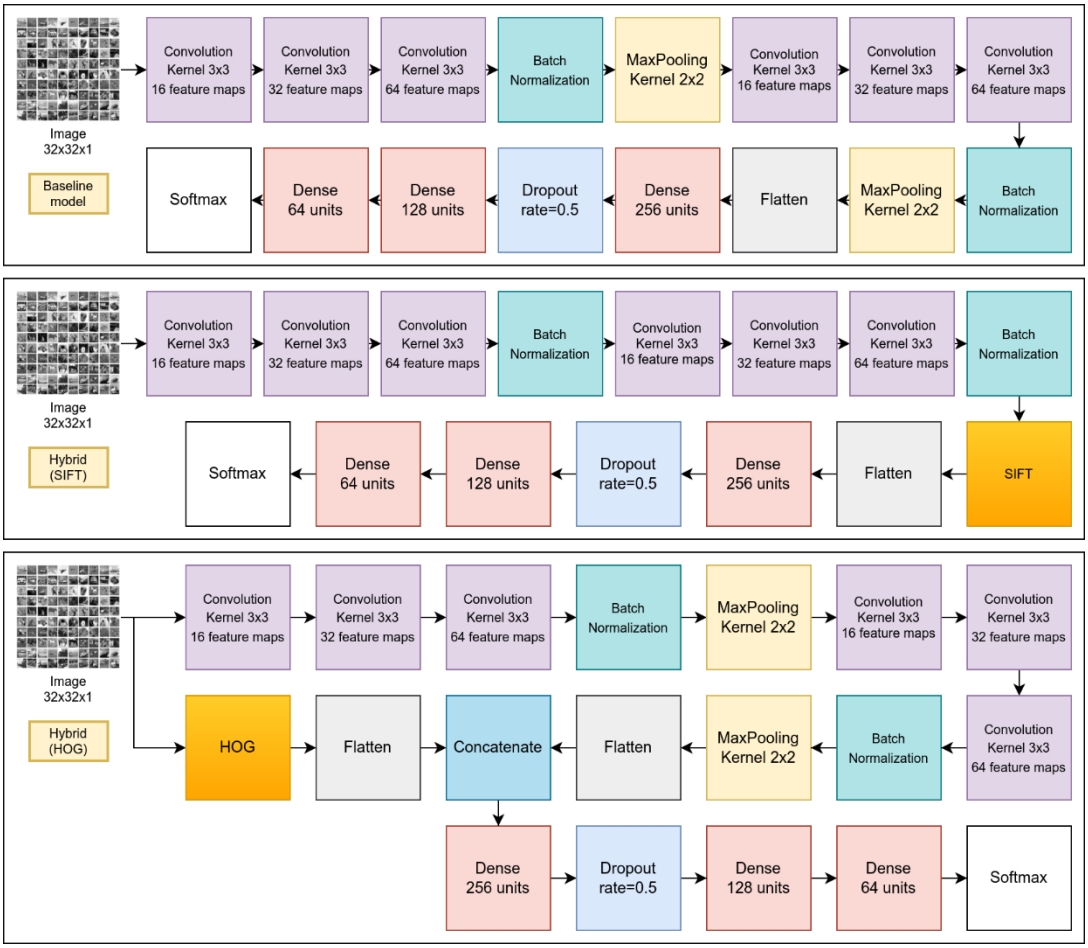

Рисунок – Схеми варіантів моделі №2: базова (baseline), гібрид SIFT (Hybrid SIFT) і гібрид HOG (hybrid HOG)

3.3. Порівняння та оцінка варіантів моделей CNN та їх гібридів

У ході дослідження було натреновано шість варіантів згорткових нейронних мереж – по три для кожного з наборів даних fashionMNIST та CIFAR-10. Тренування кожної моделі відбувалось 20 епох з використанням оптимізаційного алгоритму Adam[18, 19] зі значенням $learning\ rate = 0.001$.

На кожному з етапів тренування проводилось вимірювання і спостереження за передбачувальною точністю моделей на тренувальних даних (accuracy), а також на нових, тестувальних даних, які модель до цього не спостерігала, у якості її валідації (validation accuracy, val_accuracy). Одним з головних показників продуктивності моделі є саме її здатність правильно передбачати (класифікувати) нові дані, що в нашому випадку і оцінюватиме величина значення валідаційної точності, яка була записана на кожній епосі разом з тренувальною точністю. Розглянемо їх графіки для наших варіантів моделей.

3.3.1. Графіки тренування, швидкість тренування та точність передбачення

На наступних рисунках зображено графіки зміни під час тренування величин тренувальної та валідаційної точності для кожного з варіантів моделей №1 та №2. Графіки базової моделі позначені синім, гібриду SIFT – помаранчевим, гібриду HOG – зеленим.

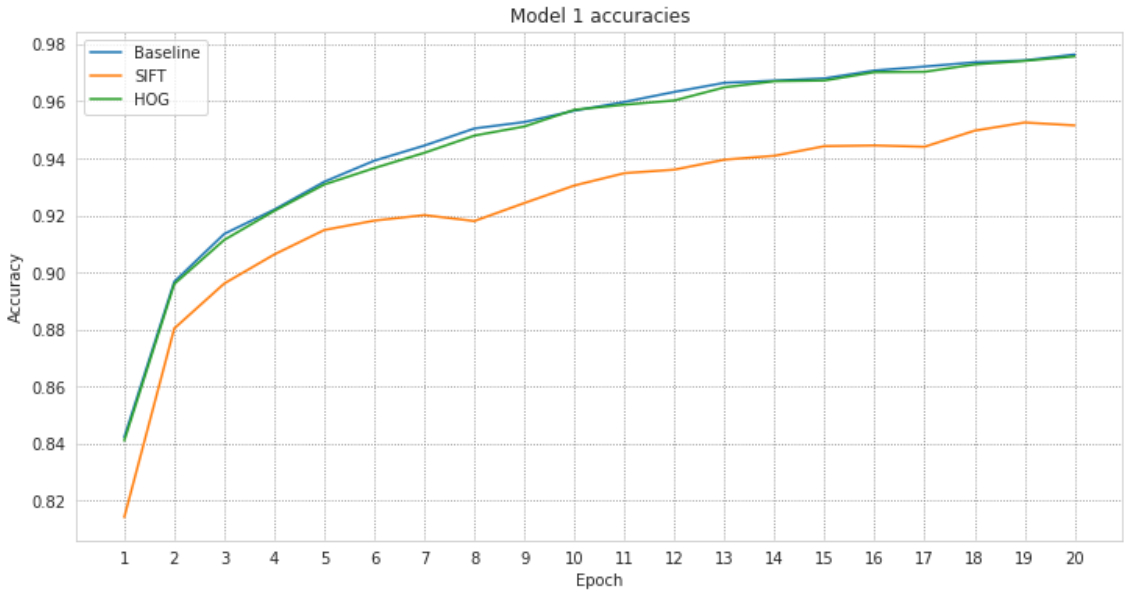

Рисунок – Графік тренувальної точності варіантів моделі №1 відносно тренувальних епох

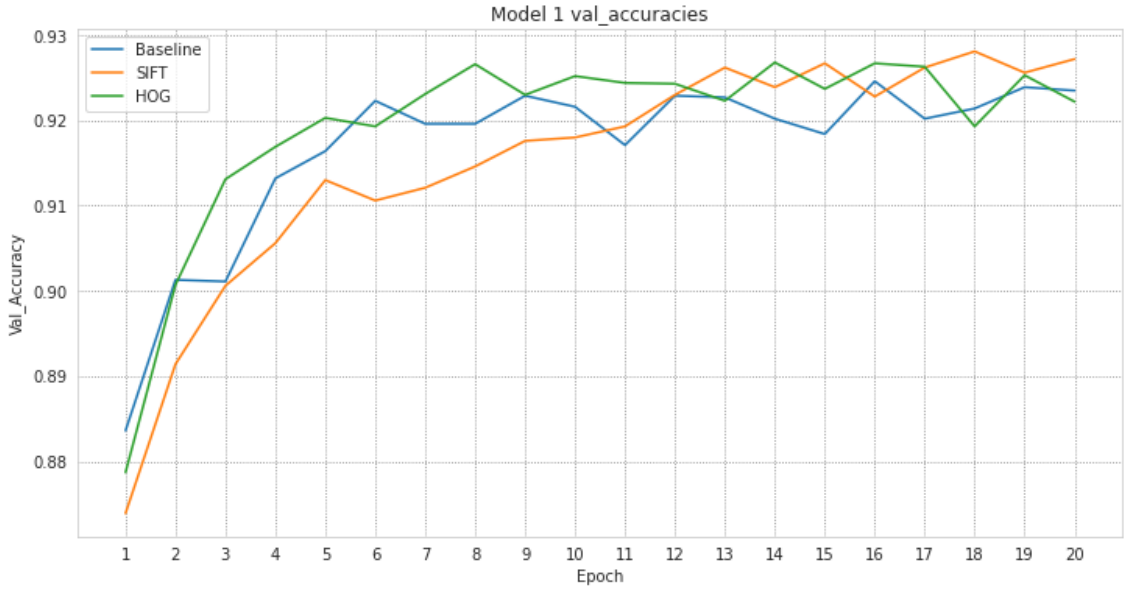

Рисунок – Графік валідаційної точності варіантів моделі №1 відносно тренувальних епох

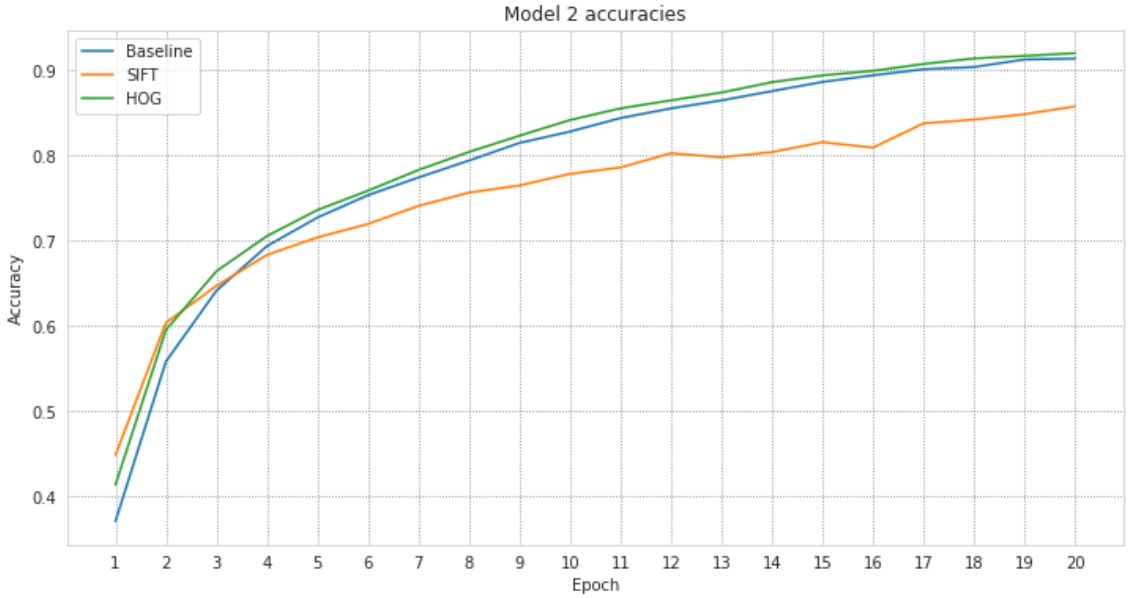

Рисунок – Графік тренувальної точності варіантів моделі №2 відносно тренувальних епох

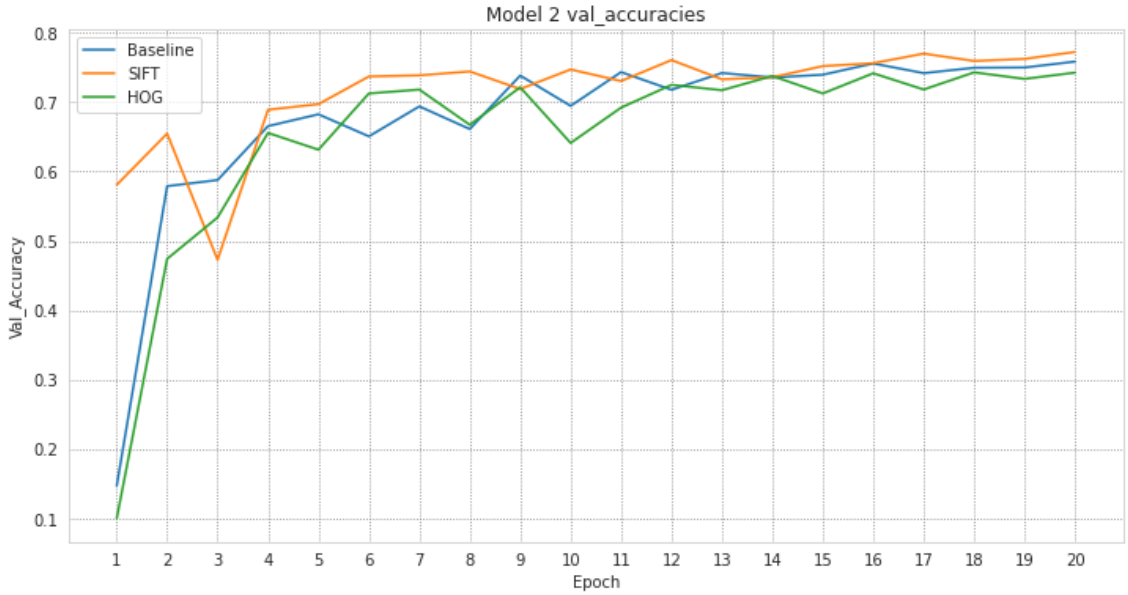

Рисунок – Графік валідаційної точності варіантів моделі №2 відносно тренувальних епох

На наступній таблиці для кожної з моделей наведено пікові значення тренувальних і валідаційних точностей, середній час тренування однієї епохи та їх відсоткова різниця відносно відповідних значень базових моделей.

Таблиця – Порівняння числових показників продуктивності стандартних моделей з гібридними

Тип моделі	Пікова тренувальна точність	Δ , %	Пікова валідаційна точність	Δ , %	Середній час тренування епохи	Δ , %
Модель 1,базова	0.9764		0.9246		2.9 с	
Модель 1, SIFT	0.9525	-2.39%	0.9281	+0.35%	57.7 с	+1990%
Модель 1, HOG	0.9757	-0.07%	0.9268	+0.22%	6.1 с	+210%
Модель 2,базова	0.9134		0.7582		4.3 с	
Модель 2, SIFT	0.8572	-5.62%	0.7721	+1.39%	92.3 с	2147%
Модель 2, HOG	0.9198	+0.64%	0.7427	-1.55%	7 с	+163%

3.3.2. Інтерпретація результатів та порівняльний аналіз моделей

Як можемо бачити, гібридна модель SIFT показала найкращий результат за показником валідаційної точності для обох варіантів моделей, не дивлячись на те, що за тренувальною точністю вона показує найгірший результат. Проте, час тренування цієї моделі, у порівнянні з базовою, займає приблизно у 20 разів більше часу у представлених прикладах. І чим більше згорткова нейронна мережа надаватиме мап ознак після згорткових шарів на вхід в дескриптор SIFT (у нашому випадку це було 32 мапи для моделі №1 і 64 мапи для моделі №2), тим

повільнішою буде робота моделі, оскільки SIFT обчислює дескрипторні вектори для кожної такої мапи ознак як для ключової точки, що потребує інтенсивних обчислень при великій кількості цих ознак. Тому використання подібної моделі може бути доречним лише у відносно нескладних згорткових мережах, або коли можливості на затрату часу не є обмеженими, а головною ціллю є покращення точності моделі.

Розглянемо результати гібридних моделей HOG. Як бачимо, гібрид моделі №1 показує кращий результат, ніж базова модель, а час його тренування є більш прийнятним і не настільки значною мірою відрізняється від базової моделі – приблизно у 1,5-2 рази, що свідчить про те, що така модель працює і показує прийнятний результат. А отже, можемо зробити висновок, що інші традиційні дескриптори, окрім SIFT, також можуть бути застосовані для створення гібридних моделей згорткових нейронних мереж, принаймні у деяких ситуаціях, як і запропонований у цій роботі другий метод гібридизації глобальних дескрипторів. Гібрид HOG моделі №2 ж навпаки, показав гірший результат. Це може бути пов’язано з більшою складністю моделі, в яку ми включаємо цей дескриптор, що негативно на неї впливає. Чим глибшою і складнішою є модель, тим більш комплексні ознаки зображень вона може вивчити. У порівнянні з такими ознаками, ті, що обраховує дескриптор HOG є відносно більш простими, менш інформативними і їх додавання у модель, яка замість цього могла б отримати зі згорткових шарів значно якісніші ознаки, негативно впливає на її продуктивність. Тому HOG може бути доцільно використовувати у відносно простих мережах, які вивчають більш прості ознаки і можуть отримати користь від додаткових ознак дескриптора.

ВИСНОВОК

Отже, в ході даної роботи було проведено огляд традиційних дескрипторів ознак SIFT та HOG. Було програмно створено гібрид SIFT за вже існуючим методом гібридизації, а також створено новий гібрид моделі згорткової нейронної мережі з дескриптором HOG за допомогою нового запропонованого методу гібридизації. Створені гібридні і стандартні моделі порівняли між собою за показниками передбачувальної точності та швидкості тренування. При цьому з'ясували, що гібридизувати згорткові нейронні мережі з іншими традиційними дескрипторами ознак, окрім вже запропонованого в літературі варіанту SIFT, можливо, проте, як показали експериментальні дослідження, не завжди доречно. Найкраще гібриди себе проявили у більш простій архітектурі згорткових нейронних мереж, перевершивши стандартну. Гібридизація складних і глибоких мереж з дескрипторами не є доречною, оскільки призводить до значного сповільнення тренування моделі або до погіршення якості її роботи.

СПИСОК ЛІТЕРАТУРИ

1. Russel S., Norvig P. Artificial intelligence: a modern approach. 3rd ed. Prentice Hall, 2010. 1132 p.
2. Anthony M. Discrete mathematics of neural networks: selected topics. Society for Industrial and Applied Mathematics, 2001. 131 p.
3. Goodfellow I., Bengio Y., Courville A. Deep learning. MIT Press, 2016. 800p.
4. LeCun Y., Kavukcuoglu K., Farabet C. Convolutional networks and applications in vision. Proceedings of 2010 IEEE international symposium on circuits and systems, Paris, France, 30 May – 2 June 2010. 2010. URL: https://doi.org/10.1109/ISCAS.2010.5537907.
5. Gholamalinezhad H., Khosravi H. Pooling methods in deep neural networks, a review. CoRR. 2020. URL: https://doi.org/10.48550/arXiv.2009.07485.
6. Exploiting SIFT descriptor for rotation invariant convolutional neural network / A. Kumar et al. 2018 15th IEEE India council international conference (INDICON), Coimbatore, India, 16–18 December 2018. 2018. URL: https://doi.org/10.1109/indicon45594.2018.8987153.
7. Lowe D. G. Distinctive image features from scale-invariant keypoints. International journal of computer vision. 2004. Vol. 60, no. 2. P. 91–110. URL: https://doi.org/10.1023/b:visi.0000029664.99615.94.
8. Dalal N., Triggs B. Histograms of oriented gradients for human detection. 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), San Diego, CA, USA. URL: https://doi.org/10.1109/cvpr.2005.177.
9. Mallick S. Histogram of Oriented Gradients explained using OpenCV. LearnOpenCV. URL: https://learnopencv.com/histogram-of-oriented-gradients.
10. Rumelhart D. E., Hinton G. E., Williams R. J. Learning internal representations by error propagation. Fort Belvoir, VA: Defense Technical Information Center, 1985. URL: https://doi.org/10.21236/ada164453.

11. Keras: The Python deep learning API. URL: https://keras.io.
12. TensorFlow. URL: https://www.tensorflow.org/.
13. Xiao H., Rasul K., Vollgraf R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. CoRR. 2017. URL: https://doi.org/10.48550/arXiv.1708.07747.
14. Fashion-MNIST. A MNIST-like fashion product database. GitHub. URL: https://github.com/zalandoresearch/fashion-mnist.
15. Deng L. The MNIST database of handwritten digit images for machine learning research [Best of the Web]. IEEE signal processing magazine. 2012. Vol. 29, no. 6. P. 141–142. URL: https://doi.org/10.1109/msp.2012.2211477.
16. Figure 7 Fashion-MNIST dataset. ResearchGate. URL: https://www.researchgate.net/figure/Fashion-MNIST-dataset_fig4_349913991.
17. Krizhevsky A. CIFAR-10 and CIFAR-100 datasets. Department of Computer Science, University of Toronto. URL: https://www.cs.toronto.edu/~kriz/cifar.html.
18. Keras documentation: Adam. Keras: The Python deep learning API. URL: https://keras.io/api/optimizers/adam/.
19. Kingma D., Ba J. Adam: a method for stochastic optimization. 3rd international conference on learning representations, San Diego, 7–9 May 2015. URL: https://doi.org/10.48550/arXiv.1412.6980.
20. PyTorch-SIFT. PyTorch implementation of SIFT descriptor. GitHub. URL: https://github.com/zalandoresearch/fashion-mnist.
21. tf-HOG. The histogram of oriented gradients (HOG) implemented in TensorFlow. GitHub. URL: https://github.com/KCC13/tf-HOG.

ДОДАТОК А (обов'язковий)

Лістинг програмного коду

```
# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T15:26:37.147989Z","iopub.execute_input":"2022-06-26T15:26:37.149103Z","iopub.status.idle":"2022-06-
26T15:26:48.98329Z","shell.execute_reply.started":"2022-06-
26T15:26:37.14844Z","shell.execute_reply":"2022-06-
26T15:26:48.982019Z"},"jupyter":{"outputs_hidden":false}}
import numpy as np
import pandas as pd
import math

import cv2
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_style("whitegrid", {"grid.color": "grey", "grid.linestyle": ":"})
import pickle

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.datasets import fashion_mnist, cifar10
from keras import backend as K
from keras.models import Model
from keras.layers import Flatten, Activation, Input
from keras.layers import Concatenate
from keras.layers import Add, Multiply, Subtract, Conv2D, DepthwiseConv2D, Lambda, Reshape,
ZeroPadding2D

np.random.seed(1)
tf.random.set_seed(1)

# %% [markdown]
# ## Utility functions

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:15:45.042094Z","iopub.execute_input":"2022-06-26T09:15:45.042499Z","iopub.status.idle":"2022-06-
26T09:15:45.059919Z","shell.execute_reply.started":"2022-06-
26T09:15:45.042467Z","shell.execute_reply":"2022-06-
26T09:15:45.058753Z"},"jupyter":{"outputs_hidden":false}}
def rad2deg(rad):
    return 180 * rad / tf.constant(np.pi)

def get_circular_gaussian(kernlen = 21):
    halfSize = kernlen / 2
    r2 = halfSize**2
    sigma_mul_2 = 0.9 * r2
    disq = 0;
    kernel = np.zeros((kernlen, kernlen))
    for y in range(kernlen):
        for x in range(kernlen):
            disq = (y - halfSize + 0.5)**2 + (x - halfSize + 0.5)**2
            if disq < r2:
                kernel[y,x] = math.exp(-disq / sigma_mul_2)
            else:
                kernel[y,x] = 0
    return kernel

def get_sift_pooling_kernel(ksize = 25):
    ks_2 = float(ksize) / 2.0
    xc2 = ks_2 - np.abs((np.arange(ksize) + 0.5 - ks_2))
    kernel = np.outer(xc2, xc2) / (ks_2 ** 2)
    return kernel

def get_sift_bin_ksize_stride_pad(patch_size, num_spatial_bins):
    ksize = 2 * int(patch_size / (num_spatial_bins + 1))
    stride = patch_size // num_spatial_bins
    pad = ksize // 4
    return ksize, stride, pad

def grad_x_initializer(shape, dtype=None):
    return tf.constant(np.reshape([-1, 0, 1], newshape=(1,3,1,1)).repeat(repeats=shape[2], axis=2),
shape=shape, dtype=dtype)
```

```
def grad_y_initializer(shape, dtype=None):
    return tf.constant(np.reshape([-1, 0, 1], newshape=(3,1,1,1)).repeat(repeats=shape[2], axis=2),
shape=shape, dtype=dtype)

def pk_initializer(shape, dtype=None):
    nw = get_sift_pooling_kernel(ksize=shape[0])
    return tf.constant(np.reshape(nw, newshape=(shape[0],shape[0],1,1)).repeat(repeats=shape[2],
axis=2), shape=shape, dtype=dtype)

# %% [markdown]
# # Descriptors

# %% [markdown]
# ## SIFT

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:15:52.731883Z","iopub.execute_input":"2022-06-26T09:15:52.732303Z","iopub.status.idle":"2022-06-
26T09:15:52.754703Z","shell.execute_reply.started":"2022-06-
26T09:15:52.732271Z","shell.execute_reply":"2022-06-
26T09:15:52.753667Z"},"jupyter":{"outputs_hidden":false}}
def get_sift_model(inputs, patch_size = 16, num_ang_bins = 8, num_spatial_bins = 4, clipval = 0.2):
    eps = 1e-10
    gk = tf.Variable(get_circular_gaussian(kernlen=patch_size), trainable=False, dtype=tf.float32)
    bin_ksize, bin_stride, pad = get_sift_bin_ksize_stride_pad(patch_size, num_spatial_bins)

    gx = DepthwiseConv2D(kernel_size=(1, 3), use_bias=False, trainable=False, padding='valid',
                depthwise_initializer=grad_x_initializer)
    gx = gx(tf.pad(inputs, paddings = [[0, 0], [0, 0], [1, 1], [0, 0]], mode='REFLECT'))

    gy = DepthwiseConv2D(kernel_size=(3, 1), use_bias=False, trainable=False, padding='valid',
                depthwise_initializer=grad_y_initializer)
    gy = gy(tf.pad(inputs, paddings = [[0, 0], [1, 1], [0, 0], [0, 0]], mode='REFLECT'))

    nw = get_sift_pooling_kernel(bin_ksize)
    pk = DepthwiseConv2D(kernel_size=(nw.shape[0], nw.shape[1]), use_bias=False, trainable=False,
                         strides = (bin_stride, bin_stride),
                         padding = 'valid', depthwise_initializer=pk_initializer)

    magnitude = tf.sqrt(gx * gx + gy * gy + 1e-10)
    orientation = tf.atan2(gy, gx + 1e-10)
    magnitude = tf.multiply(magnitude, tf.expand_dims(tf.repeat(tf.expand_dims(gk, 2),
magnitude.shape[-1], axis=2), axis=0))

    o_big = (orientation + 2 * math.pi) / (2 * math.pi) * num_ang_bins

    bo0_big = tf.floor(o_big) % num_ang_bins
    bo1_big = (bo0_big + 1) % num_ang_bins

    wo0_big = (1 - (o_big - tf.floor(o_big))) * magnitude
    wo1_big = (o_big - tf.floor(o_big)) * magnitude

    ang_bins = []

    for i in range(0, num_ang_bins):
        pk_input = tf.pad(tf.cast(bo0_big == i, dtype=tf.float32) * wo0_big + tf.cast(bo1_big == i,
dtype=tf.float32) * wo1_big,
                          paddings = [[0, 0], [pad, pad], [pad, pad], [0, 0]])
        out = pk(pk_input)
        ang_bins.append(out)

    ang_bins = Concatenate()(ang_bins)
    ang_bins = Flatten()(ang_bins)
    ang_bins = Lambda(lambda x: K.l2_normalize(x, axis=1))(ang_bins)
    ang_bins = Lambda(lambda x: tf.clip_by_value(x, 0, clipval))(ang_bins)
    ang_bins = Lambda(lambda x: K.l2_normalize(x, axis=1))(ang_bins)
    return ang_bins

def add_sift(x):
    patch_size = x.shape[1]
    x = get_sift_model(inputs=x, patch_size=patch_size)
    return x

# %% [markdown]
# ## HOG

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:15:54.711141Z","iopub.execute_input":"2022-06-26T09:15:54.711629Z","iopub.status.idle":"2022-06-
26T09:15:54.732692Z","shell.execute_reply.started":"2022-06-
```

```
26T09:15:54.711593Z","shell.execute_reply":"2022-06-
26T09:15:54.731209Z"},"jupyter":{"outputs_hidden":false}}
def get_hog_model(image, cell_size=8, block_size=2, block_stride=1, n_bins=9):

    _, height, width, depth = image.shape
    batch_size = tf.shape(image)[0]
    scale_factor = tf.constant(180 / n_bins, dtype=tf.float32)

    if (height % cell_size) != 0 or (width % cell_size) != 0:
        height = height + (cell_size - (height % cell_size)) % cell_size
        width = width + (cell_size - (width % cell_size)) % cell_size
        image = tf.image.resize_with_crop_or_pad(image, height, width)

    g_x = Conv2D(filters=1, kernel_size=(1, 3), use_bias=False, trainable=False, padding='valid',
                 kernel_initializer=grad_x_initializer)
    g_x = g_x(tf.pad(image, paddings = [[0, 0], [0, 0], [1, 1], [0, 0]], mode='SYMMETRIC'))

    g_y = Conv2D(filters=1, kernel_size=(3, 1), use_bias=False, trainable=False, padding='valid',
                 kernel_initializer=grad_y_initializer)
    g_y = g_y(tf.pad(image, paddings = [[0, 0], [1, 1], [0, 0], [0, 0]], mode='SYMMETRIC'))

    g_norm = tf.add(tf.abs(g_x), tf.abs(g_y))

    g_dir = rad2deg(tf.atan2(g_y, g_x)) % 180
    g_bin = tf.cast(g_dir / scale_factor, dtype=tf.int32)

    cell_norm = tf.nn.space_to_depth(g_norm, cell_size)
    cell_bins = tf.nn.space_to_depth(g_bin,  cell_size)

    hist = list()
    zero = tf.zeros_like(cell_bins, dtype=tf.float32)
    for i in range(n_bins):
        mask = tf.equal(cell_bins, tf.constant(i))
        hist.append(tf.reduce_mean(tf.where(mask, cell_norm, zero), 3))
    hist = tf.transpose(tf.stack(hist), [1,2,3,0])

    block_hist = tf.image.extract_patches(hist, sizes  = [1, block_size, block_size, 1],
                                          strides = [1, block_stride, block_stride, 1],
                                          rates   = [1, 1, 1, 1], padding = 'VALID')
    block_hist = tf.nn.l2_normalize(block_hist, 3, epsilon=1.0)

    hog_descriptor = tf.reshape(block_hist, [batch_size, int(block_hist.get_shape()[1]) *
                                int(block_hist.get_shape()[2]) *
                                int(block_hist.get_shape()[3])])

    return hog_descriptor

def add_hog(x):
    hog_descriptor = get_hog_model(image=x)
    return hog_descriptor

# %% [markdown]
# # Models

# %% [markdown]
# ## Model 1 - Baseline

# %% [code] {"execution":{"iopub.status.busy":"2022-06-26T09:15:57.36104Z","iopub.execute_input":"2022-
06-26T09:15:57.361891Z","iopub.status.idle":"2022-06-
26T09:15:57.37324Z","shell.execute_reply.started":"2022-06-
26T09:15:57.361849Z","shell.execute_reply":"2022-06-
26T09:15:57.371874Z"},"jupyter":{"outputs_hidden":false}}
def make_model_1_baseline():

    inputs = keras.Input(shape=(28, 28, 1))

    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu')(inputs)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='valid', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2, padding="valid")(x)
    x = layers.Flatten()(x)

    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)
```

```
    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

# %% [markdown]
# ## Model 1 - SIFT hybrid

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:15:59.043746Z","iopub.execute_input":"2022-06-26T09:15:59.044345Z","iopub.status.idle":"2022-06-
26T09:15:59.053899Z","shell.execute_reply.started":"2022-06-
26T09:15:59.044313Z","shell.execute_reply":"2022-06-
26T09:15:59.052627Z"},"jupyter":{"outputs_hidden":false}}
def make_model_1_SIFT():

    inputs = keras.Input(shape=(28, 28, 1))

    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu')(inputs)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='valid', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = add_sift(x)

    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

    for layer in model.layers[4:-3]:
        layer.trainable = False

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

# %% [markdown]
# ## Model 1 - HOG hybrid

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:00.882142Z","iopub.execute_input":"2022-06-26T09:16:00.882624Z","iopub.status.idle":"2022-06-
26T09:16:00.893323Z","shell.execute_reply.started":"2022-06-
26T09:16:00.882579Z","shell.execute_reply":"2022-06-
26T09:16:00.892284Z"},"jupyter":{"outputs_hidden":false}}
def make_model_1_HOG():

    inputs = keras.Input(shape=(28, 28, 1))

    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu')(inputs)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='valid', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2)(x)
    x_1 = layers.Flatten()(x)

    x_2 = layers.Rescaling(scale = 255.)(inputs)
    x_2 = add_hog(x_2)

    x = layers.Concatenate()([x_1, x_2])

    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

```
    return model

# %% [markdown]
# ## Model 2 - Baseline

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:02.176196Z","iopub.execute_input":"2022-06-26T09:16:02.177366Z","iopub.status.idle":"2022-06-
26T09:16:02.191608Z","shell.execute_reply.started":"2022-06-
26T09:16:02.177324Z","shell.execute_reply":"2022-06-
26T09:16:02.190173Z"},"jupyter":{"outputs_hidden":false}}
def make_model_2_baseline():

    inputs = keras.Input(shape=(32, 32, 1))

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(inputs)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2)(x)

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2)(x)
    x = layers.Flatten()(x)

    x = layers.Dense(units=256,activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dense(units=64, activation='relu')(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

# %% [markdown]
# ## Model 2 - SIFT hybrid

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:03.472747Z","iopub.execute_input":"2022-06-26T09:16:03.473901Z","iopub.status.idle":"2022-06-
26T09:16:03.489118Z","shell.execute_reply.started":"2022-06-
26T09:16:03.473856Z","shell.execute_reply":"2022-06-
26T09:16:03.487602Z"},"jupyter":{"outputs_hidden":false}}
def make_model_2_SIFT():

    inputs = keras.Input(shape=(32, 32, 1))

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(inputs)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = add_sift(x)

    x = layers.Dense(units=256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dense(units=64, activation='relu')(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

    for layer in model.layers[9:-5]:
```

```
        layer.trainable = False

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

# %% [markdown]
# ## Model 2 - HOG hybrid

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:04.746378Z","iopub.execute_input":"2022-06-26T09:16:04.746753Z","iopub.status.idle":"2022-06-
26T09:16:04.763053Z","shell.execute_reply.started":"2022-06-
26T09:16:04.746724Z","shell.execute_reply":"2022-06-
26T09:16:04.762051Z"},"jupyter":{"outputs_hidden":false}}
def make_model_2_HOG():

    inputs = keras.Input(shape=(32, 32, 1))

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(inputs)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2)(x)

    x = layers.Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(pool_size=2)(x)
    x_1 = layers.Flatten()(x)

    x_2 = layers.Rescaling(scale = 255.)(inputs)
    x_2 = add_hog(x_2)

    x = layers.Concatenate()([x_1, x_2])

    x = layers.Dense(units=256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(units=128, activation='relu')(x)
    x = layers.Dense(units=64, activation='relu')(x)

    outputs = layers.Dense(units=10, activation='softmax')(x)

    model = keras.Model(inputs, outputs)

    model.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

# %% [markdown]
# # Training and evaluation

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:07.714473Z","iopub.execute_input":"2022-06-26T09:16:07.714895Z","iopub.status.idle":"2022-06-
26T09:16:10.467421Z","shell.execute_reply.started":"2022-06-
26T09:16:07.714862Z","shell.execute_reply":"2022-06-
26T09:16:10.466103Z"},"jupyter":{"outputs_hidden":false}}
# Model / data parameters
num_classes = 10
batch_size = 128
epochs = 20

# the data, split between train and test sets
(x_train_1, y_train_1), (x_test_1, y_test_1) = fashion_mnist.load_data()
(x_train_2, y_train_2), (x_test_2, y_test_2) = cifar10.load_data()

x_train_2 = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in x_train_2])
x_test_2 = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in x_test_2])
y_train_2 = np.squeeze(y_train_2)
y_test_2 = np.squeeze(y_test_2)
```

```
# Scale images to the [0, 1] range
x_train_1 = x_train_1.astype("float32") / 255
x_test_1 = x_test_1.astype("float32") / 255
x_train_2 = x_train_2.astype("float32") / 255
x_test_2 = x_test_2.astype("float32") / 255
# Make sure images have shape (28, 28, 1) and (32, 32, 1)
x_train_1 = np.expand_dims(x_train_1, -1)
x_test_1 = np.expand_dims(x_test_1, -1)
x_train_2 = np.expand_dims(x_train_2, -1)
x_test_2 = np.expand_dims(x_test_2, -1)
print("x_train_1 shape (fashionMNIST):", x_train_1.shape)
print("x_train_2 shape (cifar-10):", x_train_2.shape)
print(x_train_1.shape[0], "train samples (fashionMNIST)")
print(x_test_1.shape[0], "test samples (fashionMNIST)")
print(x_train_2.shape[0], "train samples (cifar-10)")
print(x_test_2.shape[0], "test samples (cifar-10)")

# convert class vectors to binary class matrices
y_train_1 = keras.utils.to_categorical(y_train_1, num_classes)
y_test_1 = keras.utils.to_categorical(y_test_1, num_classes)
y_train_2 = keras.utils.to_categorical(y_train_2, num_classes)
y_test_2 = keras.utils.to_categorical(y_test_2, num_classes)

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T09:16:10.469733Z","iopub.execute_input":"2022-06-26T09:16:10.470114Z","iopub.status.idle":"2022-06-
26T09:16:12.081588Z","shell.execute_reply.started":"2022-06-
26T09:16:10.470083Z","shell.execute_reply":"2022-06-
26T09:16:12.080068Z"},"jupyter":{"outputs_hidden":false}}
model_1_baseline = make_model_1_baseline()
model_1_SIFT = make_model_1_SIFT()
model_1_HOG = make_model_1_HOG()
model_2_baseline = make_model_2_baseline()
model_2_SIFT = make_model_2_SIFT()
model_2_HOG = make_model_2_HOG()

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T02:10:05.212585Z","iopub.execute_input":"2022-06-26T02:10:05.213095Z","iopub.status.idle":"2022-06-
26T02:11:45.857341Z","shell.execute_reply.started":"2022-06-
26T02:10:05.213056Z","shell.execute_reply":"2022-06-
26T02:11:45.855772Z"},"jupyter":{"outputs_hidden":false}}
history_model_1_baseline = model_1_baseline.fit(x_train_1, y_train_1, batch_size=batch_size,
                                                epochs=epochs, validation_data=(x_test_1, y_test_1),
verbose=1)

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T02:31:18.976102Z","iopub.execute_input":"2022-06-26T02:31:18.976468Z","iopub.status.idle":"2022-06-
26T02:31:18.98212Z","shell.execute_reply.started":"2022-06-
26T02:31:18.976439Z","shell.execute_reply":"2022-06-
26T02:31:18.980903Z"},"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_1_baseline', 'wb') as file_pi:
    pickle.dump(history_model_1_baseline.history, file_pi)

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T07:38:33.221204Z","iopub.execute_input":"2022-06-26T07:38:33.2227Z","iopub.status.idle":"2022-06-
26T07:38:54.915191Z","shell.execute_reply.started":"2022-06-
26T07:38:33.222653Z","shell.execute_reply":"2022-06-
26T07:38:54.912884Z"},"jupyter":{"outputs_hidden":false}}
history_model_1_SIFT = model_1_SIFT.fit(x_train_1, y_train_1, batch_size=batch_size,
                                        epochs=epochs, validation_data=(x_test_1, y_test_1), verbose=1)

# %% [code] {"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_1_SIFT', 'wb') as file_pi:
    pickle.dump(history_model_1_SIFT.history, file_pi)

# %% [code] {"execution":{"iopub.status.busy":"2022-06-25T23:20:17.43556Z","iopub.execute_input":"2022-
06-25T23:20:17.435951Z","iopub.status.idle":"2022-06-
25T23:20:54.396225Z","shell.execute_reply.started":"2022-06-
25T23:20:17.435918Z","shell.execute_reply":"2022-06-
25T23:20:54.395105Z"},"jupyter":{"outputs_hidden":false}}
history_model_1_HOG = model_1_HOG.fit(x_train_1, y_train_1, batch_size=batch_size,
                                      epochs=epochs, validation_data=(x_test_1, y_test_1), verbose=1)

# %% [code] {"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_1_HOG', 'wb') as file_pi:
    pickle.dump(history_model_1_HOG.history, file_pi)

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T01:57:25.711496Z","iopub.execute_input":"2022-06-26T01:57:25.711917Z","iopub.status.idle":"2022-06-
26T01:58:54.612473Z","shell.execute_reply.started":"2022-06-
```

```
26T01:57:25.711879Z","shell.execute_reply":"2022-06-
26T01:58:54.610821Z"},"jupyter":{"outputs_hidden":false}}
history_model_2_baseline = model_2_baseline.fit(x_train_2, y_train_2,
                                                batch_size=batch_size, epochs=epochs,
validation_data=(x_test_2, y_test_2), verbose=1)

# %% [code] {"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_2_baseline', 'wb') as file_pi:
    pickle.dump(history_model_2_baseline.history, file_pi)

# %% [code] {"jupyter":{"outputs_hidden":false}}
history_model_2_SIFT = model_2_SIFT.fit(x_train_2, y_train_2, batch_size=batch_size,
                                        epochs=epochs, validation_data=(x_test_2, y_test_2), verbose=1)

# %% [code] {"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_2_SIFT', 'wb') as file_pi:
    pickle.dump(history_model_2_SIFT.history, file_pi)

# %% [code] {"jupyter":{"outputs_hidden":false}}
history_model_2_HOG = model_2_HOG.fit(x_train_2, y_train_2, batch_size=batch_size,
                                      epochs=epochs, validation_data=(x_test_2, y_test_2), verbose=1)

# %% [code] {"jupyter":{"outputs_hidden":false}}
with open('/kaggle/working/history_model_2_HOG', 'wb') as file_pi:
    pickle.dump(history_model_2_HOG.history, file_pi)

# %% [code] {"jupyter":{"outputs_hidden":false}}


# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T15:27:11.520466Z","iopub.execute_input":"2022-06-26T15:27:11.521189Z","iopub.status.idle":"2022-06-
26T15:27:11.570218Z","shell.execute_reply.started":"2022-06-
26T15:27:11.521152Z","shell.execute_reply":"2022-06-
26T15:27:11.568861Z"},"jupyter":{"outputs_hidden":false}}
# history_m_1_base=np.load('../input/masters-thesis-3/history_model_1_baseline',allow_pickle='TRUE')
# history_m_1_SIFT=np.load('../input/masters-thesis-3/history_model_1_SIFT',allow_pickle='TRUE')
# history_m_1_HOG=np.load('../input/masters-thesis-3/history_model_1_HOG',allow_pickle='TRUE')
# history_m_2_base=np.load('../input/masters-thesis-3/history_model_2_baseline',allow_pickle='TRUE')
# history_m_2_SIFT=np.load('../input/masters-thesis-3/history_model_2_SIFT',allow_pickle='TRUE')
# history_m_2_HOG=np.load('../input/masters-thesis-3/history_model_2_HOG',allow_pickle='TRUE')
# epochs = 20

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T15:27:15.254671Z","iopub.execute_input":"2022-06-26T15:27:15.255094Z","iopub.status.idle":"2022-06-
26T15:27:15.264773Z","shell.execute_reply.started":"2022-06-
26T15:27:15.255063Z","shell.execute_reply":"2022-06-
26T15:27:15.263897Z"},"jupyter":{"outputs_hidden":false}}
print('Model 1 (base) peak accuracy:', max(history_m_1_base['accuracy']))
print('Model 1 (SIFT) peak accuracy:', max(history_m_1_SIFT['accuracy']))
print('Model 1 (HOG) peak accuracy:', max(history_m_1_HOG['accuracy']))
print('Model 2 (base) peak accuracy:', max(history_m_2_base['accuracy']))
print('Model 2 (SIFT) peak accuracy:', max(history_m_2_SIFT['accuracy']))
print('Model 2 (HOG) peak accuracy:', max(history_m_2_HOG['accuracy']))

print('Model 1 (base) peak val_accuracy:', max(history_m_1_base['val_accuracy']))
print('Model 1 (SIFT) peak val_accuracy:', max(history_m_1_SIFT['val_accuracy']))
print('Model 1 (HOG) peak val_accuracy:', max(history_m_1_HOG['val_accuracy']))
print('Model 2 (base) peak val_accuracy:', max(history_m_2_base['val_accuracy']))
print('Model 2 (SIFT) peak val_accuracy:', max(history_m_2_SIFT['val_accuracy']))
print('Model 2 (HOG) peak val_accuracy:', max(history_m_2_HOG['val_accuracy']))

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T12:10:39.714254Z","iopub.execute_input":"2022-06-26T12:10:39.714733Z","iopub.status.idle":"2022-06-
26T12:10:40.089239Z","shell.execute_reply.started":"2022-06-
26T12:10:39.714692Z","shell.execute_reply":"2022-06-
26T12:10:40.087811Z"},"jupyter":{"outputs_hidden":false}}
plt.figure(figsize=(12,6))
plt.title('Model 1 accuracies')
plt.xticks(np.arange(epochs), labels = np.arange(1, epochs+1))
plt.plot(history_m_1_base['accuracy'])
plt.plot(history_m_1_SIFT['accuracy'])
plt.plot(history_m_1_HOG['accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Baseline', 'SIFT', 'HOG'], loc='upper left')
plt.show()

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T12:10:48.133527Z","iopub.execute_input":"2022-06-26T12:10:48.134014Z","iopub.status.idle":"2022-06-
```

```
26T12:10:48.490205Z","shell.execute_reply.started":"2022-06-
26T12:10:48.133978Z","shell.execute_reply":"2022-06-
26T12:10:48.488868Z"},"jupyter":{"outputs_hidden":false}}
plt.figure(figsize=(12,6))
plt.title('Model 1 val_accuracies')
plt.xticks(np.arange(epochs), labels = np.arange(1, epochs+1))
plt.plot(history_m_1_base['val_accuracy'])
plt.plot(history_m_1_SIFT['val_accuracy'])
plt.plot(history_m_1_HOG['val_accuracy'])
plt.ylabel('Val_Accuracy')
plt.xlabel('Epoch')
plt.legend(['Baseline', 'SIFT', 'HOG'], loc='upper left')
plt.show()

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T12:10:48.961906Z","iopub.execute_input":"2022-06-26T12:10:48.962651Z","iopub.status.idle":"2022-06-
26T12:10:49.315375Z","shell.execute_reply.started":"2022-06-
26T12:10:48.962608Z","shell.execute_reply":"2022-06-
26T12:10:49.314213Z"},"jupyter":{"outputs_hidden":false}}
plt.figure(figsize=(12,6))
plt.title('Model 2 accuracies')
plt.xticks(np.arange(epochs), labels = np.arange(1, epochs+1))
plt.plot(history_m_2_base['accuracy'])
plt.plot(history_m_2_SIFT['accuracy'])
plt.plot(history_m_2_HOG['accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Baseline', 'SIFT', 'HOG'], loc='upper left')
plt.show()

# %% [code] {"execution":{"iopub.status.busy":"2022-06-
26T12:10:52.040207Z","iopub.execute_input":"2022-06-26T12:10:52.040731Z","iopub.status.idle":"2022-06-
26T12:10:52.398706Z","shell.execute_reply.started":"2022-06-
26T12:10:52.040688Z","shell.execute_reply":"2022-06-
26T12:10:52.397114Z"},"jupyter":{"outputs_hidden":false}}
plt.figure(figsize=(12,6))
plt.title('Model 2 val_accuracies')
plt.xticks(np.arange(epochs), labels = np.arange(1, epochs+1))
plt.plot(history_m_2_base['val_accuracy'])
plt.plot(history_m_2_SIFT['val_accuracy'])
plt.plot(history_m_2_HOG['val_accuracy'])
plt.ylabel('Val_Accuracy')
plt.xlabel('Epoch')
plt.legend(['Baseline', 'SIFT', 'HOG'], loc='upper left')
plt.show()
```