UNIVERSITÀ DEGLI STUDI FIRENZE

PARALLEL IMAGE COMPOSITION FOR DATA AUGMENTATION

Miruna Alexandrescu 7163599

OBIETTIVO

DATA AUGMENTATION

CODICE

RISULTATI

CONSCLUSIONI

- **PARALLELIZZARE** un algoritmo di Data Augmentation per image composition

- **CONFRONTARE** diverse strategie di parallelizzazione

- **VALUTARE** speedup ed efficienza rispetto alla versione sequenziale

DATA AUGMENTATION:

tecnica per aumentare artificialmente la dimensione di un dataset senza raccogliere nuovi dati. Molto utile nell'addestramento delle reti neurali.

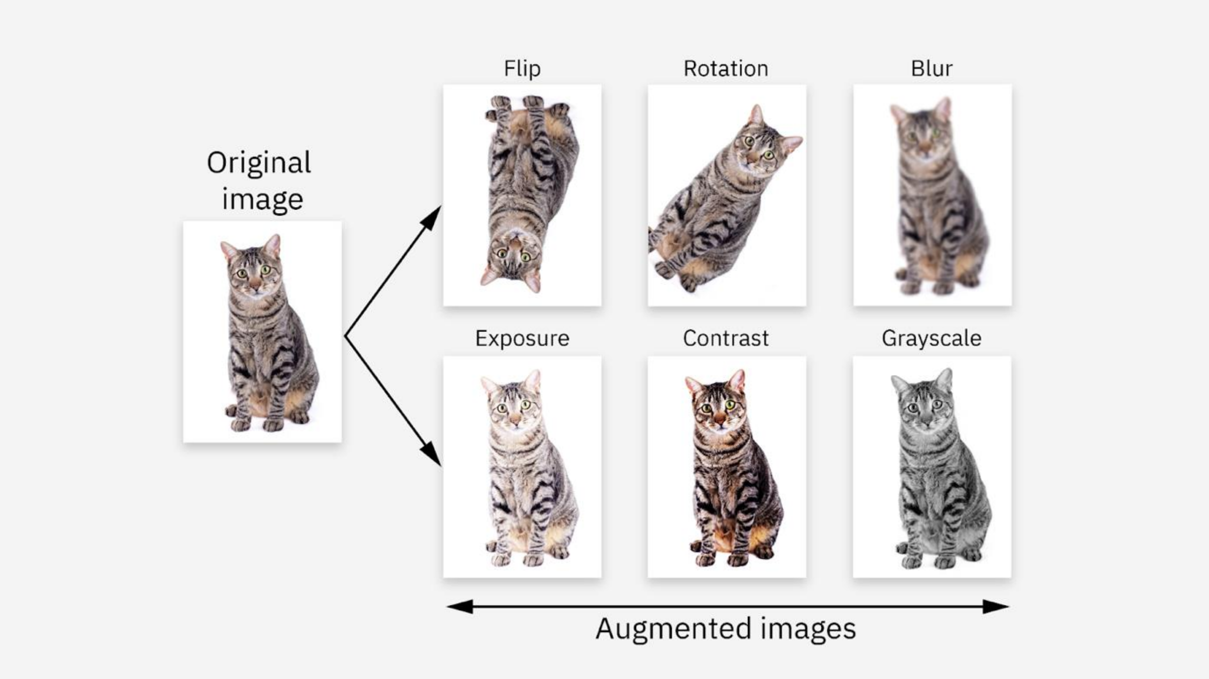
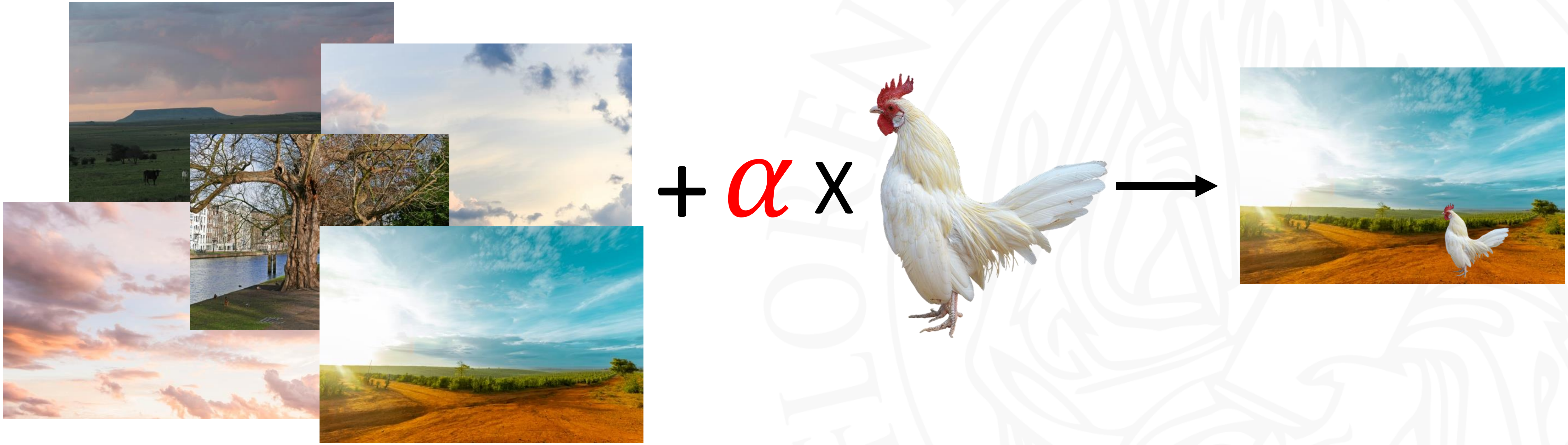


IMAGE COMPOSITION:

L'esperimento utilizza 5 immagini di background (JPEG) e 1 immagine foreground (PNG con canale alpha). Tramite Image Composition, il foreground viene posizionato casualmente sui background con trasparenza randomica (128-255).



SPECIFICHE ESPERIMENTO

- Embarrassingly parallel
- Parallelizzazione tramite multiprocessing
- Parallelizzazione tramite Joblib
- Test da 50 a 500 immagini con incremento di 50

Model	MacBook Pro 13-inch, M1, 2020
Processor	Apple M1 chip (8-core CPU) 4 performance + 4 efficiency cores
Memory	16 GB unified memory

IMAGE COMPOSITION SEQUENZIALE

```python
def compose_images_sequential(foreground,
    backgrounds, num_images, save_images=True):
    """Sequential image composition"""
    output_dir = None
    if save_images:
        timestamp = datetime.datetime.now()
        output_dir = f'output/{timestamp}'
        os.makedirs(output_dir, exist_ok=True)

    for i in range(num_images):
        # Choose random background
        bg_index = random.randint(0, len(
            backgrounds) - 1)
        background = copy(backgrounds[bg_index])

        # Calculate random position for
            foreground
        max_row = background.shape[0] -
            foreground.shape[0]
        max_col = background.shape[1] -
            foreground.shape[1]
        row = random.randint(0, max_row - 1)
        col = random.randint(0, max_col - 1)
```

```python
alpha_blend = random.randint(128, 255) /
    255.0
for j in range(foreground.shape[0]):
    for k in range(foreground.shape[1]):
        f_pixel = foreground[j, k]
        b_pixel = background[row + j, col
             + k]
        f_alpha = f_pixel[3] / 255.0

        if f_alpha > 0.9:  # Only non-
            transparent pixels
            for c in range(3):  # RGB
                channels
                background[row + j, col +
                     k, c] = (
                    b_pixel[c] * (1 -
                        alpha_blend) +
                    f_pixel[c] *
                        alpha_blend *
                        f_alpha
                )

# Save image if requested
if save_images:
    cv2.imwrite(f"{output_dir}/composed_{
        i}.png", background)
```

LIBRERIA MULTIPROCESSING

- Permette di istanziare e far partire manualmente i processi su una certa funzione target
- Lavoro equamente diviso tra i processi

IMAGE COMPOSITION PARALLELO MULTIPROCESSING

```python
def parallel_processes(foreground, backgrounds,
     num_images, num_processes, save_images=True):
    """Parallelization with Process"""
    output_dir = None
    if save_images:
        timestamp = datetime.datetime.now()
        output_dir = f'output/{timestamp}'
        os.makedirs(output_dir, exist_ok=True)

    images_per_process = math.ceil(num_images /
        num_processes)
```

```python
processes = []
for i in range(num_processes):
    p = Process(target=compose_batch_pool,
                args=(foreground, backgrounds
                    , output_dir,
                    images_per_process,
                    save_images))
    processes.append(p)
    p.start()

for p in processes:
    p.join()

return output_dir
```

IMAGE COMPOSITION PARALLELO POOL MULTIPROCESSING

```python
def parallel_pool(foreground, backgrounds,
    num_images, num_processes, save_images=True):
    """Parallelization with Pool"""
    output_dir = None
    if save_images:
        timestamp = datetime.datetime.now()
        output_dir = f'output/{timestamp}'
        os.makedirs(output_dir, exist_ok=True)

    images_per_process = math.ceil(num_images /
        num_processes)

    with Pool(processes=num_processes) as pool:
        args = [(foreground, backgrounds,
            output_dir, images_per_process,
            save_images)
                for _ in range(num_processes)]
        pool.starmap(compose_batch_pool, args)

    return output_dir
```

```python
def compose_batch_pool(foreground, backgrounds,
    output_dir, num_images, save_images=True):
    """Compose a batch of images (for Pool)"""
    for i in range(num_images):
        bg_index = random.randint(0, len(
            backgrounds) - 1)
        background = copy(backgrounds[bg_index])
        # ... alpha blending logic ...
        if save_images and output_dir:
            process_name = multiprocessing.
                current_process().name
            cv2.imwrite(f"{output_dir}/composed_{
                process_name}_{i}.png",
                background)
```

LIBRERIA JOBLIB

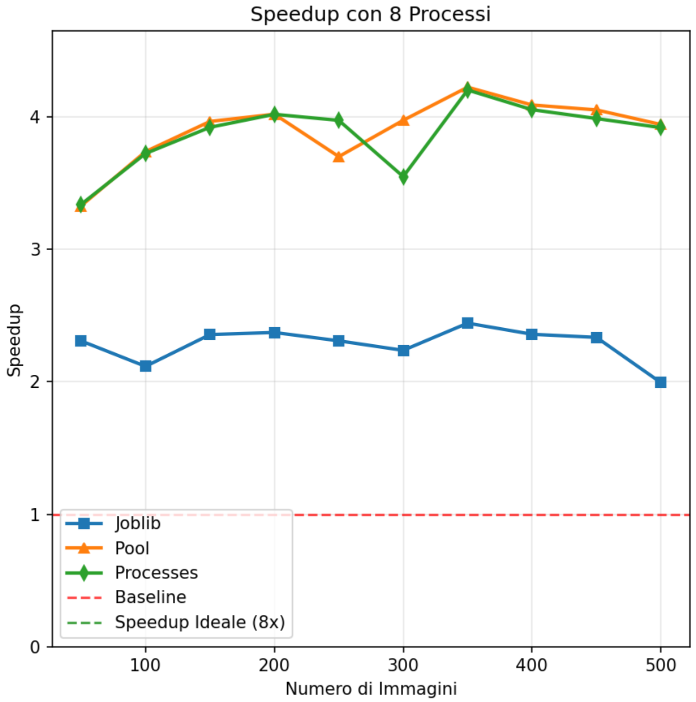
- Suddivisione del lavoro in modo autonomo
- Processi eseguiti in modo asincrono sulla funzione target
- Ad ogni nuova esecuzione gli argomenti devono essere ripassati in ingresso alla funzione

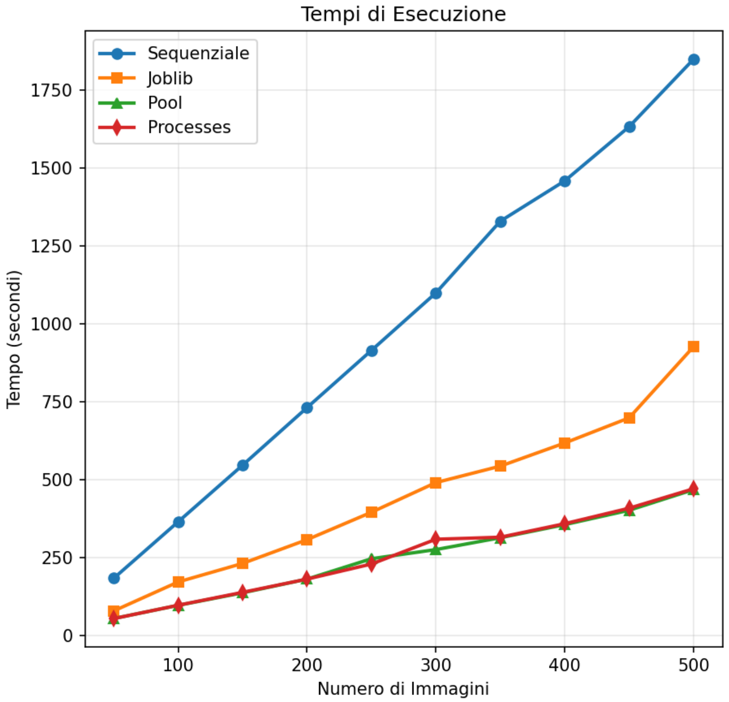
IMAGE COMPOSITION PARALLELO JOBLIB

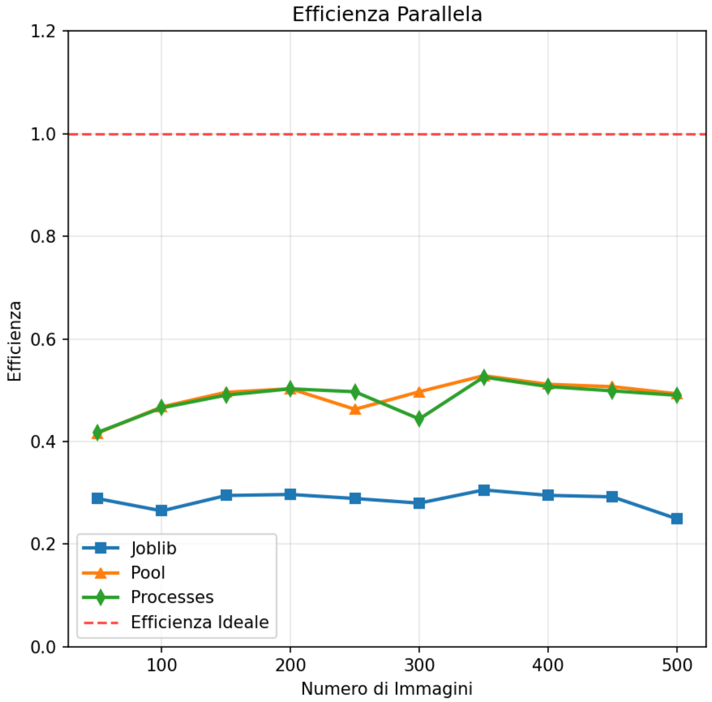
```
def parallel_joblib(foreground, backgrounds,
    num_images, num_processes, save_images=True):
    """Parallelization with joblib"""
    output_dir = None
    if save_images:
        timestamp = datetime.datetime.now()
        output_dir = f'output/{timestamp}'
        os.makedirs(output_dir, exist_ok=True)

    # Choose one background for all tasks
    bg_index = random.randint(0, len(backgrounds)
         - 1)
    background = backgrounds[bg_index]

    Parallel(n_jobs=num_processes)(
        delayed(compose_single_image)(foreground,
             background, output_dir, i,
            save_images)
        for i in range(num_images)
    )

    return output_dir
```

```
def compose_single_image(foreground,
    background_img, output_dir, image_id,
    save_images=True):

    """Compose a single image (for joblib)"""
    background = copy(background_img)
    # ... positioning and alpha blending logic
        ...
    if save_images and output_dir:
        cv2.imwrite(f"{output_dir}/composed_{
            image_id}.png", background)
```

RISULTATI


Speedup con 8 Processi
Speedup
Numero di Immagini
0
1
2
3
4
100
200
300
400
500
Joblib
Pool
Processes
Baseline
Speedup Ideale (8x)


Tempi di Esecuzione
Tempo (secondi)
Numero di Immagini
0
250
500
750
1000
1250
1500
1750
100
200
300
400
500
Sequenziale
Joblib
Pool
Processes


Efficienza Parallela
Efficienza
Numero di Immagini
0.0
0.2
0.4
0.6
0.8
1.0
1.2
100
200
300
400
500
Joblib
Pool
Processes
Efficienza Ideale

CONCLUSIONI

- Il metodo Pool risulta essere il migliore con uno speedup medio 3.91x con un picco di 4.23x a 350 immagini e un’efficienza del 50% (E=Sp/p)
- Il metodo Joblib mostra performance limitate a 2.28x

Table 2. Performance Summary - Key Results

Dataset Size	Sequential (s)	Joblib (s)	Pool (s)	Process (s)	Best Speedup
50	185.02	80.13	55.57	55.38	3.34x (Process)
100	366.29	173.05	97.95	98.33	3.74x (Pool)
200	732.51	308.70	182.12	182.16	4.02x (Pool)
300	1100.55	491.81	276.82	309.99	3.98x (Pool)
350	1330.57	544.59	314.85	316.46	4.23x (Pool)
500	1850.37	927.88	469.11	471.99	3.94x (Pool)
Average	**1012.22**	**447.26**	**264.09**	**267.40**	**3.91x (Pool)**