



UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato d'Esame in Information Retrieval Systems

*Dense Vector Search Application:
Query By Example*

Anno Accademico 2022/2023

Relatore
Prof. Antonio Maria Rinaldi

Candidato
Mennillo Valerio
matr. M63001277

Indice

1	Introduzione	1
2	Tecnologie	3
2.1	Apache Solr 9.2.1	3
2.2	TensorFlow 1.15.0	4
2.3	Keras 2.10.0	4
2.4	Apache Maven 3.9.2	5
2.5	GitHub	5
3	Preparazione della collezione documentale	7
3.1	Dataset	7
3.2	Embedding	8
3.3	VGG-16	10
3.4	Transfer Learning	11
3.5	Feature Extraction	14
4	Information Retrieval System: Apache Solr	16
4.1	Apache Solr	16
4.2	Dense Retrieval	17
4.2.1	Small World Graph	18
4.2.2	Hierchical Navigable Small World Graph	20
4.2.3	Costruzione del grafo HNSW	22
4.2.4	Ricerca nel grafo HNSW	26
4.2.5	Dimensione dell'indice	28
4.3	Dense Vector Search	29
4.4	Configurazione Solr	30
4.5	Apache Solr plugin	33
4.6	Caricamento del plugin in Solr	39
4.7	Valutazione del retrieval	40
4.8	Visualizzazione dei risultati	40

5 Esempi di utilizzo	42
5.1 Query by Example (Monarch)	42
5.2 Query by Example (Painted Lady)	43
5.3 Standard text query	44
6 Conclusione	45

Capitolo 1

Introduzione

Query by Example (QBE) è un approccio di ricerca che consente agli utenti di trovare informazioni o oggetti simili a un esempio o un campione di riferimento fornito. Invece di formulare una query utilizzando parole chiave o parametri specifici, l'utente presenta un esempio o un'istanza di ciò che sta cercando, e il sistema cerca elementi simili in base a quella rappresentazione.

La QBE è importante oggi per diversi motivi:

1. **Semplificazione della ricerca:** La QBE semplifica il processo di ricerca per gli utenti che potrebbero avere difficoltà nel formulare query complesse o che potrebbero non avere una chiara idea di come esprimere ciò che cercano. Con la QBE, possono fornire un esempio concreto o un'immagine e ottenere risultati pertinenti.
2. **Ricerca basata sul contenuto:** La QBE consente di effettuare ricerche basate sul contenuto o sulle caratteristiche degli oggetti, come l'aspetto visivo, il testo associato o altre caratteristiche specifiche.
3. **Personalizzazione e raccomandazioni:** La QBE può essere utilizzata per fornire raccomandazioni personalizzate in base agli

oggetti o alle preferenze degli utenti. Ad esempio, utilizzando un’immagine di un prodotto che piace, un sistema può suggerire prodotti simili che potrebbero interessare all’utente.

4. **Ricerca basata sull’esperienza:** La QBE consente di effettuare ricerche basate sull’esperienza, consentendo agli utenti di trovare elementi simili a quelli che hanno già identificato o utilizzato in precedenza. Questo può essere utile per la ricerca di documenti simili, brani musicali correlati o altri oggetti che si desidera trovare in base a un esempio di riferimento.

Obiettivo dell’elaborato è costruire un sistema di Query by Example, sfruttando una rappresentazione densa dei contenuti multimediali.

Capitolo 2

Tecnologie

Di seguito verranno elencate le principali tecnologie utilizzate per il progetto.

2.1 Apache Solr 9.2.1

Solr è un motore di ricerca open source basato su Apache Lucene. È progettato per l'indicizzazione e il recupero rapido di grandi volumi di dati. Solr offre funzionalità avanzate per la ricerca, inclusa la ricerca full-text, il filtraggio dei risultati e la classificazione dei documenti. Può essere utilizzato come sistema di information retrieval, poiché consente agli utenti di cercare e recuperare informazioni pertinenti da un insieme di dati strutturati o non strutturati.



Figura 2.1: Apache Solr Logo

2.2 TensorFlow 1.15.0

TensorFlow è una libreria open source per l'apprendimento automatico e l'intelligenza artificiale. È ampiamente utilizzato per creare reti neurali e modelli di machine learning. TensorFlow offre un'ampia gamma di strumenti e funzionalità per la costruzione, l'addestramento e l'implementazione di modelli di apprendimento automatico. La sua caratteristica principale è la gestione efficiente dei calcoli su tensori multidimensionali, che consente di eseguire operazioni complesse su grandi set di dati. TensorFlow è estremamente popolare e supportato da una vasta comunità di sviluppatori, rendendolo uno degli strumenti più utilizzati nell'ambito dell'apprendimento automatico.



Figura 2.2: TensorFlow Logo

2.3 Keras 2.10.0

Keras è una libreria open source di alto livello per il deep learning, scritta in Python. Essa fornisce un'interfaccia semplice e intuitiva per la creazione, l'addestramento e la valutazione dei modelli di deep learning. Keras è progettato per essere modulare, flessibile e adatto sia ai principianti che agli esperti. È in grado di eseguire su diversi backend di calcolo, tra cui TensorFlow, CNTK e Theano. Keras semplifica lo sviluppo di reti neurali profonde, permettendo agli sviluppatori di concentrarsi sulla progettazione del modello e sulle stra-

tegie di addestramento, senza dover affrontare dettagli complessi di implementazione.



Figura 2.3: Keras Logo

2.4 Apache Maven 3.9.2

Apache Maven è uno strumento di gestione dei progetti software ampiamente utilizzato per la compilazione, il testing, il packaging e la distribuzione. È basato su un modello di configurazione dichiarativo, definito nel file pom.xml, che permette di specificare le dipendenze, le fasi di build e gli obiettivi da raggiungere. Maven automatizza le operazioni comuni di sviluppo, semplificando il processo di gestione dei progetti e favorisce l'interoperabilità tra i vari ambienti di sviluppo.



Figura 2.4: Apache Maven Logo

2.5 GitHub

GitHub è una piattaforma web-based che offre servizi di hosting per lo sviluppo collaborativo del software. È ampiamente utilizzato dai team di sviluppatori per gestire e condividere il codice sorgente dei progetti software. GitHub fornisce funzionalità di controllo versione distribuito utilizzando Git, consentendo agli sviluppatori di tenere traccia delle modifiche al codice nel tempo. Oltre al controllo versione, GitHub

offre strumenti per la gestione delle richieste di pull, il tracciamento degli errori e la collaborazione tra i membri del team. È diventato uno degli hub principali per la condivisione del codice open source e facilita la collaborazione e la condivisione del lavoro tra gli sviluppatori di tutto il mondo.



Figura 2.5: GitHub Logo

Capitolo 3

Preparazione della collezione documentale

3.1 Dataset

Il Leeds Butterfly Dataset [8] è una collezione di oltre 800 immagini di farfalle provenienti da diverse specie. Creato da Josiah Wang, il dataset è ampiamente utilizzato nel campo della visione artificiale e dell'elaborazione delle immagini. Le immagini, con una risoluzione variabile, coprono dieci specie di farfalle provenienti da diverse parti del mondo. Ogni immagine è associata a un'etichetta di classe che indica la specie di farfalla rappresentata. Il dataset è organizzato in cartelle corrispondenti alle specie di farfalle, consentendo un'organizzazione chiara e facile accessibilità alle immagini. Il Leeds Butterfly Dataset è disponibile pubblicamente e gratuitamente, permettendo a ricercatori e sviluppatori di utilizzarlo per scopi di ricerca e sviluppo nel campo del riconoscimento delle farfalle e dell'elaborazione delle immagini.



Figura 3.1: Esempi del dataset[8]

Il dataset è stato diviso in tre parti: train set, validation set e test set. Questa suddivisione viene eseguita utilizzando le proporzioni specificate (70% per il train set, 20% per il validation set e il restante 10% per il test set).

3.2 Embedding

Un vettore di feature è essenzialmente una rappresentazione numerica delle caratteristiche rilevanti di un'immagine, che può essere utilizzata per scopi come la classificazione, il riconoscimento o il clustering. Tali vettori di feature possono anche essere confrontati tra loro attraverso una metrica di similarità.

Per l'estrazione di vettori di feature, le reti neurali sono diventate uno strumento molto potente ed efficace. In particolare, le reti neurali convoluzionali (CNN) sono ampiamente utilizzate per la classificazione delle immagini. Le CNN sono progettate per apprendere gerarchie di caratteristiche dalle immagini attraverso una serie di strati convoluzionali e di pooling.

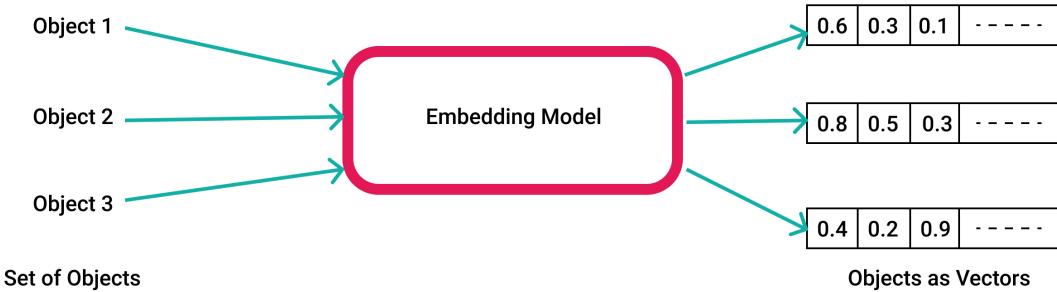


Figura 3.2: Embedding [6]

L'estrazione di vettori di feature da una rete neurale avviene tipicamente sfruttando le informazioni contenute negli strati intermedi della rete, noti anche come strati nascosti. Questi strati intermedi rappresentano livelli gerarchici di astrazione delle caratteristiche dell'immagine. Man mano che i dati attraversano la rete, gli strati convoluzionali estraggono informazioni sempre più complesse e significative.

Per ottenere un vettore di feature, possiamo prendere l'output di uno di questi strati intermedi, e utilizzarlo come rappresentazione numerica dell'immagine. Questo vettore di feature cattura le caratteristiche salienti dell'immagine in uno spazio di dimensioni ridotte e fisse. Ad esempio, se utilizziamo una CNN pre-addestrata come VGG o ResNet, l'output di uno degli strati convoluzionali può essere considerato come un vettore di feature.

L'estrazione di vettori di feature con una rete neurale consente di ottenere una rappresentazione compatta e informativa delle immagini, che può essere utilizzata per l'elaborazione successiva come la classificazione delle immagini. Questi vettori di feature possono essere passati a un classificatore tradizionale o utilizzati come input per un'altra rete neurale, ad esempio un classificatore softmax per la classificazione delle immagini.

3.3 VGG-16

VGG16 è una rete neurale convoluzionale (CNN) profonda e molto popolare, sviluppata da Karen Simonyan e Andrew Zisserman presso l’Università di Oxford. "VGG" sta per Visual Geometry Group, il gruppo di ricerca che ha sviluppato questa architettura. VGG16 è stato originariamente progettato per la classificazione delle immagini nel contesto del concorso ImageNet Large Scale Visual Recognition Challenge (ILSVRC) del 2014.

L’architettura VGG16 è caratterizzata da 16 strati, tra cui 13 strati convoluzionali e 3 strati completamente connessi.

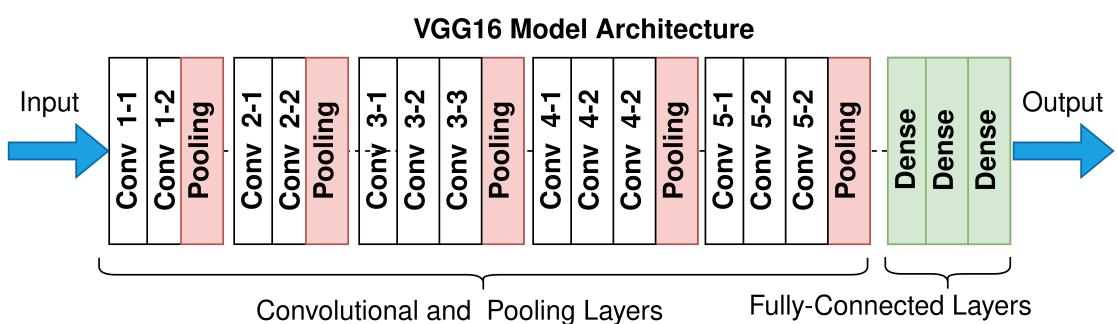


Figura 3.3: VGG16 Architecture [1]

E’ possibile rimuovere il top layer (ultimo strato completamente connesso) e sostituirlo con ulteriori layer personalizzati per adattare la rete al proprio problema specifico. Ciò può essere fatto per diverse ragioni:

1. Adattamento al numero di classi: L’architettura VGG16 originale è stata addestrata per classificare immagini in 1000 classi diverse. Se il problema richiede una classificazione in un numero diverso di classi (come nel nostro caso), si può rimuovere il top layer e sostituirlo con un nuovo strato completamente connesso che produca un numero di output corrispondente alle classi.

2. Trasferimento di conoscenza (Transfer Learning): VGG16 è stato pre-addestrato su un vasto set di dati di immagini e ha dimostrato ottime capacità di estrazione delle caratteristiche. Rimuovendo il top layer e aggiungendo nuovi strati personalizzati, si può sfruttare l'apprendimento precedente della rete su dati generici per migliorare le prestazioni del modello su un task specifico con un set di dati limitato. Congelando la parte di rete di VGG16, si va praticamente ad allenare solo la nuova parte di rete aggiunta.

Prima di dare un'immagine a VGG16, è necessario normalizzare i valori dei pixel, ridimensionare l'immagine a una dimensione di 224x224 pixel e scambiare i canali di colore da RGB a BGR.

3.4 Transfer Learning

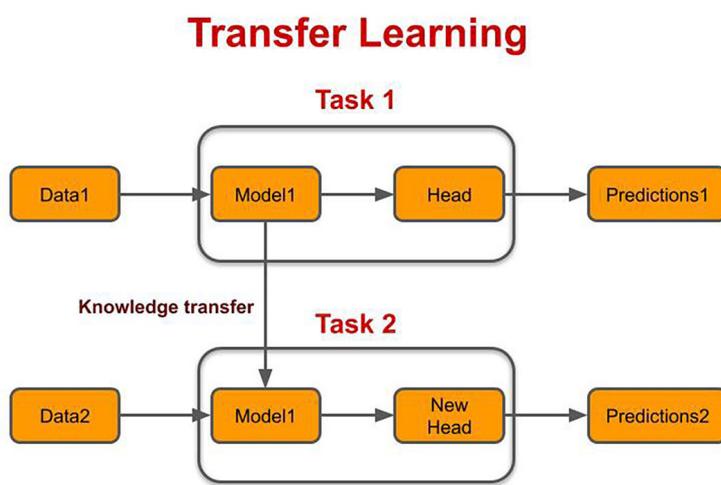


Figura 3.4: Transfer Learning process [7]

L'azione di congelare i layer di VGG16 e addestrare solo i propri layer personalizzati è un esempio di transfer learning, un approccio comune

CAPITOLO 3. PREPARAZIONE DELLA COLLEZIONE DOCUMENTALE

ed efficace per adattare modelli pre-addestrati a compiti specifici o set di dati più piccoli.

Nel transfer learning, si parte da un modello pre-addestrato, come VGG16, su un grande set di dati e lo si adatta al proprio compito specifico o set di dati più piccolo.

Nel caso in esame, il modello VGG16 è stato utilizzato come base e sono stati aggiunti ulteriori layer personalizzati. Congelando i layer di VGG16, si è deciso che non vengano addestrati durante l'addestramento del modello completo, mantenendo invariati i pesi e le caratteristiche estratte da VGG16 durante l'addestramento dei nuovi layer.

Successivamente, sono stati allenati solo i layer personalizzati, sfruttando le conoscenze acquisite da VGG16 sulle caratteristiche generali delle immagini per adattarle al compito specifico di classificazione. I layer personalizzati impareranno a riconoscere ed estrarre caratteristiche specifiche per il compito utilizzando un set di dati più piccolo, senza la necessità di addestrare l'intero modello VGG16 da zero.

Questo approccio di transfer learning è vantaggioso quando si ha a disposizione un set di dati limitato, poiché consente di ottenere buone prestazioni anche con un addestramento limitato, riducendo il rischio di overfitting e il tempo complessivo di addestramento.

Il modello ad alto livello che è stato definito utilizza l'architettura della rete neurale convoluzionale VGG16 come base. Il modello base VGG16 è stato pre-addestrato su un ampio set di dati di immagini chiamato ImageNet, apprendendo filtri e caratteristiche utili per vari compiti di visione artificiale.

Successivamente, sono stati aggiunti ulteriori layer personalizzati al modello base per adattarlo al compito specifico di classificazione. Un layer di flatten è stato utilizzato per convertire l'output bidimensionale del modello base in un vettore unidimensionale, permettendo il collegamento con i successivi layer fully-connected.

CAPITOLO 3. PREPARAZIONE DELLA COLLEZIONE DOCUMENTALE

Dopo il layer di flatten, è stato aggiunto un layer fully-connected con una funzione di attivazione LeakyReLU per apprendere rappresentazioni delle caratteristiche specifiche al compito di classificazione. Infine, è stato aggiunto un ultimo layer fully-connected con una funzione di attivazione softmax, producendo l'output finale del modello come distribuzione di probabilità tra le dieci classi di output.

Tabella 3.1: Resoconto del modello

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten (Flatten)	(None, 25088)	0
dense	(None, 64)	1,605,696
dense_1	(None, 10)	650
Total params:		16,321,034
Trainable params:		1,606,346
Non-trainable params:		14,714,688

Com'è possibile vedere dal resoconto, i parametri di vgg16 sono stati congelati, bensì etichettati come *Non-trainable params*.

Epochs	Loss	Accuracy	Val Loss	Val Accuracy
1	4.1700	0.5218	0.8735	0.7937
2	0.3289	0.8909	0.3729	0.9250
3	0.0643	0.9782	0.3956	0.8875
4	0.0173	0.9982	0.1889	0.9500
5	0.0037	1.0000	0.1643	0.9688
6	0.0012	1.0000	0.1675	0.9750
7	0.0009	1.0000	0.1430	0.9812
8	0.0009	1.0000	0.1577	0.9750
9	0.0008	1.0000	0.1383	0.9812
10	0.0008	1.0000	0.1528	0.9750

Tabella 3.2: Metriche di training e validazione

Si è raggiunta una Accuracy di circa 96% sul test set.

3.5 Feature Extraction

Dopo aver utilizzato VGG16 per la classificazione delle immagini, la rete viene ora scomposta per estrarre i vettori di feature anziché per la classificazione diretta. Questo approccio, noto come "feature extraction", sfrutta la capacità della rete di apprendere rappresentazioni ricche di informazioni a vari livelli di astrazione.

Nello specifico, la rete VGG16 viene "scoperchiata" rimuovendo il layer di output finale, che è responsabile della classificazione. Ciò consente di accedere ai livelli interni della rete, che rappresentano feature di alto livello, come forme, texture e altre caratteristiche semantiche.

Per estrarre i vettori di feature, le immagini vengono passate attraverso la rete fino a un certo punto specificato, ad esempio fino a un layer intermedio. I valori di output di questo layer intermedio rappresentano i vettori di feature corrispondenti alle immagini di input.

Questo approccio consente di ottenere rappresentazioni dense e significative delle immagini, che possono essere utilizzate in ulteriori analisi o compiti, come il clustering, la ricerca basata sulla similarità o l'estrazione di informazioni rilevanti.

L'utilizzo di vettori di feature estratti da VGG16 può essere utile per quantificare la similarità tra vettori di feature e quindi i corrispondenti contenuti multimediali.

Il modello "scoperchiato" restituirà un vettore di feature denso di dimensione fissa 64. Tale modello viene salvato in un formato standard SavedModel. Un SavedModel è un formato di salvataggio standard per modelli TensorFlow. È una directory che contiene l'intera rappresentazione di un modello TensorFlow, compresi i pesi, l'architettura del grafo computazionale e le informazioni necessarie per l'inferenza. Il formato SavedModel è progettato per essere flessibile, portatile e compatibile tra le diverse versioni di TensorFlow.

Listing 3.1: Schema dei dati

```
{  
    "image_id": String,  
    "image_path": String,  
    "scientific_name": String,  
    "common_name": String,  
    "description": String,  
    "feature_vector": Float [64]  
}
```

Capitolo 4

Information Retrieval System: Apache Solr

4.1 Apache Solr

Apache Solr è un sistema di ricerca e indicizzazione open source basato su Apache Lucene. È progettato per fornire funzionalità avanzate di Information Retrieval (IR) per la ricerca e l'indicizzazione di grandi quantità di dati non strutturati, come documenti, testo, immagini e file multimediali.

Solr è un motore di ricerca full-text altamente scalabile e ad alte prestazioni. Utilizza l'indicizzazione invertita, che crea un indice che mappa le parole presenti nei documenti al loro posizionamento all'interno di essi. Questo consente una rapida ricerca e recupero delle informazioni corrispondenti a determinate query.

Apache Solr è un sistema di indicizzazione e ricerca progettato principalmente per l'elaborazione e l'indicizzazione di dati testuali. Ciò significa che Solr non è in grado di salvare direttamente le immagini stesse all'interno del suo indice. Tuttavia, Solr può gestire e indicizzare i vettori di feature che rappresentano le immagini.

Quando si lavora con immagini in Apache Solr, l'approccio co-

mune consiste nel trasformare le immagini in vettori di feature utilizzando tecniche di estrazione delle caratteristiche o di elaborazione delle immagini. Queste tecniche convertono le immagini in una rappresentazione numerica compatta che può essere indicizzata da Solr.

Una volta estratti i vettori di feature, questi possono essere salvati come campi nel documento Solr insieme ai relativi metadati. Ad esempio, è possibile avere un campo "feature_vector" nel documento Solr che rappresenta il vettore di feature dell'immagine. Questo campo può essere indicizzato da Solr per consentire ricerche e recupero basati sulla similarità delle caratteristiche delle immagini.

L'immagine stessa non viene memorizzata nell'indice, bensì vengono indicizzati i vettori di feature. Questo significa che il vettore di feature diventa il principale elemento utilizzato per il calcolo delle metriche di similarità e la ricerca di immagini simili.

È importante notare che l'estrazione dei vettori di feature deve essere gestita attraverso processi esterni o plugin personalizzati. Ciò significa che è necessario utilizzare strumenti o librerie specifiche di elaborazione delle immagini per estrarre i vettori di feature e poi utilizzare l'API di Solr per indicizzare i dati nel campo appropriato. A tale scopo, è stato progettato un plugin di Solr il cui scopo è prendere l'immagine ed estrarne il vettore di feature, per verificare la similarità con i vettori di feature già contenuti in Solr.

4.2 Dense Retrieval

Dato una vettore denso v , l'approccio più semplice per trovare i vettori densi "vicini" sarebbe calcolare la distanza (euclidea, prodotto scalare, ecc.) tra v e ogni vettore d nei documenti della collezione documentale e riordinare i risultati in ordine discendente. Questo approccio riscontra una complessità lineare rispetto al numero di documenti quindi

non è utilizzabile per collezioni documentali di dimensione elevata. Si sono quindi esplorati approcci alternativi per efficientare il retrieval.

4.2.1 Small World Graph

Una possibile formalizzazione del problema è quella di creare un grafo $G(V,E)$ dove i vertici sono i documenti, collegati tra loro da link pesati in base ad una metrica di distanza.

È importante notare che i link nel grafo svolgono due scopi distinti:

1. Esiste un sottoinsieme di collegamenti a breve raggio.
2. Un altro sottoinsieme è costituito dai collegamenti a lungo raggio, utilizzati per la scalabilità logaritmica della ricerca greedy. I collegamenti a lungo raggio sono responsabili delle proprietà di navigazione del piccolo mondo del grafo costruito.

Una delle proprietà su cui si basa il grafo a "mondo piccolo" è la proprietà di **six degrees of separation**, che sostiene che due persone qualsiasi nel mondo possono essere collegate attraverso una catena di conoscenze di lunghezza media di sei persone. L'obiettivo di questo grafo è riuscire a raggiungere tutti i nodi nel numero minore di passi possibili.

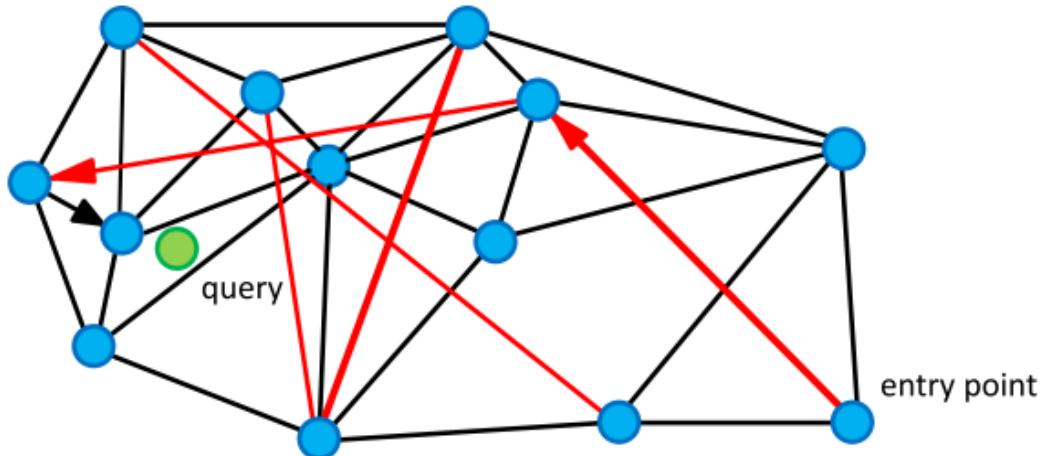


Figura 4.1: Esempio di Small World Graph: I link neri sono i "collegamenti a breve raggio" e i link rossi sono i "collegamenti a lungo raggio". Le frecce mostrano un percorso di esempio dell'algoritmo greedy dall'entry point alla query (mostrata in verde). [4]

Tale grafo viene costruito in maniera incrementale andando a posizionare spazialmente i documenti rispetto al loro vettore di feature nello spazio vettoriale, collegando il nuovo documento solo ai documenti più vicini. Per ricercare i vettori più simili a v , partendo da un entry point predefinito, è possibile utilizzare un approccio greedy il cui scopo è quello di valutare tutti i nodi vicini (friends) e misurare la similarità rispetto a v . Se si trova un nodo con distanza minore, si salva e il processo si ripete iterativamente. La condizione di stop dell'algoritmo è quando i vicini del nodo corrente non sono più "vicini" al vettore v .

```

Greedy_Search(q: object, ventry_point: object)
1   vcurr←ventry_point;
2   δmin←δ(q, vcurr); vnext←NIL;
3   foreach vfriend∈vcurr.getFriends() do
4     δfr←d(query, vfriend)
5     if δfr<δmin then
6       δmin←δfr;
7       vnext←vfriend;
8   if vnext=NIL then return vcurr;
9   else return Greedy_Search(q, vnext);

```

Figura 4.2: Basic greedy search algorithm [4]

Nonostante sia più efficiente di una ricerca esaustiva, la ricerca greedy può portare ad un minimo locale non globale. Per cercare di massimizzare la probabilità di trovare minimo globale, ci sono euristiche e strategie come Simulated Annealing, algoritmi genetici o ottimizzazioni con più entry point. Inoltre è possibile aumentare il "grado" (ovvero il numero di collegamenti) dei nodi e utilizzare tali nodi come entry point a discapito della complessità computazionale.

4.2.2 Hierarchical Navigable Small World Graph

A differenza dello Small World Graph introdotto precedentemente, questo tipo di grafo è molto più efficiente (a discapito dello storage).

Il grafo HNSW è un grafo multilivello che si basa sullo stesso principio della Ricerca dicotomica e della Probability Skip List.

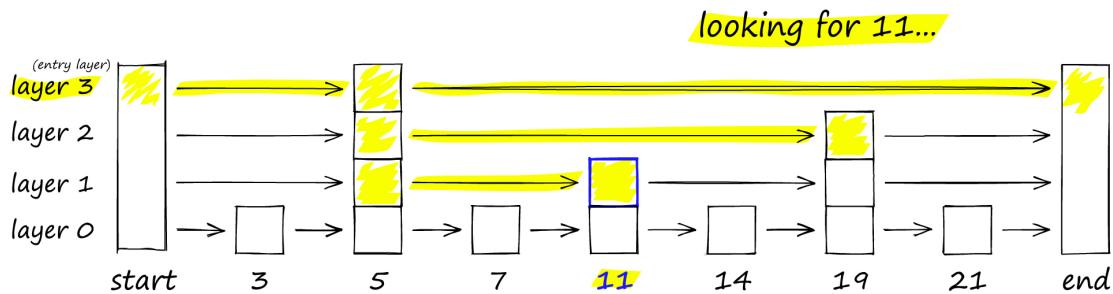


Figura 4.3: Esempio di ricerca del valore 11 in una Probability Skip List [5]

Per cercare in una Probability Skip List, iniziamo dal livello più alto con i "salti" più lunghi. Se scopriamo che il valore del nodo corrente è maggiore del valore che stiamo cercando, sappiamo di aver superato il nostro obiettivo, quindi ci spostiamo al nodo precedente ma del livello più basso di uno.

L'idea dell'algoritmo usato in HNSW è quella di separare i collegamenti in diversi layer in base alla loro scala di lunghezza per poi effettuare la ricerca in un grafo multilayer.

In questo modo, possiamo valutare solo una porzione fissa necessaria delle connessioni per ciascun elemento, indipendentemente dalla dimensione della rete, consentendo così una scalabilità logaritmica.

Il numero massimo di connessioni per elemento in tutti i layer può essere mantenuto costante, consentendo quindi una complessità logaritmica per l'instradamento in una rete navigabile del piccolo mondo. Questo approccio permette di evitare minimi locali non globali e di ottenere una navigazione efficiente nel grafo.

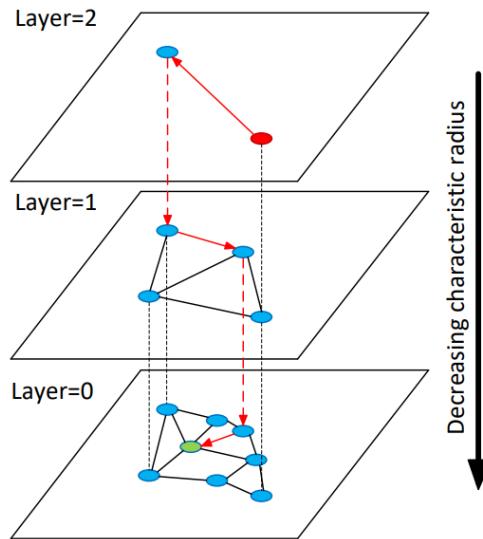


Figura 4.4: Struttura grafo multilivello HNSW [3]

Aggiungendo la gerarchia a NSW si ottiene un grafo in cui i collegamenti sono separati in diversi livelli. Nel livello superiore abbiamo i collegamenti più lunghi, mentre nel livello inferiore abbiamo quelli più corti. Si può navigare questo grafo multilivello "orizzontalmente" sullo stesso livello e "verticalmente" per scendere di livello. La fase di "zoom-out" in HNSW prevede il passaggio attraverso i vertici a basso grado (pochi collegamenti), mentre la fase di "zoom-in" coinvolge il passaggio attraverso i vertici ad alto grado (tanti collegamenti) per restringere la ricerca ai punti più vicini e rilevanti.

4.2.3 Costruzione del grafo HNSW

L'inserimento di un nuovo nodo nel grafo consiste in due fasi:

1. La probabilità di inserimento di un vettore in un determinato livello è data da una funzione di probabilità normalizzata

$$\lfloor -\ln(unif(0..1)) \cdot m_L \rfloor$$

dove $m_L = 0$ significa che i vettori vengono inseriti solo al livello 0.

La costruzione del grafo inizia al livello superiore. Dopo essere entrato nel grafo, l'algoritmo attraversa in maniera greedy gli archi, cercando i vicini più vicini al nostro vettore inserito q.

Dopo aver trovato il minimo locale, si passa al livello successivo (come avviene durante la ricerca). Questo processo viene ripetuto fino a raggiungere il livello di inserimento scelto. Qui inizia la fase due della costruzione.

2. Nella fase due, questi vicini più vicini sono candidati per i collegamenti con il nuovo elemento inserito q e come punti di ingresso al livello successivo. Sono aggiunti M vicini come collegamenti da questi candidati, cioè i vettori più vicini.

Ci sono due parametri che vengono presi in considerazione durante l'aggiunta dei collegamenti. M_{max} , che definisce il numero massimo di collegamenti che un vertice può avere, e M_{max_0} , che definisce lo stesso ma per i vertici nel livello 0. La condizione di arresto per l'inserimento è raggiungere il minimo locale nel livello 0.

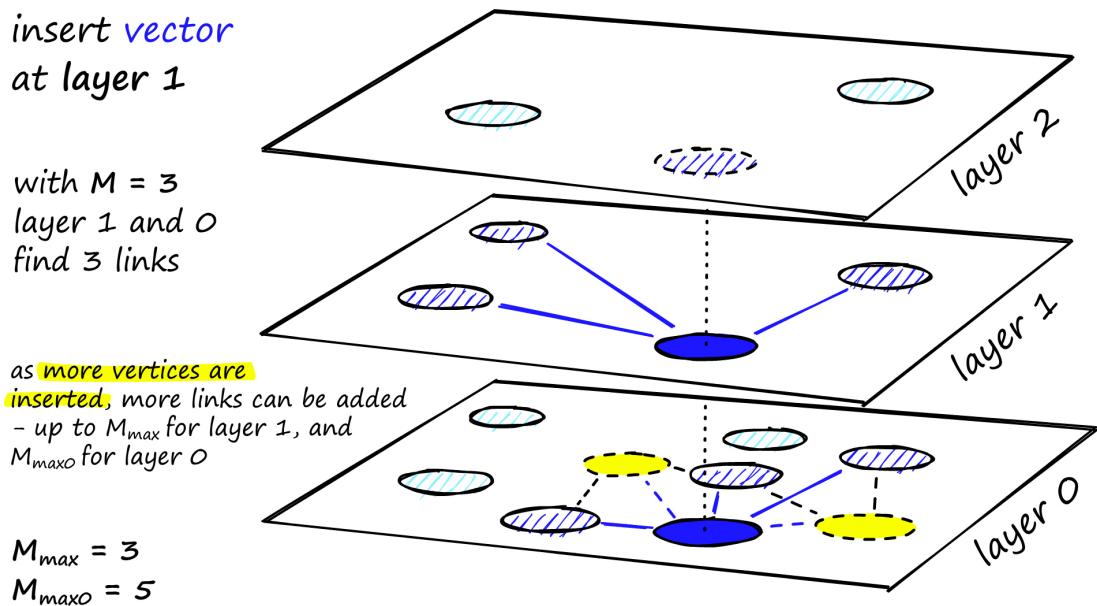


Figura 4.5: Esempio di inserimento [5]

INSERT($hnsn$, q , M , M_{max} , $efConstruction$, ml)

Input: multilayer graph $hnsn$, new element q , number of established connections M , maximum number of connections for each element per layer M_{max} , size of the dynamic candidate list $efConstruction$, normalization factor for level generation ml

Output: update $hnsn$ inserting element q

```

1  $W \leftarrow \emptyset$  // list for the currently found nearest elements
2  $ep \leftarrow$  get enter point for  $hnsn$ 
3  $L \leftarrow$  level of  $ep$  // top layer for  $hnsn$ 
4  $l \leftarrow \lfloor -\ln(unif(0..1)) \cdot ml \rfloor$  // new element's level
5 for  $l_c \leftarrow L \dots l+1$ 
6    $W \leftarrow \text{SEARCH-LAYER}(q, ep, ef=1, l_c)$ 
7    $ep \leftarrow$  get the nearest element from  $W$  to  $q$ 
8 for  $l_c \leftarrow \min(L, l) \dots 0$ 
9    $W \leftarrow \text{SEARCH-LAYER}(q, ep, efConstruction, l_c)$ 
10   $neighbors \leftarrow \text{SELECT-NEIGHBORS}(q, W, M, l_c)$  // alg. 3 or alg. 4
11  add bidirectionall connections from  $neighbors$  to  $q$  at layer  $l_c$ 
12 for each  $e \in neighbors$  // shrink connections if needed
13    $eConn \leftarrow \text{neighbourhood}(e)$  at layer  $l_c$ 
14   if  $|eConn| > M_{max}$  // shrink connections of  $e$ 
      // if  $l_c = 0$  then  $M_{max} = M_{max0}$ 
15    $eNewConn \leftarrow \text{SELECT-NEIGHBORS}(e, eConn, M_{max}, l_c)$ 
      // alg. 3 or alg. 4
16   set  $\text{neighbourhood}(e)$  at layer  $l_c$  to  $eNewConn$ 
17    $ep \leftarrow W$ 
18 if  $l > L$ 
19   set enter point for  $hnsn$  to  $q$ 

```

Figura 4.6: Pseudocodice di inserimento nel grafo HNSW [3]

Questo algoritmo consente di scartare i candidati per la valutazione che si trovano più lontani dalla query rispetto all'elemento più lontano nella lista, evitando così un ingrossamento delle strutture di ricerca.

La costruzione viene effettuata mediante inserimenti iterativi di

tutti gli elementi, mentre l'inserimento di un elemento è semplicemente una sequenza di ricerche K-ANN a diversi livelli con l'utilizzo successivo di un'euristica (che ha complessità fissa a $efConstruction$ fissato) o alternativamente approccio greedy. Il numero medio di livelli in cui viene aggiunto un elemento è una costante che dipende da m_L :

$$E[l + 1] = E[-\ln(\text{unif}(0, 1)) \cdot m_L] + 1 = m_L + 1$$

La complessità di $O(N \log N)$ deriva dal fatto che durante la costruzione del grafo HNSW, vengono eseguite operazioni di inserimento iterativo per ogni elemento nel set di dati. L'inserimento di ogni elemento comporta ricerche K-ANN a diversi livelli per determinare la posizione ottimale del nuovo elemento nel grafo.

Come vedremo subito dopo, le ricerche K-ANN hanno una complessità logaritmica rispetto al numero di punti di dati nel grafo. Poiché ogni elemento viene inserito una volta, il numero totale di ricerche K-ANN eseguite durante la costruzione è proporzionale al numero di punti di dati nel set di dati, che è N .

4.2.4 Ricerca nel grafo HNSW

La ricerca nel grafo HNSW si svolge in due fasi:

1. Fase di ricerca approssimata (Approximate Search): Durante questa fase, si utilizza un approccio greedy per individuare rapidamente un vicino approssimato alla query. Il parametro ef (expansion factor) viene inizialmente impostato a 1, il che significa che si cerca solo il vicino più vicino. La ricerca procede attraverso il grafo, spostandosi da un livello all'altro in base alle connessioni dei vicini più vicini. L'obiettivo è di trovare un vicino che sia abbastanza simile alla query.

2. Fase di ricerca precisa (Refine Search): Una volta raggiunto un determinato livello, ad esempio l'ultimo, si passa alla fase di ricerca precisa. Durante questa fase, il parametro ef viene aumentato da 1 a un valore maggiore chiamato $efConstruction$. Questo permette di espandere la ricerca per includere un numero maggiore di vicini e migliorare il recupero dei risultati. I vicini più vicini trovati su ciascun livello vengono anche utilizzati come candidati per le connessioni dell'elemento inserito, contribuendo alla costruzione del grafo stesso.

Complessivamente, la ricerca nel grafo HNSW combina un'efficace ricerca approssimata con una fase di ricerca precisa che permette di ottenere risultati più accurati. Questo approccio consente di effettuare ricerche efficienti all'interno del grafo, individuando gli elementi più simili alla query in modo rapido e preciso.

K-NN-SEARCH($hnsn$, q , K , ef)

Input: multilayer graph $hnsn$, query element q , number of nearest neighbors to return K , size of the dynamic candidate list ef

Output: K nearest elements to q

```

1  $W \leftarrow \emptyset$  // set for the current nearest elements
2  $ep \leftarrow$  get enter point for  $hnsn$ 
3  $L \leftarrow$  level of  $ep$  // top layer for  $hnsn$ 
4 for  $l_c \leftarrow L \dots 1$ 
5    $W \leftarrow$  SEARCH-LAYER( $q$ ,  $ep$ ,  $ef=1$ ,  $l_c$ )
6    $ep \leftarrow$  get nearest element from  $W$  to  $q$ 
7    $W \leftarrow$  SEARCH-LAYER( $q$ ,  $ep$ ,  $ef$ ,  $l_c = 0$ )
8 return  $K$  nearest elements from  $W$  to  $q$ 
```

Figura 4.7: Pseudocodice di ricerca nel grafo HNSW [3]

La complessità di ricerca approssimata è dell'ordine di $O(\log N)$, dove N è la dimensione del grafo.

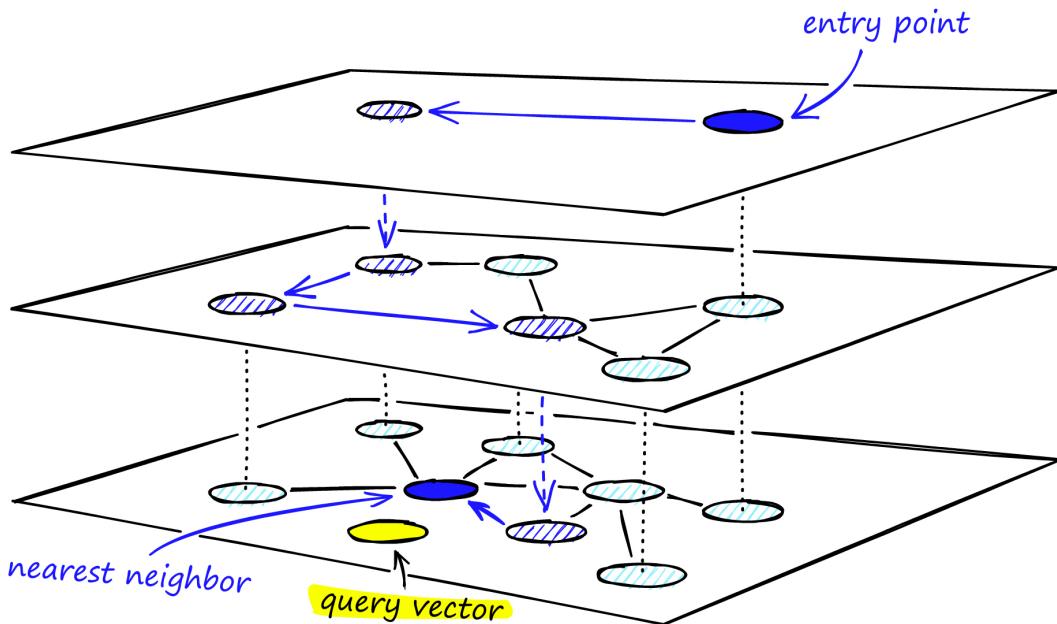


Figura 4.8: Flow di ricerca in grafo HNSW [5]

4.2.5 Dimensione dell'indice

La memoria necessaria per conservare il grafo HNSW è principalmente determinata dalla memorizzazione delle connessioni del grafo. Il numero di connessioni per elemento è M_{max_0} per il livello 0 e M_{max} per tutti gli altri livelli. Pertanto, il consumo medio di memoria per nodo è

$$(M_{max_0} + m_L * M_{max}) * \text{bytes_per_link}$$

Se limitiamo il numero massimo totale di elementi a circa quattro miliardi, possiamo utilizzare interi senza segno a 4 byte per memorizzare le connessioni. I test suggeriscono che i valori M_{max_0} e M_{max} tipici, vicini all'ottimale, si trovino di solito in un intervallo tra 6 e 48. Ciò significa che i requisiti tipici di memoria per l'indice (escludendo la dimensione dei dati) sono di circa 60-450 byte per oggetto. [3]

4.3 Dense Vector Search

HNSW è implementato in Faiss (Facebook AI Similarity Search) e Lucene (dal commit *b36b4af22bb76dc42b466b818b417cbc0deb006*) da cui è possibile usare in Solr tramite Dense Vector Search.

Dense Vector Search di Solr aggiunge il supporto per l'indicizzazione e la ricerca di vettori numerici densi, sfruttando proprio grafi HNSW descritto sopra.

Il deep learning può essere utilizzato per produrre una rappresentazione vettoriale sia della query che dei documenti in un corpus di informazioni.

Apache Solr consente di gestire la ricerca di vettori densi, che sono rappresentazioni numeriche multidimensionali di dati. Questa funzionalità permette di eseguire ricerche basate sulla similarità dei vettori di feature.

Per utilizzare questa funzionalità, è necessario definire un campo nel schema di Solr per rappresentare i vettori densi. Questo permette di indicizzare e recuperare efficientemente i dati.

La dense vector search è un approccio per la ricerca di dati basato sulla similarità dei vettori densi. Nel contesto di Apache Solr, la dense vector search viene gestita utilizzando il modulo di ricerca di vettori densi, integrato in Solr.

Il processo di dense vector search si svolge in diverse fasi:

- **Indicizzazione:** Durante la fase di indicizzazione, i vettori densi vengono associati ai documenti Solr. È necessario definire un campo nel proprio schema Solr per rappresentare i vettori densi e specificare il tipo di campo come "knn_vector", dichiarato in *solr.DenseVectorField*. In questa fase, i vettori densi vengono immagazzinati in un formato ottimizzato per il recupero efficiente.

- **Query:** Per effettuare una ricerca di vettori densi, si creano query specificando i parametri corretti. È possibile utilizzare parametri come "feature_vector" per specificare il vettore di query da confrontare con i vettori densi nell'indice. Inoltre, si possono specificare le metriche di similarità da utilizzare, come la distanza euclidea o la similarità del coseno.
- **Calcolo della similarità:** Durante la ricerca, Solr calcola la similarità tra il vettore di query e i vettori densi presenti nell'indice. La similarità viene calcolata utilizzando le metriche specificate nella query. Solr confronta i vettori e restituisce i risultati ordinati in base alla similarità. In realtà, la similarità non viene calcolata su tutti i vettori di feature presenti nella collezione documentale ma bensì si utilizza un approccio più efficiente, HNSW.
- **Restituzione dei risultati:** Solr restituisce i risultati della ricerca, ordinati in base alla similarità dei vettori densi. È possibile limitare il numero di risultati restituiti e applicare filtri o ulteriori criteri di ordinamento.

4.4 Configurazione Solr

Come prima cosa è stato creato un Core su Apache Solr, al cui interno è stato definito uno schema. Lo schema è specificato nel file managed-schema.xml. Si è quindi adattato lo schema alle esigenze di rappresentazione dei propri dati, specificati precedentemente.

Listing 4.1: Definizione schema dei dati in Solr

```
<fieldType name="knn_vector"
    class="solr.DenseVectorField" vectorDimension="64"
    similarityFunction="cosine"/>
<field name="feature_vector" type="knn_vector"
    indexed="true" stored="true" required="true"/>

<field name="image_id" type="string" indexed="false"
    required="true" />
<field name="image_path" type="string" indexed="false"
    required="true" />
<field name="scientific_name" type="text_general"
    indexed="true"/>
<field name="common_name" type="text_general"
    indexed="true" />
<field name="description" type="text_general"
    indexed="true" />

<uniqueKey>image_id</uniqueKey>
```

Sono stati quindi definiti campi di tipo `text_general` che conterrano le stringhe estratte dal dataset o elaborate a posteriori. Sono stati indicizzati alcuni dei campi dello schema come `scientific_name`, `common_name`, `description` e `feature_vector`.

Nel caso di `text_general`, la pipeline di indicizzazione è completamente integrata in Solr, dalla tokenizzazione all'aggiunta dei termini nell'indice inverso.

L'indice inverso è il cuore del meccanismo di indicizzazione di Solr. È un'astrazione che consente di recuperare rapidamente i documenti che corrispondono a una determinata parola o termine di ricerca. L'indicizzazione per il campo di tipo "knn_vector" denominato "feature_vector" invece è stata spiegata precedentemente ed è anch'essa integrata in Solr, grazie a Lucene.

Si vuole porre attenzione particolare sul campo `feature_vector`. Queste righe del file `managed-schema.xml` di Solr definiscono un nuovo

tipo di campo chiamato "knn_vector" utilizzando la classe *solr.DenseVectorField*

Nella definizione del vettore di classe *solr.DenseVectorField* è possibile specificare:

- **vectorDimension** La dimensione del vettore denso da passare.

Valori accettati: Qualsiasi numero intero ≤ 1024 .

- **similarityFunction (default=euclidean)** Funzione di similarità tra vettori; utilizzata nella ricerca per restituire i K vettori più simili a un vettore di destinazione.

- **knnAlgorithm (default=hnsrw)** Specifica l'algoritmo knn sottostante da utilizzare.

- **hnsrwMaxConnections (default=16)** Controlla quanti dei candidati vicini più vicini sono collegati al nuovo nodo.

Ha lo stesso significato di M_{max} .

- **hnsrwBeamWidth (default=100)** È il numero di candidati vicini più vicini da seguire durante la ricerca del grafo per ogni nuovo nodo inserito.

Ha lo stesso significato di *efConstruction*.

Il campo "knn_vector" viene utilizzato per rappresentare vettori densi di dimensione 64. La funzione di similarità specificata per questo campo è la "cosine", che indica che la similarità tra i vettori sarà calcolata utilizzando la similarità coseno.

Questo campo viene indicizzato ("indexed") e memorizzato ("stored"). La direttiva "required" indica che il valore del campo è obbligatorio per ogni documento. Tale feature_vector conterrà il vettore di feature estratto dalla rete neurale.

Fondamentalmente, non vi è alcuna differenza nell'occupazione dello spazio tra *DenseVectorField* e *FloatPointField* multivalore [2].

Una volta definito lo schema, sono stati aggiunti i veri e propri documenti a Solr e quindi Solr ha indicizzato tali documenti, in base alle direttive specificate nel managed-schema.xml. Tale aggiunta poteva essere fatta in due modi: tramite una richiesta curl a linea di comando oppure tramite interfaccia Solr. Per comodità, è stato fatto tramite interfaccia grafica di Solr, passando come input un file JSON contenente la collezione documentale. Tale file JSON è stato generato dallo script python "feature_vector_extractor.py", presente nella repository. Lo scopo di questo script è costruire i documenti da caricare su Solr. Prende le informazioni delle singole immagini e ne aggiunge altri campi tra cui il vettore di feature calcolato tramite la rete neurale costruita e allenata precedentemente.

4.5 Apache Solr plugin

Come detto precedentemente, lo scopo del progetto è quello di passare a Solr non il vettore di feature ma l'immagine. Sarà quindi compito del plugin caricare l'immagine in Solr, estrarne il vettore di feature e interrogare il core di Solr per ottenere le immagini più simili, ordinate secondo la metrica di similarità scelta (nel caso corrente la similarità coseno).

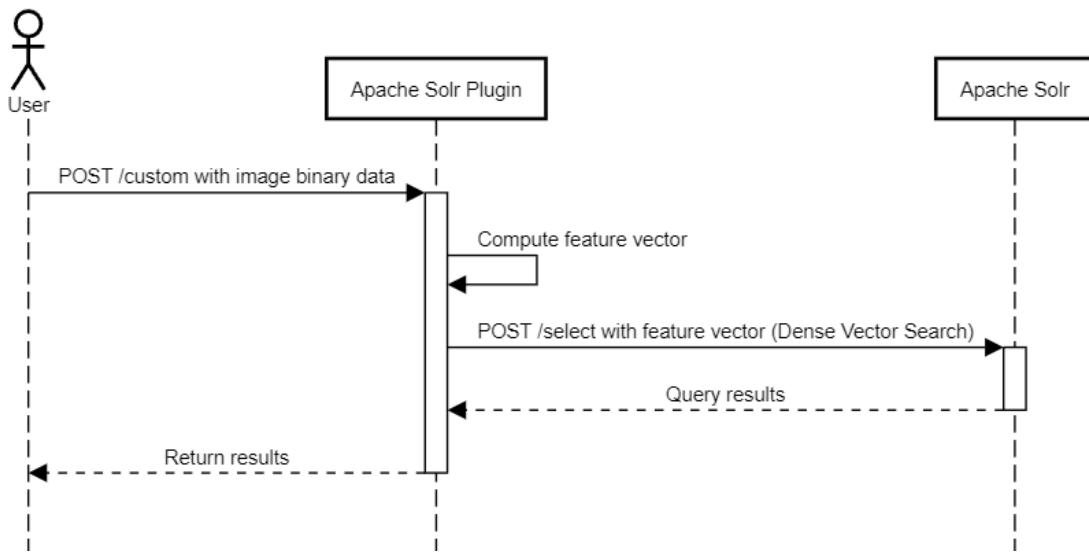


Figura 4.9: Diagramma di sequenza di alto livello

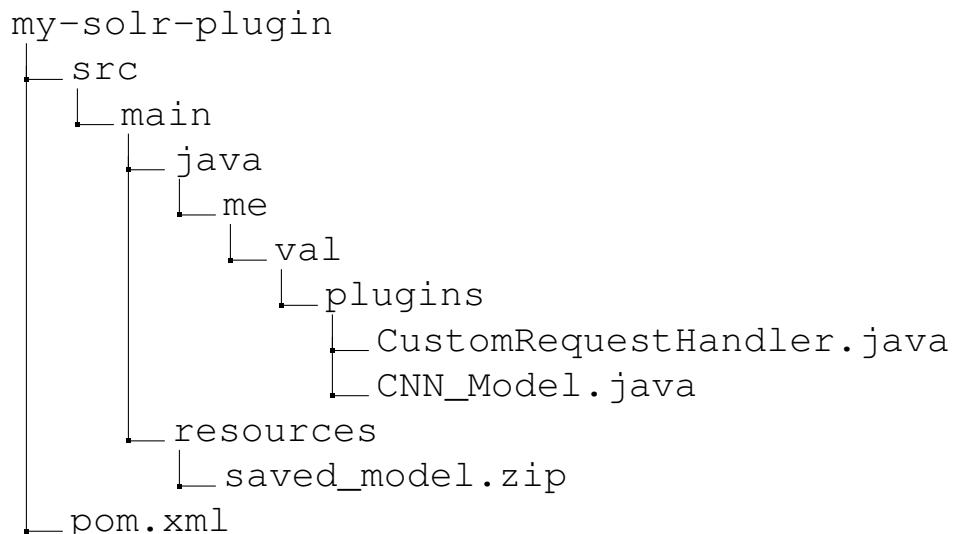


Figura 4.10: Struttura interna file jar del plugin

Le fasi che segue il plugin sono:

- 1. Caricamento dell'immagine tramite un RequestHandler personalizzato:** il plugin deve avere un punto di accesso particolare per accettare non testo ma bensì dati binari, corrispondenti all'immagine. è stato aggiunto al file "solrconfig.xml"

una dichiarazione di un nuovo endpoint il cui comportamento è definito dal percorso dove risiedono i file java del plugin. L'endpoint è stato collegato alla classe *CustomRequestHandler* e tale classe farà 2 cose:

- inizializzerà il modello contenuto in *CNN_Model.java* all'avvio di Solr facendo un override della funzione *init()*. Tale inizializzazione viene fatta una sola volta all'avvio di Solr invece che ogni volta che arriva una immagine, per velocizzare il querytime;
- quando l'end point riceverà una immagine, stimolerà il modello pre-caricato nel jar per estrarre il vettore di feature per poi restituire i documenti relativi alle immagini più simili. Tale comportamento viene specificato facendo override della funzione **handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp)**

Listing 4.2: Definizione RequestHandler personalizzato

```
<requestHandler name="/custom"
                 class="me.val.plugins.CustomRequestHandler" />
```

2. **Preprocessing:** bisogna effettuare del preprocessing sull'immagine, lo stesso applicato durante l'allenamento e specificato dalla documentazione di VGG16. In particolare, bisogna fare un ri-dimensionamento dell'immagine a 224x224x3, normalizzare, sottrarre dei valori noti per ogni canale di colore e poi invertire il canale rosso col canale blu.

Le operazioni effettuate dal preprocessing sono:

- (a) *resizeImage*: se l'immagine è di dimensione diversa da 224x224x3, viene portata a tale risoluzione;

- (b) *convertImageToArray*: per lavorare con l’immagine, è necessario cambiare la sua rappresentazione in un array multidimensionale;
- (c) *normalizeImage*: normalizzazione dell’immagine sottraendo un valore specifico per ogni canale e inversione dei canali rosso e blu;

```
BufferedImage resizedImage = resizeImage(image,  
224, 224);  
  
float[][][][][] inputArray =  
convertImageToArray(resizedImage);  
  
float[][][][][] inputArray2 =  
normalizeImage(inputArray);
```

3. Feature Extraction: Il plugin deve contenere al suo interno uno snapshot della rete neurale, così da fare le predizioni.

La rete neurale, per comodità e requisiti di dimensione, è stata compressa in un file chiamato *saved_model.zip* nel plugin jar e all’avvio del programma, sempre tramite funzione *init()*, verrà estratta nella cartella temporanea di solr, ovvero *./solr/tmp*. Il plugin sa che lì troverà il modello preallenato.

```
String savedModelPath = extractModelFromResource();
```

Quindi, una volta ottenuta l’immagine preprocessata, la si dà alla rete neurale che fornisce in uscita un vettore di 64 elementi. Tale vettore è il vettore di feature, rappresentante le caratteristiche salienti dell’immagine, che dovrà essere utilizzato per interrogare il core di Solr sulla nostra collezione documentale.

La funzione **calculateFeatureVector** prende in input l’immagine e ne estrae il vettore di feature (a valle di preprocessing).

```
public float[] calculateFeatureVector(BufferedImage
    image) {

    String inputTensorName =
        "serving_default_vgg16_input";
    String outputTensorName =
        "StatefulPartitionedCall";

    BufferedImage resizedImage = resizeImage(image,
        224, 224);

    float[][][][][] inputArray =
        convertImageToArray(resizedImage);

    float[][][][][] inputArray2 =
        normalizeImage(inputArray);

    // Create the input tensor
    Tensor<Float> inputTensor =
        Tensors.create(inputArray2);

    // Run inference
    Tensor<Float> outputTensor = session.runner()
        .feed(inputTensorName, inputTensor)
        .fetch(outputTensorName)
        .run()
        .get(0)
        .expect(Float.class);

    FloatBuffer floatBuffer =
        FloatBuffer.allocate(outputTensor.numElements());
    outputTensor.writeTo(floatBuffer);

    // Copy the float values from the buffer to a
    // float array
    float[] features = new
        float[outputTensor.numElements()];
    floatBuffer.flip();
```

```
    floatBuffer.get(features);

    return features;
}
```

4. **Interrogazione:** Avendo precedentemente indicizzato i nostri documenti su vettori densi, si interagisce con l'endpoint di Solr "/select" specificando come query testuale una query contenente il metodo di ricerca KNN, il numero di risultati che si vuole avere (ordinati per similarità), il vettore con cui confrontare il vettore in input e ovviamente il vettore input (vettore di feature).

Quando si desidera trovare le immagini più simili a una data immagine di query, il plugin utilizza il vettore di feature della query per confrontarlo con i vettori di feature dei documenti Solr. Vengono calcolate le metriche di similarità (la similarità del coseno), per identificare i documenti con vettori di feature più simili.

I documenti con i vettori di feature più simili vengono quindi restituiti come risultato della ricerca, consentendo di individuare le immagini che condividono caratteristiche visive simili alla query.

`List<SolrDocument> searchResults = searchDocuments(solrUrl, collectionName, featureVector, topK);` è la funzione che prende in input il vettore di feature e interagisce con l'endpoint */select* specificando come query:

```
solrQuery.setQuery("={!knn f=feature_vector topK=" +
    topK + "}" +
    featureVectorToString(featureVector));
```

Si mostreranno così i primi 10 risultati.

5. **Elaborazione dei risultati:** Una volta ottenuti i risultati, il plugin li gestisce e trasforma in un formato fisso.

Listing 4.3: Schema logico dei risultati del plugin

```
{  
    response_headers: <status, QTime>  
    image_info: <width, height, feature_vector>  
    documents: topK* [<document, similarity>]  
}
```

Viene inoltre ricalcolata la cosine similarity per ognuno dei risultati. Viene restituito oltre all'array dei documenti anche la corrispondente similarità rispetto al vettore di feature in input. Sarà poi compito di chi riceve (soggetto esterno a Solr) questo risultato in formato JSON di visualizzarlo.

4.6 Caricamento del plugin in Solr

Per compilare il progetto in java è stato usato Maven. Attraverso il comando **mvn clean install** si otteneva il file jar da importare in Solr. Per caricare il plugin in Solr è stato necessario aggiungere nella cartella lib il jar compilato del nostro plugin, denominato **CustomHandler-1.jar**. Inoltre, visto che si utilizzano comandi di TensorFlow all'interno del plugin e Solr non contiene tali librerie, sono state aggiunte sempre nella cartella lib anche le librerie necessarie per utilizzare TensorFlow. Le librerie aggiuntive importate sono state:

- libtensorflow-1.15.0-javadoc.jar;
- libtensorflow-1.15.0-sources.jar;
- libtensorflow-1.15.0.jar;
- libtensorflow_jni-1.15.0.jar;

In particolare, per fare ciò, è stata disabilitato il Security Manager che limitava l'accesso da parte di Solr alle librerie nel filesystem. Soluzioni alternative non sono state esplorate per mancanza di tempo.

```
IF NOT DEFINED SOLR_SECURITY_MANAGER_ENABLED (
    set SOLR_SECURITY_MANAGER_ENABLED=false
)
```

4.7 Valutazione del retrieval

Interagendo con l'endpoint */custom* si riceveranno i 10 documenti più simili a quello in input. Ipotizzando un documento restituito "rilevante" se è della stessa classe del documento fornito in input, si misura l'efficacia del sistema tramite una metrica P@10 iterando su tutti i possibili documenti in input e valutando quanti dei 10 documenti più simili appartengono alla stessa classe. Si è raggiunta una precision@10 media del 97.98%.

4.8 Visualizzazione dei risultati

Si è pensato possibile interagire con Solr sottoponendo l'immagine e ottenendo i documenti più simili in due modi:

- Da linea di comando, tramite il modulo *curl*: Essendo i risultati in formato JSON, la linea di comando non è molto adatta alla visualizzazione dei risultati.
- *Da pagina HTML*: è stata costruita una pagina html il cui unico scopo è interagire con l'entry point */custom* di Solr (l'entry point collegato al plugin) mandando il contenuto multimediale in formato binario. Tramite AJAX, la risposta di Solr verrà catturata e visualizzata in maniera adatta. Visto che la richiesta proviene da localhost e viene effettuata tramite JavaScript, è stato necessario modificare le politiche CORS nel file web.xml di Solr.

Nel caso di Apache Solr, il file web.xml viene utilizzato per configurare il contesto dell'applicazione Solr e definire le servlet e altre risorse associate a Solr.

Listing 4.4: Filtro per abilitare CORS

```
<filter>
<filter-name>cross-origin</filter-name>
<filter-class>org.eclipse.jetty.servlets.CrossOriginFilter
</filter-class>
<init-param>
    <param-name>allowedOrigins</param-name>
    <param-value>*</param-value>
</init-param>
<init-param>
    <param-name>allowedMethods</param-name>
    <param-value>GET, POST, OPTIONS, DELETE, PUT, HEAD</param-value>
</init-param>
<init-param>
    <param-name>allowedHeaders</param-name>
    <param-value>origin, content-type, accept</param-value>
</init-param>
</filter>
```

Questa configurazione consente di gestire le richieste cross-origin nel contesto di Apache Solr. Il filtro CrossOriginFilter viene utilizzato per consentire alle applicazioni web di effettuare richieste al server Solr da un dominio diverso, specificando le origini, i metodi HTTP e gli header consentiti.

Capitolo 5

Esempi di utilizzo

5.1 Query by Example (Monarch)

IRS Project

Dense Vector Search Application: Query By Example

```
curl -X POST -H "Content-Type: image/png" --data-binary "@images/0010005.png" http://localhost:8983/solr/new_core123/custom
```

0010005.png

Response Headers

```
status 0
QTime 151
```

Image Info

width	[840]
height	[600]
[201.05666, 360.45374, -171.49918, 115.67646, 215.46649, 333.70874, -141.73596, -72.99226, 158.39317, -149.22823, -137.66122, -57.42324, -148.3629, 336.5366, -150.29764, -20.638247, 198.59195, 5.868116, 171.05849, 80.73076, 121.002556, -212.62183, 111.325836, -2.4827316, -28.049095, -148.35732, 67.27934, 92.501, -40.875618, 243.88466, 131.89734, 18.00063, 82.05918, -31.512362, 132.89929, 295.39377, -183.02823, 84.95258, -27.894417, -244.43506, -212.37576, -22.3734, -142.43011, -196.46509, 370.3197, 163.72206, 319.03824, 368.22748, -181.81758, 323.33783, 44.47537, 56.877502, 111.232506, -174.23651, -54.094112, 1.6529857, 164.66223, -210.64247, -37.86024, 476.4699, 235.10359, -213.84296, -84.70515, 188.33008]	

Results:

			
ID: 0010005 Similarity: 1.0000000030451381 Common Name: Monarch	ID: 0010004 Similarity: 0.9860059456671039 Common Name: Monarch	ID: 0010054 Similarity: 0.9927028239219378 Common Name: Monarch	ID: 0010002 Similarity: 0.9923771613962836 Common Name: Monarch

Figura 5.1: Esempio 1

In questo esempio, è stato usata l'interfaccia di *index.html* che non fa altro che visualizzare i risultati in maniera adatta ai contenuti trattati.

Si è caricata l'immagine di una farfalla di tipo Monarch e si vede come i risultati sono ordinati in base alla similarità coseno. Nella pagina c'è anche il comando per interagire con Solr, tramite linea di comando.

5.2 Query by Example (Painted Lady)

IRS Project

Dense Vector Search Application: Query By Example

```
curl -X POST -H "Content-Type: image/png" --data-binary "@images/0100162.png" http://localhost:8983/solr/new_core123/custom
```

0100162.png

Response Headers

status	0
QTime	353

Image Info

width	[200]
height	[145]
feature_vector	[-29.101912, 107.38926, -65.56172, -7.059354, 265.6365, -31.442953, 13.283454, -74.3302, -11.239751, -26.255991, -37.76881, -90.36857, -75.25123, 113.7151, -90.12242, 6.513312, -11.102213, -82.97127, 138.02005, 37.94528, 118.15065, -126.01226, 44.251495, -133.31339, -133.88548, -124.47945, -53.17857, 114.321014, -103.763504, 150.59888, -52.46435, -8.224417, -42.658745, 61.253242, -30.46023, 228.38869, 77.513699, -24.963734, -31.404284, -68.843666, -61.186512, 129.40051, -8.428271, -60.298016, 24.42566, -78.71272, -11.322376, 77.513699, -24.963734, -31.404284, -68.843666, -61.186512, 129.40051, -8.428271, -60.298016, 24.42566, -78.71272, -11.322376]

ID: 0100162 Similarity: 0.999999942473742 Common Name: Painted Lady	ID: 0100025 Similarity: 0.9755388342446679 Common Name: Painted Lady	ID: 0100113 Similarity: 0.9744159942690321 Common Name: Painted Lady	ID: 0100031 Similarity: 0.9739539162200034 Common Name: Painted Lady	ID: 0100163 Similarity: 0.9711579628209575 Common Name: Painted Lady
ID: 0100019 Similarity: 0.9706358950740629 Common Name: Painted Lady	ID: 0100147 Similarity: 0.9694318213739714 Common Name: Painted Lady	ID: 0100012 Similarity: 0.9685280541341692 Common Name: Painted Lady	ID: 0100092 Similarity: 0.9676680786426592 Common Name: Painted Lady	ID: 0100077 Similarity: 0.9659890678538698 Common Name: Painted Lady

Figura 5.2: Esempio 2

Anche in questo esempio è stata usata l'interfaccia di *index.html* che non fa altro che visualizzare i risultati in maniera adatta ai contenuti trattati. Si è caricata l'immagine di una farfalla di tipo Painted Lady e si vede come i risultati sono ordinati in base alla similarità coseno. In questo caso lo screen mostra tutti e 10 i risultati, sempre ordinati per similarità. Si nota come nel caso precedente, che il primo risultato è proprio l'immagine che si è data in input.

5.3 Standard text query

The screenshot shows the Apache Solr Request Handler interface. On the left, there is a form with various input fields and checkboxes. The 'q' field contains 'common_name:monarch'. The 'wt' field is set to 'json'. The 'Execute Query' button is at the bottom. On the right, the results are displayed as a JSON response. The response header includes status 0, QTTime 0, and params. The response itself shows a single document with image_id '0010001', image_path 'images\\0010001.png', scientific_name 'Danaus plexippus', common_name 'Monarch', and a detailed description: '3 1/2-4" (89-102 mm). Very large, with FW long and drawn out. Above, bright, burnt-orange with black veins and stripes. Below, yellowish-orange with black veins and stripes. Immature forms are yellow with black veins and stripes.' The feature_vector is listed as [210.66495, 249.39471, -131.13228, 82.472984, 342.92215, 169.3665, -112.81071, -202.39215, 115.50172, -81.53707, -165.56093, -132.93753, -180.75223, 297.58374, -149.63603, -36.739357, 227.1478, -98.76953, 183.85046, 214.80443, 129.38162, -264.27585, 101.52017, -83.26537, -65.44441, -219.72499, 58.953506, 232.76485, -62.237858, 265.6889].

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "common_name:monarch",
      "indent": "true",
      "q.op": "OR",
      "useParams": "",
      "_": "1685971215281"
    }
  },
  "response": {
    "numFound": 82,
    "start": 0,
    "numFoundExact": true,
    "docs": [
      {
        "image_id": "0010001",
        "image_path": "images\\0010001.png",
        "scientific_name": ["Danaus plexippus"],
        "common_name": ["Monarch"],
        "description": "3 1/2-4\" (89-102 mm). Very large, with FW long and drawn out. Above, bright, burnt-orange with black veins and stripes. Below, yellowish-orange with black veins and stripes. Immature forms are yellow with black veins and stripes."
      }
    ]
  }
}
  
```

Figura 5.3: Esempio 3

Si può interrogare Apache Solr anche specificando un campo indicizzato tramite indice inverso, ad esempio *common_name*.

Capitolo 6

Conclusione

La realizzazione e l’implementazione del sistema di Query by Example hanno rappresentato un’opportunità significativa per esplorare l’efficacia e l’utilità di questa metodologia.

Le caratteristiche chiave delle immagini di input sono state identificate grazie a tecniche di apprendimento automatico e la ricerca di immagini simili è stata eseguita in modo efficiente, grazie all’utilizzo di rappresentazioni efficienti dei dati. Ciò ha contribuito a garantire tempi di risposta ragionevoli e una gestione ottimale di grandi volumi di dati.

Durante le prove e le valutazioni del sistema, sono state effettuate diverse query utilizzando esempi di contenuto multimediale e i risultati ottenuti sono stati confrontati con quelli attesi. I risultati hanno dimostrato una buona precisione e coerenza. Ciò indica che il sistema di Query by Example ha raggiunto l’obiettivo di fornire risultati accurati e rilevanti ai suoi utenti.

Bibliografia

- [1] Gorlapraveen. VGG 16. <https://commons.wikimedia.org/wiki/File:VGG16.png>.
- [2] SEASE Ltd. Apache solr neural search knn benchmark, 2022.
- [3] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. 2018.
- [4] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. 2016.
- [5] Pinecone. Pinecone - learn - hierarchical navigable small world.
- [6] Pinecone. Vector embeddings.
- [7] TopBots. Transfer learning nlp - top bots.
- [8] Josiah Wang, Katja Markert, and Mark Everingham. Learning models for object recognition from natural language descriptions. In *Proceedings of the British Machine Vision Conference*, 2009.