



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA

DIMES

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

MACHINE AND DEEP LEARNING

Relazione Elaborato Finale

"Learning Hierarchical Graph Neural Networks for Image Clustering"

Professori

Prof. Fabrizio Angiulli

Prof. Francesco De Luca

Studente

Stefano Francesco Monea

Matr. 256972

Indice

1	Introduzione	2
1.1	Fase di Encoding	4
1.2	Fase di predizione di link e densità	4
1.3	Fase di Decoding	5
1.4	Loss	5
2	Panoramica dataset e metriche di valutazione	6
2.1	Dataset	6
2.2	Metriche di valutazione	9
2.2.1	Clustering	9
2.2.2	Riconoscimento di visi	10
3	Ricreazione esperimento	11
3.1	Evaluation	12
4	Modifiche Architetturali	14
4.1	Modifica 1: normalizzazione del peso per l'aggregazione	14
4.1.1	Train ed Eval	14
4.2	Modifica 2: un nuovo modo di pesare l'aggregazione	16
4.2.1	Train ed Eval	17
4.3	Modifica 3: modo alternativo di aggregare nel GAT-layer	18
4.3.1	Train ed Eval	19
4.4	Modifica 4: modifica del MLP usato per classificazione	19
4.4.1	Train ed Eval	20
4.5	Modifica 5: nuovo modo di concatenare le features	23
4.5.1	Train ed Eval	23
4.6	Modifica 6: nuove features per il grafo al prossimo livello	25
4.6.1	Train ed Eval	25
4.7	Modifica 7: approccio ibrido	27
4.7.1	Train ed Eval	27
5	Conclusioni	31

1 Introduzione

Il clustering è una tecnica fondamentale nell’ambito dell’apprendimento non supervisionato, che mira a raggruppare un insieme di dati sulla base di un criterio di somiglianza. L’obiettivo è identificare strutture o pattern naturali nei dati senza l’ausilio di etichette. Tuttavia, la natura non supervisionata del clustering presenta una difficoltà: in assenza di etichette, la scelta del criterio di raggruppamento e della granularità della partizione è arbitraria e può portare a molteplici soluzioni differenti, rendendo difficile determinare quale sia la migliore o più significativa per un dato problema. L’approccio presentato nel lavoro *Learning Hierarchical Graph Neural Networks for Image Clustering* introduce **Hi-LANDER** (Hierarchical Link Approximation and Density Estimation Refinement), un metodo innovativo che affronta il problema del clustering gerarchico agglomerativo attraverso l’apprendimento da un insieme di dati etichettati (noto anche come meta-training set), disgiunto da quello su cui si effettua l’inferenza (test set). L’uso del meta-training è motivato dagli autori, in quanto il loro obiettivo è quello di sfruttare il training set così costituito in modo da far sì che il modello impari a raggruppare diversi test set con un numero sconosciuto di identità, questo si avvicina molto a quello che in gergo prende il nome di *open-set* oppure *open-universe classification*.

Hi-LANDER si basa sull’uso delle *Graph Neural Networks* (GNNs) in quanto sono uno strumento naturale per fare clustering, visto che forniscono un modo per predire la connettività in un grafo. La tecnica presentata è una procedura iterativa che parte da un grafo k -NN costruito su rappresentazioni visuali (*features embedding*) delle immagini e, ad ogni livello gerarchico, seleziona sottoinsiemi di archi da mantenere in base a due proprietà/quantità chiave: la **probabilità di connessione** tra due nodi e la **densità locale** di ciascun nodo.

A differenza dei precedenti approcci GNN-based che utilizzano modelli distinti per stimare la densità e la connettività, Hi-LANDER impiega un singolo modello unificato denominato **LANDER**, che predice congiuntamente entrambe le quantità, riducendo così i costi computazionali e migliorando l’efficienza. L’output del modello a ciascun livello è un insieme di componenti connesse, che vengono aggregate in *super-nodi*, i quali a loro volta costituiscono il grafo del livello successivo. Il processo viene ripetuto fino alla convergenza, cioè fino a quando il modello non aggiunge più nuovi archi.

La procedura è formalizzata come segue:

1. Costruzione del grafo iniziale (G_1):

- Dato un insieme di N immagini e i loro *visual embedding*, si costruisce un grafo di affinità iniziale $G_1 = \langle V_1, E_1 \rangle$;
- I nodi V_1 del grafo corrispondono ai *visual embedding* di ciascuna immagine;
- Gli archi E_1 sono stabiliti connettendo ogni nodo ai suoi k vicini più simili.

2. Selezione degli archi e formazione delle componenti connesse (livello ℓ):

- A ogni livello gerarchico ℓ , il modello **LANDER** viene utilizzato per predire congiuntamente la **probabilità di connessione** tra due nodi e la **densità locale** di ciascun nodo;
- Sulla base di queste predizioni, viene selezionato un sottoinsieme di archi $E'_\ell \subset E_\ell$, risultando in un grafo sparso $G'_\ell = \langle V_\ell, E'_\ell \rangle$.
- Le componenti connesse di G'_ℓ sono identificate e denominate $\{c_i^{(\ell)}\}$.

3. Aggregazione in Super-Nodi e creazione del livello successivo ($V_{\ell+1}$):

- Ogni componente connessa $\{c_i^{(\ell)}\}$ viene aggregata per formare un nuovo nodo (un “super-nodo”) per il livello successivo. Questi super-nodi costituiscono l’insieme $V_{\ell+1}$;
- Le features di ciascun super-nodo $V_{\ell+1}$ sono calcolati attraverso una funzione di aggregazione ψ che combina le feature dei nodi della componente connessa corrispondente.

4. Iterazione:

- Viene costruito un nuovo grafo $G_{\ell+1}$ connettendo i super-nodi di $V_{\ell+1}$ anch’essi tramite una relazione k -NN basata sulle loro nuove rappresentazioni;
- Il processo (dal punto 2 al 4) viene ripetuto iterativamente fino alla convergenza.

La convergenza è raggiunta quando $E'_\ell = \emptyset$, cioè quando non ci sono più archi aggiunti al grafo. A quel punto, l’assegnazione finale dei cluster viene ottenuta propagando gli identificatori delle componenti connesse dal livello più alto fino ai nodi originari.

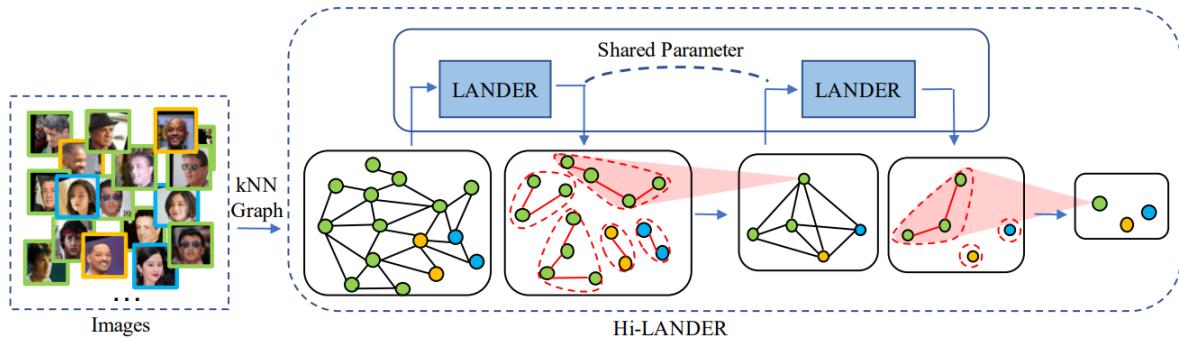


Figura 1: Schema di Hi-Lander presente sul paper

Viene riportato lo pseudo-codice per semplicità di comprensione: In cui:

Algorithm 1 Hierarchical Generalization

Input: N, F, k ;

$l \leftarrow 1$

$H_1 \leftarrow F$

while not converged **do**

$G_l \leftarrow$ k-nearest-neighbor(H_l, k)

$E'_l \leftarrow \phi(G_l, H_l)$

$G'_l \leftarrow$ connected-components(E'_l)

$H_{l+1} \leftarrow \psi(H_l, G'_l)$

$l \leftarrow l + 1$

end while

$ID \leftarrow$ id-propagation($\{G_l\}, \{G'_l\}$)

return ID

- La funzione ϕ : predice congiuntamente la probabilità di connessione tra nodi e una stima della densità locale di un nodo;

- La funzione ψ : ha il compito di costruire le feature dei super-nodi per il livello successivo della gerarchia.

Per ogni componente connessa individuata da ϕ , ψ calcola due vettori: uno basato sul nodo con densità massima (peak feature) e uno come media delle feature dei nodi nel cluster (average feature). Questi due vettori vengono concatenati e utilizzati per costruire il grafo del livello successivo. Questo schema consente al modello di iterare il processo gerarchico in modo efficiente e strutturato.

1.1 Fase di Encoding

La fase di encoding ha lo scopo di trasformare le feature iniziali dei nodi h_i (visual embeddings) in rappresentazioni arricchite h'_i tramite un modello GNN, più precisamente una *Graph Attention Network* (GAT). L'uso di GAT consente al nodo di aggregare informazioni dai vicini pesandole in base alla loro rilevanza. In alternativa, gli autori osservano che anche layer più semplici come GCN (Graph Convolution Network) producono risultati simili, ma GAT è usato come default.

Il risultato dell'encoder è una nuova rappresentazione, una nuova features h'_i per ogni nodo v_i .

1.2 Fase di predizione di link e densità

A partire dai nuovi embeddings h'_i , il modello LANDER esegue due predizioni per ogni coppia di nodi connessi (v_i, v_j) :

- **Linkage probability** p_{ij} , la probabilità che v_i e v_j appartengano allo stesso cluster;
- **Pseudo-densità** \hat{d}_i , una stima, un *proxy* della densità locale del nodo v_i .

La predizione di p_{ij} è ottenuta concatenando gli embeddings dei due nodi:

$$[h'_i, h'_j]$$

e passando tale concatenazione in un MLP (Multi Layer Perceptron) seguito da softmax:

$$p_{ij} = \text{softmax}(\text{MLP}([h'_i, h'_j]))$$

Per stimare la pseudo-densità \hat{d}_i , si calcola una media pesata delle similarità con i vicini:

$$\hat{d}_i = \frac{1}{k} \sum_{j \in \mathcal{N}_k(i)} \hat{e}_{ij} \cdot a_{ij}$$

dove:

- $a_{ij} = \langle h'_i, h'_j \rangle$ è la similarità coseno (in gergo noto come dot product) tra i due embeddings;
- $e_{ij} = P(y_i = y_j) - P(y_i \neq y_j)$ è una stima di quanto è probabile che i nodi appartengano alla stessa classe (valori positivi indicano alta probabilità di stesso cluster, negativi altrimenti), il pedice j indica i k vicini di v_i . Con riferimento al paper, questa quantità viene chiamata *edge-coefficient*. Nell'equazione precedente è scritta, vi è \hat{e}_{ij} al posto di e_{ij} , ma che può essere ottenuto in modo molto semplice: $\hat{e}_{ij} = \mathbf{1}(y_i = y_j) - \mathbf{1}(y_i \neq y_j)$.

Questa stima di densità riflette quanto i vicini più simili appartengano allo stesso cluster e aiuta a stabilire un ordinamento tra i nodi (i nodi con bassa densità sono tipicamente ai confini).

1.3 Fase di Decoding

Dato p_{ij} e \hat{d}_i , per ciascun nodo v_i si definisce il sottoinsieme di candidati:

$$E(i) = \{j \mid (v_i, v_j) \in E \wedge \hat{d}_i \leq \hat{d}_j \wedge p_{ij} \geq \tau\}$$

dove τ è un iperparametro.

Se $E(i) \neq \emptyset$, si seleziona il nodo j che massimizza e_{ij} :

$$j^* = \arg \max_{j \in E(i)} e_{ij}$$

e si aggiunge l'arco (v_i, j^*) a E' , che forma il grafo sparso $G' = (V, E')$. Le componenti connesse di G' costituiscono i cluster a questo livello.

1.4 Loss

La loss globale è composta da due termini:

$$\mathcal{L} = \mathcal{L}_{\text{conn}} + \mathcal{L}_{\text{den}}$$

Connection Loss $\mathcal{L}_{\text{conn}}$

Misura la correttezza della predizione di p_{ij} rispetto alle etichette ground truth $q_{ij} = \mathbf{1}[y_i = y_j]$, cioè se i due nodi appartengono allo stesso cluster. Si usa dunque la *binary cross-entropy*:

$$\mathcal{L}_{\text{conn}} = -\frac{1}{|E|} \sum_{(i,j) \in E} \begin{cases} q_{ij} \log p_{ij} + (1 - q_{ij}) \log(1 - p_{ij}), & \text{se } d_i \leq d_j \\ 0, & \text{altrimenti} \end{cases}$$

Density Loss \mathcal{L}_{den}

Confronta la densità stimata \hat{d}_i con la densità ground-truth d_i (calcolata come media dei vicini con stessa etichetta), non è altro che l'errore quadratico medio (MSE):

$$\mathcal{L}_{\text{den}} = \frac{1}{|V|} \sum_{i=1}^{|V|} \|d_i - \hat{d}_i\|_2^2$$

Entrambe le perdite sono calcolate e sommate per tutti i livelli gerarchici durante il training. Il vantaggio dell'approccio è che un singolo modello predice congiuntamente connessione e densità, condividendo parametri e beneficiando di una supervisione condivisa.

Analizzando il codice si può osservare che è possibile specificare se utilizzare la *Focal Loss*, funzione di perdita utilizzata nel caso in cui si voglia gestire uno sbilanciamento marcato tra classi. Questa funzione di perdita introduce un termine di modulazione che riduce il peso assegnato agli esempi “facili” (cioè quelli con alta confidenza), concentrando l’attenzione sugli esempi “difficili”. Di seguito per completezza, viene riportata la forma generale della focal loss per una classificazione binaria:

$$\mathcal{L}_{\text{focal}} = -\alpha_t (1 - p_t)^\gamma \log(p_t)$$

2 Panoramica dataset e metriche di valutazione

Gli esperimenti condotti nel paper impiegano un'ampia varietà di dataset per l'addestramento e la valutazione del metodo di clustering presentato, focalizzandosi sia sul clustering di volti che sul clustering di specie animali. Questa diversità permette una valutazione robusta delle capacità di generalizzazione del modello in scenari open-set.

2.1 Dataset

Per il clustering di volti, vengono utilizzati i seguenti dataset:

- **TrillionPairs**: Un dataset di immagini su larga scala, da cui viene selezionato casualmente un decimo ($\sim 660K$ volti) per il processo di addestramento. La dimensione media dei cluster in questo dataset è di 37 volti;
- **IMDB**: Utilizzato per il testing, include 1,2 milioni di volti. Presenta una sovrapposizione minima (inferiore al 2%) di identità di persona rispetto al set di addestramento TrillionPairs. La dimensione media dei cluster è di 25.
- **Hannah**: Un altro dataset per il testing, contenente circa 200K volti. È cruciale notare che Hannah non ha nessuna identità dipersona sovrapposte con il set di addestramento TrillionPairs, rendendolo particolarmente adatto per valutare le capacità open-set. La dimensione media dei cluster qui è significativamente più grande, pari a 800.

I dataset Deepglint (TrillionPairs) e IMDB vengono inoltre impiegati per l'addestramento con pseudo-etichette per il riconoscimento facciale.

Per il clustering di specie animali, il lavoro si avvale di:

- **iNaturalist2018**: Questo dataset viene suddiviso, garantendo che le classi utilizzate per l'addestramento ($\sim 320K$ istanze) siano completamente disgiunte da quelle utilizzate per il testing ($\sim 130K$ istanze). Entrambe le suddivisioni mostrano distribuzioni di dimensione dei cluster simili, con una media di 56 istanze per classe. Le feature per questo task sono estratte da un modello ResNet50 pre-addestrato.

I dataset utilizzati sono distribuiti sul web in formato `pickle`, ciascuno dei quali contiene una tupla nella forma:

$$(\text{features}, \text{labels})$$

dove `features` è un array `numpy.ndarray` di vettori numerici estratti da un modello di deep learning (ad esempio, ResNet50 o un backbone per il riconoscimento facciale), mentre `labels` è un array di interi rappresentanti le etichette di classe corrispondenti.

Un esempio semplificato del contenuto è il seguente:

```
[array([[ 0.04876965, -0.0022702 , -0.09742823, ..., -0.10640518,
       0.0167092 ,  0.20930775],
       [ 0.04028706,  0.00432362, -0.08870611, ..., -0.112983 ,
       0.04768377,  0.18264517],
       [ 0.04569606,  0.00269087, -0.09117389, ..., -0.12082147,
       0.02699837,  0.17943028],
       ...,
       [ 0.02597456,  0.03862995,  0.00876153, ..., -0.02229541,
       -0.00136784,  0.08818514],
       [ 0.07475369, -0.08642773,  0.06043139, ...,  0.04062082,
       -0.12228334,  0.06424055]],
```

```
[ 0.06698991,  0.02681589, -0.07973656, ...,  0.00929351,
 -0.03647897,  0.10198697]],  
array([206, 206, 206, ..., 106, 102, 204])]  
  
(2,) <- shape
```

Ogni riga dell'array delle `features` rappresenta un'immagine trasformata in uno spazio vettoriale ad alta dimensione, tipicamente con 128, 512 o 1024 dimensioni, a seconda del modello. Queste rappresentazioni non sono interpretabili visivamente come immagini, ma possono essere visualizzate sotto forma di *heatmap*, dove l'intensità del colore rappresenta l'ampiezza di ciascun valore nel vettore.

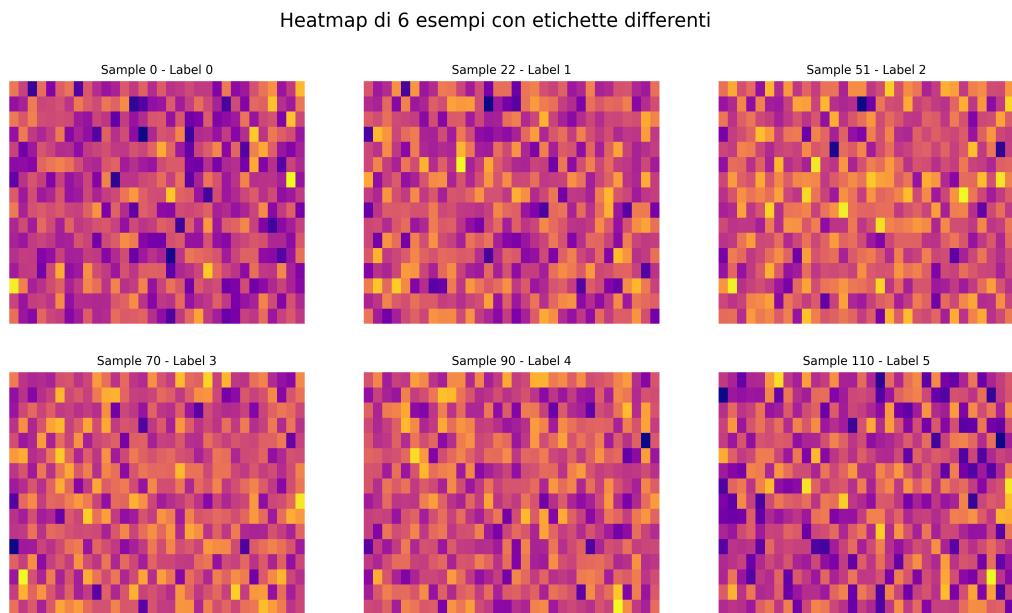


Figura 2: Heatmap di 6 esempi del dataset INAT

Oltre alla heatmap che, in Figura 3 viene presentato anche una proiezione delle stesse in uno spazio bidimensionale ottenuta tramite PCA (Principal Component Analysis). Questa visualizzazione permette di valutare qualitativamente la separabilità tra classi.

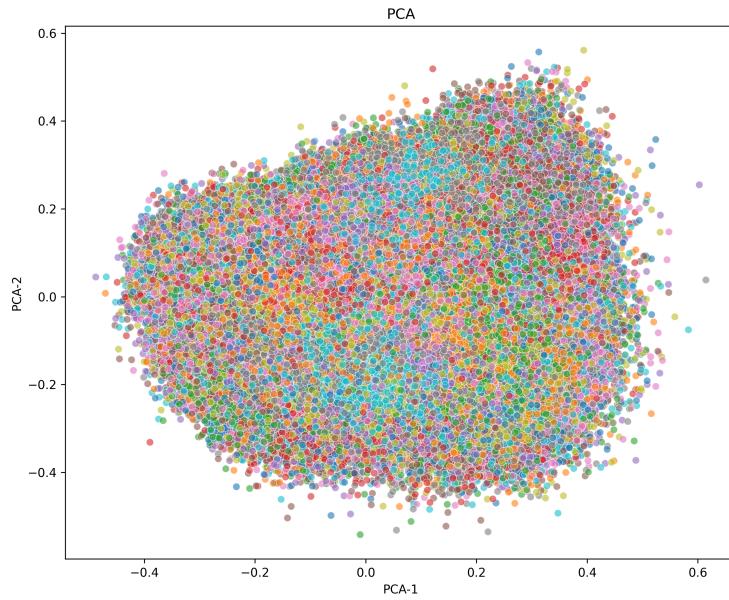


Figura 3: PCA del dataset INAT

È importante sottolineare che, sebbene nella proiezione PCA le feature appaiano visivamente addensate o sovrapposte, ciò è una conseguenza naturale della riduzione della dimensionalità. La PCA conserva principalmente la varianza globale dei dati, ma non necessariamente la struttura locale o le separazioni non lineari tra classi, che possono essere presenti nello spazio originale ad alta dimensionalità. Di conseguenza, una separazione visivamente non marcata in 2D non implica una scarsa discriminabilità. Viene dunque riportato un'altra visualizzazione sfruttando in questo caso *t-SNE*, una tecnica di embedding non lineare progettata per preservare le relazioni locali nei dati, offrendo una rappresentazione più chiara della distribuzione e separazione tra le classi.

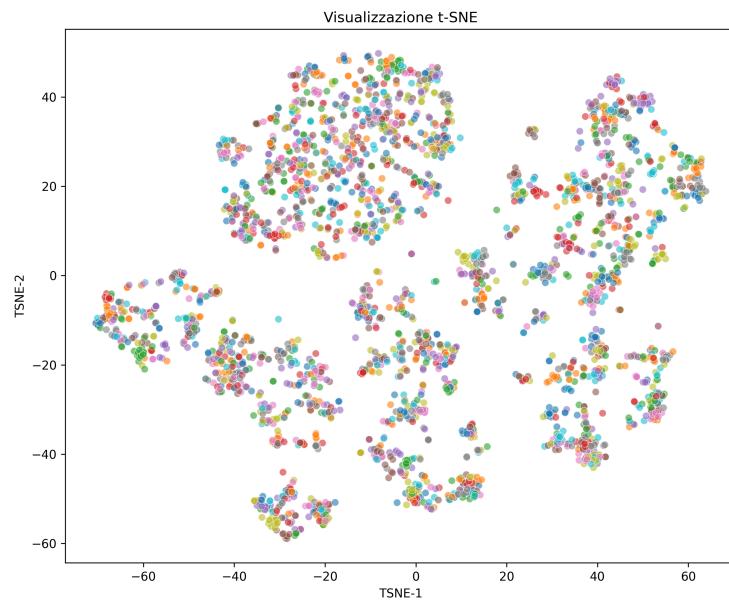


Figura 4: t-SNE del dataset INAT, campionando circa 3000 esempi

A differenza della **PCA**, che è un metodo lineare che cerca di massimizzare la varianza globale proiettando i dati lungo assi ortogonali di massima varianza, t-SNE si concentra sulle similitudini locali e può catturare relazioni complesse e non lineari tra i dati. Questo rende t-SNE particolarmente efficace per visualizzare cluster separati anche quando la struttura globale è complessa. La visualizzazione di quest'ultimo grafico è utile per capire che il dataset non si pone come in Figura 3, ma hanno una struttura complessa difficilmente rappresentabile in uno spazio bidimensionale.

2.2 Metriche di valutazione

La performance del modello è valutata utilizzando diverse metriche standard, specifiche per i task di clustering e riconoscimento dei volti.

2.2.1 Clustering

- **Normalized Mutual Information (NMI)**: questa metrica cattura sia l'omogeneità (quanto i cluster contengono solo membri di una singola classe) che la completezza (quanto tutti i membri di una data classe sono assegnati allo stesso cluster);

$$\text{NMI}(U, V) = \frac{2 \cdot I(U; V)}{H(U) + H(V)}$$

dove $I(U; V)$ è l'informazione mutua tra U e V , mentre $H(U)$ e $H(V)$ sono le rispettive entropie.

- **F-score Pairwise (Fp)**: un tipo di media armonica tra precision e recall. Si definisce:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$F_p = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

dove:

- TP : coppie correttamente assegnate allo stesso cluster;
- FP : coppie assegnate allo stesso cluster ma con etichette diverse;
- FN : coppie con la stessa etichetta ma assegnate a cluster diversi.

- **B-cubed F-score (Fb)**: a differenza di quella introdotta precedentemente, essa valuta la coerenza tra clustering predetto ed etichetta reale per ciascun elemento, aggregando i risultati individuali in una media complessiva.

$$\text{Precision}(x) = \frac{|\text{Cluster}(x) \cap \text{Class}(x)|}{|\text{Cluster}(x)|}, \quad \text{Recall}(x) = \frac{|\text{Cluster}(x) \cap \text{Class}(x)|}{|\text{Class}(x)|}$$

$$F_b = \frac{1}{n} \sum_x \frac{2 \cdot \text{Precision}(x) \cdot \text{Recall}(x)}{\text{Precision}(x) + \text{Recall}(x)}$$

dove $\text{Cluster}(x)$ è l'insieme degli elementi nel cluster di x , e $\text{Class}(x)$ l'insieme degli elementi con la stessa etichetta reale.

2.2.2 Riconoscimento di visi

- **False Non-Match Rate (FNMR)**: indica la proporzione di coppie corrispondenti che non vengono riconosciute come tali, misurata a diversi livelli di **False Match Rate (FMR)**.

$$\text{FNMR} = \frac{\text{FN}}{\text{TP} + \text{FN}} \quad @\text{FMR}(\text{ad un dato FMR})$$

- **False Negative Identification Rate (FNIR)**: è la frazione di identità vere non identificate dal sistema, misurata rispetto a vari livelli di **False Positive Identification Rate (FPIR)**.

$$\text{FNIR} = \frac{\text{FN}}{\text{TP} + \text{FN}} \quad @\text{FPIR}(\text{ad un dato FPIR})$$

3 Ricreazione esperimento

Gli autori del paper hanno condotto 3 diversi addestramenti per valutare le prestazioni dell'architettura *Hi-LANDER* con i seguenti iperparametri:

Tabella 1: Iperparametri generici

Iperparametro	Valore
k_nn	10,5,3
levels	2,3,4
hidden	512
epochs	250
lr	0.01
batch_size	4096
num_conv	1
optm	SGD
momentum	0.9

Con le eventuali seguenti flag attive:

- *faiss_gpu* o *faiss_cpu*: la presenza indica quale strategia adottare per FAISS, una libreria sviluppata da META per la ricerca efficiente di similarità e il clustering di vettori densi. Il suo scopo principale è quello di trovare rapidamente i "vicini" più prossimi (i cosiddetti nearest neighbors) di un dato vettore di query;
- *balance*: la cui presenza indica il riferirsi ad una strategia di bilanciamento dei campioni durante il calcolo della loss di connessione. Questo bilanciamento viene applicato per affrontare lo squilibrio tra il numero di esempi positivi e negativi all'interno di un batch di dati. L'obiettivo di questa operazione è evitare che la loss sia dominata dalla classe maggioritaria, contribuendo a una migliore generalizzazione del modello;
- *gat*: la cui presenza indica l'uso del Graph Attention Convolution layer, se diversamente specificato si farà riferimento ad layer Graph Convolution tradizionale;
- *use_cluster_feat*: la cui presenza determina se il modello incorpora informazioni aggiuntive sul cluster degli elementi, potenzialmente fornendo un contesto più ricco per l'apprendimento delle connessioni e delle densità.

Vengono riportate le loss relative ai tre dataset precedentemente menzionati:

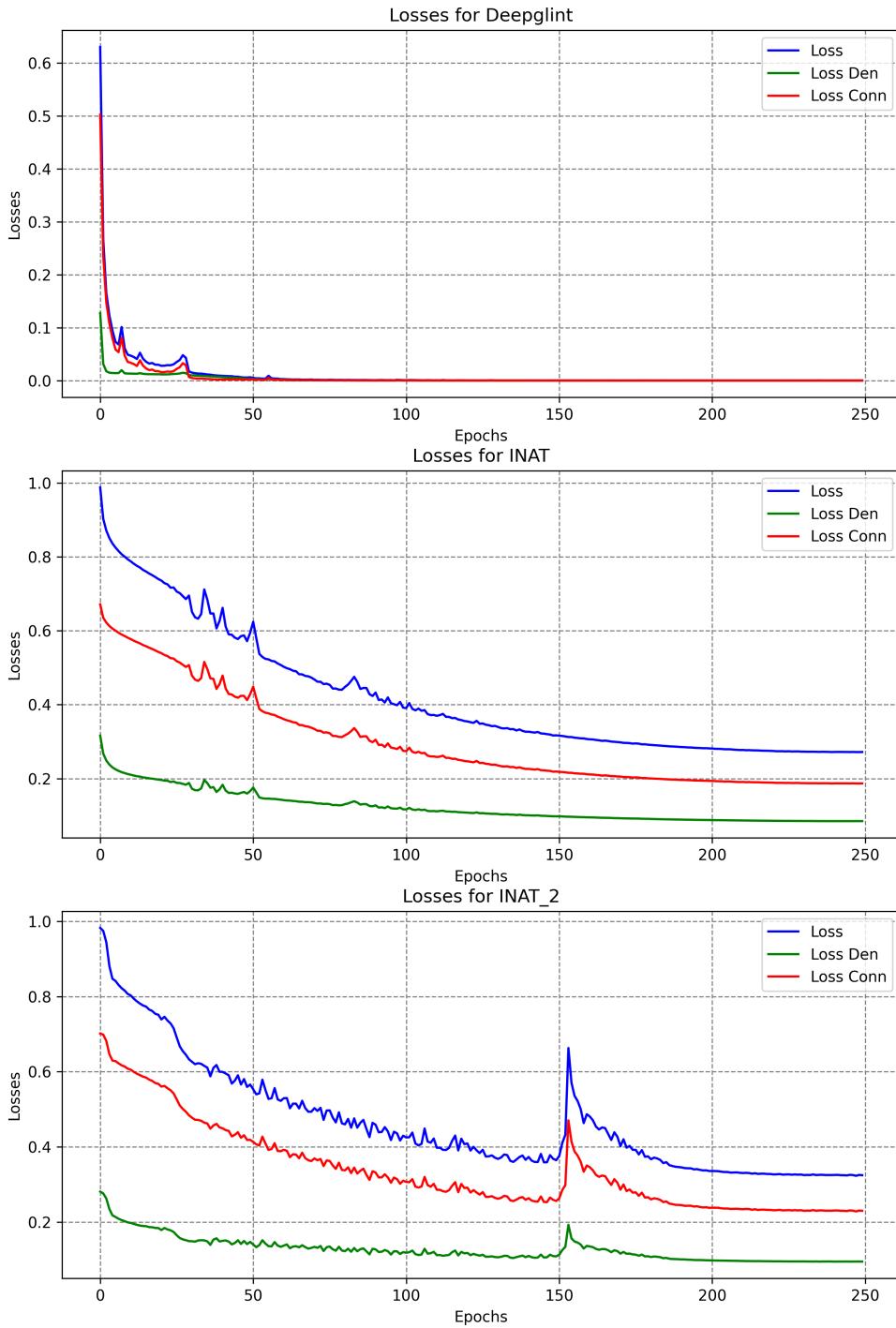


Figura 5: Loss dei diversi dataset con l'architettura base senza apportare modifiche

3.1 Evaluation

In fase di inferenza gli autori utilizzano due setting:

- *Clustering with Unseen Test Data Distribution:* in questo setting abbiamo che la distribuzione a tempo di test è sconosciuta e differente da quella del meta-training. In altre parole, parametri (come τ e k-NN k) non possono essere regolati in anticipo utilizzando informazioni provenienti nella fase di test;

- *Clustering with Seen Test Data Distribution*: setting opposto e più semplificato rispetto a quello in precedenza, in quanto le due distribuzioni delle dimensioni dei cluster dei dati di addestramento e test sono simili.

In particolare abbiamo di seguito i valori ottenuti, che sono confrontati con quelli presenti nel paper.

Tabella 2: Confronto dei risultati in seguito alla riproduzione dell'esperimento vs baseline (paper)

Setting	Source	IMDB-Test-SameDist			iNat2018-Test		
		Fp	Fb	NMI	Fp	Fb	NMI
Seen Test Data	Riproduzione	0.7851	0.8198	0.9490	0.3175	0.3507	0.7699
	Paper Originale	0.779	0.8198	0.94	0.330	0.350	0.774

Tabella 3: Confronto dei risultati in seguito alla riproduzione dell'esperimento vs baseline (paper)

Setting	Source	Hannah			IMDB		
		Fp	Fb	NMI	Fp	Fb	NMI
Unseen Test Data	Riproduzione	0.7698	0.7267	0.8179	0.7154	0.8097	0.9528
	Paper Originale	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	iNat2018-Test		
		Fp	Fb	NMI
Unseen Test Data	Riproduzione	0.2913	0.3374	0.7651
	Paper Originale	0.294	0.352	0.764

I riscontri ottenuti sono ottimi, gli autori stessi del paper esplicitamente avvertono che i risultati ottenuti potrebbero presentare delle fluttuazioni. Nello specifico, queste variazioni sono da ricondurre all'introduzione della randomicità utilizzando la libreria FAISS.

4 Modifiche Architetturali

In questa sezione vengono riportate le modifiche architetturali effettuate con i relativi addestramenti e testing.

4.1 Modifica 1: normalizzazione del peso per l'aggregazione

La prima modifica architetturale implementata riguarda la modalità di aggregazione dei messaggi all'interno del layer convoluzionale di base del grafo, specificamente nella classe ‘GraphConvLayer’ (file ‘graphconv.py’).

```
1 graph.update_all(fn.u_mul_e("h", "affine", "m"), fn.sum(msg="m", out="h"))
```

In precedenza, la funzione di aggiornamento dei nodi utilizzava il peso `edges.data["affine"]` per ponderare i messaggi provenienti dai nodi sorgente prima della loro somma. Questa impostazione è stata modificata per utilizzare `edges.data["raw_affine"]` al posto di `edges.data["affine"]`, risultando nella seguente operazione:

```
1 graph.update_all(fn.u_mul_e("h", "raw_affine", "m"), fn.sum(msg="m", out="h"))
```

La differenza fondamentale tra ‘affine’ e ‘raw_affine’ risiede nella loro definizione all’interno della pipeline di costruzione del grafo. Il campo ‘affine’ viene popolato direttamente dai valori normalizzati della matrice di adiacenza K-NN, ottenuti tramite una normalizzazione per riga. Successivamente, ‘raw_affine’ viene derivato da ‘affine’ dividendo quest’ultimo per `edges.dst["norm"]`, dove rappresenta la somma delle righe non normalizzate della matrice di adiacenza per il nodo di destinazione.

Questo approccio introduce una forma di doppia normalizzazione o una normalizzazione aggiuntiva che tiene conto non solo dei pesi di similarità iniziali, ma anche della connettività locale o "densità" del nodo di destinazione. Si è implementato tale modifica perché idealmente potrebbe portare ad una maggiore stabilità numerica durante l’addestramento, ed una ponderazione più equa dei messaggi proveniente da nodi con *gradi diversi*.

4.1.1 Train ed Eval

Per valutare l’impatto delle modifiche architetturali introdotte si mette a confronto la loss del modello con le modifiche apportate solo sul layer convoluzionale standard (senza l’utilizzo di GAT) con quello di partenza.

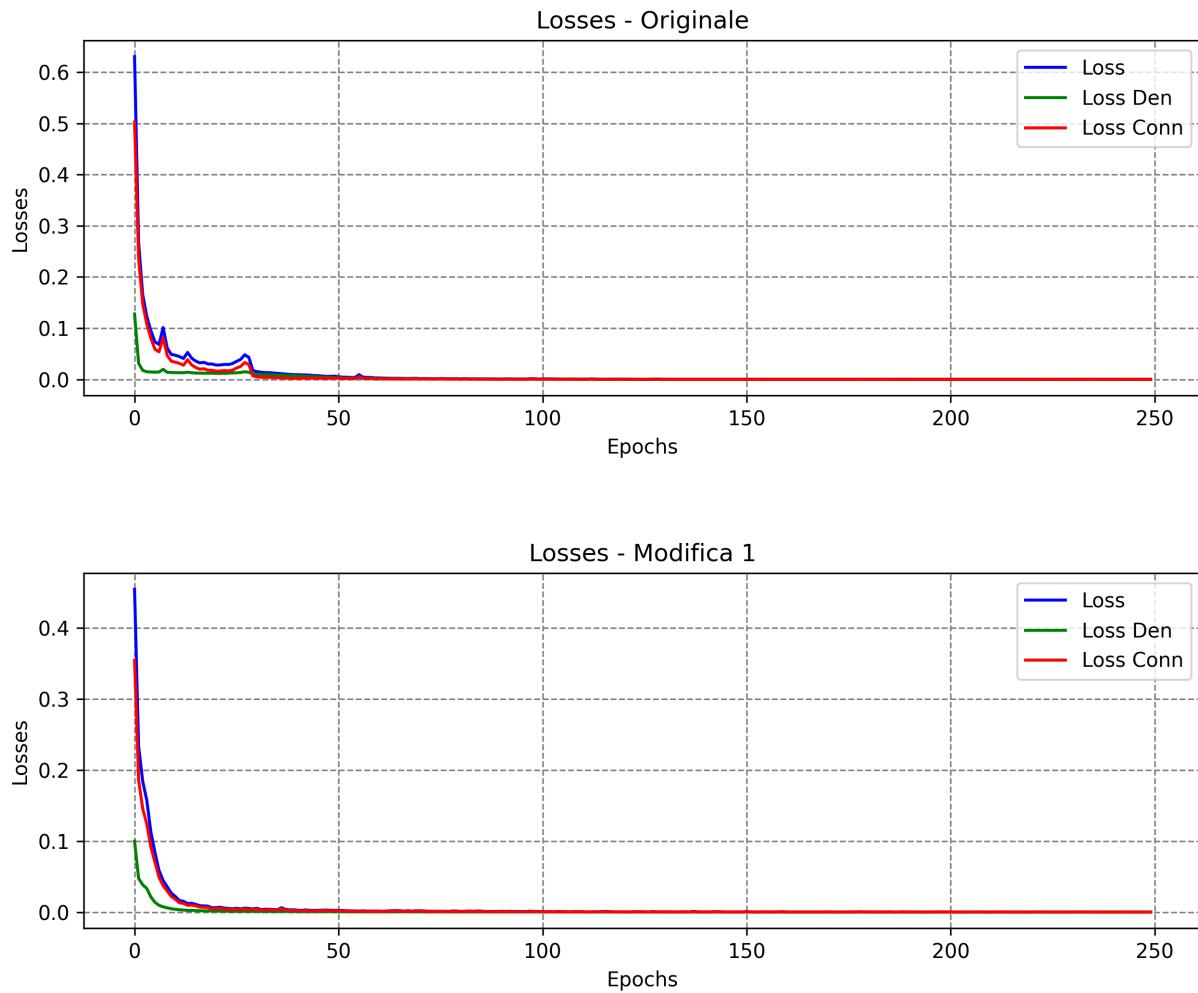


Figura 6: Loss di addestramento in seguito alla modifica 1

Tabella 4: Confronto dei risultati in seguito alla Modifica 1 vs baseline (paper)

Setting	Source	Hannah			IMDB		
		Fp	Fb	NMI	Fp	Fb	NMI
Unseen Test Data	Modifica 1	0.5925	0.7584	0.9255	0.3429	0.7222	0.9256
	Baseline (paper)	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	IMDB-Test-SameDist		
		Fp	Fb	NMI
Seen Test Data	Modifica 1	0.5410	0.7596	0.9253
	Baseline (paper)	0.779	0.819	0.949

L'introduzione della normalizzazione dei pesi degli archi basata su raw_affine all'interno del message passing nel GraphConvLayer ha dimostrato un impatto significativo sulla dinamica della loss durante l'addestramento.

Analizzando il grafico infatti, e confrontandolo con la loss del modello baseline, si osserva immediatamente una convergenza più rapida e stabile della loss totale (Loss), della loss di

densità (Loss Den) e della loss di connettività (Loss Conn). Il modello modificato raggiunge valori di loss prossimi allo zero in un numero significativamente inferiore di epoche rispetto alla baseline. Già dopo circa 15/20 epoche, la loss per la Modifica 1 è quasi azzerata, mentre il modello originale impiega circa 30 epoche per stabilizzarsi, e con valori iniziali di loss più elevati.

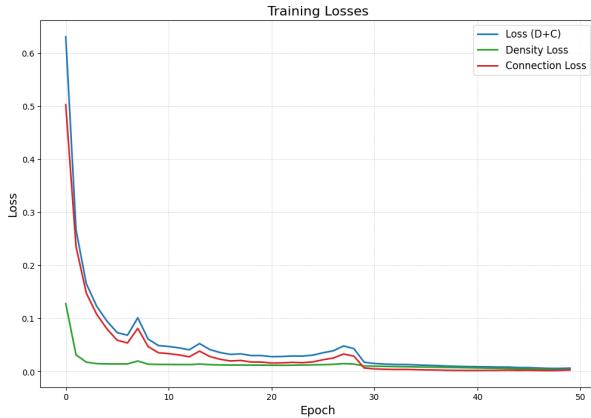


Figura 7: Loss baseline

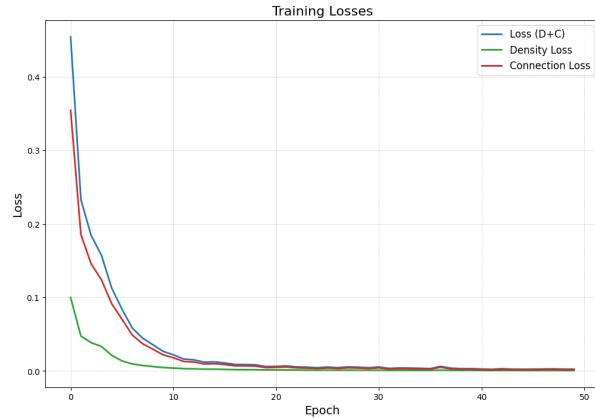


Figura 8: Loss modifica 1

Figura 9: Confronto delle curve di loss entro la 50'esima epoca in seguito alla modifica.

Tuttavia, i risultati mostrati nelle tabelle presentano un quadro più articolato. Sebbene la Modifica 1 mostri un NMI (Normalized Mutual Information) superiore o comparabile rispetto alla baseline su alcuni dataset, indicando una migliore qualità del clustering in termini di raggruppamento delle istanze, si registra un peggioramento notevole di Fp, soprattutto sul dataset Hannah Unseen e IMDB Seen. La normalizzazione aggiuntiva, pur migliorando la stabilità e la convergenza della loss e l'NMI (che valuta la qualità del raggruppamento indipendentemente dall'etichettatura), potrebbe aver reso il modello meno preciso, o aver influenzato la capacità del modello di distinguere con precisione tra istanze positive e negative.

4.2 Modifica 2: un nuovo modo di pesare l'aggregazione

Bene a questo punto ci chiediamo, ma non teniamo in considerazione la densità del nodo sorgente? Anche in questo caso la modifica riguarda la modalità di aggregazione dei messaggi all'interno del layer convoluzionale di base del grafo. In particolare, sempre nel file *graphconv.py* la seguente riga:

```
1 graph.update_all(fn.u_mul_e("h", "affine", "m"), fn.sum(msg="m", out="h"))
```

è stata sostituita con:

```
1 graph.srcdata["h_density"] = graph.srcdata["h"] * graph.srcdata["density"]
    ].unsqueeze(1)
2 graph.update_all(fn.u_mul_e("h_density", "affine", "m"), fn.sum(msg="m",
    out="h"))
```

Viene introdotto un passaggio intermedio prima dell'aggregazione, ossia, calcola una nuova feature temporanea chiamata *h_density* per i nodi sorgente. Successivamente si prende la feature corrente del nodo sorgente (*graph.srcdata["h"]*) e lo si moltiplica elemento per elemento per la density del nodo sorgente. Per l'aggregazione, invece di usare direttamente "h", si usa la nuova feature *h_density* come informazione da propagare. Qual è l'obiettivo di

questa modifica? Quello di pesare la feature del nodo sorgente in base alla sua densità locale prima che questi vengano propagati e aggregati. Moltiplicando la feature del nodo sorgente per la sua densità, i messaggi provenienti da nodi più densi avranno un peso o un'influenza maggiore sui nodi vicini durante l'aggregazione.

Perchè si è scelto di usare proprio la densità come peso? L'architettura proposta nel paper, utilizza la densità dei nodi per il processo di decode e la selezione dei "peak nodes" (nodi che rappresentano i centri dei cluster per il livello successivo). Incorporare la densità già nella fase di aggregazione dei messaggi della GNN potrebbe rendere il modello più intrinsecamente allineato con la sua logica di clustering. In quanto, ci si aspetterebbe che i nodi centrali e più densi di un cluster (noti spesso anche come i "core points") abbiano un'influenza predominante nella formazione delle rappresentazioni dei nodi vicini. Questo dovrebbe portare a cluster più compatti e ben definiti, poichè l'informazione più rappresentativa dei cluster verrebbe propagata con maggiore forza. C'è da tenere in considerazione però, che possono esistere degli scenari dove i cluster sono a bassa densità ma ben separati, questa modifica potrebbe penalizzare eccessivamente la propagazione dei messaggi all'interno di tali cluster, rendendoli più difficili da identificare correttamente. Oppure, se i nodi più densi iniziano a dominare eccessivamente il controllo del flusso dell'informazione, il modello potrebbe diventare meno sensibile alle sfumature e alle interconnessioni tra i cluster.

4.2.1 Train ed Eval

Anche in questo caso per valutare l'impatto della modifica precedentemente introdotta, si deve fare riferimento all'architettura senza che faccia d'uso del GAT-layer.

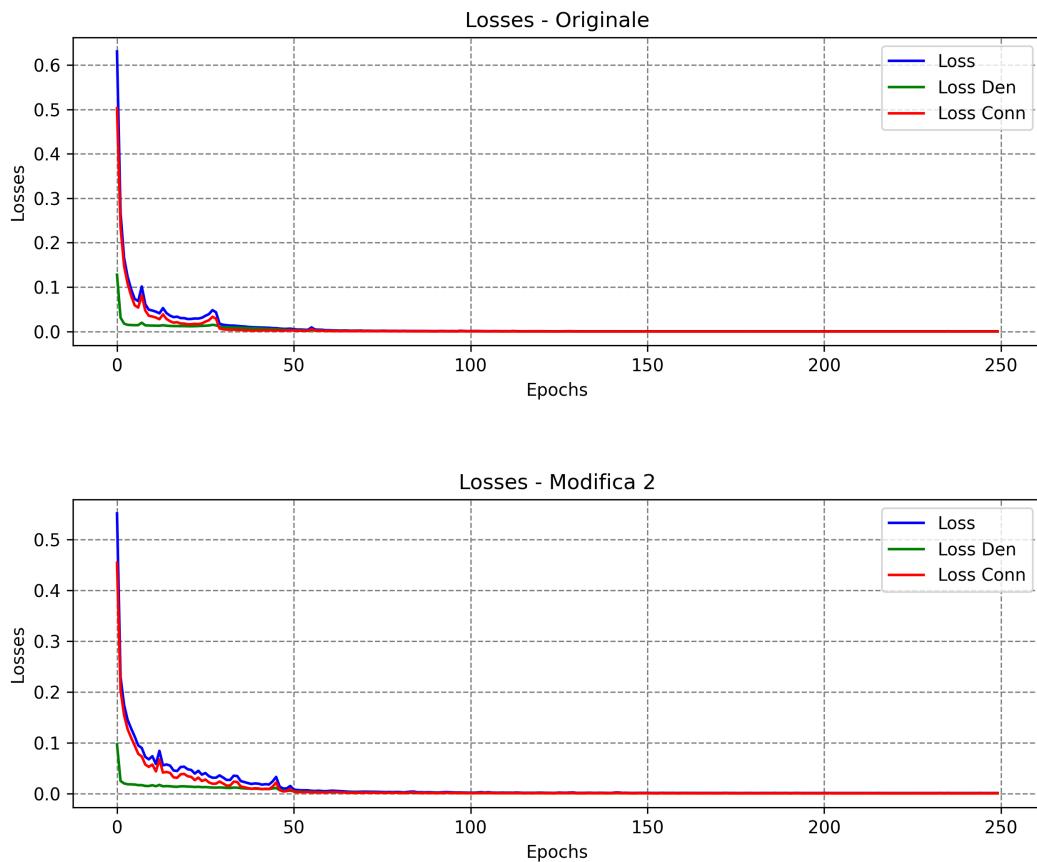


Figura 10: Loss di addestramento in seguito alla modifica 2

Tabella 5: Confronto dei risultati in seguito alla Modifica 2 vs baseline (paper)

Setting	Source	Hannah			IMDB		
		Fp	Fb	NMI	Fp	Fb	NMI
Unseen Test Data	Modifica 2	0.7261	0.6794	0.7996	0.7948	0.8065	0.9561
	Baseline (paper)	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	IMDB-Test-SameDist		
		Fp	Fb	NMI
Seen Test Data	Modifica 2	0.8213	0.8165	0.9524
	Baseline (paper)	0.779	0.819	0.949

Anche in questo caso, entrambi i grafici mostrano un comportamento simile: le perdite diminuiscono molto rapidamente nelle prime epoche (circa le prime 50) e poi si stabilizzano su valori molto bassi per il resto dell’addestramento. Osservando il grafico si può notare che la modifica apportata abbia una perdita complessiva leggermente inferiore o comunque molto simile a quella dell’architettura originale, una volta che il modello si stabilizza. Questo suggerisce che la modifica non compromette la capacità del modello di minimizzare la funzione di perdita generale. Un’altra cosa, visibile ad occhio nudo, sono le performance, decisamente superiori a quelle ottenute nella modifica precedente, abbiamo dei valori che non si discostano molto da quelli trovati. Ed inoltre la loss di densità, parte da valori più bassi, risultando anche leggermente più stabilizzata.

4.3 Modifica 3: modo alternativo di aggregare nel GAT-layer

Visto che la maggior parte degli addestramenti utilizza il GAT-layer, ed una caratteristica tipica dei livelli di attention è che hanno più teste, e queste teste hanno tutti la stessa importanza perché a questo punto non si lascia libero il modello di scegliere a quale testa dare più contributo nell’aggregazione? In questo caso, tale modifica introduce un cambiamento nel modo in cui gli output delle diverse teste di attenzione, all’interno del modulo GAT (Graph Attention Network) vengono combinati. Nella versione originale, quando si utilizza GAT, gli output delle K teste di attenzione vengono semplicemente mediati dopo aver aggiunto un bias. Questo significa che ogni testa contribuisce in modo uguale all’output finale. La modifica proposta invece cosa fa, introduce un parametro apprendibile (inizializzato a uno). L’output finale quindi, viene calcolato non più come media, ma come una somma ponderata degli output di ciascuna testa, dove i pesi sono quelli appresi e normalizzati dal modello.

```

1 out_weighted = torch.zeros_like(out[:, 0, :])
2 for i in range(out.shape[1]): # Iterazione sulle K teste
3     out_weighted += weights[i] * (out[:, i, :] + self.bias[i])
4 out = out_weighted

```

Perchè si è introdotta questa modifica? Perchè permettere al modello di apprendere i pesi per ciascuna testa di attenzione, fa sì che esso aumenti la sua capacità di catturare relazioni complesse nei dati. Il modello può dare maggiore importanza alle teste che hanno appreso rappresentazioni più rilevanti per il compito specifico, ad esempio in alcuni scenari, diverse teste di attenzione potrebbero concentrarsi su aspetti differenti del grafo (relazioni locali o globali). Nonostante questi possibili vantaggi c’è da tenere in considerazione che se le diverse

teste di attenzione apprendono informazioni molto simili, l'aggiunta di pesi potrebbe non portare a benefici significativi e quindi aggiungere complessità non necessaria.

4.3.1 Train ed Eval

L'unico addestramento che è possibile condurre senza esaurire le risorse disponibili è l'uso di INAT campionato. Di seguito riportati i risultati:

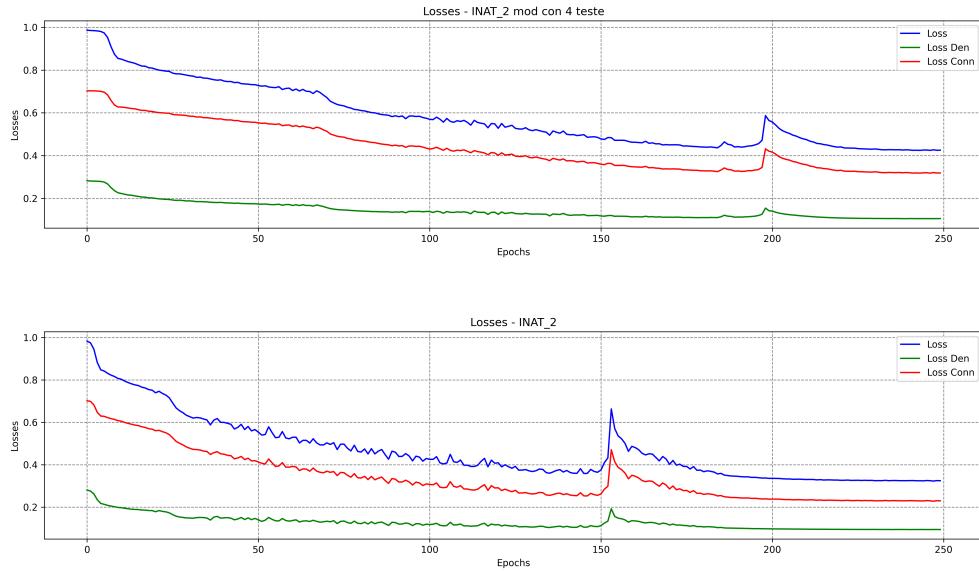


Figura 11: Loss di addestramento in seguito alla modifica 3 vs loss modello baseline

Setting	Source	iNat2018-Test		
		Fp	Fb	NMI
Unseen Test Data	Modifica 3	0.3001	0.3711	0.7572
	Baseline (paper)	0.294	0.352	0.764

Questo risultato non ci sorprende, nel grafico notiamo un leggero miglioramento verso la 150'epoca, ma i valori della loss di connessione si mantiene più alta per tutto l'addestramento. Dai risultati ottenuti si può concludere quindi che tale modifica al modello rende molto più lenta la convergenza, oltre al fatto che per abbassare tale loss sarebbe necessario effettuare più epoche. Questo è dovuto all'aggiunta dei pesi associati ai livelli che devono essere appresi. Ma a parità di epoche, il modello a cui viene apportata la modifica ottiene dei risultati molto fedeli alla baseline, se non leggermente migliori per *Fp* ed *Fb*.

4.4 Modifica 4: modifica del MLP usato per classificazione

Da tutte le modifiche precedenti notiamo che tra le due loss (densità e connessione), quella più alta è quella di connessione, allora proviamo a modificare l'elemento dell'architettura responsabile di ciò. Questa modifica architettonale, a differenza delle altre, si concentra sul classificatore delle connessioni, responsabile di stabilire se esiste o meno una connessione tra due nodi. Il classificatore originale (`classifier_conn_og`) e la versione modificata (`classifier_conn_4`) sono definiti come segue:

```

1 ##### OG
2 self.classifier_conn_og = nn.Sequential(
3     nn.PReLU(nhid_half),
4     nn.Linear(nhid_half, nhid_half),
5     nn.PReLU(nhid_half),
6     nn.Linear(nhid_half, 2),
7 )
8
9 ##### 4 MODIFICA
10 self.classifier_conn_4 = nn.Sequential(
11     nn.PReLU(nhid_half),
12     nn.Linear(nhid_half, nhid_half),
13     nn.BatchNorm1d(nhid_half),
14     nn.PReLU(nhid_half),
15     nn.Linear(nhid_half, nhid_half),
16     nn.BatchNorm1d(nhid_half),
17     nn.PReLU(nhid_half),
18     nn.Linear(nhid_half, 2),
19 )

```

Un normale MLP (Multi Layer Perceptron), tale modifica introduce i seguenti cambiamenti:

- *nn.BatchNorm1d(nhid_half)*: Vengono inseriti due strati di Batch Normalization. Utile per migliorare la stabilità dell’addestramento e previnire l’overfitting. Ma è stata aggiunta principalmente per affrontare una delle principali problematiche riscontrate durante l’addestramento, come l’esplosione del gradiente (gradient explosion);
- *nn.Linear(nhid_half, nhid_half)*: Nuovo livello lineare, aggiunto per aumentarne la profondità della rete;
- *nn.PReLU(nhid_half)*: la funzione di attivazione non lineare aggiunta subito dopo la Batch Normalization.

L’idea che sta alla base è molto semplice, ossia quella di aumentare la capacità del modello in modo tale che apprenda pattern più complessi dalle feature per stabilire la presenza o meno, di un collegamento tra i due nodi.

4.4.1 Train ed Eval

Anche in questo caso per valutare l’impatto della modifica precedentemente introdotta, si fa riferimento all’architettura senza che faccia d’uso della modifica. Prima di vedere il confronto, viene riportata la loss durante il training.

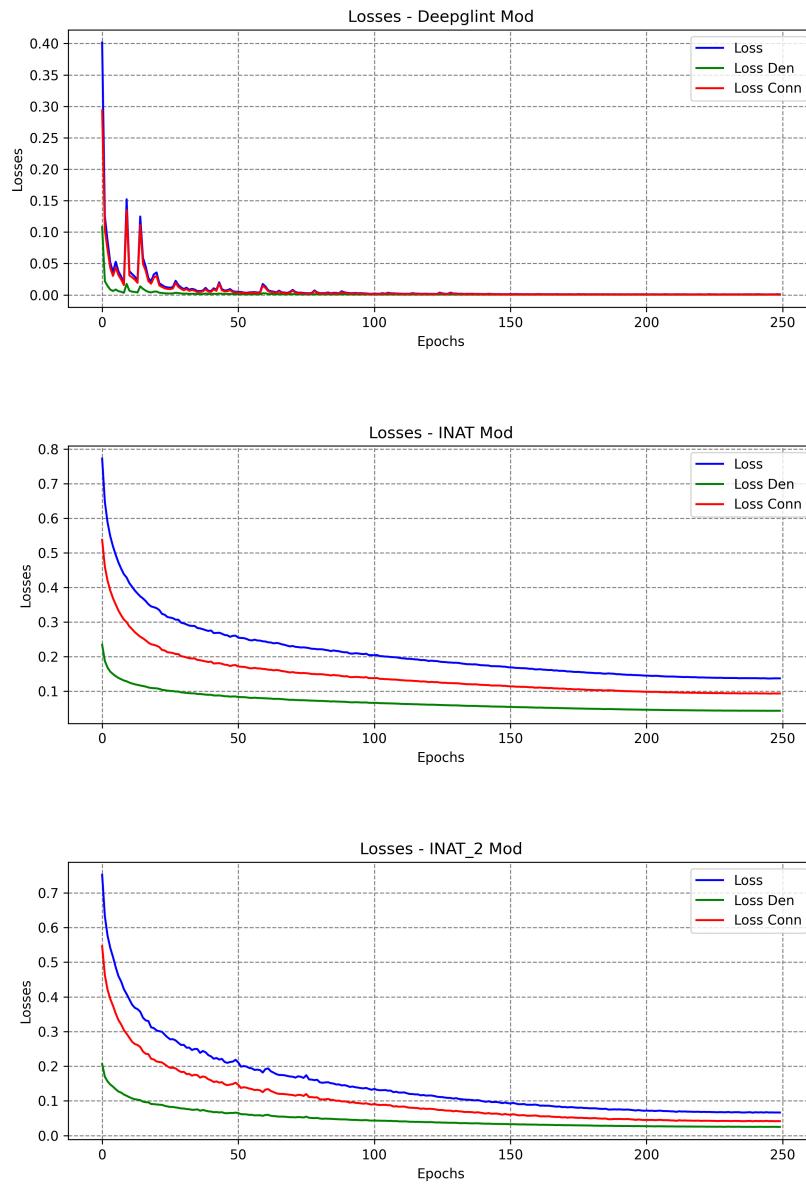


Figura 12: Loss di addestramento in seguito alla modifica 4

Tabella 6: Confronto dei risultati in seguito alla Modifica 4 vs baseline (paper)

Setting	Source	IMDB-Test-SameDist			iNat2018-Test		
		Fp	Fb	NMI	Fp	Fb	NMI
Seen Test Data	Modifica 4	0.7883	0.8122	0.9457	0.2952	0.3272	0.7673
	Paper	0.779	0.8198	0.94	0.330	0.350	0.774

Tabella 7: Confronto dei risultati in seguito alla Modifica 4 vs baseline (paper)

Setting	Source	Hannah			IMDB		
		Fp	Fb	NMI	Fp	Fb	NMI
Unseen Test Data	Modifica 4	0.7462	0.7183	0.8022	0.6810	0.7945	0.9485
	Paper	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	iNat2018-Test		
		Fp	Fb	NMI
Unseen Test Data	Modifica 4	0.3023	0.3176	0.7589
	Paper	0.294	0.352	0.764

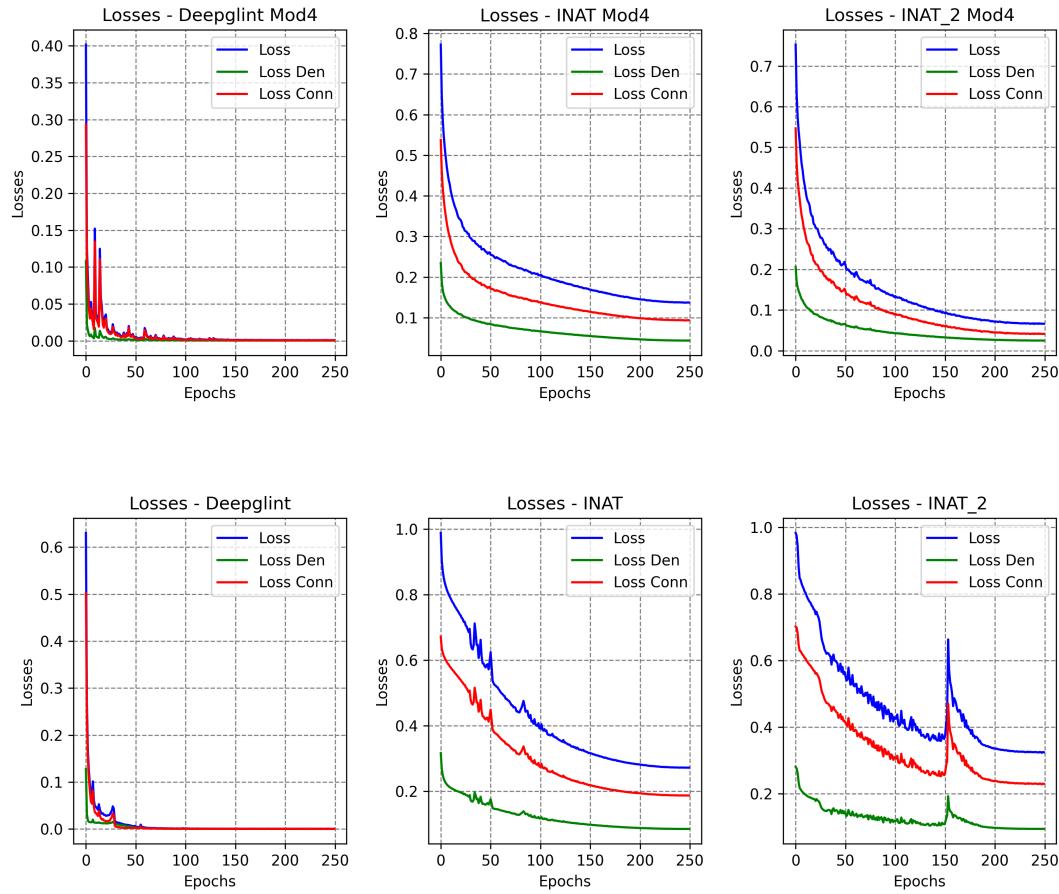


Figura 13: Loss di addestramento in seguito alla modifica 4 (sopra) vs baseline (sotto)

Dal grafico sul confronto possiamo notare che:

- La modifica introduce stabilità (riga superiore) rispetto alla baseline, tranne per il dataset Deepglint, in quanto si può osservare un innalzamento della loss di connessione durante le prime epoche;

- Nel grafico relativo ad IANT_2 si può osservare come manca il punto innalzato nella 150'esima epoca;
- La modifica fa ottenere dei valori di loss relativamente più bassi rispetto alla baseline e sono più regolarizzati, infatti non hanno un adamento altalenante;

Ma anche guardando le tabelle in cui vengono riportati gli score di valutazione, si ottengono delle performance che sono fedeli alla baseline. Questo non ci stupisce, aumentando la profondità del MLP ne stiamo aumentando la complessità e quindi riesce a catturare anche delle relazioni più complesse.

4.5 Modifica 5: nuovo modo di concatenare le features

Anche questa modifica si concentra sul classificatore introdotto precedentemente. A differenza delle modifiche precedenti però, questa non solo altera l'architettura interna del classificatore, ma anche il modo in cui le feature dei nodi sorgente e destinazione vengono combinate prima di essere passate al classificatore stesso.

```

1 def pred_conn(self, edges):
2     src_feat = self.src_mlp(edges.src["conv_features"])
3     dst_feat = self.dst_mlp(edges.dst["conv_features"])
4     ##### OG
5     #pred_conn = self.classifier_conn(src_feat + dst_feat)
6
7     ##### MODIFICA
8     combined_feat = torch.cat([src_feat, dst_feat], dim=-1) #
9     pred_conn = self.classifier_conn(combined_feat) #

```

Originariamente le feature del nodo sorgente (src_feat) e del nodo destinazione (dst_feat) venivano combinate tramite una somma elemento per elemento (src_feat + dst_feat). Il risultato era un vettore di input di dimensione nhid_half per il classificatore, in cui le informazioni dei due nodi sono aggregate. La modifica attuata invece cosa comporta? Le feature vengono ora combinate tramite concatenazione (torch.cat([src_feat, dst_feat], dim=-1)). Questo crea un vettore di input di dimensione 2 * nhid_half, fornendo al classificatore una rappresentazione più ricca e dettagliata, mantenendo separate le identità delle feature di origine e destinazione prima che il classificatore le elabori.

Il classificatore è stato anch'esso modificato, in cui è stata raddoppiata la dimensione dell'input ed è stato aggiunto il layer di normalizzazione, ma la struttura complessiva ricalca quella originale, quindi ci sono solamente 2 livelli. L'introduzione del livello di normalizzazione fa sì che anche in questo caso, il classificatore goda delle proprietà introdotte nella 4 modifica.

La concatenazione delle features dovrebbe portare a un classificatore più potente e accurato, questo perchè gli stiamo passando più informazioni. Questo, a sua volta, è fondamentale per migliorare la qualità del clustering gerarchico complessivo del modello. Nonostante questi vantaggio, il primo strato lineare nel classificatore ha un numero maggiore di parametri, letteralmente il doppio (input 2*nhid_half vs nhid_half nell'originale), aumentando la complessità del modello, il tempo di addestramento ed il consumo di memoria.

4.5.1 Train ed Eval

Per valutare l'impatto di questa modifica, anche in questo caso si fa un confronto tra le prestazioni del modello con la modifica appena introdotta, con il modello baseline che utilizzava la combinazione delle feature tramite somma.

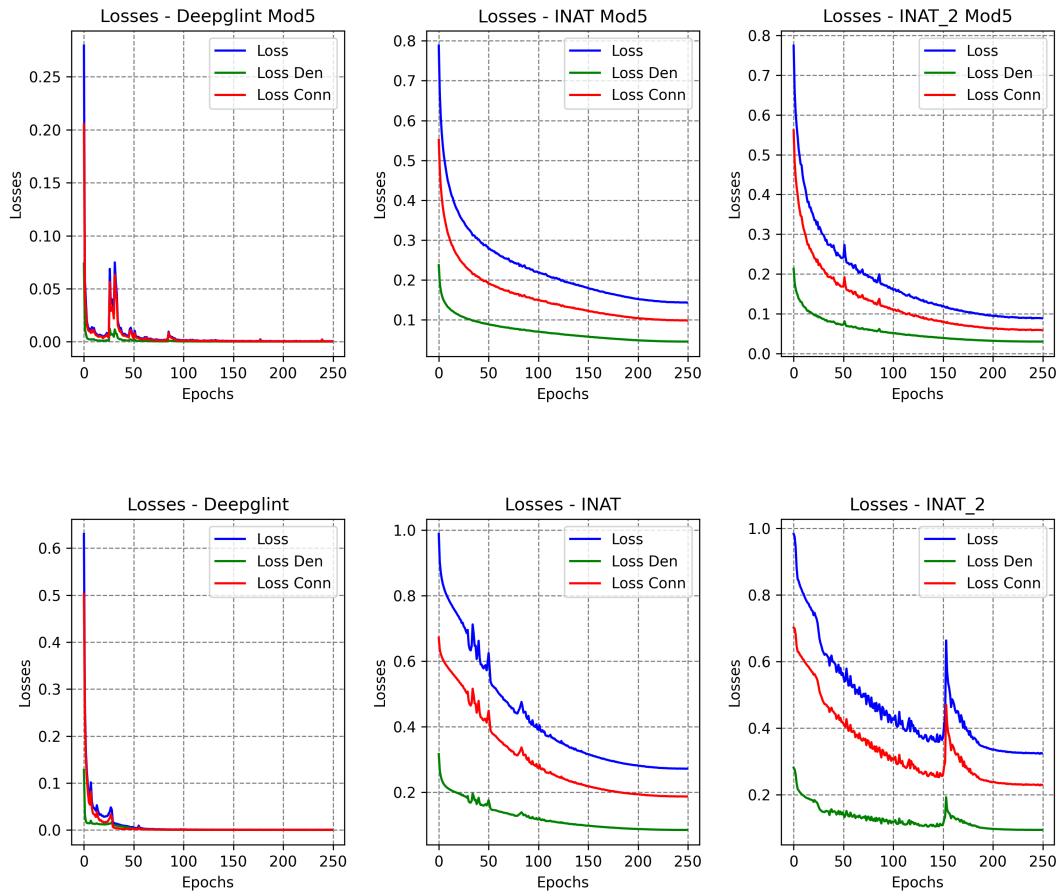


Figura 14: Loss di addestramento in seguito alla modifica 5 (sopra) vs baseline (sotto)

Rispetto alla baseline notiamo che la loss ha valori molto più bassi, si guardi come nel primo dataset la loss totale parte da $\sim 0,30$ mentre nella baseline la stessa loss parte da $\sim 0,65$, inoltre sono tutte più stabilizzate, basti guardare all'ultimo dataset. Come nella modifica apportata precedentemente, abbiamo un risultato simile, quindi potrebbe essere prevedibile che anche in fase di testing il modello si comporti nello stesso modo.

Tabella 8: Confronto dei risultati in seguito alla Modifica 5 vs baseline (paper)

Setting	Source	IMDB-Test-SameDist			iNat2018-Test		
		Fp	Fb	NMI	Fp	Fb	NMI
Seen Test Data	Modifica 5	0.7397	0.8287	0.9483	0.3116	0.3328	0.7617
	Paper	0.779	0.8198	0.94	0.330	0.350	0.774

Tabella 9: Confronto dei risultati in seguito alla Modifica 5 vs baseline (paper)

Setting	Source	Hannah			IMDB		
		Fp	Fb	NMI	Fp	Fb	NMI
Unseen Test Data	Modifica 5	0.7381	0.7174	0.8094	0.5613	0.8144	0.9513
	Paper	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	iNat2018-Test		
		Fp	Fb	NMI
Unseen Test Data	Modifica 5	0.2730	0.3274	0.7492
	Paper	0.294	0.352	0.764

4.6 Modifica 6: nuove features per il grafo al prossimo livello

Questa modifica si riflette, a differenza delle altre, sulla funzione *build_next_level*, utilizzata per la costruzione delle features del grafo al livello successivo. La funzione originale calcola le *cluster_features* (le featurese per il livello successivo) prendendo la media delle *global_features* di tutti i nodi che appartengono a un determinato cluster. Invece, la modifica proposta calcola le *cluster_features* utilizzando direttamente le *global_features* del "peak node", il nodo di picco, associato a quel cluster. Perchè proprio il peak node? Teoricamente è il nodo che rappresenta il centro o il punto di densità più alta di un cluster.

```

1 cluster_features = np.zeros((len(peaks), global_features.shape[1]))
2 for pi in range(len(peaks)):
3     peak_node_idx = global_peaks[global_label_to_peak[pi]]
4     cluster_features[global_label_to_peak[pi], :] = global_features[
    peak_node_idx, :]

```

Utilizzare solo le features del peak node potrebbe aiutare a ridurre il rumore derivante dall'aggregazione delle features di tutti i nodi presenti in un cluster, specialmente se alcuni nodi periferici sono meno rappresentativi. Potremmo avere che usare tale features potrebbe garantire che l'informazione più saliente del cluster sia propagata al livello successivo, questo perchè il peak node per definizione, è il più denso o rappresentativo del cluster. Infine, non andremmo a fare la media quindi avremmo anche un piccolo vantaggio computazionale. Però c'è da sottolineare che il principale svantaggio è la potenziale perdita di informazione. La media delle features di tutti i nodi di un cluster cattura una rappresentazione più olistica del cluster, tenendo conto della variabilità e delle caratteristiche di tutti i suoi membri. Usare solo il nodo di picco ignora le feature degli altri nodi, che potrebbero contenere informazioni utili per il clustering a livelli superiori. Ma non solo, se un nodo di picco non è effettivamente rappresentativo dell'intero cluster, questa modifica potrebbe portare a rappresentazioni meno accurate per il livello successivo.

4.6.1 Train ed Eval

Per valutare l'impatto della modifica appena presentata, verrà presentato un confronto tra le prestazioni del modello che utilizza l'aggregazione basata sul nodo di picco rispetto al modello baseline che impiega l'aggregazione tramite media per la costruzione delle feature del grafo al livello successivo.

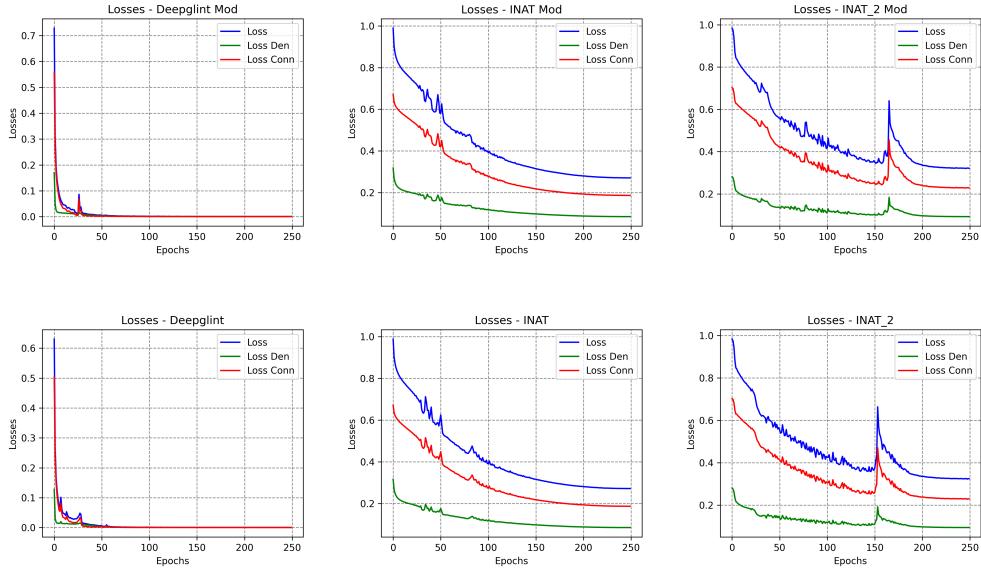


Figura 15: Loss di addestramento in seguito alla modifica 6 (sopra) vs baseline (sotto)

Da un attenta analisi ai grafici si può notare come la modifica introdotta non ha portato grossi cambiamenti, l'andamento delle loss di per se è molto simile se non per le prime epoche del dataset Deepglint nel quale per le primissime epoche abbiamo una loss di connessione che converge più rapidamente assumendo dei valori più alti. Questo è logicamente attribuibile al fatto che si perde di spiegazione attraverso l'uso solo delle features del peak node, specialmente nelle situazioni in cui si ha un cluster in cui è presente una zona periferica, tale contributo informativo si perde. Inoltre, dalle tabelle di seguito riportate, per alcuni dataset il modello peggiora la sua capacità di generalizzare penalizzando l' F_p .

Tabella 10: Confronto dei risultati in seguito alla Modifica 6 vs baseline (paper)

Setting	Source	IMDB-Test-SameDist			iNat2018-Test		
		F_p	F_b	NMI	F_p	F_b	NMI
Seen Test Data	Modifica 6	0.5550	0.8092	0.9443	0.3319	0.3576	0.7710
	Paper	0.779	0.8198	0.94	0.330	0.350	0.774

Tabella 11: Confronto dei risultati in seguito alla Modifica 6 vs baseline (paper)

Setting	Source	Hannah			IMDB		
		F_p	F_b	NMI	F_p	F_b	NMI
Unseen Test Data	Modifica 6	0.7579	0.7189	0.8147	0.4741	0.7962	0.9475
	Paper	0.741	0.706	0.810	0.717	0.810	0.953

Setting	Source	iNat2018-Test		
		Fp	Fb	NMI
Unseen Test Data	Modifica 6	0.2889	0.3357	0.7628
	Paper	0.294	0.352	0.764

4.7 Modifica 7: approccio ibrido

Poichè la modifica introdotto in precedenza si basa su una assunzione molto semplice, ossia che le features del peak node sono tra le più rilevanti, viene proposta questa modifica, successivamente motivata. Si introduce un approccio ibrido e adattivo per la costruzione delle feature del grafo al livello successivo, combinando le logiche della funzione originale e della modifica 6 precedentemente discussa. Tale funzione riceve un parametro aggiuntivo: size_threshold, utilizzato per stabilire quali dei due approcci utilizzare. Nello specifico, viene riportata la funzione:

```

1 ##### MODIFICA 7
2 def build_next_level_adaptive(
3     features, labels, peaks, global_features, global_pred_labels,
4     global_peaks, size_threshold=5
5 ):
6     ## CODICE UGUALE A PRIMA, NON REPUTO NECESSARIO RITRASCRIVERLO
7     for pi in range(len(peaks)):
8         cluster_indices = cluster_ind[pi]
9         cluster_size = len(cluster_indices)
10        peak_node_idx = global_peaks[global_label_to_peak[pi]]
11
12        if cluster_size <= size_threshold:
13            # Per i cluster piccoli usiamo solo il peak
14            cluster_features[global_label_to_peak[pi], :] =
15            global_features[peak_node_idx, :]
16        else:
17            # Per i cluster grandi usiamo la media
18            cluster_features[global_label_to_peak[pi], :] = np.mean(
19                global_features[cluster_indices, :], axis=0
20            )
21
22        features = features[peaks]
23        labels = labels[peaks]
24    return features, labels, cluster_features

```

Questo perchè logicamente pensando all'intera pipeline di lavoro di Hi-LANDER, quando si aumenta di livello di gerarchia tipicamente si tende ad avere sempre meno nodi (in quanto raggruppati in super-nodi), non si potrebbe provare ad utilizzare le features del peak node anche nei cluster di grandi dimensioni? I vantaggi sono facilmente intuibili, con questo approccio si godrebbe dei benefici visti con la modifica introdotta precedentemente. Però c'è da sottolineare che la scelta del size_threshold diventa un iperparametro cruciale per ottenere le migliori prestazioni. Quindi un size_threshold non ottimale potrebbe vanificare i benefici attesi, oltre al fatto che noi non possediamo apriori una chiara comprensione delle dimensioni tipiche dei cluster nel dataset.

4.7.1 Train ed Eval

Per il training ed il test, viene utilizzato un unico dataset, ossia INAT_2018, in quanto il più piccolo tra tutti e ciò ci permette di poter definire anche un analisi anche sul parametro

size_threshold.

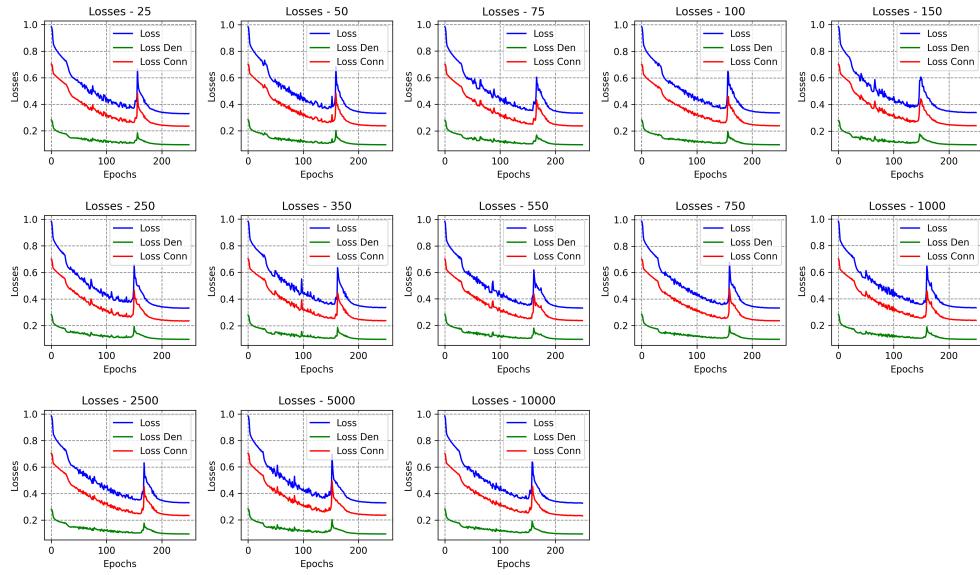


Figura 16: Loss al variare dell’iperparametro *size_threshold*

Osservando attentamente alle loss notiamo che sono molto, ma molto simili tra loro, penseremmo identiche, viene riportato per completezza un grafico in cui abbiamo solo la loss finale (costituita quindi dalla somma delle loss di densità e connessione).

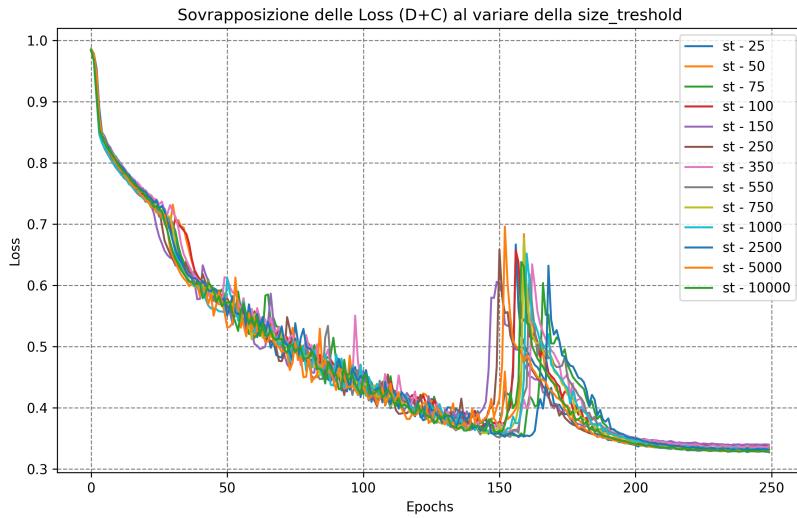
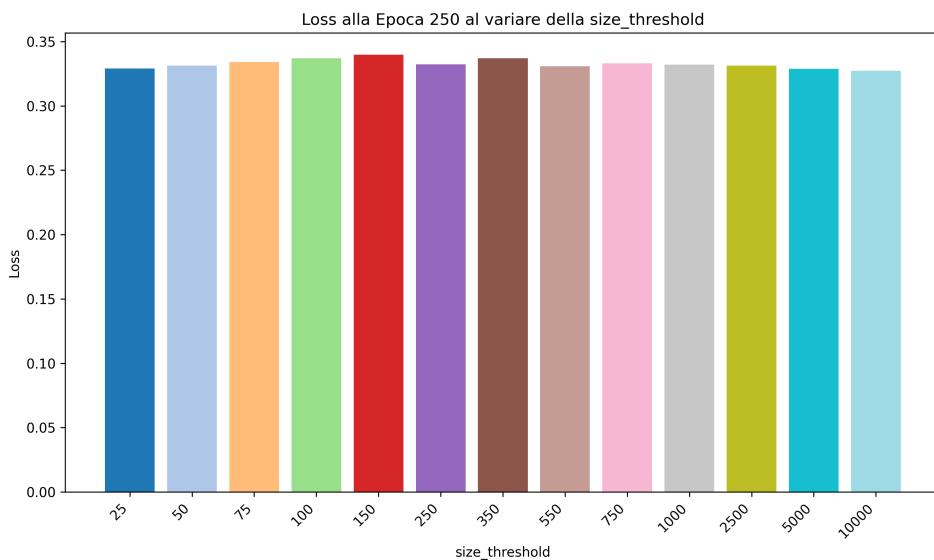


Figura 17: Loss totale al variare dell’iperparametro *size_threshold*

Questo grafico è molto confusionario, e non si può trarre nessuna conclusione da esso, l’unica cosa intuibile è che seppur molto simili le loss, differiscono sempre l’una dall’altra.



Intuitivamente ci aspetteremmo che il modello addestrato con *size_threshold* pari a 250 e 550 abbiano i valori migliori. Per determinare quale è la situazione migliore al variare dell'iperparametro, consideriamo la fase di test:

Tabella 12: Risultati su iNat2018-Test al variare di size

Size	Fp	Fb	NMI
25	0.2973	0.3417	0.7659
50	0.3055	0.3588	0.7658
75	0.3154	0.3497	0.7659
100	0.2910	0.3430	0.7654
150	0.3118	0.3525	0.7660
250	0.3130	0.3605	0.7667
350	0.2968	0.3372	0.7647
550	0.3190	0.3588	0.7663
750	0.3091	0.3450	0.7653
1000	0.2999	0.3438	0.7649
2500	0.3137	0.3438	0.7639
5000	0.2912	0.3437	0.7657
10000	0.3015	0.3585	0.7639

Con dei valori di *size_threshold* pari a 250 ed 550 si ottengono le migliori performance complessive. Chiaramente è un guadagno "ristretto", e questo avere loss e valori delle metriche pressocchè simili al variare di tale iperparametro è giustificato da come opera l'architettura stessa. Questo metodo ha impatto specialmente nei primi livelli della gerarchia di clustering, dove ci sono pochi/assenza totale di "super nodi" e clustering, salendo di livello i numeri di nodi presenti nei cluster tende a diminuire in quanto aggregati nei super nodi e di conseguenza si prende il peak nodo. Quando il dataset è grande, non avrebbe senso considerare solo il peak-node in quanto non sempre ci fornisce una visuale globale dell'intero cluster. Si basta pensare ad un cluster in cui c'è una zona periferica, tale zona usando solo il peak node se ne perderebbe il contenuto informativo.

Tabella 13: Risultati su iNat2018-Test best iperparametri vs baseline

Size	Fp	Fb	NMI
250	0.3130	0.3605	0.7667
550	0.3190	0.3588	0.7663
baseline	0.294	0.352	0.764

5 Conclusioni

Il presente elaborato ha esplorato e approfondito il funzionamento di Hi-LANDER, un modello che fa uso di Graph Neural Network per il clustering di immagini.

Un aspetto centrale di questa analisi ha riguardato lo studio delle modifiche architetturali proposte, in particolare per quanto concerne le strategie di aggregazione delle feature a diversi livelli gerarchici. In particolare, l'uso della densità dei nodi nel processo di message passing, e una concatenazione più espressiva delle feature nei classificatori, hanno mostrato performance competitive rispetto alla baseline, suggerendo che la qualità delle rappresentazioni apprese può essere ulteriormente migliorata con accorgimenti mirati.

Tuttavia, alcune modifiche, come l'introduzione dei pesi appresi per le teste nel GAT-layer o l'utilizzo esclusivo delle feature del nodo di picco, hanno evidenziato come l'equilibrio tra complessità del modello e capacità di generalizzazione debba essere attentamente mantenuto. Questo sottolinea l'importanza di testare ogni variazione in contesti realistici e su più dataset, specialmente in scenari open-set.

L'architettura analizzata rappresenta un progresso significativo rispetto agli approcci precedenti, tant'è che è stata utilizzata in tante applicazioni pratiche, come nel paper denominato "Cross-Camera Data Association via GNN for Supervised Graph Clustering" oppure come "A Meta-learning based Graph-Hierarchical Clustering Method for Single Cell RNA-Seq Data", progetti in cui si utilizza l'architettura Hi-Lander.

In definitiva, questo elaborato ha non solo chiarito i meccanismi interni e le potenziali modifiche architetturali di Hi-LANDER, ma ha anche aperto la strada a future ricerche. Queste potrebbero concentrarsi sull'implementazione di meccanismi adattivi per la gestione dei cluster di dimensioni variabili, sull'esplorazione di architetture ibride che combinino i punti di forza delle diverse modifiche proposte, e sull'applicazione del modello a nuovi domini di dati per valutarne ulteriormente la generalizzabilità e l'impatto. Il lavoro svolto ribadisce il valore delle GNN come strumento potente per l'analisi e l'organizzazione di dati complessi, offrendo prospettive molto promettenti, magari adottando anche tecniche più avanzate come i Graph-Transformer.