



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Salvatore Alfio Sambataro

Tecniche avanzate per la segmentazione di immagini

MULTIMEDIA - PROGETTO FINALE

Anno Accademico 2023 - 2024

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 3 |
| 1.1 | Tipologie di segmentazione | 4 |
| 1.2 | Tecniche di segmentazione classiche | 5 |
| 1.2.1 | Edge Based Segmentation | 5 |
| 1.2.2 | Thresholding Based Segmentation | 5 |
| 1.2.3 | Region Based Segmentation | 6 |
| 2 | Clustering-based Segmentation | 7 |
| 2.1 | Algoritmo "K-means" | 8 |
| 2.2 | Vantaggi e svantaggi degli algoritmi di clustering classici | 8 |
| 2.3 | Clustering e Deep Learning | 10 |
| 2.4 | Pipeline generale per il clustering basato su Deep Learning | 10 |
| 2.4.1 | Architettura della rete neurale | 11 |
| 2.4.2 | Funzioni di loss | 11 |
| 2.4.3 | Aggiornamento dei cluster | 13 |
| 2.4.4 | Applicazione opzionale di un ulteriore algoritmo di clustering | 14 |
| 3 | DeepCluster | 15 |
| 3.1 | Definizioni preliminari | 15 |
| 3.2 | Unsupervised Learning con il clustering | 16 |
| 3.3 | Come evitare soluzioni banali | 17 |
| 3.4 | Prestazioni di Deep Cluster sulla segmentazione | 18 |
| 4 | SAM: Segment Anything Model | 19 |
| 4.1 | Task, Modello, Dataset | 19 |
| 4.2 | SAM Task | 20 |
| 4.3 | SAM Model | 22 |
| 4.3.1 | Image Encoder | 22 |
| 4.3.2 | Prompt Encoder | 22 |
| 4.3.3 | Mask Decoder | 22 |

| | |
|---|-----------|
| <i>INDICE</i> | 2 |
| 4.4 SAM Dataset | 23 |
| 4.4.1 Generazione delle maschere del dataset SA-1B | 23 |
| 5 Implementazione di SAM | 25 |
| 5.1 Linguaggio e librerie utilizzate | 25 |
| 5.2 Funzionalità del software sviluppato | 25 |
| 5.2.1 Generazione di maschere per l'intera immagine | 26 |
| 5.2.2 Esecuzione dell'algoritmo SAM | 27 |
| 5.3 Generazione di maschere da prompt testuali | 32 |
| 5.3.1 OWL-ViT - Vision Transformer for Open-World Localization | 32 |
| 5.3.2 Pipeline di generazione delle maschere a partire dal prompt testuale | 32 |
| 5.3.3 Esecuzione dell'algoritmo di generazione di maschere da prompt testuale | 33 |
| Conclusione | 38 |
| Appendice: repository con codice | 39 |
| Bibliografia | 40 |

Capitolo 1

Introduzione

La segmentazione è definita come un processo che partiziona una data immagine in regioni omogenee, nella quale tutti i pixel che corrispondono ad uno stesso "oggetto" sono raggruppati insieme; queste informazioni possono essere poi utilizzate per identificare il contorno di una particolare regione d'interesse.

Il raggruppamento dei pixel è basato su criteri di omogeneità specifici, per esempio valori di similarità di attributi come il colore e l'intensità, oppure valori di prossimità spaziale (distanza euclidea, etc.).

Formalmente, sia R l'intera regione spaziale occupata dall'immagine. Il processo di segmentazione può essere visto come il partizionamento di R in n sottoregioni, R_1, R_2, \dots, R_n tali che:

- $\bigcup_{i=1}^n R_i = R$: ogni pixel deve appartenere ad una regione, e l'unione delle regioni restituisce l'immagine originale;
- R_i è un insieme连通的;
- $R_i \cap R_j \neq \emptyset$ per tutti i valori i, j tali che $i \neq j$: le regioni devono essere disgiunte;
- $Q(R_i) = \text{TRUE}$ per $i = 1, 2, \dots, n$: i pixel appartenenti ad una certa regione R_i devono soddisfare un certo predicato Q .
- $Q(R_i \cup R_j) = \text{FALSE}$ per ogni coppia di regioni adiacenti R_i, R_j con $Q(R_k)$ predicato definito sui punti di una regione R_k : regioni adiacenti sono diverse nel senso del predicato Q

1.1 Tipologie di segmentazione

La segmentazione di immagini può essere suddivisa in due principali tipologie:

- **Semantic Segmentation:** consiste nell'associare ad ogni pixel di un'immagine un'etichetta di classe. In questo caso, istanze di uno stesso "oggetto" avranno la stessa etichetta.
- **Instance Segmentation:** consiste nel rilevare ogni istanza distinta di un oggetto in un'immagine. In questo caso, istanze di uno stesso "oggetto" avranno comunque etichette diverse.

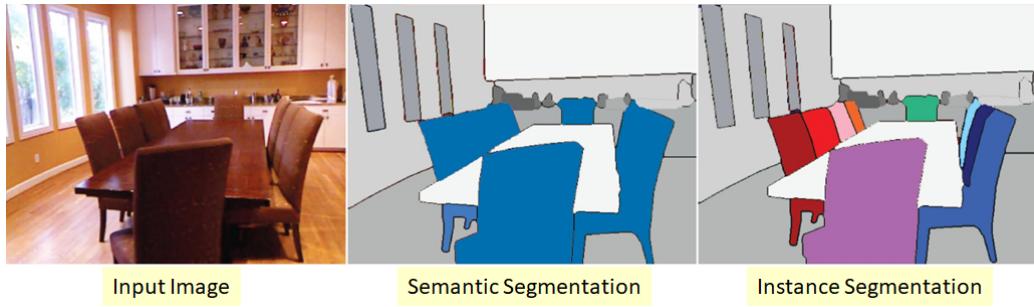


Figura 1.1: Tipologie di segmentazione

1.2 Tecniche di segmentazione classiche

La segmentazione rimane tutt'oggi un'operazione molto complessa, a causa della grande variabilità delle forme degli oggetti e della variazione della qualità delle immagini. Inoltre la difficoltà di utilizzo delle varie tecniche può aumentare a causa della presenza di rumore.

Esistono diversi approcci tradizionalmente utilizzati per effettuare la segmentazione di immagini.

I più comuni sono:

- **Edge Based Segmentation**
- **Thresholding Segmentation**
- **Region Based Segmentation**

1.2.1 Edge Based Segmentation

La segmentazione edge-based è una tecnica di segmentazione delle immagini che utilizza i contorni dell'oggetto per separare quest'ultimo dallo sfondo. Tali edge possono essere individuati con varie tecniche di edge detection, tra cui, ad esempio, il filtro Sobel o il metodo di Canny.

La segmentazione basata su contorni può essere utile quando l'obiettivo è segmentare oggetti con contorni ben definiti e contrastanti rispetto allo sfondo. Tuttavia, questa tecnica può avere difficoltà a segmentare oggetti con contorni sfumati o sfondi complessi.

1.2.2 Thresholding Based Segmentation

La segmentazione basata su thresholding (o "sogliatura") è uno dei metodi più semplici e comuni per segmentare un'immagine. Questo metodo utilizza una soglia per separare i pixel dell'immagine in due classi, una classe che corrisponde agli oggetti (o alle regioni) dell'immagine e l'altra classe che corrisponde allo sfondo.

La soglia viene definita come un valore numerico che viene confrontato con il valore di intensità di ogni pixel dell'immagine. Se il valore di intensità del pixel supera la soglia, il pixel viene assegnato alla classe degli oggetti; altrimenti viene assegnato alla classe dello sfondo.

Tuttavia, questo metodo ha alcune limitazioni, poiché può essere influenzato

da fattori come il rumore dell'immagine e la variazione della luminosità.

1.2.3 Region Based Segmentation

La segmentazione region-based è una tecnica di segmentazione delle immagini che, a partire da un certo pixel, detto "seed", agglomera ad esso i pixel a lui vicini che soddisfano un certo criterio di omogeneità, dando così vita ad una regione. Combinando diversi processi di growing da più seed, si ottiene la segmentazione dell'intera immagine.

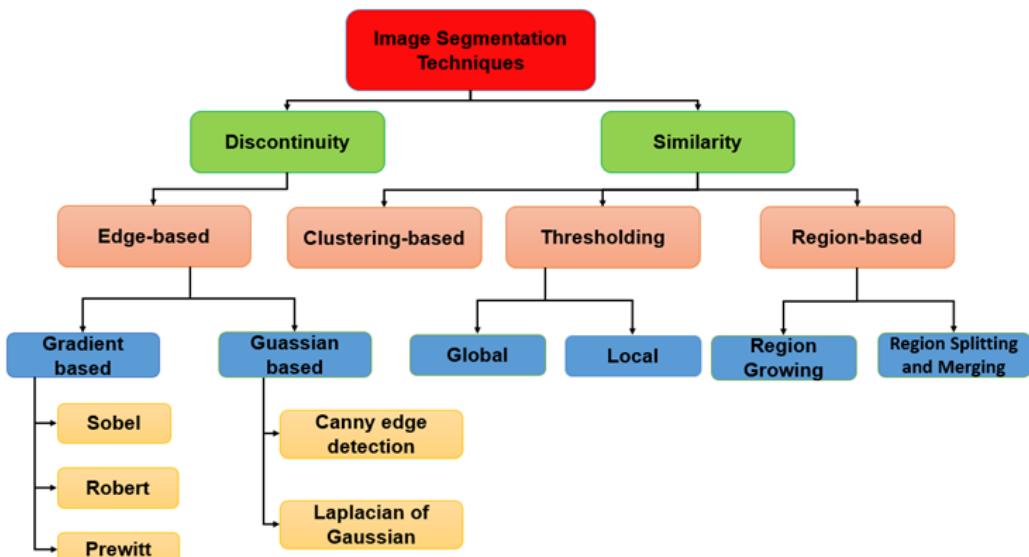


Figura 1.2: Tecniche di segmentazione classiche

Capitolo 2

Clustering-based Segmentation

La segmentazione basata sul clustering è una tecnica di segmentazione delle immagini che utilizza algoritmi di clustering per suddividere l'immagine in regioni omogenee basate sulle proprietà dei pixel dell'immagine stessa.

L'obiettivo del clustering è dividere un dato insieme di dati in K gruppi, detti "cluster". In particolare, ci aspettiamo che i cluster siano tali che gli elementi all'interno di un cluster siano simili tra loro, mentre gli elementi di cluster diversi sono diversi gli uni dagli altri.

2.1 Algoritmo "K-means"

Un algoritmo di clustering classico utilizzato per la segmentazione di immagini è l'algoritmo K-means. L'obiettivo dell'algoritmo è quello di dividere i dati di input in un dato numero K di gruppi (cluster), ove K deve essere deciso a priori. Ogni cluster sarà "rappresentato" da uno specifico punto, detto "centroide": tale punto è solitamente definito come il punto medio tra i punti che appartengono al cluster che esso rappresenta.

L'algoritmo K-means è il seguente:

Algorithm 1 K-means

```

1: initialize K random centroids  $\mu_1, \dots, \mu_K$ 
2: while centroids vary do
3:   for each point do
4:     Calculate the distances from every point to all centroids
5:     Assign every point to the nearest group/centroid
6:   end for
7:   Update centroids
8: end while
```

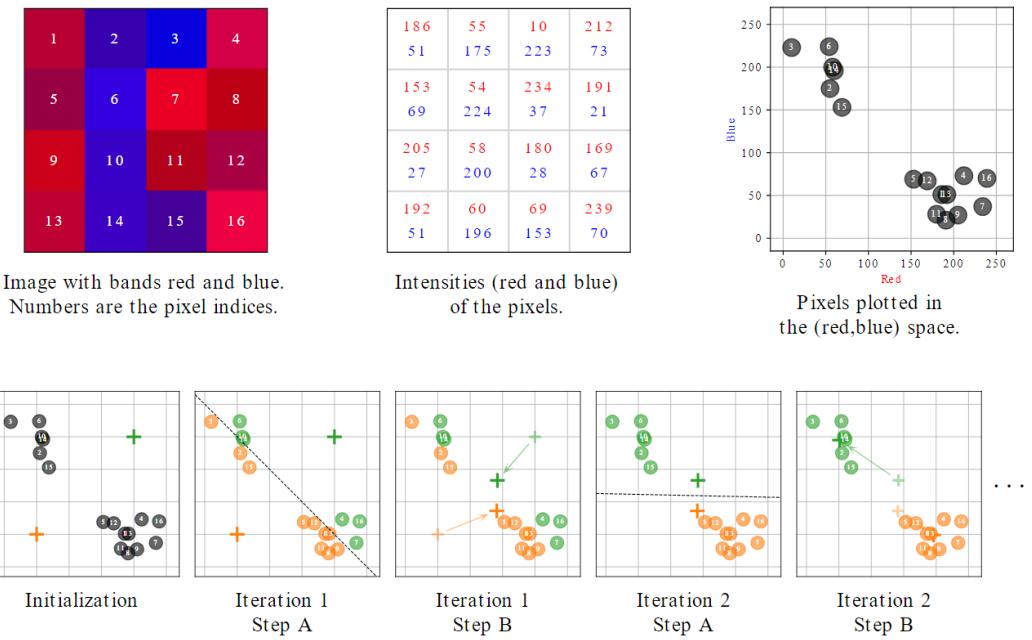
2.2 Vantaggi e svantaggi degli algoritmi di clustering classici

Nell'ambito della segmentazione di immagini, i vantaggi dell'utilizzo dell'algoritmo K-Means sono i seguenti:

- **Semplicità**
- **Facile da implementare**
- **Generalmente veloce**

Gli svantaggi sono invece i seguenti:

- **Richiede che il numero K di cluster sia noto a priori**
- **Sensibile all'inizializzazione dei centroidi**
- **Può essere molto lento con una quantità elevata di dati**
- **L'algoritmo non funziona nel caso di forme "non sferiche"**
- **Sensibile agli outlier**



Progression of the K-means algorithm (only the initialization and the first two iterations are illustrated).
Centroids are represented by +, the dotted line indicates the boundary between the two classes.

Figura 2.1: Fonte: Vincmazet

2.3 Clustering e Deep Learning

L'obiettivo principale del clustering è separare i dati in gruppi di punti dati simili. Le prestazioni degli algoritmi di clustering classici tuttavia dipendono fortemente dai dati di input. Differenti dataset infatti richiedono differenti misure di similarità e di separazione dei cluster. Per questo motivo, gli algoritmi di clustering sono spesso utilizzati dopo aver applicato delle tecniche di Dimensionality Reduction, representation learning, o più in generale tecniche che permettono di mappare i dati di input in uno "spazio di feature" ove la separazione in cluster è più semplice.

Utilizzando il deep learning e le reti neurali profonde, è possibile effettuare il learning di mapping non lineari, i quali permettono di trasformare i dati in una rappresentazione più "cluster-friendly", senza la necessità di dover effettuare manualmente la selezione o l'estrazione di feature.

2.4 Pipeline generale per il clustering basato su Deep Learning

La maggior parte degli algoritmi di clustering che fanno uso di reti neurali funzionano come segue:

1. representation learning sui dati, usando le reti neurali
2. usare la rappresentazione ottenuta come input di un generico algoritmo di clustering

Più nello specifico, la pipeline è la seguente:

1. Scelta dell'architettura della rete neurale
2. Scelta delle funzioni di loss
3. Scelta delle feature da utilizzare per il clustering
4. Training della rete neurale
5. Aggiornamento dei cluster
6. (Opzionale) eseguire di nuovo un algoritmo di clustering dopo aver effettuato il training della rete

2.4.1 Architettura della rete neurale

Nella maggior parte dei metodi di clustering basati su reti neurali, quest'ultima è usata per ottenere una nuova rappresentazione dell'input da usare poi per effettuare il clustering.

Le architetture utilizzate maggiormente sono:

- **Multylayer Perceptron (MLP)**
- **Convolutional Neural Netowrk (CNN)**
- **Deep Belief Network (DBN)**
- **Generative Adversarial Network (GAN)**
- **Variational AutoEncoder (VAE)**

2.4.2 Funzioni di loss

Esistono tre tipologie di funzioni di loss utilizzabili in questi casi:

- **Non-Clustering Loss.** Alcuni esempi:
 - **Autoencoder Reconstruction Loss:** un autoencoder è formato da un encoder e un decoder. L'encoder mappa un input x in una nuova rappresentazione z in uno spazio latente Z . Durante il training, il decoder tenta di ricostruire x a partire da z . Nell'ambito del clustering, dopo la fase di training il decoder non viene più usato, mentre l'encoder è usato per mappare nuovi dati in Z . La Reconstruction Loss è una misura di distanza) tra l'input dell'autoencoder x_i e la corrispondente ricostruzione $f(x_i)$. Una sua possibile formulazione, basata sull'errore quadratico medio, è la seguente:

$$L = d_{AE}(x_i, f(x_i)) = \sum_i \|x_i - f(x_i)\|^2 \quad (2.1)$$

ove x_i è l'input e $f(x_i)$ è la ricostruzione calcolata dall'autoencoder.

- **Self Augmentation Loss:** funzione di loss che considera la rappresentazione originale dell'input e la sua "augmentation":

$$L = \frac{1}{N} \sum_N s(f(x), f(T(x))) \quad (2.2)$$

ove x è il dato originale, T è la "augmentation function", $f(x)$ è la rappresentazione generata dal modello, s è una qualche misura di similarità.

- **Clustering Loss.** Alcuni esempi:

- **k-Means Loss:** questa loss assicura che la nuova rappresentazione appresa dalla rete neurale sia "k-Means friendly", ovvero sia adatta per l'esecuzione dell'algoritmo k-Means (ad esempio, i data-point sono ugualmente distribuiti attorno i centroidi scelti). Tale funzione di loss può essere definita come segue:

$$L(\theta) = \sum_{i=1}^N \sum_{k=1}^K s_{ik} \|z_i - \mu_k\|^2 \quad (2.3)$$

ove z_i è un punto del nuovo spazio, μ_k è un centroide e s_{ik} è una variabile booleana che indica l'assegnazione di z_i al centroide μ_k . Minimizzare questa funzione di loss rispetto ai parametri θ della rete neurale assicura che la distanza tra i data point e il centroide del rispettivo cluster sia piccola. Applicando poi k-Means si avranno di conseguenza dei risultati migliori.

- **Balanced Assignment Loss:** questa funzione di loss è usata in combinazione con altre funzioni di loss. L'obiettivo è "forzare" assegnamenti bilanciati tra i diversi cluster.

Può essere formulata come segue:

$$L_{ba} = KL(G\|U) \quad (2.4)$$

ove U è la distribuzione di probabilità uniforme e G è la distribuzione di probabilità di assegnare un punto ad un certo cluster:

$$g_k = P(y = k) = \frac{1}{N} \sum_i q_{ik} \quad (2.5)$$

Minimizzando l'equazione 2.4, la probabilità di assegnare ogni data point ad un certo cluster è uniforme per tutti i possibili cluster. Tale proprietà di "assegnazione uniforme" non sempre è desiderabile. Inoltre, è possibile sostituire la distribuzione uniforme con altre distribuzioni note.

- **Locality-preserving Loss:** questa funzione di loss mira a preservare la località dei cluster raggruppando tra loro punti dati vicini.

Può essere formulata come segue:

$$L_{lp} = \sum_i \sum_{j \in N_k(i)} s(x_i, x_j) \|z_i - z_j\|^2 \quad (2.6)$$

ove $N_k(i)$ è l'insieme dei k *nearest neighbors* del data point x_i , mentre $s(x_i, x_j)$ è una qualche misura di similarità tra i data point x_i e x_j .

- **Combinazione di Clustering Loss e Non-Clustering Loss:** è possibile combinare funzioni loss di diverso tipo.
Una funzione di loss di questo tipo può essere definita come segue:

$$L(\theta) = \alpha L_c(\theta) + (1 - \alpha) L_n(\theta) \quad (2.7)$$

ove $L_c(\theta)$ è una funzione di tipo Clustering Loss, $L_n(\theta)$ è una funzione di tipo Non-Clustering Loss, mentre $\alpha \in [0, 1]$ è un iper-parametro della rete usato per pesare l'influenza delle due funzioni di loss.

2.4.3 Aggiornamento dei cluster

Durante il training della rete, vengono aggiornati l'assegnamento dei data point ai cluster e i centroidi (se si usa un algoritmo di clustering di tipo "centroid-based").

L'aggiornamento degli assegnamenti ai cluster può avvenire in due modi:

- **Aggiornato in contemporanea con la rete neurale:** gli assegnamenti ai cluster sono formulati come probabilità (valori continui tra 0 e 1). In questo caso, possono essere considerati come dei parametri della rete ed essere quindi ottimizzati, ad esempio tramite backpropagation.
- **Aggiornamento separato dalla rete:** gli assegnamenti ai cluster sono "rigidi" e aggiornati in passi differenti rispetto ai passi di aggiornamento della rete.

In questo caso sono possibili diversi scenari, sulla base di:

- **Numero di iterazioni:** numero di iterazioni dell'algoritmo di clustering scelto, eseguite ad ogni passo di aggiornamento dei cluster.
- **Frequenza degli aggiornamenti:** in pratica, quanto spesso viene eseguito l'aggiornamento dei cluster.

2.4.4 Applicazione opzionale di un ulteriore algoritmo di clustering

Una volta terminato il training della rete, anche se sono stati individuati dei cluster, può essere utile ri-eseguire un algoritmo di clustering usando le feature apprese precedentemente.

Ciò può essere fatto per diverse ragioni:

- **Clustering su un dataset simile**
- **Migliorare i risultati ottenuti:** è stato dimostrato che in certi casi i risultati del clustering dopo il training della rete sono migliori rispetto al clustering ottenuto durante la fase di learning. Una possibile ragione per cui ciò accade è che la fase di aggiornamento dei cluster non viene conclusa (ad esempio a causa di un numero troppo basso di iterazioni dell'algoritmo di clustering scelto).

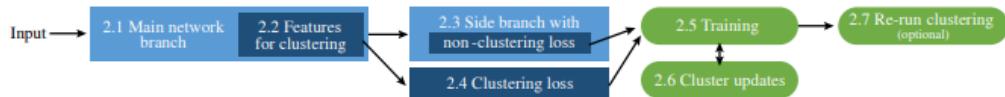


Figura 2.2: Pipeline di un algoritmo di clustering basato su Deep Learning

Capitolo 3

DeepCluster

DeepCluster, sviluppato da Meta, è un diverso approccio al clustering, basato su reti neurali convoluzionali, o *"convnet"*, che permette di **ottenere delle "visual feature"** utilizzabili per diversi task ("general purpose"). Tale approccio consiste nell'alternare l'applicazione del clustering ai descrittori dell'immagine e l'aggiornamento dei pesi della rete neurale in modo da predire l'assegnamento dei cluster.

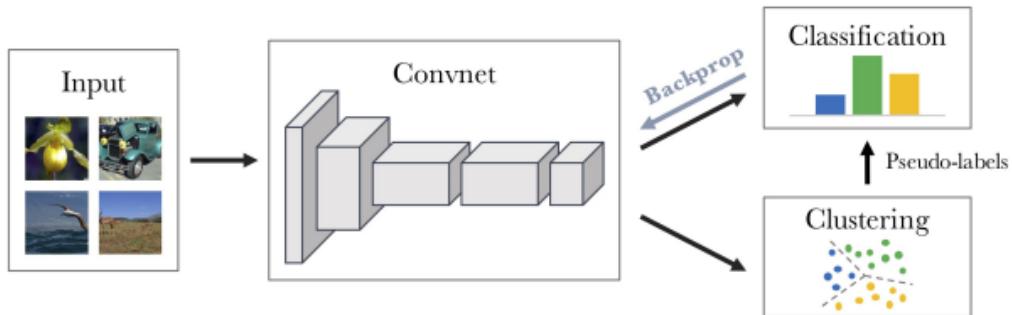


Figura 3.1: Pipeline di DeepCluster: in maniera iterativa, si effettua il clustering delle feature e poi si usano gli assegnamenti dei cluster come delle "pseudo-label" usate per apprendere i parametri della rete

3.1 Definizioni preliminari

Nel contesto dell'estrazione di feature da un'immagine, le convnets sono la scelta ottimale per mappare immagini raw in un vettore appartenente ad uno spazio con una certa dimensionalità.

Denotiamo con f_θ il mapping ottenuto dalla convnet, ove θ è l'insieme dei parametri della rete. Per indicare il vettore ottenuto applicando il mapping f_θ ad un immagine, esso verrà detto **"feature"** o **"representation"**.

Dato un training set $X = x_1, x_2, \dots, x_N$ con N immagini, vogliamo trovare l'insieme dei parametri θ^* tali che il mapping f_{θ^*} produca delle buone **"visual feature"**.

Solitamente il learning dei parametri viene effettuato con **metodi supervisionati**: ad ogni immagine x_n è associata una label $y_n \in \{0, 1\}^k$. Tale etichetta indica l'appartenenza dell'immagine ad una specifica **"classe"** tra k possibili classi predefinite.

Un classificatore parametrizzato g_W predice le etichette corrette sulla base delle feature $f_\theta(x_n)$. I parametri W del classificatore e i parametri θ del mapping possono essere appresi contemporaneamente, minimizzando la seguente equazione:

$$\min_{\theta, W} \frac{1}{N} \sum_{n=1}^N \ell(g_W(f_\theta(x_n)), y_n) \quad (3.1)$$

ove ℓ è la funzione di loss logistica multinomiale.

3.2 Unsupervised Learning con il clustering

Nel caso in cui i parametri sono campionati da una distribuzione Gaussiana, senza alcuna fase di learning, il mapping f_θ non permette di ottenere delle feature randomiche, che sorprendentemente offrono risultati ottimi per i task più comuni.

Nel caso di Deep Cluster, si effettua il clustering dell'output della rete neurale, per poi usare gli assegnamenti trovati come **"pseudo-label"** per ottimizzare l'equazione 3.1. Quello che si fa è quindi **apprendere iterativamente le feature e raggrupparle**.

Sebbene sia possibile usare un qualunque algoritmo di clustering, è possibile usare un algoritmo standard, ad esempio K-Means, in quanto le prestazioni non variano in maniera significativa.

Formalmente, K-Means riceve in input un insieme di **vettori di feature**, ovvero nel caso di Deep Cluster i vettori $f_\theta(x_n)$ prodotti dalla rete neurale, e li raggruppa in k diversi gruppi sulla base di **criteri geometrici**. Più nello specifico, l'algoritmo permette di calcolare una **matrice di centroidi** C di dimensioni $d \times k$ e gli **assegnamenti ai cluster** y_n di ogni immagine n risolvendo il seguente problema:

$$\min_{C \in \mathbb{R}^{d \times k}} \frac{1}{N} \sum_{n=1}^N \|f_\theta(x_n) - Cy_n\|_2^2, \text{ tale che } y_n^\top 1_k = 1 \quad (3.2)$$

Risolvere questo problema permette di ottenere un **insieme di assegnamenti ottimali** $(y_n^*)_{n \leq N}$ e una **matrice di centroidi** C^* . Tali assegnamenti sono poi usati come "pseudo-label", mentre la matrice dei centroidi non verrà utilizzata.

In generale, DeepCluster alterna tra l'applicazione del clustering alle feature al fine di produrre delle "pseudo-label" utilizzando l'equazione 3.2, e l'aggiornamento dei parametri della convnet usando l'equazione 3.1.

3.3 Come evitare soluzioni banali

Il procedimento adottato da Deep Cluster è incline a proporre **soluzioni banali**. Per risolvere questo problema, le soluzioni proposte riguardano principalmente l'introduzione di vincoli nella fase di clustering, ad esempio penalizzando cluster con un numero minimo di punti.

Più nello specifico, le cause principali per cui l'algoritmo propone soluzioni banali possono essere:

- **Cluster vuoti:** una soluzione semplice consiste nel riassegnare i cluster vuoti durante l'esecuzione dell'algoritmo di clustering: nello specifico, nel momento in cui un cluster è vuoto, si seleziona randomicamente un cluster non vuoto e si usa il relativo centroide, con piccole modifiche, come centroide del cluster che risultava essere vuoto. A questo punto, si procede a riassegnare i punti facenti parte del cluster non vuoto, suddividendoli tra i due cluster risultanti.
- **Scelta banale dei valori dei parametri:** se la maggior parte delle immagini è assegnata a pochi cluster, i parametri θ considereranno solo tali cluster. Nel caso peggiore, tutti i cluster tranne uno avranno una sola immagine al loro interno, e in questi casi minimizzare l'equazione 3.1 equivale a ottenere un insieme di parametri θ tali che la rete neurale restituisce sempre lo stesso output, indipendentemente dall'input. Una strategia per affrontare questo problema è quella di campionare le immagini sulla base di una distribuzione uniforme delle classi di appartenenza, o sulla base delle pseudo-label: ciò è equivalente a pesare il contributo di un input nella funzione di loss sulla base dell'inverso della dimensione del cluster a cui esso è stato assegnato.

3.4 Prestazioni di Deep Cluster sulla segmentazione

La valutazione da parte di Meta per Deep Cluster, per quanto riguarda la Semantic Segmentation, è stata effettuata sul Dataset ***Pascal VOC 2012***. In tabella 3.1 le performance di Deep Cluster sono confrontate con altri approcci di feature-learning che rappresentano lo stato dell'arte, in particolare riguardo il task di segmentazione. Come possiamo notare, il miglioramento introdotto da Deep Cluster rispetto agli altri metodi, nell'ambito della segmentazione, è del 7.5%.

| Method | Segmentation Results |
|-------------------------|----------------------|
| Random-rgb | 30.1 |
| Random-sobel | 32.0 |
| Pathak <i>et al.</i> | 29.7 |
| Donahue <i>et al.</i> * | 35.2 |
| Pathak <i>et al.</i> | - |
| Owens <i>et al.</i> * | - |
| Wang and Gupta * | 35.4† |
| Doersch <i>et al.</i> * | - |
| Bojanowski and Joulin * | 37.1 |
| Zhang <i>et al.</i> * | 35.6 |
| Zhang <i>et al.</i> * | 36.0 |
| Noroozi and Favaro * | 37.6 |
| Noroozi <i>et al.</i> * | 36.6 |
| Deep Cluster | 45.1 |

Tabella 3.1: Performance di diversi metodi sulla segmentazione.

* indica l'uso dell'inizializzazione proposta in *Krähenbühl et al.*

Capitolo 4

SAM: Segment Anything Model

Segment Anything Model, abbreviato **SAM**, è un modello di Image Segmentation sviluppatto da Meta AI, rilasciato in Aprile 2023. Esso permette di identificare l'esatta posizione di ognuno degli oggetti presenti in un'immagine, o in alternativa solo alcuni di essi.

La particolarità di SAM è quella di essere un modello **promptable**, ovvero è possibile scegliere quali parti dell'immagine segmentare.

Inoltre, è un modello di tipo **zero-shot** : ciò significa che può generare delle **maschere di segmentazione** anche per oggetti che non ha mai incontrato prima, senza necessità di effettuare una nuova fase di training del modello.

Grazie a queste sue caratteristiche, SAM può essere considerato un **foundation model per l'Image Segmentation**, ovvero è adatto ad una vasta gamma di casi d'uso.

4.1 Task, Modello, Dataset

La costruzione del modello SAM ha richiesto la definizione di tre "componenti":

- Task
- Modello
- Dataset

4.2 SAM Task

La definizione di un "task" è effettuata prendendo ispirazione dalla branca del *Natural Language Processing*. Inizialmente bisogna trasferire l'idea di prompt in NLP (ovvero del testo) nell'ambito della segmentazione: degli esempi di prompt possono essere:

- Insieme di punti di background/foreground
- Box/Area dell'immagine che racchiude gli oggetti da segmentare
- Testo. In realtà questa possibilità non è ancora implementata dal modello, per cui si tenterà di implementarla, come mostrato in sez. 5.3
- In generale, **qualsiasi informazione utile ad indicare cosa segmentare nell'immagine**

Possiamo quindi definire il task ricercato come "**restituire una maschera di segmentazione valida dato un qualsiasi prompt**", ove "**valida**" significa che, in presenza di ambiguità, l'output sia in qualche modo "**valido**" (esempio in fig. 4.1).

Il task proposto in questa forma suggerisce anche un metodo di training molto semplice, che consiste nel **simulare una sequenza di prompt e comparare le maschere prodotte dall'algoritmo con le maschere "ground-truth"**



Figura 4.1: Esempi di prompt ambigui (punti verdi) e relative maschere " valide" (evidenziate con bordo rosso)

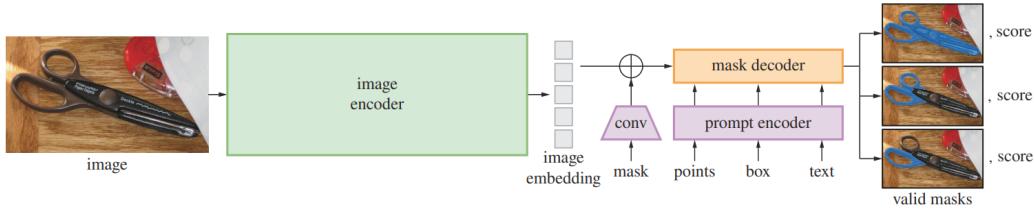


Figura 4.2: Overview del Modello SAM: un encoder restituisce un embedding dell’immagine che può essere interrogato tramite prompt in maniera efficiente, al fine di produrre delle maschere di segmentazione ottimali. Il modello inoltre permette di restituire diverse maschere, ognuna con un proprio ”confidence score”.

4.3 SAM Model

Il modello SAM prevede tre componenti principali (fig. 4.2):

- un **Image Encoder**
- un **Prompt Encoder**
- un **Mask Decoder**

4.3.1 Image Encoder

Viene fatto uso di un **Vision Transformer (ViT)** adattatto per processare input ad alta risoluzione. Tale encoder viene usato una volta per immagine, e può essere applicato ancor prima della fase di prompting.

4.3.2 Prompt Encoder

Consideriamo due tipologie di prompt: **prompt sparsi** (punti, box, testo) e **prompt densi** (maschere). Punti e Box sono rappresentati con ”**positional encoding**”, oltre a embedding imparati per essi. Per quanto riguarda il testo, viene fatto uso dell’encoder CLIP [82].

I prompt densi, ovvero maschere, sono invece rappresentati tramite convoluzione dell’embedding dell’immagine.

4.3.3 Mask Decoder

Il mask decoder mappa l’embedding dell’immagine, l’embedding del prompt e un ”token di output” in una maschera.

In caso di prompt ambiguo, poichè un output unico avrebbe la necessità di effettuare una media di diverse maschere valide e restituirne una sola, in

SAM vengono invece restituite 3 maschere valide: la scelta di restituirne 3 è dettata dal fatto che, nella maggior parte dei casi, "maschere annidate" rappresentano spesso tre "profondità" dell'immagine: **immagine intera, parte dell'immagine, "sottoparte" dell'immagine**.

Per ognuna delle maschere, il modello predice un "confidence score", detto "**estimated IoU**".

4.4 SAM Dataset

Poichè prima della pubblicazione di SAM non erano presenti dataset contenenti maschere di segmentazione abbastanza grandi, l'idea è stata quella di costruire un nuovo dataaset per la Image Segmentation, chiamato **SA-1B**. Esso contiene **11M** immagini e **11.B** maschere di segmentazione create come spiegato in sezione 4.4.1.

Le immagini avevano in origine una risoluzione di 3300×4950 pixel, ma al fine di evitare problemi di memorizzazione e accessibilità, le immagini sono state "sottocampionate" portando il lato più corto di ognuna di esse a 1500 pixel. Nonostante ciò, la risoluzione delle immagini rimane molto maggiore rispetto ai più comuni dataset disponibili in rete.

4.4.1 Generazione delle maschere del dataset SA-1B

Poichè, come detto, non sono presenti dataset di maschere di segmentazione molto estesi, SAM introduce una **pipeline per la creazione automatica di maschere di segmentazione**, la quale prevede i seguenti passi:

- **Model-assisted manual annotation:** in questa fase, un team di professionisti ha etichettato le maschere di segmentazione distinguendo i punti di foreground e background dell'immagine, e poi le maschere generate dal modello sono state rifinite tramite appositi tool. All'inizio di questa fase, SAM è stato allenato usando dataset di segmentazione standard; poi, dopo un numero sufficiente di annotazioni da parte degli esperti, il modello è stato nuovamente allenato usando solo queste ultime.

In questa fase sono state raccolte **4.3M di maschere da 120k immagini**.

- **Semi-automatic Generation:** in questa fase, per incrementare la diversità delle maschere e migliorare la capacità del modello di segmentare un'ampia gamma di oggetti, inizialmente sono state generate automaticamente delle maschere, poi esse sono state mostrate a dei

professionisti, i quali hanno migliorato e rifinito le maschere generate. In questa fase sono state raccolte altre **5.9M di maschere da 180k immagini**.

- **Fully Automatic Generation:** in questa fase finale, l'annotazione delle maschere è del tutto automatica, ovvero interamente affidata al modello, ed ha riguardato tutte le 11M di immagini del dataset. Alla fine di questa fase, è stato raccolto un totale di 1.1B di maschere.



Figura 4.3: Esempi di segmentazione con SAM, con accanto il numero di maschere per immagine

Capitolo 5

Implementazione di SAM

Il presente progetto ha previsto l'implementazione di SAM, ”***Segment Anything Model***”, in linguaggio Python, per verificarne il funzionamento. Esso è stato implementato in linguaggio Python, con l'utilizzo di apposite librerie, e la valutazione delle sue funzionalità è stata effettuata su immagini con contesti totalmente differenti, scaricate dalla rete. Mediante tali esperimenti, si è cercato di ottenere una comprensione chiara e dettagliata delle capacità e dei limiti di SAM.

5.1 Linguaggio e librerie utilizzate

Il linguaggio utilizzato per l'implementazione del presente progetto è il linguaggio Python, principalmente per la facilità di scrittura del codice e la vasta gamma di librerie disponibili.

Le librerie utilizzate per il presente progetto sono le seguenti:

- **Matplotlib**: creazione di grafici e immagini interattivi
- **OpenCV**: libreria open source che fornisce i più famosi algoritmi di Computer Vision e Machine Learning
- **PyTorch**: utile per task di deep learning e computer vision
- **Cuda**: usata per l'implementazione di procedure parallelizzate su GPU

5.2 Funzionalità del software sviluppato

Il software sviluppato prevede la possibilità di caricare un'immagine e scegliere il modello di SAM da utilizzare. Successivamente, è data la possibilità di

effettuare la segmentazione dell’immagine ricevuta in input attraverso diverse tipologie di prompt:

- **Single Point**: l’utente ha la possibilità di cliccare col mouse sul punto che intende usare come prompt/input per l’algoritmo
- **Multiple Point**: l’utente ha la possibilità di cliccare col tasto sinistro del mouse tutti i punti di interesse. Si può inoltre cliccare col tasto destro del mouse per evidenziare invece i punti che NON sono di interesse per la segmentazione.
- **Box**: l’utente ha la possibilità di creare all’interno dell’immagine un “box” che indica l’area di interesse in cui effettuare la segmentazione

Alla fine dell’esecuzione dell’algoritmo, verranno mostrate a schermo le maschere calcolate, ognuna con il relativo **”IoU score”**.

5.2.1 Generazione di maschere per l’intera immagine

Il software offre la possibilità di calcolare tutte le possibili maschere per l’immagine ricevuta in input.

A tal fine, è possibile modificare alcuni parametri dell’algoritmo: sono riportati di seguito i più importanti (per ulteriori parametri, fare affidamento alla documentazione di SAM):

- **points_per_side**: numero di punti campionati lungo ciascun lato dell’immagine. Il numero di punti totali sarà pari a point_per_side^2 . Più è alto tale valore, più le maschere saranno “precise”, ma sarà richiesto un tempo maggiore per il calcolo delle maschere;
- **pred_iou_thresh**: soglia minima di ”IoU score” per considerare ”valida” una specifica maschera, compresa tra 0 e 1. Se tale valore è alto, consideriamo le maschere con punteggi più elevati.
- **stability_score_thresh**: soglia minima di ”stability score” per considerare ”valida” una specifica maschera, compresa tra 0 e 1. Se tale valore è alto, consideriamo le maschere con punteggi più elevati.
- **crop_n_layers**: numero di layer da rimuovere dall’output del modello. Se settato a 1, solo il primo layer dell’output del modello sarà usato per la generazione delle maschere
- **crop_n_points**: fattore di downscaling applicato ai punti nel momento in cui si effettua il crop dell’output del modello. Se settato a 2, i punti subiranno un downscaling di un fattore 2 prima del crop dell’output

- **min_mask_region_area**: area minima in pixel per considerare "valida" una specifica maschera.

Una volta eseguita la generazione di tutte le maschere, saranno restituite diverse informazioni riguardanti ognuna di esse (rif. 5.1)

5.2.2 Esecuzione dell'algoritmo SAM

Per eseguire l'algoritmo, basta posizionarsi all'interno della cartella fornita in allegato al presente progetto, aprire una finestra CLI ed eseguire il seguente comando:

```
python SAM.py <image_path> <model>
```

Il significato e i possibili valori dei parametri sono i seguenti:

| Parametro | Significato | Possibili valori |
|------------|--------------------|--|
| image_path | Path dell'immagine | - |
| model | Modello utilizzato | h : usa il modello <i>Vit_h</i> b : usa il modello <i>Vit_b</i> l : usa il modello <i>Vit_l</i> |

Tabella 5.1: Parametri di ognuna delle maschere generate

| Valore | Tipo | Significato |
|-----------------|-------------------|--|
| segmentation | np.ndarray | la maschera vera e propria |
| area | int | dimensione della maschera, in pixel |
| bbox | List[int] | il contorno della maschera, in formato xywh |
| predicted_iou | float | IoU score predetto dal modello |
| point_coords | List[List[float]] | lista di input point che hanno generato tale maschera |
| stability_score | float | altra misura di qualità della maschera, oltre lo IoU score |
| crop_box | List[int] | sezione dell'immagine usata per la generazione di tale maschera, in formato xywh |



Figura 5.1: Esempio di segmentazione "Single point": l'algoritmo genera 3 maschere con punteggi differenti. La prima e la terza maschera risultano essere le migliori (esse ricoprono quasi per intero l'oggetto), mentre la seconda risulta essere la peggiore (possiamo notare che non tutto l'oggetto viene da essa coperto)

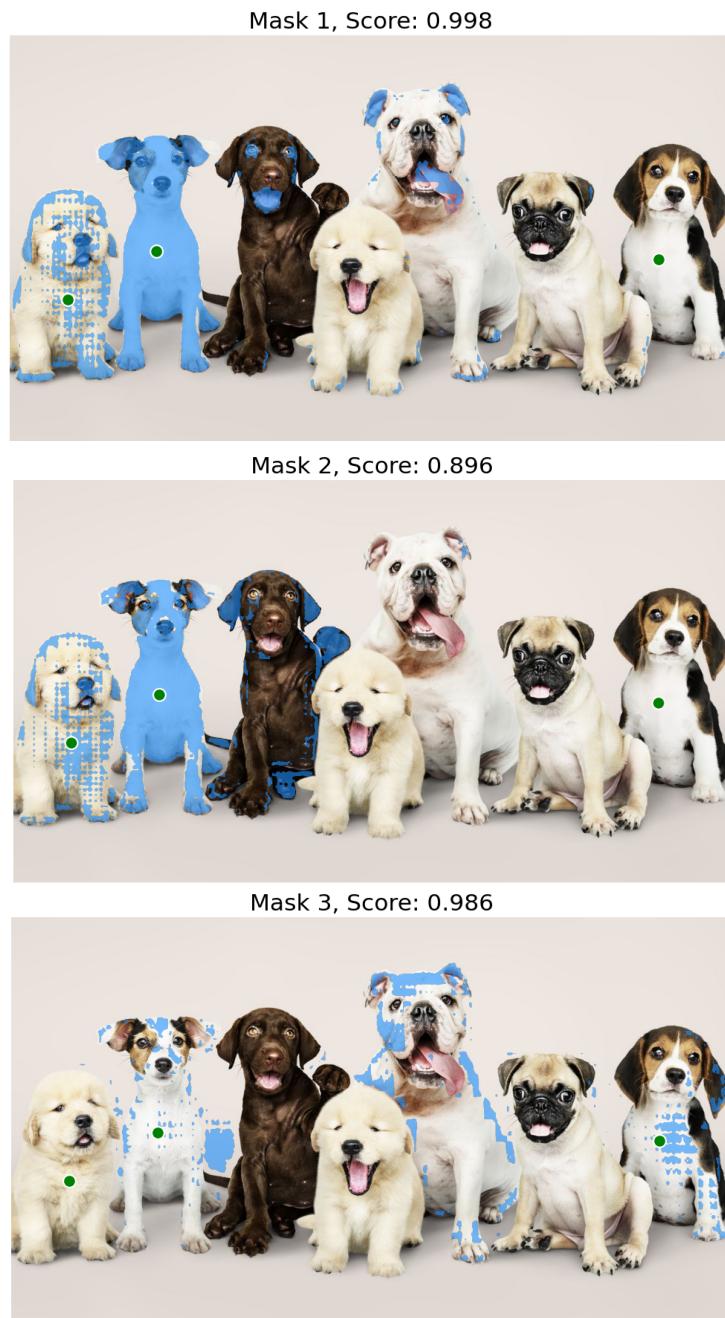


Figura 5.2: Esempio di segmentazione "Multiple point": nonostante i punteggi siano elevati, nessuna delle maschere sembra essere "valida")



Figura 5.3: Esempio di segmentazione con "Box"



Figura 5.4: Esempio di generazione di maschere per ognuno degli oggetti dell'immagine.

Parametri utilizzati: `points_per_side=10`, `pred_iou_thresh=0.995`,
`stability_score_thresh=0.95`, `crop_n_layers=1`,
`crop_n_points_downscale_factor=2`, `min_mask_region_area=100`

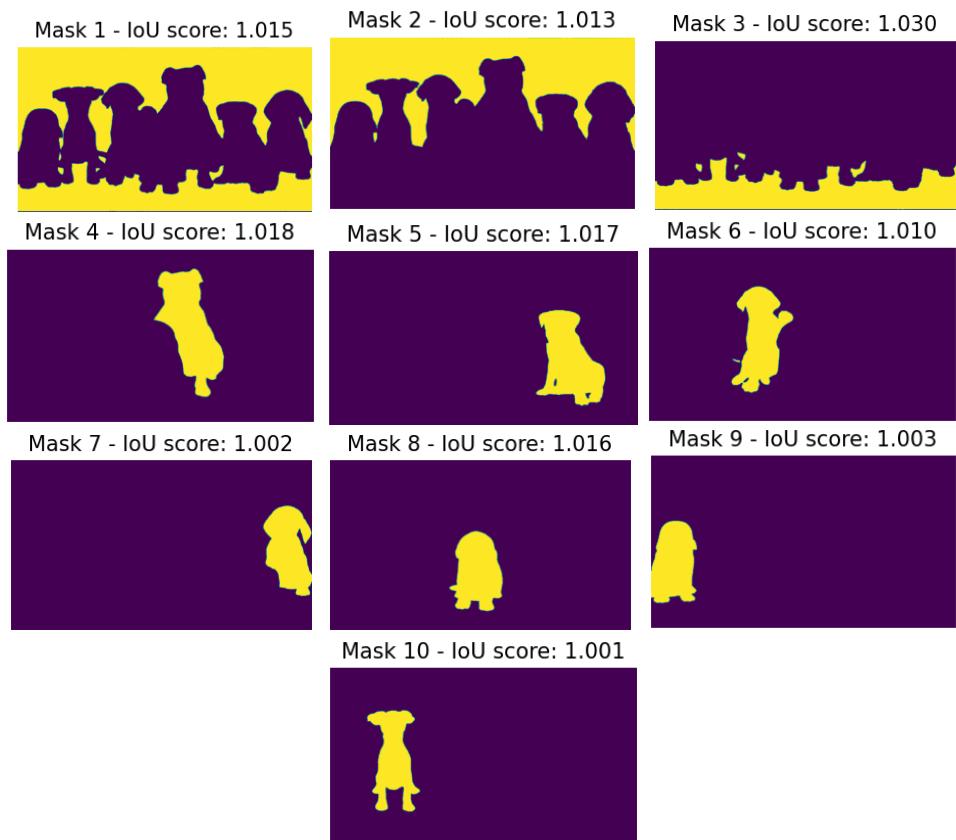


Figura 5.5: Maschere generate nell'esempio precedente

5.3 Generazione di maschere da prompt testuali

L'implementazione ufficiale di SAM non prevede attualmente la generazione di maschere di segmentazione di una data immagine sulla base di un prompt testuale.

In questa seconda parte del progetto si è tentato quindi combinare SAM con un modello di Object Detection per dare modo all'utente di effettuare la segmentazione con un prompt testuale. Si è scelto a tal fine di usare il modello di Object Detection **OWL-ViT**.

5.3.1 OWL-ViT - Vision Transformer for Open-World Localization

OWL-ViT (abbreviazione di **Vision Transformer for Open-World Localization**) è un'architettura basata su Trasformers che permette di cercare ed individuare oggetti all'interno di un'immagine a partire da un prompt testuale.

La scelta è ricaduta su tale modello in quanto **permette di effettuare task di object detection senza alcuna fase di training pregressa**, in quanto si tratta di un modello zero-shot.

5.3.2 Pipeline di generazione delle maschere a partire dal prompt testuale

La generazione delle maschere a partire dal prompt testuale fa uso della seguente pipeline:

1. Inserimento del prompt testuale da parte dell'utente
2. Calcolo dei box contenenti le istanze dell'oggetto richiesto dall'utente
3. Esecuzione di SAM usando la Segmentazione basata su Multiple Box.

5.3.3 Esecuzione dell'algoritmo di generazione di maschere da prompt testuale

Per eseguire l'algoritmo, basta posizionarsi all'interno della cartella fornita in allegato al presente progetto, aprire una finestra CLI ed eseguire il seguente comando:

```
python textSam.py <image_path> <mask_mode>
```

Il significato e i possibili valori dei parametri sono i seguenti:

| Parametro | Significato | Possibili valori |
|------------|---|---|
| image_path | Path dell'immagine | - |
| mask_mode | Modalità di visualizzazione delle maschere generate | s : le maschere vengono visualizzate singolarmente a : le maschere vengono visualizzate tutte in una volta |



Figura 5.6: Immagine di input

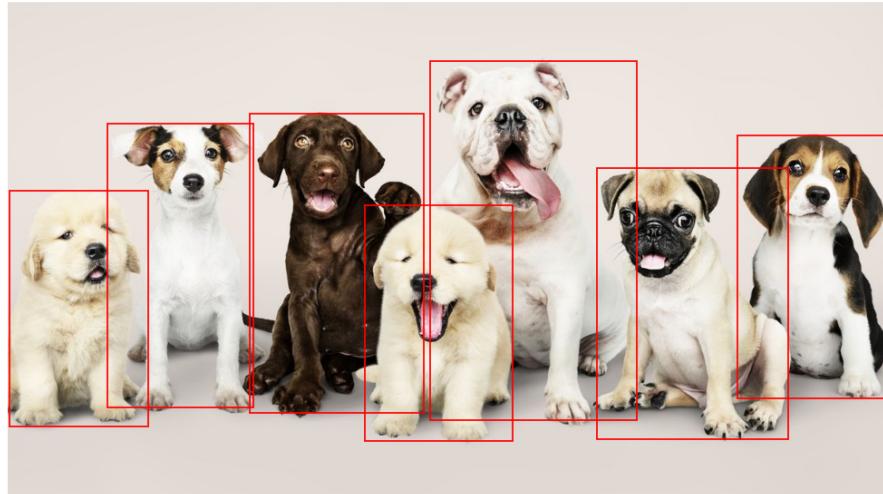


Figura 5.7: Risultato dell'esecuzione dell'Object Detection tramite OWL-ViT, usando il prompt "*dog*"

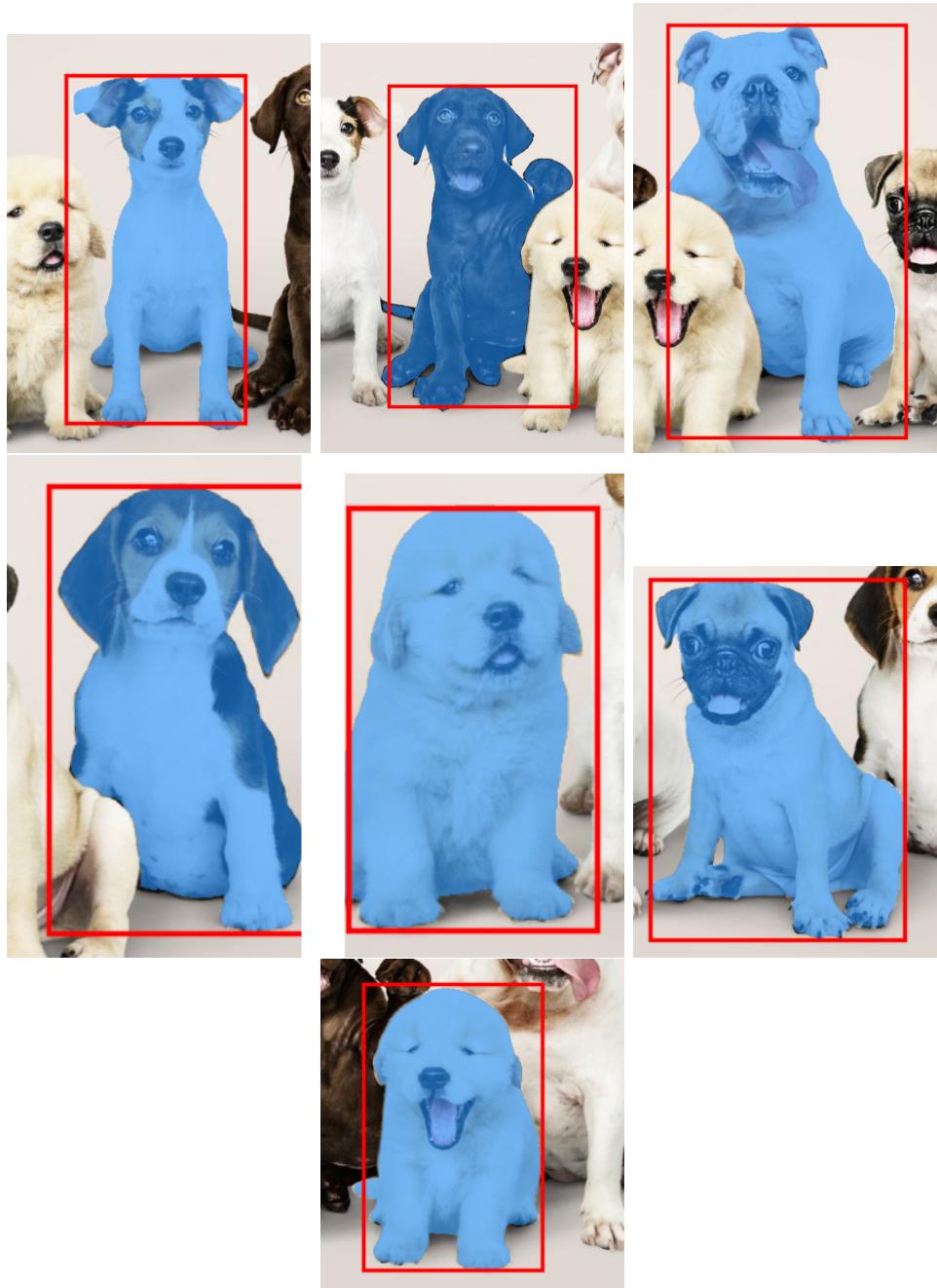


Figura 5.8: Risultati della segmentazione sulla stessa immagine di input (fig.5.6) usando l'algoritmo proposto, in modalità **”single_mask”**.

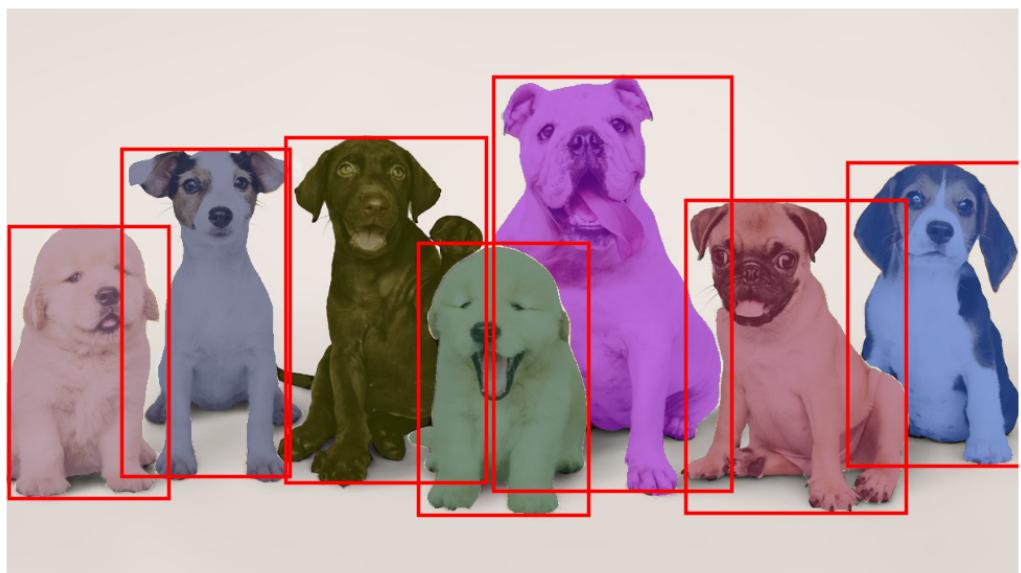


Figura 5.9: Risultati della segmentazione sulla stessa immagine di input (fig.5.6) usando l'algoritmo proposto, in modalità ”**multiple_mask**”.

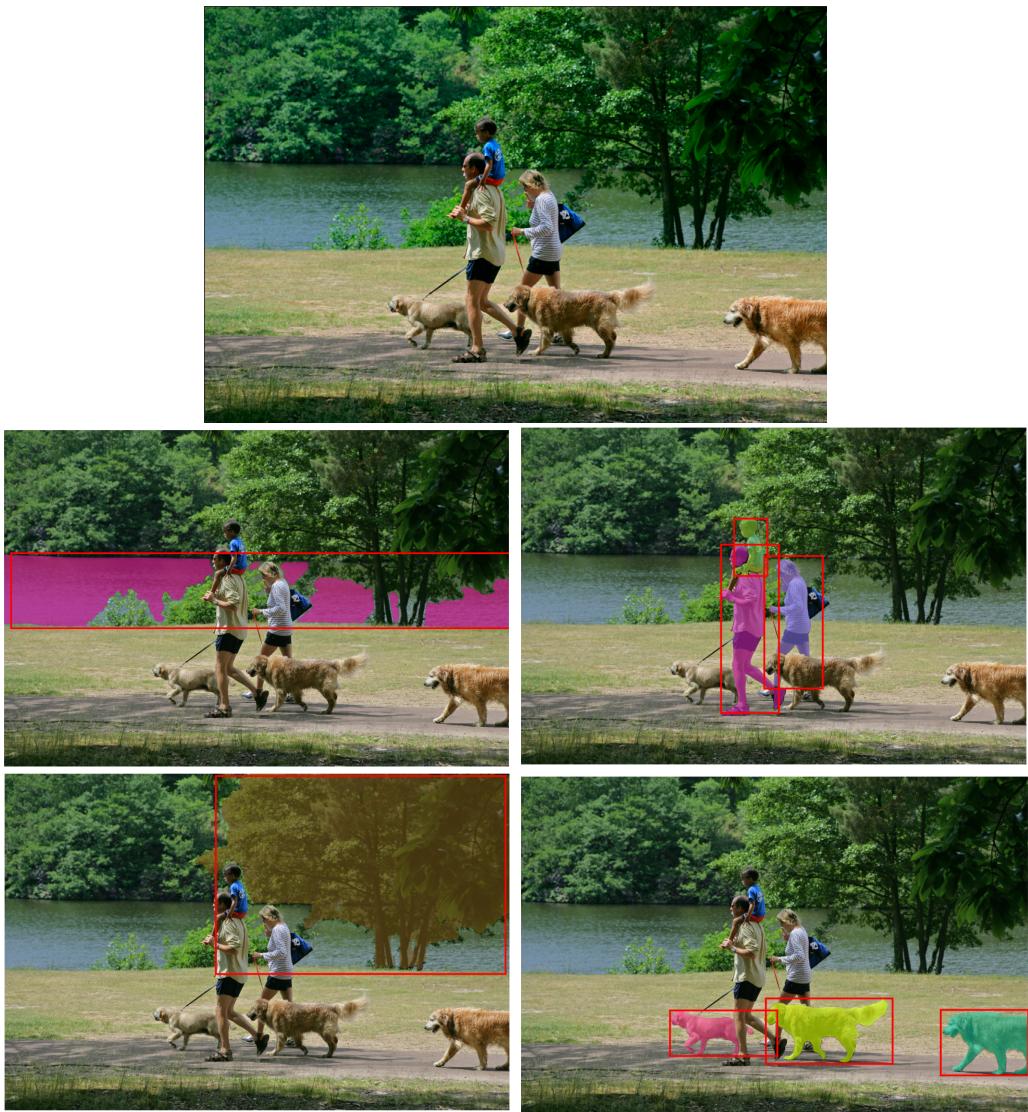


Figura 5.10: Segmentazione della stessa immagine con diversi prompt
a) Immagine di input b) prompt "water" c) prompt "person" d) prompt "tree"
e) prompt "dog"

Conclusione

In questo progetto sono state esplorate diverse tecniche di segmentazione, dalle classiche metodologie edge-based, thresholding-based e region-based, fino all'approccio basato su clustering, con particolare attenzione all'algoritmo K-means. Abbiamo analizzato i vantaggi e gli svantaggi di queste metodologie e discusso poi il ruolo del deep learning nel clustering, e come esso incrementi di molto le prestazioni degli algoritmi di clustering.

Successivamente, ci si è concentrati sul modello DeepCluster, che utilizza un approccio di apprendimento non supervisionato tramite clustering per effettuare il learning delle feature di un'immagine in maniera efficiente.

Successivamente, è stato presentato il modello SAM (Segment Anything Model), progettato per affrontare il task della segmentazione di immagini di ampio genere. E' stata discussa l'architettura del modello, il dataset utilizzato e le funzionalità implementate, inclusa la generazione di maschere a partire da diversi tipi di prompt.

Infine, si è tentato di implementare un algoritmo che permetesse di utilizzare SAM attraverso prompt testuali, una funzionalità non ancora resa disponibile dallo sviluppatore di SAM.

Ciò è stato possibile grazie all'utilizzo del modello di Object Detection "OWL-ViT": l'algoritmo infatti prevede una prima fase di object detection sulla base di un prompt testuale dato in input dall'utente, la quale restituisce dei bounding box degli oggetti presenti nell'immagine che corrispondono al prompt ricevuto in input; successivamente, si esegue la segmentazione usando i box prodotti da OWL-ViT come prompt/input di SAM.

Appendice: repository con codice

I codici prodotti per il presente progetto possono essere trovati all'interno del seguente repository GitHub:

<https://github.com/SalvoSamba01/textual-image-segmentation>

I file presenti nel repository sono i seguenti:

- ***SAM.py***: implementazione di SAM con prompt di diverso tipo (single point, multiple point, box)
- ***textSam.py***: implementazione di SAM con prompt testuali
- ***requirements.txt***: file contenente le librerie necessarie all'esecuzione degli script proposti
- ***testImages***: cartella contenente le immagini usate per testare le implementazioni proposte

Bibliografia

- [1] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features, 2019.
- [2] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.
- [3] Yazhou Ren, Jingyu Pu, Zhimeng Yang, Jie Xu, Guofeng Li, Xiaorong Pu, Philip S. Yu, and Lifang He. Deep clustering: A comprehensive survey, 2022.
- [4] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao li, Jiajun Bu, Jia Wu, Xin Wang, Wenwu Zhu, and Martin Ester. A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions, 2022.