UNIVERSIT DI FIRENZE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Images Augmentation, Comparing Sequential and Parallel version

Docente:

Marco Bertini

Candidato:

Daniele Morganti

ANNO ACCADEMICO 2024/2025

Indice

1 Introduzione

1.1 Contesto e obbiettivi

Nell'ambito della computer vision e del machine learning, la disponibilit di dataset di immagini di elevata qualit e variet rappresenta un elemento essenziale per ottenere modelli predittivi accurati e generalizzabili. Tuttavia, la raccolta di grandi volumi di immagini pu essere costosa e logisticamente complessa, limitando cos l'accesso a dataset sufficientemente ampi e diversificati. Per affrontare questa sfida, una tecnica comunemente utilizzata l'augmentation delle immagini, che consiste nel generare varianti sintetiche delle immagini originali applicando una serie di trasformazioni, come rotazioni, ridimensionamenti, aggiunta di rumore e altre modifiche che non alterano il contenuto semantico dell'immagine.

La libreria Albumentations uno strumento particolarmente efficace per eseguire l'augmentation delle immagini in Python, offrendo un'ampia gamma di trasformazioni predefinite e personalizzabili. Tuttavia, quando si ha a che fare con dataset di grandi dimensioni, l'augmentation di immagini pu risultare un'operazione computazionalmente intensiva. Questo crea la necessit di esplorare strategie di ottimizzazione per ridurre i tempi di esecuzione e migliorare l'efficienza del processo.

L'obiettivo del progetto confrontare due implementazioni di uno script di augmentation delle immagini: una versione sequenziale, che esegue le trasformazioni su ogni immagine in serie,

e una versione parallela, che sfrutta tecniche di parallel computing per suddividere il carico di lavoro tra pi core della CPU. Attraverso questo confronto, si intende valutare il potenziale guadagno in termini di velocit e risorse computazionali della versione parallela rispetto alla versione sequenziale.

1.2 Approccio Generale

Per il confronto tra le due versioni dello script, il progetto sar strutturato in due fasi principali. Nella prima fase, verr implementata una versione sequenziale del processo di augmentation delle immagini, utilizzando Albumentations per applicare una serie di trasformazioni a ogni immagine del dataset in modo indipendente e seriale. Nella seconda fase, verr sviluppata una versione parallela dello script, che far uso di librerie Python per il parallel computing, come multiprocessing o joblib, al fine di distribuire l'elaborazione delle immagini su pi processori.

Ogni versione dello script verr quindi eseguita su un campione rappresentativo del dataset, registrando il tempo di esecuzione e l'utilizzo delle risorse di sistema per ciascuna implementazione. Il confronto si baser su metriche come il tempo medio di elaborazione per immagine, l'uso complessivo della CPU e il consumo di memoria, consentendo cos di trarre conclusioni sulla scalabilit e l'efficienza del processo parallelo rispetto a quello sequenziale.

Il progetto permetter non solo di evidenziare i vantaggi e gli svantaggi del parallel computing in un contesto di data augmentation, ma anche di offrire una guida pratica per chiunque desideri implementare una pipeline di preprocessing delle immagini ottimizzata per dataset di grandi dimensioni.

2 Implementazione

Funzione di Base per l'Augmentation delle Immagini L'implementazione dell'augmentation di singole immagini gestita dalla funzione process_image, che é possibile vedere nel codice in figura 2.1. Questa funzione riceve in input il nome del file immagine (img_name), la trasformazione da applicare (transform), e le directory di input e output (input_dir e output_dir). La prima operazione eseguita é il caricamento dell'immagine da disco tramite cv2.imread(). Se il caricamento fallisce, ad esempio a causa di un file corrotto o di un percorso non valido, viene restituito un messaggio di errore e limmagine viene esclusa dal processo.

Dopo aver caricato limmagine, la funzione applica la trasformazione definita da transform, che utilizza la libreria Albumentations per modificare limmagine. La trasformazione pu includere operazioni come distorsioni geometriche, ribaltamenti, modifiche di contrasto e rotazioni, che arricchiscono la variabilit del dataset. Infine, limmagine risultante viene salvata nella directory di output, mantenendo il nome originale del file, utilizzando cv2.imwrite(). La funzione restituisce infine il nome dellimmagine trasformata per confermare lesecuzione.

```
def process_image(img_name, transform, input_dir, output_dir):
    img_path =os.path.join(input_dir, img_name)
    image =cv2.imread(img_path)

    if image is None:
        print(f"Errore nel caricare l'immagine {img_name}. Saltata.")
        return None

    augmented =transform(image=image)
    augmented_image =augmented["image"]

    output_path =os.path.join(output_dir, img_name)
    cv2.imwrite(output_path, augmented_image)
    return img_name
```

Figura 2.1: Codice per augmentation immagine

Implementazione dellAugmentation Parallela Per elaborare un'intera directory di immagini in parallelo, viene utilizzata la funzione augmentDir_par. Questa funzione permette di applicare trasformazioni di augmentation in modo efficiente, sfruttando il parallelismo della CPU e consentendo di distribuire il carico di lavoro tra pi core.

Allavvio, augmentDir_par accede alla directory di input e genera una lista dei file immagine presenti, preparandoli per lelaborazione. Successivamente, utilizza ThreadPoolExecutor (dalla libreria concurrent.futures) per gestire lesecuzione parallela delle operazioni. Il parametro max_workers di ThreadPoolExecutor viene impostato sul numero di processi desiderato, che di default é pari al numero di core disponibili sulla CPU, ottimizzando cos lutilizzo delle risorse di sistema.

Ogni immagine viene quindi elaborata tramite una chiamata alla funzione process_image, inviata al pool di thread tramite executor.submit(). In questo modo, ogni immagine pu essere trasformata simultaneamente a pi altre, riducendo significativamente il tempo complessivo di elaborazione rispetto a un approccio sequenziale. Durante lesecuzione, tqdm mostra una barra di avanzamento che indica lo stato di completamento delle immagini, permettendo di monitorare l'avanzamento del processo.

Una volta completata lelaborazione di tutte le immagini, viene stampato un messaggio di conferma, indicando che il processo di augmentation parallela stato portato a termine con successo.

```
def augmentDir_par(input_dir,
    output_dir,
    num_processes=os.cpu_count(),
    transform =A.Compose([
                        A.GridDistortion(num_steps=5, distort_limit=0.3, p=0.5),
                        A.HorizontalFlip(p=0.5),
                        A.RandomBrightnessContrast(p=0.2),
                        A.Rotate(limit=40, p=0.5)
                    ])):
    image_list =os.listdir(input_dir)

    with ThreadPoolExecutor(max_workers=num_processes) as executor:
        futures =[executor.submit(process_image, img_name, transform, input_dir,
                                                    output_dir) for img_name in
                                                    image_list]

        for future in tqdm(as_completed(futures), total=len(futures)):
            img_name =future.result()

    print("Processo di augmentazione parallela completato!")
```

Dettagli sulle Trasformazioni Utilizzate La trasformazione applicata nella funzione augmentDir_par é composta da una serie di operazioni fornite da Albumentations, con lo scopo di generare varianti realistiche delle immagini originali e aumentare la robustezza di un potenziale modello addestrato sul dataset risultante. Le operazioni di default di questa funzione includono: una distorsione della griglia per variare la geometria dell'immagine; un ribaltamento orizzontale casuale; modifiche casuali della luminosit e del contrasto; una rotazione casuale fino a 40 gradi. Queste trasformazioni contribuiscono a creare un dataset diversificato, migliorando la capacit del modello di apprendere e generalizzare su nuovi dati, tuttavia queste sono sovrascrivibile a runtime dal codice client con trasformazioni custum.

Esecuzione e Benchmarking Il codice esegue laugmentation di un dataset di immagini in parallelo utilizzando la funzione augmentDir_par, impostando progressivamente un numero crescente di thread di elaborazione per osservare come varia il tempo di esecuzione. L'obiettivo ottenere una misura quantitativa dell'impatto del parallelismo in funzione del numero di processi, in modo da confrontare lefficienza delle diverse configurazioni di parallel computing e valutare il potenziale speedup.

La directory di input input_dir contiene il dataset di immagini originale (in questo caso, il dataset imagenette), mentre output_dir la directory di output in cui vengono salvate le im-

magini trasformate. Il codice verifica che output_dir esista, creando la directory se necessario con os.makedirs(output_dir, exist_ok=True). Questa operazione assicura che la struttura delle directory sia pronta per salvare i risultati dellaugmentation.

Viene definito un numero di iterazioni, che rappresenta il numero massimo di thread da testare. In ciascuna iterazione, il codice incrementa il numero di worker di uno, a partire da un singolo processo. Per ogni esecuzione, viene utilizzata la trasformazione transform definita con Albumentations.

Il codice itera attraverso 25 cicli, eseguendo augmentDir_par con un numero progressivo di worker (da 1 a 25). In ogni iterazione viene inizializzato un array result per raccogliere il numero di worker utilizzati e il tempo di esecuzione corrispondente, viene registrato il tempo di inizio dell'operazione tramite start = time.time(), infine la funzione augmentDir_par viene chiamata con i parametri input_dir, output_dir, il numero di worker i+1 e la trasformazione transform. Al termine della funzione, viene calcolato il tempo totale di esecuzione sottraendo il valore di start al tempo corrente. Il numero di worker e il tempo di esecuzione vengono salvati in result, e larray result viene aggiunto alla lista report.

```
os.makedirs(output_dir, exist_ok=True)

ITERATIONS =25
transform =A.Compose([
                    A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=1),
                    A.GridDistortion(num_steps=5, distort_limit=0.3, p=1),
                    A.HorizontalFlip(p=1),
                    A.RandomBrightnessContrast(p=1),
                    A.Rotate(limit=40, p=1)
                ])

report =[]

for i in range(ITERATIONS):
    result =[]
    result.append(i+1)
    start =time.time()
    print(f"num workers: {i+1}")
    augmentDir_par(input_dir, output_dir, i+1, transform)
    result.append(time.time() -start)
    report.append(result)

with open("results", mode='w', newline='', encoding='utf-8') as file_csv:
    scrittore =csv.writer(file_csv)
    scrittore.writerows(report)
```

3 Esperimenti e Risultati

In questo capitolo vengono presentati e analizzati i risultati sperimentali della procedura di augmentation presentata nel capitolo precedente. L'obiettivo é valutare le prestazioni in termini di tempo di esecuzione, efficienza e scalabilit.

Gli esperimenti di seguito sono stati eseguiti su un processore Intel Core i7-12700K con 12 core.

Obiettivi dell'Analisi L'analisi dei risultati sperimentali si propone di rispondere a diverse domande:

- Qual é il tempo di esecuzione dell'implementazione parallela in funzione del numero di processi lanciati?
- L'implementazione parallela scala meglio con l'aumento della complessit delle trasformazioni effettuate sulle immagini?

Metodologia dei Test I test sono stati eseguiti utilizzando il dataset Imagenette (https://github.com/fastai/i variando il numero di processi e la tipologia delle trasformazioni per valutare come la procedura si comporta al crescere del carico computazionale. In particolare sono stati effettuati 3 esperimenti, ognuno effettua trasformazioni sulle immagini diverse, progressivamente pi ”pe-

santi". All'interno di ogni esperimento si varia il numero di worker tenendo traccia dei tempi di esecuzione.

Metriche di Valutazione Le principali metriche utilizzate per il confronto tra le due versioni sono le seguenti:

Tempo di esecuzione: tempo necessario per completare l'algoritmo, misurato in millisecondi. Questa metrica permette di osservare la riduzione di tempo ottenuta dalla parallelizzazione. Speedup: definito come il rapporto tra il tempo di esecuzione della versione seriale e il tempo di esecuzione della versione parallela con n thread. Lo speedup ideale direttamente proporzionale al numero di thread, ma in pratica é influenzato dall'overhead di sincronizzazione.

3.1 Primo Esperimento

In questo primo esperimento sono state applicate 3 trasformazioni di augmentation al dataset:

- HorizontalFlip: ribaltamento orizzontale.
- RandomBrightnessContrast: variazione casuale della luminosit e del contrasto.
- Rotate: rotazione casuale dellimmagine entro un limite di 40 gradi.

Si tratta di tre trasformazioni molto leggere da applicare a un immagine che non rappresentano un carico computazionale pesante.

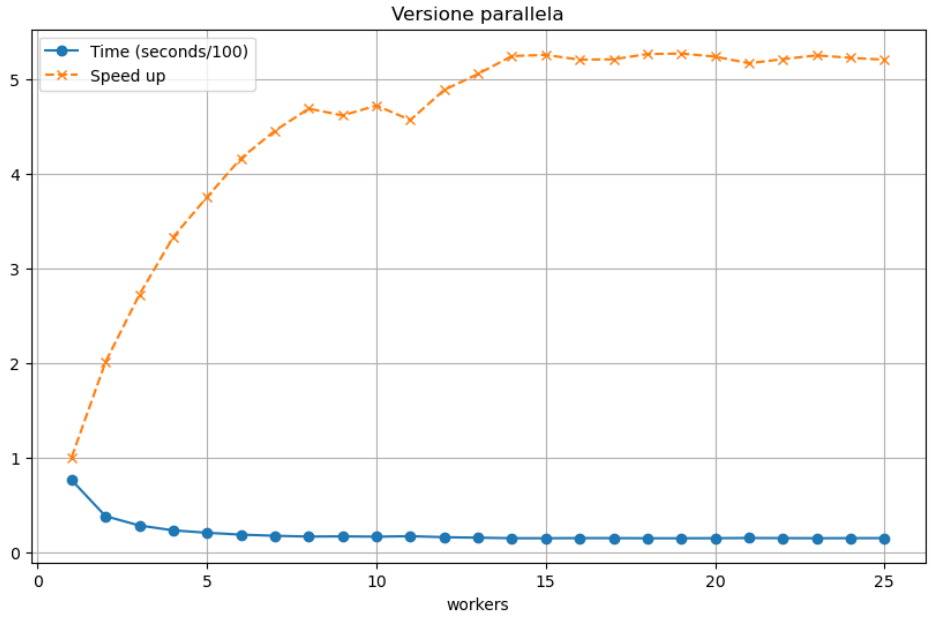

Figura 3.1: Tempo di esecuzione in funzione del numero di workers

workers	time (seconds)
1	75.88809061050415
2	37.81303811073303
3	27.912081003189087
4	22.79630947113037
5	20.225557327270508
6	18.23984384536743
7	17.045785188674927
8	16.196183443069458
9	16.447453022003174
10	16.085492849349976
11	16.621712923049927
12	15.53949522972107
13	15.019898891448975
13	15.019898891448975
14	14.47722840309143
15	14.44535207748413
16	14.584312200546265
17	14.572691917419434
18	14.419632911682129
19	14.405098915100098
20	14.494438648223877
21	14.691521644592285
22	14.568217754364014
23	14.459544897079468
24	14.5306236743927
25	14.585738182067871

Tabella 3.1: Tempi di esecuzione secondo esperimento

Possiamo osservare come lo speed up salga rapidamente con i primo 8 workers per poi assestarsi dai 14 in poi. Questo é ragionevole considerando il processore da 8 core di performance e 4 di efficienza che é stato usato per condurre questo esperimento.

3.2 Secondo Esperimento

Questo secondo esperimento é stato condotto utilizzando, oltre a quelle precedenti, una trasformazione computazionalmente pi complessa, ossia la grid distorsion. Questa trasformazione effettua una distorsione dell'immagine su una griglia di definizione data.

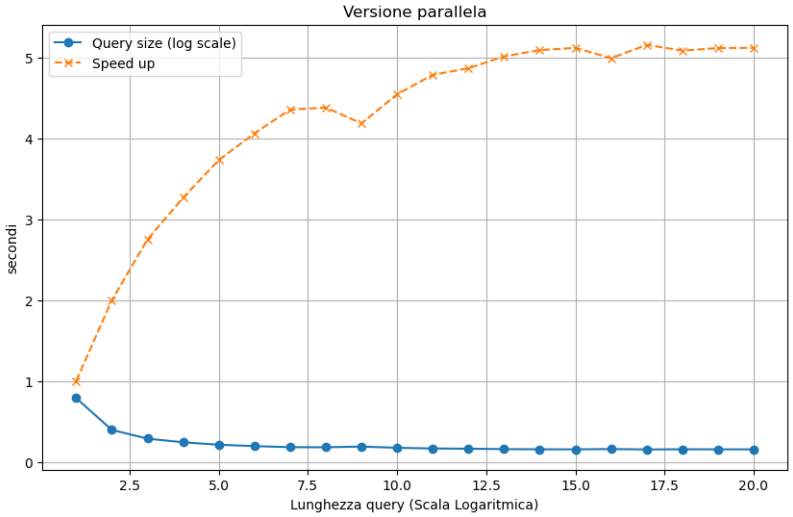

Figura 3.2: Tempo di esecuzione in secondo esperimento

workers	time (seconds)
1	92.95418763160706
2	51.55107831954956
3	39.712486267089844
4	34.102256774902344
5	31.646128177642822
6	29.608182668685913
7	28.151955604553223
8	28.38787078857422
9	27.720837116241455
10	27.676270246505737
11	27.691027879714966
12	28.0363552570343
13	28.109331130981445
14	28.06713628768921
15	28.17850351333618
16	28.043840408325195
17	28.26502799987793
18	29.069459438323975
19	28.72594690322876
20	28.575819492340088
21	28.505796909332275
22	28.708466291427612
23	28.222480058670044
24	28.36543869972229
25	28.263317584991455

Tabella 3.2: Tempi di esecuzione secondo esperimento

Come si puó vedere i risultati ci dicono che le performance della parallelizzazione sono calate rispetto al rpimo esperimento.

3.3 Terzo esperimento

L'ultimo esperimento stato condotto aggiungendo nuovamente un'altra trasformazione a quelle precedenti, ossia ElasticTransform. Questa trasformazione modifica la struttura dellimmagine per simularne una deformazione elastica.

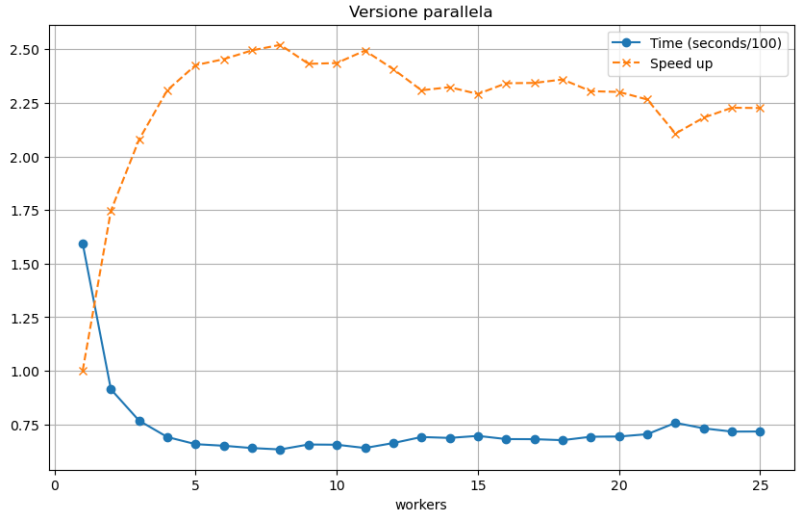

Figura 3.3: Tempo di esecuzione in terzo esperimento

workers	time (seconds)
1	159.4784710407257
2	91.33796215057373
3	76.69728136062622
4	69.05409574508667
5	65.73287963867188
6	64.9834496974945
7	63.91303610801697
8	63.26288032531738
9	65.56867003440857
10	65.46667218208313
11	63.94161033630371
12	66.23613715171814
13	69.07031178474426
14	68.64087796211243
15	69.59494805335999
16	68.1019036769867
17	68.0585868358612
18	67.59454393386841
19	69.19629549980164
20	69.30482912063599
21	70.38147950172424
22	75.69312024116516
23	73.10872840881348
24	71.59385776519775
25	71.6435797214508

Tabella 3.3: Tempi di esecuzione secondo esperimento

Da questi risultati possiamo nuovamente osservare un calo del beneficio della parallelizzazione nell'operazione di augmentation del dataset. Ció ci conferma che pi le operazioni di trasformazione sono complesse meno abbiamo beneficio a parallelizzare con processi separati.

4 Conclusioni

Il progetto ha esplorato e confrontato le prestazioni di un processo di image augmentation eseguito in modo sequenziale e parallelo, utilizzando la libreria Albumentations. I risultati mostrano chiaramente che la parallelizzazione pu accelerare il tempo di elaborazione complessivo, in particolare quando il numero di worker viene incrementato, permettendo di processare pi immagini contemporaneamente. Tuttavia, l'efficacia di questo approccio varia sensibilmente in base alla complessit computazionale delle trasformazioni applicate alle immagini.

In scenari con trasformazioni semplici (ad esempio ribaltamenti o rotazioni leggere), loverhead introdotto dalla gestione dei thread é minimo rispetto al tempo di elaborazione, e di conseguenza la parallelizzazione produce un miglioramento significativo delle prestazioni. In questi casi, lo speedup ottenuto elevato poich il sistema pu dedicare la maggior parte delle risorse al calcolo effettivo delle trasformazioni.

Tuttavia, con laumentare della complessit delle trasformazioni come nel caso di ElasticTransform o GridDistortion, che richiedono risorse computazionali pi intensive l'efficienza della parallelizzazione diminuisce. Questo comportamento spiegabile da diversi fattori:

- **Overhead di Coordinamento e Sincronizzazione**: quando i processi sono impegnati in compiti pi pesanti, il tempo necessario per distribuire, sincronizzare e raccogliere i risultati tra i vari thread o processi aumenta. Di conseguenza, il tempo speso per loverhead cresce in proporzione al carico di lavoro, riducendo il guadagno netto in termini di speedup.

- **Competizione per le Risorse di Sistema**: le trasformazioni pesanti richiedono un uso intensivo di CPU e memoria. Quando piú thread o processi competono per queste risorse, si creano colli di bottiglia che impediscono un accesso ottimale ai dati, limitando il parallelismo.

In conclusione, lapproccio parallelo offre vantaggi significativi nelle operazioni di augmentation di immagini, soprattutto in scenari con trasformazioni leggere. Tuttavia, il beneficio della parallelizzazione tende a ridursi con laumentare della complessit delle trasformazioni. Per ottimizzare le prestazioni in questi contesti, potrebbe essere utile adottare approcci di parallelismo che minimizzino l'overhead o utilizzare hardware specializzato come GPU, piú adatto per eseguire calcoli complessi su larga scala.