UNIVERSITÀ
degli STUDI
di CATANIA

CORSO DI LAUREA MAGISTRALE (LM-18) IN Informatica

Un modello basato su Graph Convolutional Network per la stratificazione di Pazienti con patologie non trasmissibili

TESI DI LAUREA MAGISTRALE

Relatore
Chiar.mo prof. Alfredo Pulvirenti

Laureando
dott. Francesco Grasso

Luglio 25, 2024

” Ringrazio il mio relatore Prof. Alfredo Pulvirenti per la supervisione di questa tesi. Il suo contributo ha permesso di portare a termine il lavoro e di ampliare le mie conoscenze nell'ambito di studio”.

” Un ringraziamento speciale a Simona, il suo costante supporto e la sua positività sono stati preziosi, soprattutto nei momenti più impegnativi di questo lavoro”.

” Un ringraziamento alla mia famiglia per la loro pazienza e comprensione durante questo percorso. Il loro atteggiamento privo di pressioni mi ha permesso di affrontare questo traguardo con serenità”.

” Infine, un profondo ringraziamento a me stesso”.

Sommario

Il cancro al seno è la forma tumorale più comune tra le donne. Si tratta di una malattia molto eterogenea, che può presentarsi in diverse forme cliniche, si può originare da vari tessuti della mammella, come il tessuto adiposo o il tessuto denso, e distinguersi in forme invasive o non invasive.
Secondo i dati più recenti dell'Organizzazione Mondiale della Sanità, nel 2023 si sono registrati 2,3 milioni di nuovi casi di cancro al seno nelle donne, con ben 670.000 decessi. Riuscire a prevedere precocemente la prognosi è cruciale per impostare una terapia mirata ed efficace. Calcolare accuratamente le probabilità di sopravvivenza rappresenta infatti una sfida fondamentale nell'analisi della sopravvivenza censurata, che studia se e quando un determinato evento (come il decesso del paziente) avverrà in un certo periodo di tempo.
Questo studio propone un nuovo approccio scalabile per l'apprendimento semi-supervisionato su dati strutturati a grafo, basato su un'efficiente variante delle reti neurali convoluzionali che opera direttamente sui grafi. La scelta di una struttura convoluzionale deriva da un'approssimazione localizzata delle convoluzioni spettrali dei grafi. Questo modello scala linearmente rispetto al numero di archi del grafo e impara rappresentazioni dello strato nascosto che codificano sia la struttura locale del grafo, sia le caratteristiche dei nodi.
I risultati finali dimostrano che questo innovativo approccio offre prestazioni molto promettenti, evidenziandone l'efficacia e le potenzialità nell'elaborazione di dati strutturati a grafo in ambito sanitario.

Elenco delle figure

Indice

1 Introduzione

Il cancro al seno rappresenta la forma tumorale più comune tra le donne, con un bisogno crescente di algoritmi affidabili per prevederne la prognosi.
Studi precedenti si sono concentrati sull'utilizzo di dati di clinici per costruire sistemi predittivi con risultati promettenti, tuttavia i recenti progressi nella disponibilità di set di dati sul cancro, come quelli sull'espressione genica, hanno motivato la creazione di nuovi modelli, evidenziato che c'è ancora ampio margine di miglioramento in questo campo.

In questo contesto, il presente studio propone un modello basato su una rete convoluzionale a grafo (GCN) che incorpora dati clinici provenienti dal database METABRIC, disponibile pubblicamente. Utilizzando i grafi per estrarre informazioni strutturali, il modello mira a classificare le pazienti con cancro al seno in base alla loro prognosi, distinguendo tra sopravvivenza a breve e a lungo termine.

Il modello sfrutta tecniche avanzate di rappresentazione e convoluzione del grafo per costruire una matrice di adiacenza normalizzata, rappresentante la rete di interazioni tra geni. Per ogni paziente e gene, viene costruita una distribuzione normale basata sui valori di espressione genica rilevati, campionando ripetutamente per ottenere una matrice tridimensionale.
Successivamente, un autoencoder basato su GCN riduce le dimensioni di queste matrici, creando una nuova rete di interazioni tra pazienti.

Questa rete è fondamentale per il successivo passo di classificazione, dove le pazienti vengono categorizzate in base alla loro sopravvivenza a breve o lungo termine. Il modello così costruito è poi valutato attraverso una serie di metriche di performance per garantire la sua accuratezza e affidabilità.

Nei capitoli successivi, verranno dettagliati tutti i passaggi implementativi del modello, mostrando i risultati ottenuti in ogni fase del processo. Dalla preparazione dei dati alla costruzione del grafo, dall'addestramento del modello alla valutazione delle prestazioni, ogni fase sarà analizzata per evidenziare i punti di forza e le possibili aree di miglioramento.

2 GCN

2.1 Concetti introduttivi

I grafi sono strutture dati cardinali in ambito scientifico, in quanto descrivono in maniera chiara e intuitiva l'interazione tra le entità in sistemi reali, più o meno complessi, come illustrato nella Figura 2.1.

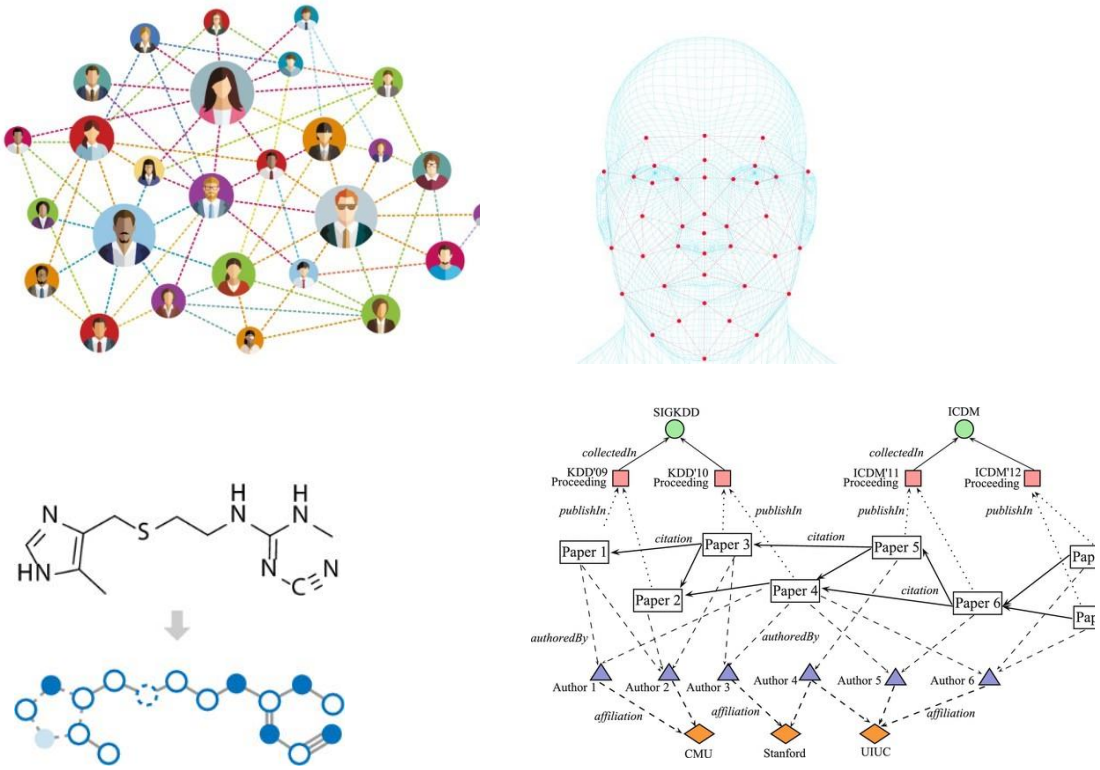


Figura 2.1: Esempi di Applicazione sui grafi

In genere, i grafi o le reti si suddividono in due categorie principali:

Natural graphs - strutture che rappresentano connessioni concrete tra elementi, basate su interazioni fisiche esistenti

- Social Networks
- Communication Networks
- Search Engine Algorithms

Information graphs - strutture che modellano connessioni tra entità, come la somiglianza o l'interazione potenziale.

- Web Page Linkage Graphs
- Knowledge graphs
- Recommendation Systems

Di conseguenza la costruzione del grafo, più precisamente i nodi e le interconnessioni tra loro, sono solitamente determinati in base a una comprensione dettagliata del problema che si sta affrontando e in alcuni casi, quando i dati astratti o potenziali, queste connessioni possono derivare anche da intuizioni piuttosto che da dati concreti.
Potrebbero rappresentare geni collegati per formare reti genetiche, pagine web connesse tramite link ipertestuali, o aeroporti interconnessi da rotte aeree, e molto altro.

La versatilità dei grafi come tipo di dato astratto porta spesso a sfide nell'estrazione delle informazioni pertinenti. D'altra parte, le tecniche di apprendimento sui grafi sono state sviluppate con l'obiettivo di creare modelli di conoscenza per contesti non lineari, sfruttando sia le caratteristiche dei singoli nodi che la topologia della rete definita dalle loro connessioni. Una volta stabilito il modello, può essere applicato efficacemente per risolvere una vasta gamma di problemi, come:

Node prediction - Predire la potenziale presenza o assenza di un nodo (fraud/fake detection in social network e articoli scientifici [1 - 11]).

Node Classification: assegnare automaticamente una categoria o un'etichetta a ciascun nodo in un grafo, utilizzando non solo gli attributi specifici del nodo ma anche le sue connessioni con altri nodi. Questo può essere applicato in contesti di apprendimento semi-supervisionato, dove solo una parte dei nodi ha etichette note, o in sistemi dinamici dove le proprietà dei nodi e le loro relazioni possono cambiare nel tempo.

Edge prediction: Predire la potenziale formazione o assenza di un arco tra due nodi in un grafo, utilizzando attributi e relazioni esistenti tra i nodi.

Graph embedding - Rappresentare graficamente i nodi di un grafo in uno spazio vettoriale continuo, preservando le relazioni e le caratteristiche del grafo originale. (embedding e predizione di fenotipi tessutali da profili molecolari spaziali) [12]

Node/network similarity: Valutare il grado di similitudine tra due nodi o due grafi.

2.2 I grafi

Un grafo è una struttura dati che rappresenta un insieme di oggetti (nodi o vertici) e le relazioni tra questi oggetti (archi o spigoli).
La distinzione tra network (o rete) e grafo è sottile e spesso i termini vengono usati in modo intercambiabile, ma ci sono alcune sfumature che possiamo considerare.
Dal punto di vista tecnico, il termine "rete" è comunemente impiegato per designare entità reali e concrete, come ad esempio Social Network oppure Reti neurali biologiche, mentre con il termine *grafo* (*graph*) ne indichiamo la rappresentazione astratta e matematica, ovvero il modello matematico che utilizziamo per rappresentare e analizzare quella rete.

Come evidenziato nella Figura 2.2, le caratteristiche degli archi definiscono la tipologia di grafo. Il grafo si classifica come orientato quando le connessioni tra i nodi sono direzionate, mentre se le connessioni non sono direzionali il grafo è considerato non orientato.
Un grafo in cui non esistono cicli tra i nodi è chiamato aciclico, abbreviato anche con l'acronimo DAG (*Directed Acyclic Graph*).

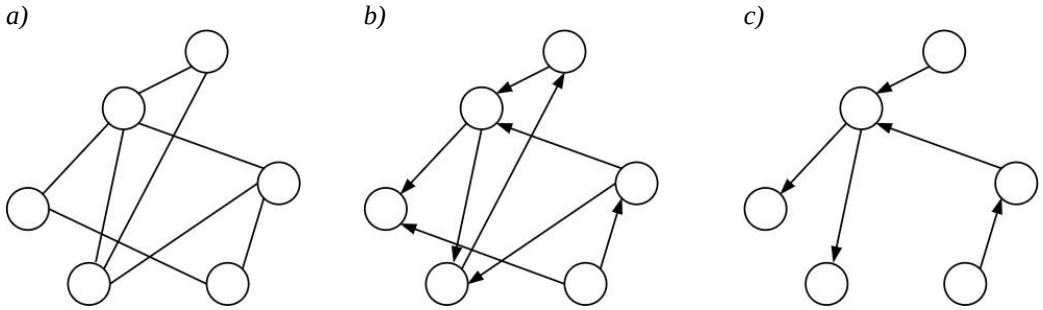


Figura 2.2: Esempi di grafo: (a) Non orientato, (b) Orientato, (c) Aciclico

Un grafo non orientato in cui vi è un unico cammino tra i nodi è chiamato *albero*.
Il grafo è *connesso*, se per qualsiasi coppia di nodi esiste almeno un percorso che li collega, *disconnesso* se è costituito da due o più sotto grafi distinti e non collegati tra loro.
Si definisce *grado* k_i del nodo *i* in un grafo non orientato, il numero di rami che convergono nel nodo *i*. Il numero medio di connessioni per nodo, $\bar{k}$ vale:

$$\bar{k} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{k_i} = \frac{2|\mathrm{E}|}{\mathrm{N}} \tag{2.1}$$

Per grafi orientati si deve invece fare distinzione tra grado di rami entranti nel nodo (*in-degree*) k_i^{in}, e uscenti dal nodo (*out-degree*) k_i^{out}. Ipotizzando un ugual numero di connessioni entranti e uscenti, il valore medio vale:

$$\bar{k}^{in} = \bar{k}^{out} = \frac{|E|}{N} \tag{2.2}$$

Un grafo si dice *completo* quando ogni coppia di nodi distinti è collegata da un arco, rendendo il grafo completamente connesso, in questo scenario il numero di rami si calcola:

$$|E| = \binom{N}{2} = \frac{N(N-1)}{2} \tag{2.3}$$

ed il grado medio si ottiene con la seguente formula:

$$\bar{k} = |N| - 1 \tag{2.4}$$

2.2.1 Rappresentazione delle connessioni

La struttura delle connessioni che è stata descritta precedentemente, viene rappresentata nella teoria dei grafi con una particolare matrice detta *Matrice di Adiacenza*:

$$A = \begin{vmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & \ddots & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{vmatrix} \tag{2.5}$$

La matrice di adiacenza è definita come una matrice binaria quadrata di dimensioni $N \times N$, il cui generico elemento A_{ij} descrive la connessione (orientata) tra il nodo *i* ed il nodo *j*. La presenza di un ramo semplice è indicata con un termine $A_{ij} = 1$; se il ramo è pesato da un valore reale W_{ij}, $A_{ij} = W_{ij}$, l'assenza di connessione è indicata da $A_{ij} = 0$.

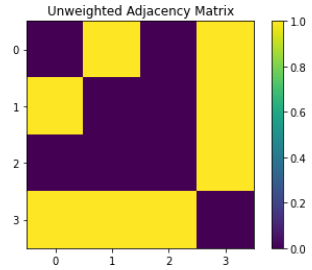
Nel caso di grafi non orientati la matrice è simmetrica poiché per ogni connessione *ij* esiste anche la connessione opposta *ji.* Nei grafi orientati invece la simmetria generalmente non vale.

Gli elementi sulla diagonale principale rappresentano la connessione di un nodo con sé stesso, che viene indicata con il termine *self loop*.

La matrice di adiacenza può risultare sconveniente dal punto di vista computazionale quando il grafo ha molti nodi e una bassa densità di connessioni, condizione che si verifica nella maggior parte dei grafi reali; in questo caso la matrice pur avendo dimensioni notevoli è *sparsa*, ovvero ha una densità di molto bassa di elementi non nulli.

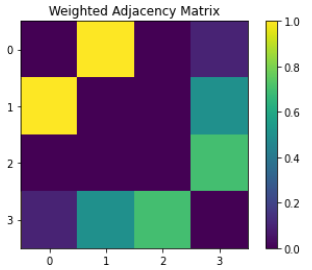
In alternativa alla rappresentazione matriciale, le connessioni di un grafo possono essere descritte con una lista di coordinate che indicano solamente i nodi connessi (*COO format*, o *Edge List*).

Le Matrici di Adiacenza possono essere rappresentate in forma grafica, con una griglia di quadrati disposti secondo le coordinate dell'elemento (i, j) e colorati in accordo alla scala di colori riportata a fianco, come nell'esempio seguente (Figura 2.3 – Figura 2.4):



$$A = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}$$

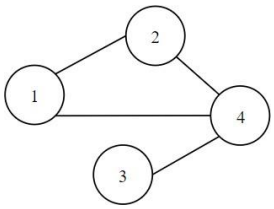
Figura 2.3: Matrice A binaria con rappresentazione grafica



$$A = \begin{vmatrix} 0.0 & 1.0 & 0.0 & 0.1 \\ 1.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.7 \\ 0.1 & 0.5 & 0.7 & 0.0 \end{vmatrix}$$

Figura 2.4: Matrice A pesata con rappresentazione grafica

Dalla matrice di adiacenza normalizzata si può ricavare il grado di ogni nodo, sommando gli elementi per riga o per colonna, come evidenziato nelle Figure 2.5 e 2.6.



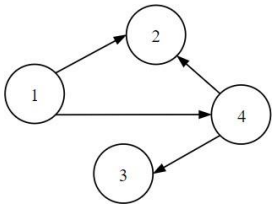
$$A = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}$$

$$A_{ij} = A_{ji}$$

$$k_i = \sum_{j=1}^{N} A_{ij} \tag{2.6}$$

$$k_j = \sum_{i=1}^{N} A_{ij} \tag{2.7}$$

Figura 2.5: Adjacency matrix e calcolo dei gradi in un grafo non diretto



$$A = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{vmatrix}$$

$$A_{ij} \neq A_{ji}$$

$$k_i^{in} = \sum_{j=1}^{N} A_{ij} \tag{2.8}$$

$$k_j^{out} = \sum_{i=1}^{N} A_{ij} \tag{2.9}$$

Figura 2.6: Adjacency matrix e calcolo dei gradi in un grafo diretto

L'informazione sul grado di tutti i nodi del grafo è contenuta nella *degree matrix D*, così definita:

$$D_{ij} = \begin{cases} k_i & per\ i = j \\ 0 & per\ i \neq j \end{cases} \tag{2.10}$$

La combinazione delle matrici *A* e *D* nella *Laplacian Matrix L* rappresenta un modo molto utilizzato per descrivere efficacemente le proprietà di un grafo, attraverso i cosiddetti metodi spettrali:

$$L = D - A \tag{2.11}$$

Ogni nodo *n* è univocamente identificato da un'etichetta *l* (*label*) ed eventualmente da una serie di attributi, valori reali o discreti che ne descrivono le proprietà.

2.3 Node embedding

Per poter applicare tecniche di machine learning su dati strutturati in forma di grafo è necessario rappresentare tali dati in spazi vettoriali di dimensione opportuna, che ci consentano di applicare gli algoritmi di classificazione. Come illustrato nella Figura 2.7, questo processo di trasformazione, noto come node embedding, mappa i nodi del grafo in uno spazio vettoriale.

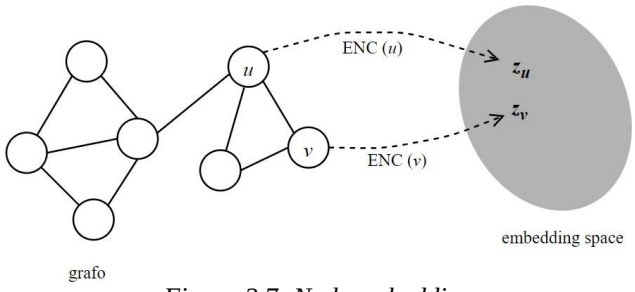


Figura 2.7: Node embedding

Ciò avviene tramite la funzione di *encoding* (ENC), che traduce ogni nodo in un punto in modo che le similitudini tra di essi vengano mantenute nello spazio di stato. Il secondo elemento fondamentale è il criterio di valutazione della "somiglianza" tra le due rappresentazioni; nello spazio vettoriale la somiglianza tra due vettori si può facilmente associare al coseno dell'angolo compreso:

$$cosine\ similarity(\boldsymbol{z_u}; \boldsymbol{z_v}) = \boldsymbol{z_u}^{\top}\boldsymbol{z_v} \tag{2.12}$$

Nel grafo invece il criterio di somiglianza è l'elemento chiave del processo di learning.
La forma più semplice di encoding può essere realizzata da una relazione lineare (*shallow encoding*):

$$ENC(v) = \boldsymbol{Zv} \tag{2.13}$$

$\boldsymbol{Z} \in \mathbb{R}^{d \times |N|}$ è una matrice in cui ogni colonna rappresenta l'embedding di un singolo nodo nello spazio di stato d-dimensionale, mentre $\boldsymbol{v} \in \mathbb{I}^{|N|}$ è un vettore composto da tutti elementi 0 ed un 1 in corrispondenza del nodo indicato *v*. Nel processo di learning la matrice *Z* viene continuamente adattata in modo da massimizzare la somiglianza di ogni coppia di nodi nei due domini.

2.4 - Graph embedding

I concetti introdotti per il node embedding possono essere estesi per rappresentare un intero grafo come un punto nello spazio di stato, così da introdurre il graph embedding, come rappresentato graficamente nella Figura 2.8:

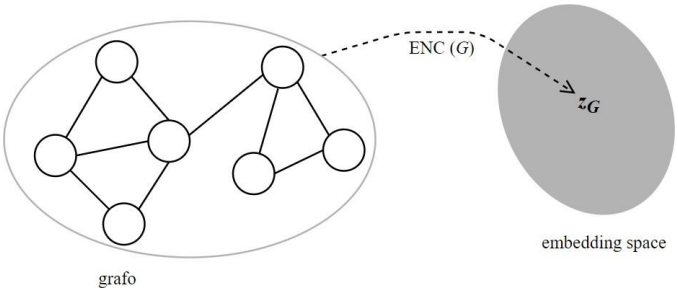


Figura 2.8: Graph embedding

Una semplice idea, introdotta in [15] ed evidenziata nella Figura 2.9, consiste nell'effettuare l'embedding di tutti i nodi nel dominio del grafo con l'introduzione di uno o più nodi virtuali:

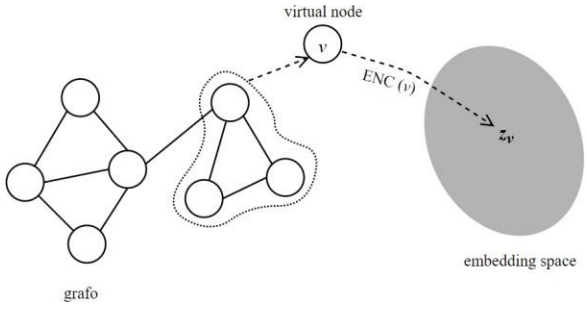


Figura 2.9: Gated Graph Sequence Neural Network [15]

2.5 Graph Neural Networks

Il semplice encoder mostrato nel capitolo precedente presenta alcune forti limitazioni dovute principalmente alla sua natura lineare; sostituire questa funzione con una composta da più strati (*deep*) di trasformazioni non lineari consente di aumentare la quantità di informazione appresa dalla struttura del grafo. Le Graph Neural Networks inoltre utilizzano un vettore di feature per ogni nodo, che lo indentifica più dettagliatamente rispetto alla sola etichetta considerata in precedenza.

Le Graph Neural Networks sono state inizialmente introdotte in [15] e [16] come una variante di reti neurali ricorrenti (RNN) sfruttando la possibilità di definire queste ultime con una struttura a grafo aciclico orientato (DAG). Il metodo utilizzato consiste nella ripetuta applicazione di operazioni di clustering e pooling per definire l'embedding di ogni nodo fino al raggiungimento di uno stato stazionario che viene poi utilizzato in una seconda rete di tipo feed-forward per la classificazione.

2.6 Graph Convolutional Networks

Per risolvere il problema delle reti neurali convoluzionali (CNN), che difficilmente riescono a elaborare dati relazionali di tipo non reticolare come i grafi, Kipf et al. hanno proposto una rete neurale convoluzionale a grafo (GCN). L'idea centrale della GCN è quella di eseguire una doppia fusione informativa per ogni nodo di un dato grafo durante ogni iterazione: la fusione delle informazioni sulla struttura del grafo e la fusione delle dimensioni delle caratteristiche dei nodi. Grazie alla caratteristica delle generalizzazioni combinatorie, il GCN è stato ampiamente utilizzato nei campi dell'analisi delle relazioni sociali nei social network, dell'elaborazione del linguaggio naturale e della biologia computazionale e genomica, ecc

Le Graph Convolutional Networks nascono dal tentativo di generalizzare i modelli consolidati di reti neurali come le Reti Convoluzionali (CNN) e le Reti Ricorrenti (RNN), applicandoli al dominio dei grafi. Il motivo per cui CNN e RNN sono estremamente efficaci è che operano su dati intrinsecamente descritti su domini regolari; osservando la Figura 2.10, si nota che le prime operano su immagini, che sono di fatto composte dalla disposizione dei pixel su una griglia ordinata, sulla quale si può definire la distanza, e di conseguenza il concetto di “vicinanza” tra due pixel, evitando di applicare una rete interamente connessa

alimentata da tutti i pixel che formano l'immagine. Nel caso delle RNN è il collegamento temporale a rappresentare la relazione d'ordine tra le sequenze di input.

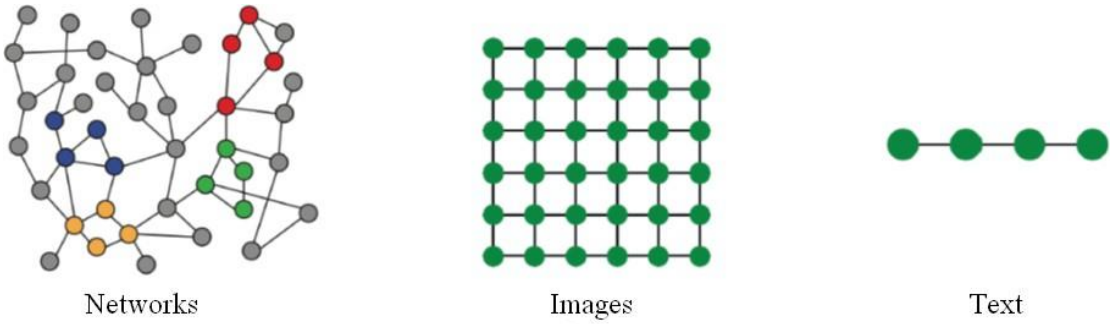


Figura 2.10: Strutture irregolari e regolari

Al contrario, l'applicazione della convoluzione sui nodi di un grafo è resa più difficile dalla mancanza del concetto di localizzazione spaziale.

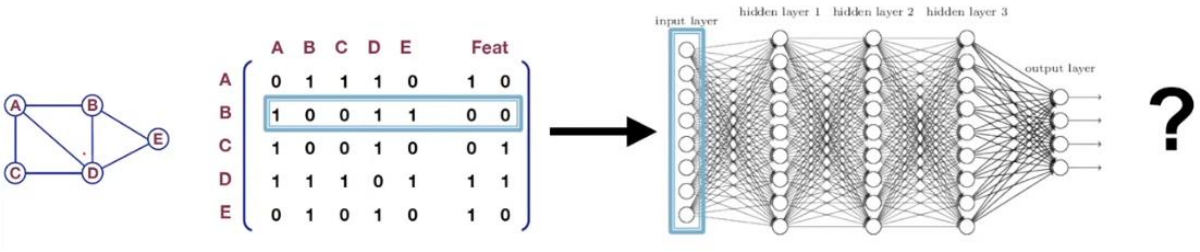


Figura 2.11: CNN, approccio naive

Un aspetto cruciale da analizzare è il motivo per cui non è possibile applicare direttamente le Reti Neurali Convoluzionali (CNN) standard a un grafo, più precisamente a una matrice di adiacenza, con le relative features, per rappresentare un grafo.
Esso, infatti, viene definito un approccio *naive* o ingenuo, come sintetizza efficacemente la Figura 2.11, di seguito ne mostro le motivazioni:

- Parametrizzazione non generalizzabile: Ovviamente questo approccio ha un certo numero di parametri, O(N) parametri, dove N sono i nostri nodi del grafo. Ciò significa che il numero di parametri cresce linearmente con la dimensione del grafo. In altre parole, non è generalizzabile a grafi differenti, questa è una grande limitazione perché non permette di utilizzare dati con dimensioni diverse sul modello su cui è stato addestrato. Un esempio banale, se addestriamo una rete su grafi con 100 nodi, non potremo utilizzarla su grafi con 101 o 99 nodi, quindi una grave mancanza di flessibilità.

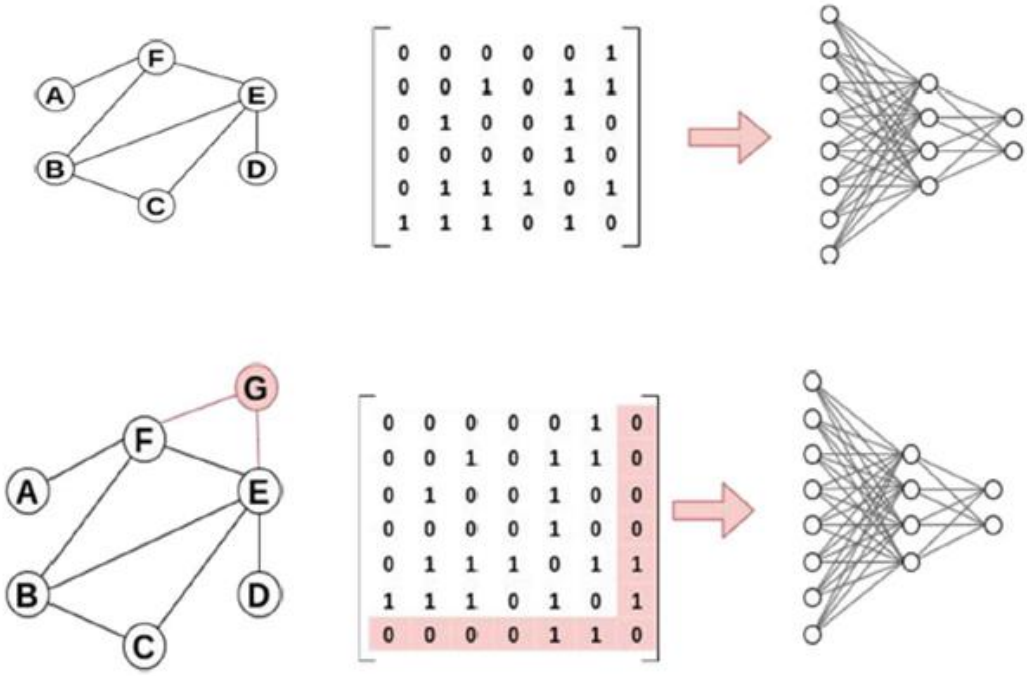


Figura 2.12: Non generalizzabilità delle CNN standard sui grafi.

Questa figura dimostra come l'aggiunta di un singolo nodo (G) al grafo originale modifica la dimensione della matrice di adiacenza e, di conseguenza, la struttura della rete neurale necessaria per processarla. La parte superiore mostra il grafo originale con la sua matrice di adiacenza e la corrispondente rete neurale, mentre la parte inferiore illustra come l'aggiunta di un nodo richieda una rete con una struttura diversa, evidenziando l'incapacità delle CNN standard di generalizzare a grafi di dimensioni variabili.

- Sensibilità all'ordinamento dei nodi: le CNN standard sono progettate per operare su dati con una struttura regolare e fissa, come le immagini. Questo avviene perché la topologia dei dati a forma di grafo è molto arbitraria e complicata; quindi, è computazionalmente difficile da eseguire sui dati dei grafi, il che implica l'assenza di localizzazione spaziale. Questo avviene perché lo stesso grafo può essere rappresentato da matrici di adiacenza diverse, semplicemente cambiando l'ordine in cui i nodi sono elencati. Le CNN sono sensibili a questi cambiamenti di ordinamento, il che può portare a risultati inconsistenti per lo stesso grafo rappresentato in modi diversi.

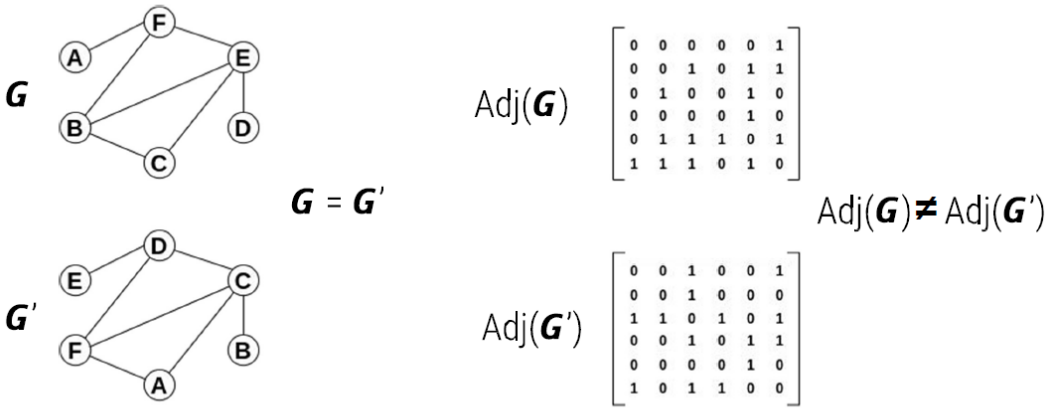


Figura 2.13: Invarianza strutturale ma non rappresentativa dei grafi.

Questa figura illustra chiaramente come lo stesso grafo (G = G') possa avere matrici di adiacenza diverse (Adj(G) ≠ Adj(G')) a causa del diverso ordinamento dei nodi. Ciò dimostra visivamente perché le CNN standard, che operano direttamente su queste matrici, possono produrre risultati inconsistenti per lo stesso grafo rappresentato in modi diversi.

- Perdita di informazioni topologiche: la rappresentazione di un grafo come matrice di adiacenza, sebbene matematicamente corretta, non preserva esplicitamente le informazioni topologiche in una forma facilmente processabile dalle CNN. Le CNN sono ottimizzate per catturare pattern locali e gerarchici in dati con struttura regolare. Nei grafi, la nozione di "località" è definita dalla connettività tra nodi, che non si traduce direttamente in vicinanza spaziale nella matrice di adiacenza.

- Sparsità e inefficienza: le matrici di adiacenza per grafi reali sono spesso estremamente sparse, con la maggior parte degli elementi uguali a zero. Questo porta a calcoli inefficienti nelle CNN standard, che non sono ottimizzate per gestire dati sparsi.

- Mancanza di invarianza alle permutazioni: come sottolineato da Bronstein et al. (2017) nel loro lavoro "Geometric Deep Learning: Going beyond Euclidean data", un requisito fondamentale per l'elaborazione dei grafi è l'invarianza alle permutazioni. Le CNN standard non soddisfano intrinsecamente questo requisito.

- Difficoltà nel catturare relazioni multi-hop: le CNN standard, operando localmente sulla matrice di adiacenza, faticano a catturare efficacemente le relazioni tra nodi distanti nel grafo. Questo limita la capacità del modello di comprendere strutture globali e pattern a lungo raggio nel grafo.

Inoltre, due matrici di adiacenza differenti possono rappresentare il medesimo grafo. L'aggiunta e la rimozione di archi necessiterebbe la creazione di un'altra rete convoluzionale specifica.
Per superare queste limitazioni, sono stati sviluppati approcci specifici per l'elaborazione dei grafi, come le Graph Convolutional Networks (GCN).

Le Graph Convolutional Networks si possono classificare in due macro-categorie sulla base del dominio in cui operano: *spectral-based* e *spatial-based.* Di seguito daremo i dettagli sui metodi spatial based che sono il focus di questa tesi.

La Figura 2.14 offre una visualizzazione dettagliata dei metodi *spatial-based*, definiti collocando il grafo in un dominio spaziale che consente di formulare una conoscenza della struttura di rete più localizzata al singolo nodo e ad un insieme di suoi "vicini", di conseguenza i pesi della rete possono essere condivisi tra più regioni della stessa che presentano strutture locali simili.

Il funzionamento può essere visto come la creazione, ad ogni strato, di una serie di nuovi grafi, disconnessi, che hanno la stessa topologia dell'originale, ma i cui nodi vengono iterativamente definiti sulla base del proprio stato e delle informazioni scambiate con l'insieme dei nodi vicini (Figura 2.15).

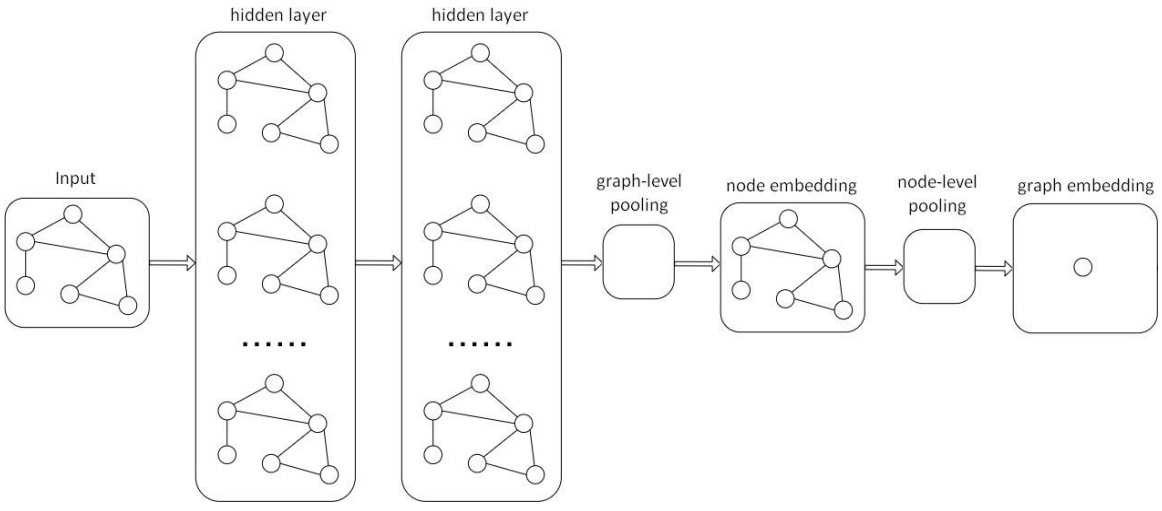


Figura 2.14: struttura GNN convoluzionale spatial-based

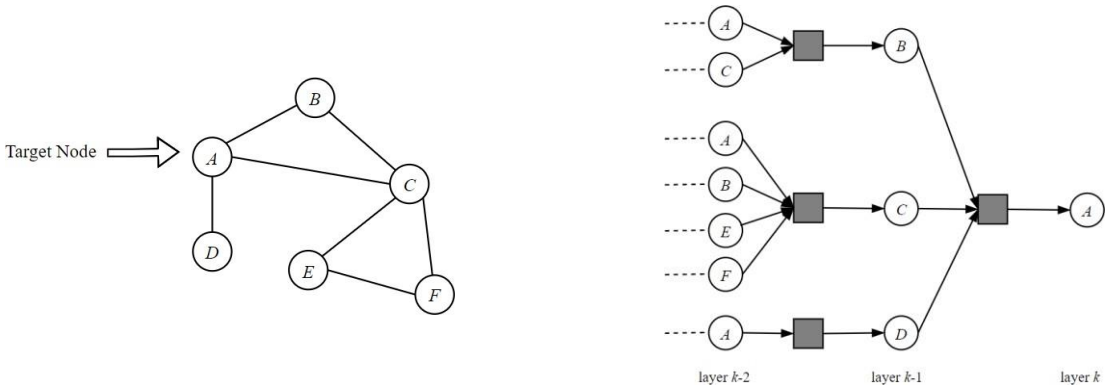


Figura 2.15: Meccanismo di node embedding basato su message passing

Le black box che elaborano le informazioni provenienti dallo strato precedente possono implementare una qualsiasi funzione, possono essere semplici somme, medie o pooling, o più genericamente reti neurali che contengono un operatore non lineare (σ), parametri addestrabili (matrici W e B) e si differenziano in base alla funzione di aggregazione degli input (AGG).

Anche in questo caso il grafo in input è descritto dalla matrice $\mathbf{H} \in \mathbb{R}^{d \times |N|}$ le cui colonne rappresentano le feature di ogni nodo. Al generico strato *k* le colonne di H vengono definite da:

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}_k \cdot AGG(\{\boldsymbol{h}_u^{k-1}, \forall\, u \in N(v)\}), \boldsymbol{B}_k \boldsymbol{h}_v^{k-1}) \tag{2.14}$$

Da queste considerazioni segue che non esiste un modello standard che definisce la convoluzione spatial-based. Una semplice implementazione può essere fatta componendo l'input come la media dei vettori provenienti dai nodi vicini:

$$\mathrm{a}\ \mathbf{h}_{\mathrm{v}}^{\mathrm{k}} = \sigma\left(\mathbf{W}_{\mathrm{k}} \cdot \sum_{\mathrm{u} \in \mathrm{N(v)}} \frac{\mathbf{h}_{\mathrm{u}}^{\mathrm{k}-1}}{|\mathrm{N(v)}|} + \mathbf{B}_{\mathrm{k}} \mathbf{h}_{\mathrm{u}}^{\mathrm{k}-1}\right) \tag{2.15}$$

2.6.1 Readout layer

Con il termine readout si intende l'insieme di operazioni che permettono di passare dalla rappresentazione dei nodi (per problemi di node classification) ad un singolo elemento che caratterizzi il grafo nel suo insieme (per problemi di graph classification).

Ancora una volta la mancanza di una struttura rende inefficienti le classiche operazioni di pooling utilizzate nelle CNN:

$$\boldsymbol{h}_G = \sum_{i=1}^{|N|} \boldsymbol{h}_i^L (sum) \tag{2.16}$$

$$\boldsymbol{h}_G = \frac{1}{|N|} \sum_{i=1}^{N} \boldsymbol{h}_i^L \, (mean\ pooling) \tag{2.17}$$

$$\boldsymbol{h}_G = \max\{\boldsymbol{h}_i^L, \forall i\} \, (\max pooling) \tag{2.18}$$

in cui $\mathbf{h}_G$ è la rappresentazione del grafo G, mentre $\mathbf{h}^L$ è la rappresentazione del nodo i all'ultimo strato L della rete.

In [17] viene proposto l'utilizzo di una distribuzione statistica dei valori assunti dai nodi, quantizzati su un numero finito di intervalli, per identificare il grafo.

Più frequentemente i valori dei nodi all'uscita della rete vengono utilizzati come input per un classificatore MLP [18]:

$$\mathbf{h}_G = \sigma([H^L] \cdot \Theta_{FC}) \tag{2.19}$$

in cui $[H^L] \in \mathbb{R}^{N \cdot f_L}$ è un vettore composto dalla concatenazione delle node feature allo strato finale, e Θ_{FC} è la matrice dei parametri addestrabili della rete MLP.

La Figura 2.16 offre una panoramica di un altro approccio comune si basa sul concetto di *hierarchical clustering* [19]: ad ogni livello della rete gruppi di nodi vengono raggruppati utilizzando vari algoritmi di clustering, per ridurre la dimensione della rappresentazione finale ad una unità che identifica l'intero grafo:

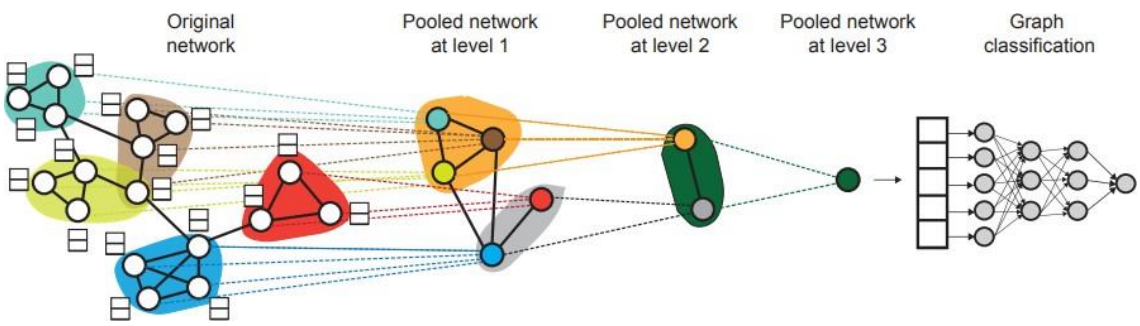


Figura 2.16: Hierarchical clustering

La tabella seguente riporta un elenco delle più note architetture Graph-Convolutional Neural Networks

Approach	Category	Inputs	Pooling	Readout	Time Complexity
GNN* (2009)	RecGNN	A, X, X^e	-	a dummy super node	$O(m)$
GraphESN (2010)	RecGNN	A, X	-	mean	$O(m)$
GGNN (2015)	RecGNN	A, X	-	attention sum	$O(m)$
SSE (2018)	RecGNN	A, X	-	-	-
Spectral CNN (2014)	Spectral-based ConvGNN	A, X	spectral clustering+max pooling	max	$O(n^3)$
Henaff et al. (2015)	Spectral-based ConvGNN	A, X	spectral clustering+max pooling		$O(n^3)$
ChebNet (2016)	Spectral-based ConvGNN	A, X	efficient pooling	sum	$O(m)$
GCN (2017)	Spectral-based ConvGNN	A, X	-	-	$O(m)$
CayleyNet (2017)	Spectral-based ConvGNN	A, X	mean/graclus pooling	-	$O(m)$
AGCN (2018)	Spectral-based ConvGNN	A, X	max pooling	sum	$O(n^2)$
DualGCN (2018)	Spectral-based ConvGNN	A, X	-	-	$O(m)$
NN4G (2009)	Spatial-based ConvGNN	A, X	-	sum/mean	$O(m)$
DCNN (2016)	Spatial-based ConvGNN	A, X	-	mean	$O(n^2)$
PATCHY-SAN (2016)	Spatial-based ConvGNN	A, X, X^e	-	sum	-
MPNN (2017)	Spatial-based ConvGNN	A, X, X^e	-	attention sum/set2set	$O(m)$
GraphSage (2017)	Spatial-based ConvGNN	A, X	-	-	-
GAT (2017)	Spatial-based ConvGNN	A, X	-	-	$O(m)$
MoNet (2017)	Spatial-based ConvGNN	A, X	-	-	$O(m)$
LGCN (2018)	Spatial-based ConvGNN	A, X	-	-	-
PGC-DGCNN (2018)	Spatial-based ConvGNN	A, X	sort pooling	attention sum	$O(n^3)$
CGMM (2018)	Spatial-based ConvGNN	A, X, X^e	-	sum	-
GAAN (2018)	Spatial-based ConvGNN	A, X	-	-	$O(m)$
FastGCN (2018)	Spatial-based ConvGNN	A, X	-	-	-
StoGCN (2018)	Spatial-based ConvGNN	A, X	-	-	-
Huang et al. (2018)	Spatial-based ConvGNN	A, X	-	-	-
DGCNN (2018)	Spatial-based ConvGNN	A, X	sort pooling	-	$O(m)$
DiffPool (2018)	Spatial-based ConvGNN	A, X	differential pooling	mean	$O(n^2)$
GeniePath (2019)	Spatial-based ConvGNN	A, X	-	-	$O(m)$
DGI (2019)	Spatial-based ConvGNN	A, X	-	-	$O(m)$
GIN (2019)	Spatial-based ConvGNN	A, X	-	sum	$O(m)$
ClusterGCN (2019)	Spatial-based ConvGNN	A, X	-	-	-

Tabella 2.1: Reti convGNN, tratto da "Comprehensive Survey on Graph Neural Networks" [20]

2.7 Attention Mechanism in Graph Neural Networks

Il concetto che sta alla base di ogni meccanismo di *attention* applicato alle reti neurali è quello di rendere il modello capace di apprendere quali tra gli elementi di input sono più rilevanti per il task che deve svolgere, e quali meno. Tipicamente ciò avviene modulando il flusso di informazioni attraverso un vettore di "attention weights" appresi dinamicamente.

Nel caso di dati strutturati in forma di grafo, le strutture in input possono avere dimensioni anche molto elevate, a causa della presenza di nodi e rami, associati a componenti di rumore piuttosto che componenti informative.

I meccanismi di attention in questo caso permettono di concentrare l'estrazione dell'informazione solo da un sottoinsieme del grafo.

Dato un insieme di nodi $\Gamma_V = \{v_0 \dots v_n\} \subset V$ non necessariamente composto da tutti i nodi in V ed un obbiettivo s, definiamo genericamente il meccanismo di attention come una funzione ϕ_s': $\Gamma_V \rightarrow [0; 1]$ che mappa ogni elemento in Γ_V in un punteggio di attention, e che soddisfa la condizione:

$$\sum_{i=0}^{|V_r|} \phi_s'^{(v_i)} = 1 \tag{2.20}$$

I meccanismi di attention si differenziano a seconda che il problema sia di node embedding o di graph embedding.

Nel caso di node embedding, l'obbiettivo s è un generico nodo v_j del grafo e Γ_V è l'insieme dei nodi vicini $N(v_j)$. Il meccanismo di attention viene applicato ad ogni nodo del grafo e definisce l' "importanza" di ciascun nodo in $N(v_j)$ nel creare il corretto embedding per v_j.

Uno dei primi lavori ad introdurre questo concetto è [21] il cui modello di rete viene appunto chiamato GAT (Graph Attention Network). In questo modello si definisce
$x_i^{(k)} = \{x_1^{(k)} \dots x_n^{(k)}\} \in \mathbb{R}^{f_k}$ il feature vector di input di ciascun nodo allo strato k;

attraverso una trasformazione lineare pesata da una matrice learnable $W \in \mathbb{R}^{f_{k+1} \times f_k}$
e una successiva funzione a si calcola il vettore di scores:

$$e_{ij} = a\left(W\, x_i^{(k)}, W\, x_j^{(k)}\right) \tag{2.21}$$

con $i \in V$ e $j \in N(i)$. Come si può osservare nella Figura 2.17, la funzione di attention a utilizzata è una rete feed-forward parametrizzata da un vettore $\boldsymbol{a} \in \mathbb{R}^{2f_{k+1}}$ e con funzione di attivazione *leaky ReLU*. Il vettore di scores viene quindi normalizzato per rendere i valori uniformi su tutti i nodi:

$$a_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N(i)} exp\,(e_{ik})} \tag{2.22}$$

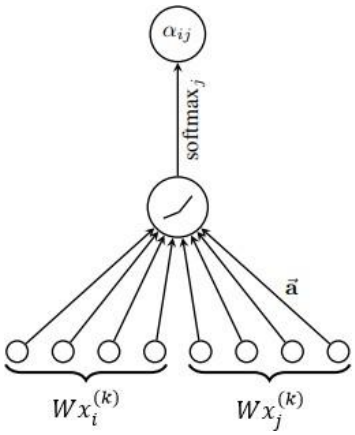


Figura 2.17: Calcolo del valore di attention in GAT

I valori di attention vengono quindi utilizzati per calcolare l'embedding dello strato successivo $x_I^{(k+1)} = \{x_1^{(k+1)} \dots\ x_n^{(k+1)}\} \in \mathbb{R}^{f_{k+1}}$, con una ulteriore funzione non lineare a parametri learnable ξ:

$$x_i^{(k+1)} = \xi\left(\sum_{j \in N(i)} \alpha_{ij} W\, x_j^{(k)}\right) \tag{2.23}$$

Nel caso di graph embedding, l'obbiettivo è quello di formare una funzione di attention che permetta di mappare l'intero grafo in un vettore unidimensionale attribuendo "importanza" differente alle diverse regioni che lo compongono.

Uno dei primi lavori proposti è "Graph Classification using Structural Attention" [22] che è basato su un meccanismo random-walk guidato da parametri di attention.

In [23] viene invece implementato un meccanismo di attention a livello di readout layer (ultimo strato della rete), dove al posto della semplice rete feed-forward $h_G = g(x^{(k)})$ viene utilizzata una struttura più complessa:

$$h_G = \xi\left(\sum_{i \in V} \alpha_i^{(k)} \cdot \tanh\left(g\left(x_i^{(k)}\right)\right)\right) \tag{2.24}$$

in cui analogamente a prima, ξ è una funzione non lineare e $\{\alpha_1{}^{(k)} \dots \alpha_n{}^{(k)}\}$ rappresenta il vettore di attention scores all'ultimo strato *k*, normalizzato tramite softmax:

$$a_1^{(k)} = softmax\left(a\left(x_i^{(k)}\right)\right) \tag{2.25}$$

In [24] il meccanismo di attention viene invece attuato a livello di pooling layer. Dopo aver calcolato i coefficienti di attention per tutti i nodi al livello *k* vengono selezionati i più "rilevanti" tramite una regola a soglia:

$$x_i^{(k+1)} = \begin{Bmatrix} a_i{}^{(k)}\, x_1^{(k)} & se\ > \bar{a} \\ 0 & altrimenti \end{Bmatrix} \tag{2.26}$$

Infine, in [25] viene calcolato il vettore di scores sulla base dell'intera struttura di rete, anziché considerare solo caratteristiche locali, utilizzando le matrici normalizzate $\widetilde{D}$ (*Degree matrix*) e $\tilde{A}$ (*Adjacency matrix*):

$$\alpha^{(k)} = \sigma\left(\widetilde{D}^{-1/2}\, \tilde{A} \widetilde{D}^{-1/2}\, X^{(k)} p\right) \tag{2.27}$$

in cui $X^{(k)}$ è la feature matrix allo strato k e $p \in \mathbb{R}^{fk}$ è il vettore dei parametri learnable.

2.8 Principal Component Analysis

La PCA (Analisi delle Componenti Principali) si è rivelata uno strumento fondamentale in questo studio, precedendo l'implementazione della GCN. Questa tecnica ha consentito una efficace visualizzazione della distribuzione dei dati originali.

L'applicazione di questo algoritmo di riduzione dimensionale ha portato alla luce importanti informazioni sulla struttura del dataset, evidenziando in particolare la presenza di uno sbilanciamento tra le classi. Tale scoperta ha fornito preziosi spunti per l'analisi successiva e l'ottimizzazione del processo di elaborazione dei dati.

Inoltre, l'implementazione della PCA in un algoritmo di machine learning offre diversi vantaggi significativi, che possono andare aldilà della visualizzazione di dati:

1. Riduzione della dimensionalità: La PCA consente di diminuire la complessità del modello, mitigando il rischio di overfitting e migliorando l'efficienza computazionale, soprattutto per la gestione di dataset di grandi dimensioni.
2. Riduzione del rumore: Concentrando l'informazione nelle componenti principali di maggior rilievo, la PCA può contribuire efficacemente al filtraggio del rumore nei dati.
3. Estrazione di caratteristiche: Rivela pattern latenti nei dati, generando nuove caratteristiche potenzialmente più informative.
4. Analisi delle anomalie: Facilita l'identificazione di outlier o punti anomali nel dataset.
5. Miglioramento della generalizzazione: Riducendo il rischio di overfitting, la PCA può contribuire allo sviluppo di modelli con maggiore capacità di generalizzazione su nuovi dati.

PCA sta per Principal Component Analysis, una tecnica utilizzata nell'analisi statistica e nell'apprendimento automatico per ridurre la dimensionalità dei dati. PCA viene utilizzata per trasformare i dati in modo che le nuove variabili (chiamate componenti principali) siano linearmente non correlate e ordinati per importanza decrescente in termini di varianza dei dati originali.

Sia $X \in R^{n \times p}$ la nostra matrice delle caratteristiche e sia

$$\Sigma = \frac{1}{N-1} \bar{X}^T \bar{X} \in R^{p \times p} \tag{2.28}$$

la matrice di Covarianza di X, dove $\bar{X}$ è la matrice dei dati centrata, ovvero
$\bar{X} = [x_1 - \bar{x}_1, ..., xp - \bar{x}p]$ (tutte le colonne di X sono centrate rispetto alla loro media campionaria).
Poiché la matrice di covarianza Σ dei nostri dati centrati è una matrice simmetrica semidefinita positiva, possiamo calcolare la sua decomposizione in autovalori:

$$\sum = V \Lambda V^T \tag{2.29}$$

con:

$$V = [v_1, v_2, \dots, v_p] \in R^{p \times p}, \quad \Lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & 0 \\ 0 & \cdots & \lambda_n \end{bmatrix}, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq 0. \tag{2.30}$$

Si può dimostrare che le colonne di V descrivono le direzioni di maggiore varianza dei dati, in funzione della grandezza degli autovalori. Ciò significa che V è una nuova base per R^p e può essere vista come una matrice di rotazione. Sia $x(i) \in R^p$, allora $y^{(i)} = V^T x^{(i)}$ è lo stesso vettore ma rappresentato con la base V.

Di conseguenza, potremmo calcolare la rotazione di ogni punto nei nostri dati originali X ottenendo la matrice $Y = \bar{X}V \in R^{n \times p}$.
La matrice di covarianza risultante Σ_Y sarebbe:

$$\Sigma_Y = \frac{1}{N-1} Y^T Y = V^T X^T X V = V^T \Sigma V = \Lambda \tag{2.31}$$

Ciò significa che i dati scritti rispetto alla nuova base V hanno una matrice di covarianza diagonale Λ e la varianza di Y rispetto all'asse v_i è λ_i.
Chiamiamo $v_1, \dots, v_p$ componenti principali e λ_i è la varianza spiegata della componente principale v_i.

Poiché gli autovalori sono calcolati in ordine decrescente, le prime componenti dei dati scritti nella base V sono quelle con la maggiore varianza spiegata.

Quindi, troncare la matrice dei dati ruotata alle prime k colonne (Y_k) causa una perdita di varianza spiegata che è quantificata dai rimanenti autovalori $\lambda_{k+1} \dots \lambda_p$. La percentuale di varianza spiegata dalle prime k componenti è quindi calcolata come:

$$ev_{ratio} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \tag{2.32}$$

2.9 Confusion matrix

Le prestazioni degli algoritmi di apprendimento automatico sono in genere valutate da una matrice di confusione, come illustrato nella Figura 2.18, specialmente in problemi di classificazione binaria come quello trattato nel nostro studio.
Si tratta di uno strumento essenziale che consente di visualizzare e analizzare il comportamento di un classificatore confrontando le predizioni del modello con le classi effettive.
Le colonne sono la classe predette (Predicted) e le righe sono la classe effettive (Actual).

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Figura 2.18: Matrice di confusione

Nella matrice di confusione, TN è il numero di esempi negativi correttamente classificati (True Negatives), FP è il numero di esempi negativi erroneamente classificati come positivi (Falsi positivi), FN è il numero di esempi positivi erroneamente classificati come negativi (Falsi negativi) e TP è il numero di esempi positivi correttamente classificati (Veri positivi).

La matrice di confusione permette di calcolare diverse metriche di valutazione delle prestazioni dell'algoritmo, tra cui:

La *Predictive accuracy* (Accuratezza*)*, ovvero la misura delle prestazioni generalmente associata agli algoritmi di apprendimento automatico, essa rappresenta la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive, ed è definita come:

accuracy = (TP +TN) / (TP +FP +TN +FN).

Nel contesto della predizione del cancro al seno, l'accuratezza da sola potrebbe non essere sufficiente per valutare l'efficacia del modello, dato il possibile sbilanciamento delle classi.
In questo caso, precisione e recall e F1-Score diventano particolarmente rilevanti:

- Precisione è critica quando i falsi positivi devono essere minimizzati, per esempio per evitare stress non necessario ai pazienti.
- Recall è essenziale per garantire che il maggior numero possibile di casi di cancro sia identificato correttamente, riducendo il rischio di mancata diagnosi.
- F1-Score: Offre un bilanciamento tra precisione e recall, fornendo una singola metrica che valuta complessivamente la performance del modello, particolarmente utile quando le classi sono sbilanciate.

La *precision* (precisione), indica la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive.
precision = (TP) / (TP +FP).

La *recall* (Sensibilità o Tasso di Rilevamento), misura la capacità del modello di identificare correttamente le istanze positive.
recall = (TP) / (TP +FN).

La *F1-score*, essa fornisce una misura combinata di precisione e recall, utile quando è necessario trovare un equilibrio tra i due.
F1-score = 2 x (precision x recall) / (precision + recall)

2.10 Smote

Nell'ambito dell'analisi dei dati e dell'apprendimento automatico, la presenza di classi fortemente sbilanciate rappresenta una sfida significativa che può compromettere l'efficacia dei modelli predittivi.

Lavorare con dati fortemente sbilanciati può essere problematico sotto diversi aspetti:

- Metriche di performance distorte: in un dataset altamente sbilanciato, ad esempio un dataset binario con un rapporto di classe di 98:2, un algoritmo che prevede sempre la classe di maggioranza e ignora completamente la classe di minoranza sarà comunque corretto al 98%. Ciò rende insignificanti misure come l'accuratezza della classificazione. Ciò a sua volta rende difficile la valutazione delle prestazioni del classificatore e può anche danneggiare l'apprendimento di un algoritmo che si sforza di massimizzare l'accuratezza.

- Dati di training insufficienti nella classe di minoranza: nei domini in cui la raccolta dati è costosa, un set di dati contenente 10.000 esempi è in genere considerato abbastanza grande. Se, tuttavia, il set di dati è sbilanciato con un rapporto di classe di 100:1, significa che contiene solo 100 esempi della classe di minoranza. Questo numero di esempi potrebbe non essere sufficiente all'algoritmo di classificazione per stabilire un buon confine di decisione e può portare a una scarsa generalizzazione

Per affrontare questa problematica, nel presente studio è stata impiegata la tecnica SMOTE (Synthetic Minority Over-sampling Technique), un approccio sofisticato per il bilanciamento delle classi.

SMOTE, introdotta da Chawla et al. nel 2002, è una metodologia di oversampling che mira a mitigare lo sbilanciamento delle classi attraverso la generazione sintetica di nuove istanze della classe minoritaria.
A differenza delle tecniche di oversampling tradizionali, che si limitano a replicare le istanze esistenti, SMOTE crea nuovi esempi sintetici interpolando tra istanze vicine della classe minoritaria nello spazio delle caratteristiche.
Il processo SMOTE opera selezionando un'istanza della classe minoritaria e individuando i suoi k-vicini più prossimi. Successivamente, vengono create nuove istanze sintetiche lungo i segmenti che congiungono l'istanza selezionata ai suoi vicini.

Questo approccio non solo aumenta la numerosità della classe minoritaria, ma arricchisce anche la varietà dei dati, contribuendo a una rappresentazione più robusta e diversificata della classe sottorappresentata.
L'implementazione di SMOTE in questo studio si è rivelata cruciale per affrontare il marcato sbilanciamento tra le classi osservato nel dataset originale. Questa tecnica ha permesso di ottenere una distribuzione più equilibrata dei dati.

Il processo SMOTE si articola nei seguenti passaggi:

Sia $X=\{x_1,x_2,...,x_n\}$ l'insieme di istanze della classe minoritaria, dove $x_i \in R^d$ rappresenta un vettore d-dimensionale nello spazio delle caratteristiche.

Per ogni istanza x_i della classe minoritaria, vengono identificati i k-vicini più prossimi utilizzando una metrica di distanza, tipicamente la distanza euclidea:

$$d(x_i, x_j) = \|x_i - x_j\|_2 \tag{2.33}$$

Successivamente si seleziona casualmente uno dei k-vicini, sia esso x_j e si genera una nuova istanza sintetica x_{new}, come combinazione lineare di x_i e x_j

$$x_{new} = x_i + \lambda(x_j - x_i) \tag{2.34}$$

dove $\lambda \in [0,1]$ è un numero casuale che determina la posizione della nuova istanza lungo il segmento che congiunge x_i e x_j.
Tutto questo processo viene eseguito fino al raggiungimento del numero desiderato di nuove istanze sintetiche.

Ecco una versione semplificata dell'algoritmo SMOTE:

```
def get_neigbours(M, k):
      nn = NearestNeighbors(n_neighbors=k+1, metric="euclidean").fit(M)
      dist, indices = nn.kneighbors(M, return_distance=True)
      return dist, indices

def SMOTE(M, N, k=5):
       t = M.shape[0] # number of minority class samplesnumattrs = M.shape[1]
       N = int(N/100)
       _, indices = get_neigbours(M, k)
       synthetic = np.empty((N * t, numattrs))
       synth_idx = 0
       for i in range(t):
              for j in range(N):
                     neighbour = randrange(1, k+1)
                     diff = M[indices[i, neighbour]] - M[i]
                     gap = random.uniform(0, 1)
                     synthetic[synth_idx] = M[i] + gap*diff
                     synth_idx += 1
       return synthetic
```

3 Framework Pythorch Geometric

3.1 Introduzione

PyTorch Geometric è una delle librerie più importanti per il machine learning su dati strutturati in forma di grafo, sviluppata nel Department Of Computer Graphics dell'Università *TU Dortmund Unversity* e presentato al workshop ICLR, International Conference on Learning Representations, del 2019 [26]

PyTorch Geometric (PyG) è estensibile e ottimizzata per la costruzione e l'addestramento di reti neurali grafiche (Graph Neural Networks, GNNs) basata su PyTorch.
Essa fornisce un insieme di strumenti e metodi per l'apprendimento profondo su grafi e strutture irregolari, noti anche come geometric deep learning.

La libreria include:

- Supporto per la manipolazione e la gestione efficiente di dati grafo-strutturati.
- Implementazioni di vari algoritmi di aggregazione e propagazione di messaggi utilizzati nelle GNN.
- Una vasta gamma di strati e modelli predefiniti per costruire reti neurali grafiche, inclusi Graph Convolutional Networks (GCN), Graph Attention Networks (GAT) e molte altre architetture avanzate.
- Meccanismi per il caricamento di dati in mini-batch e il supporto per l'addestramento distribuito su più GPU e macchine.
- Funzionalità per analizzare e visualizzare strutture grafiche e i risultati dell'addestramento dei modelli.

La libreria PyG è progettata per essere facilmente estensibile e integrabile con altri componenti dell'ecosistema PyTorch, facilitando lo sviluppo rapido e la sperimentazione di nuovi algoritmi di apprendimento su grafi.

La libreria implementa un gran numero di metodi computazionali derivati dalla teoria dei grafi e strutture proposte nei più recenti articoli di geometric deep learning:

» SplineConv from Fey et al.: SplineCNN: Fast Geometric Deep Learning with Continuous B- Spline Kernels (CVPR 2018)

» GCNConv from Kipf and Welling: Semi-Supervised Classification with Graph Convolutional Networks (ICLR 2017)

» ChebConv from Defferrard et al.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)

» NNConv from Gilmer et al.: Neural Message Passing for Quantum Chemistry (ICML 2017)

» CGConv from Xie and Grossman: Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties (Physical Review Letters 120, 2018)

» ECConv from Simonovsky and Komodakis: Edge-Conditioned Convolution on Graphs (CVPR 2017)

» GATConv from Veličković et al.: Graph Attention Networks (ICLR 2018)

» SAGEConv from Hamilton et al.: Inductive Representation Learning on Large Graphs (NIPS 2017)

» GraphConv from, e.g., Morris et al.: Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks (AAAI 2019)

» GatedGraphConv from Li et al.: Gated Graph Sequence Neural Networks (ICLR 2016)

» GINConv from Xu et al.: How Powerful are Graph Neural Networks? (ICLR 2019)

» GINEConv from Wu et al.: Strategies for Pre-training Graph Neural Networks (ICLR 2020)

» ARMAConv from Bianchi et al.: Graph Neural Networks with Convolutional ARMA Filters (CoRR 2019)

» SGConv from Wu et al.: Simplifying Graph Convolutional Networks (CoRR 2019)

» APPNP from Klicpera et al.: Predict then Propagate: Graph Neural Networks meet Personalized PageRank (ICLR 2019)

» MFConv from Duvenaud et al.: Convolutional Networks on Graphs for Learning Molecular Fingerprints (NIPS 2015)

» AGNNConv from Thekumparampil et al.: Attention-based Graph Neural Network for Semi- Supervised Learning (CoRR 2017)

» TAGConv from Du et al.: Topology Adaptive Graph Convolutional Networks (CoRR 2017)

» RGCNConv from Schlichtkrull et al.: Modeling Relational Data with Graph Convolutional Networks (ESWC 2018)

» SignedConv from Derr et al.: Signed Graph Convolutional Network (ICDM 2018)

» DNAConv from Fey: Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks (ICLR-W 2019)

» EdgeConv from Wang et al.: Dynamic Graph CNN for Learning on Point Clouds (CoRR, 2018)

» PointConv (including Iterative Farthest Point Sampling, dynamic graph generation based on nearest neighbor or maximum distance, and k-NN interpolation for upsampling) from Qi et al.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation (CVPR 2017) and PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space (NIPS 2017)

» XConv from Li et al.: PointCNN: Convolution On X-Transformed Points (official implementation) (NeurIPS 2018)

» PPFConv from Deng et al.: PPFNet: Global Context Aware Local Features for Robust 3D Point Matching (CVPR 2018)

» GMMConv from Monti et al.: Geometric Deep Learning on Graphs and Manifolds using Mixture Model CNNs (CVPR 2017)

» FeaStConv from Verma et al.: FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis (CVPR 2018)

» HypergraphConv from Bai et al.: Hypergraph Convolution and Hypergraph Attention (CoRR 2019)

» GravNetConv from Qasim et al.: Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks (European Physics Journal C, 2019)

» A MetaLayer for building any kind of graph network similar to the TensorFlow Graph Nets library from Battaglia et al.: Relational Inductive Biases, Deep Learning, and Graph Networks (CoRR 2018)

» GlobalAttention from Li et al.: Gated Graph Sequence Neural Networks (ICLR 2016)

» Set2Set from Vinyals et al.: Order Matters: Sequence to Sequence for Sets (ICLR 2016)

» Sort Pool from Zhang et al.: An End-to-End Deep Learning Architecture for Graph Classification (AAAI 2018)

» Dense Differentiable Pooling from Ying et al.: Hierarchical Graph Representation Learning with Differentiable Pooling (NeurIPS 2018)

» Dense MinCUT Pooling from Bianchi et al.: MinCUT Pooling in Graph Neural Networks (CoRR 2019)

» Graclus Pooling from Dhillon et al.: Weighted Graph Cuts without Eigenvectors: A Multilevel Approach (PAMI 2007)

» Voxel Grid Pooling from, e.g., Simonovsky and Komodakis: Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs (CVPR 2017)

» Top-K Pooling from Gao and Ji: Graph U-Nets (ICML 2019), Cangea et al.: Towards Sparse Hierarchical Graph Classifiers (NeurIPS-W 2018) and Knyazev et al.: Understanding Attention and Generalization in Graph Neural Networks (ICLR-W 2019)

» SAG Pooling from Lee et al.: Self-Attention Graph Pooling (ICML 2019) and Knyazev et al.: Understanding Attention and Generalization in Graph Neural Networks (ICLR-W 2019)

» Edge Pooling from Diehl et al.: Towards Graph Pooling by Edge Contraction (ICML-W 2019) and Diehl: Edge Contraction Pooling for Graph Neural Networks (CoRR 2019)

» ASAPooling from Ranjan et al.: ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations (AAAI 2020)

» Local Degree Profile from Cai and Wang: A Simple yet Effective Baseline for Non-attribute Graph Classification (CoRR 2018)

» Jumping Knowledge from Xu et al.: Representation Learning on Graphs with Jumping Knowledge Networks (ICML 2018)

» Node2Vec from Grover and Leskovec: node2vec: Scalable Feature Learning for Networks (KDD 2016)

» MetaPath2Vec from Dong et al.: metaphat2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

» Deep Graph Infomax from Veličković et al.: Deep Graph Infomax (ICLR 2019)

» All variants of Graph Auto-Encoders from Kipf and Welling: Variational Graph Auto-Encoders (NIPS-W 2016) and Pan et al.: Adversarially Regularized Graph Autoencoder for Graph Embedding (IJCAI 2018)

» RENet from Jin et al.: Recurrent Event Network for Reasoning over Temporal Knowledge Graphs (ICLR-W 2019)

» GraphUNet from Gao and Ji: Graph U-Nets (ICML 2019)

» SchNet from Schütt et al.: SchNet: A Continuous-filter Convolutional Neural Network for Modeling Quantum Interactions (NIPS 2017)

» DimeNet from Klicpera et al.: Directional Message Passing for Molecular Graphs (ICLR 2020)

» NeighborSampler from Hamilton et al.: Inductive Representation Learning on Large Graphs (NIPS 2017)

» ClusterGCN from Chiang et al.: Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks (KDD 2019)

» GraphSAINT from Zeng et al.: GraphSAINT: Graph Sampling Based Inductive Learning Method (ICLR 2020)

» GDC from Klicpera et al.: Diffusion Improves Graph Learning (NeurIPS 2019)

» SIGN from Rossi et al.: SIGN: Scalable Inception Graph Neural Networks (CoRR 2020)

3.2 Gestione dei dati a grafo

In PyTorch Geometric un grafo viene rappresentato con una struttura dati che contiene uno o più dei seguenti campi:

» x, matrice delle feature di ogni nodo, di dimensioni [$n.\,nodi \times n.\,features\ per\ nodo$].

» edge_index, matrice di connettività in formato COO (derivata dalla Adjacency Matrix), di dimensioni [$2 \times n.\,rami$] che contiene le coppie di nodi connessi, ordinati secondo la direzione del ramo.

» edge_attribute, matrice delle feature di ogni ramo, di dimensioni [$n.rami \times n.features\ per\ ramo$].

» pos, matrice delle coordinate "spaziali" di ogni nodo, di dimensioni [$n.\,nodi \times n.\,dimensioni\ per\ nodo$]

» y, vettore graph labels che contiene l'etichetta associata ad ogni grafo, di dimensioni [$1 \times n.\,grafi$].

» edge_labels, vettore che contiene l'etichetta associata ad ogni ramo, di dimensioni [$1 \times n.rami$].

» node_labels, vettore che contiene l'etichetta associata ad ogni nodo, di dimensioni [$1 \times n.\,nodi$].

inoltre, possono essere aggiunti ulteriori elementi su cui sviluppare algoritmi personalizzati.

Nella maggior parte dei casi i grafi non vengono definiti manualmente ma importati da dataset esistenti, che in accordo alle regole di PyTorch Geometric, devono essere composti da una serie di files *csv* (*comma separated value*), con uno specifico formato.

3.3 Esempio implementativo di una GCN

In questo paragrafo esploreremo un esempio pratico di implementazione di una GCN utilizzando PyTorch Geometric.
Per prima cosa creeremo un semplice un grafo non pesato e non orientato con tre nodi e quattro spigoli, ne è possibile vedere il plot nella Figura 3.1. Ogni nodo contiene esattamente una caratteristica o feature:

```
import torch
from torch_geometric.data import Data

edge_index = torch.tensor([[0, 1, 1, 2],
        [1, 0, 2, 1]], dtype=torch.long)
x = torch.tensor([[-1], [0], [1]], dtype=torch.float)
data = Data(x=x, edge_index=edge_index)
>>> Data(edge_index=[2, 4], x=[3, 1])
```

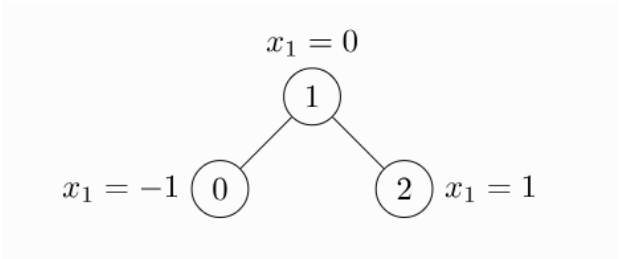


Figura 3.1: plot del grafo implementato

In questo esempio:

- `edge_index` è un tensore (2x4) che rappresenta le connessioni del grafo. La prima riga indica i nodi di origine, la seconda i nodi di destinazione.
- `x` è un tensore (3x1) che rappresenta le caratteristiche (features) di ciascun nodo.
- `Data` è una classe di PyTorch Geometric che incapsula tutte le informazioni del grafo.

Si noti che è necessario che gli elementi in edge_index contengano solo indici nell'intervallo { 0, ..., num_nodes - 1}. Questo perché vogliamo che la nostra rappresentazione dei dati finale sia il più compatta possibile, ad esempio, vogliamo indicizzare le caratteristiche del nodo sorgente e di destinazione del primo spigolo(0, 1) tramite x[0] e x[1] , rispettivamente.

3.3.1 Definiamo la GCN

Utilizzeremo un semplice layer GCN e replicheremo gli esperimenti sul dataset di citazioni Cora, un benchmark standard nel campo dell'apprendimento su grafi.

```
from torch_geometric.datasets import Planetoid

dataset = Planetoid(root='/tmp/Cora', name='Cora')
>>> Cora()
```

Implementazione di una GCN a due livelli:

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GCN(torch.nn.Module):
 def __init__(self):
  super().__init__()
  self.conv1 = GCNConv(dataset.num_node_features, 16)
  self.conv2 = GCNConv(16, dataset.num_classes)

 def forward(self, data):
  x, edge_index = data.x, data.edge_index

  x = self.conv1(x, edge_index)
  x = F.relu(x)
  x = F.dropout(x, training=self.training)
  x = self.conv2(x, edge_index)

  return F.log_softmax(x, dim=1)
model = GCN()
out = model(data)
```

Il costruttore della classe (cioè il metodo `__init__`) definisce una rete con due layer convoluzionali (GCNConv) che vengono richiamati nel passaggio successivo (forward pass): il primo trasforma le feature di input da 1 a 16 dimensioni, seguito da una funzione di attivazione ReLU e il secondo le riporta a 1 dimensione.

È importante notare che la funzione di attivazione non lineare, non è integrata nelle chiamate dei layer convoluzionali (conv) e quindi deve essere applicata successivamente.
Alla fine del processo, la rete produce una distribuzione softmax sul numero di classi.

Applicando il modello ai dati del grafo, otteniamo un output che rappresenta le nuove feature dei nodi dopo il passaggio attraverso la GCN.

Alleniamo questo modello per 200 epoche sei nodi di training:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GCN().to(device)
data = dataset[0].to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

model.train()
for epoch in range(200):
 optimizer.zero_grad()
 out = model(data)
 loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
 loss.backward()
 optimizer.step()
```

Questo blocco di codice:

1. Sposta il modello e i dati su GPU se disponibile.
2. Definisce un ottimizzatore Adam con learning rate e weight decay specifici.
3. Esegue 200 epoche di addestramento, dove in ogni epoca:
    - Azzeriamo i gradienti.
    - Calcoliamo l'output del modello.
    - Calcoliamo la loss solo sui nodi di training.
    - Eseguiamo la backpropagation.
    - Aggiorniamo i parametri del modello.

Infine, possiamo valutare il nostro modello sui nodi di test:

```
model.eval()
pred = model(data).argmax(dim=1)
correct = (pred[data.test_mask] == data.y[data.test_mask]).sum()
acc = int(correct) / int(data.test_mask.sum())
print(f'Accuracy: {acc:.4f}')
>>> Accuracy: 0.8150
```

L'accuratezza ottenuta (circa 81.50%) è notevole considerando la semplicità del modello, dimostrando l'efficacia delle GCN nell'apprendimento su grafi.

Questo esempio illustra come PyTorch Geometric semplifichi notevolmente l'implementazione di modelli complessi su grafi, permettendo di concentrarsi sulla struttura del modello piuttosto che sui dettagli di basso livello della gestione dei dati grafici.

3.3.2 Librerie

Per il presente lavoro di tesi sono state utilizzate ulteriori tre librerie di supporto fornite dagli stessi sviluppatori:

PyTorch Scatter, libreria di ottimizzazione delle operazioni *sum*, *min*, *mean* e *max*, su tensori.

PyTorch Sparse, ottimizzazione delle operazioni su matrici sparse: *coalesce* (fusione), *transpose*, *sparse - dense matrix multiplication* e *sparse - sparse matrix multiplication*.

PyTorch Cluster, algoritmi di clustering: *k-NN*.

Infine, sono state utilizzate le librerie NetworkX e Matplotlib per la visualizzazione dei grafi nelle fasi di debug.

3.3.3 Configurazione Hardware

Le simulazioni sono state svolte su due piattaforme hardware differenti: un personal computer con CPU Intel Core i7-7700HQ @ 2.80 GHz, 16GB RAM e GPU Nvidia GeForce GTX 1050 Ti, e una macchina virtuale con CPU Intel Xeon Platinum 8272CL @ 2.60 GHz, 31GB RAM e GPU Microsoft Hyper-V virtual VGA.

Le migliori performance sono state ottenute dalla macchina virtuale, dei leggeri miglioramenti rispetto alla prima macchina.

4 Modello e metodologia sviluppata

In questo capitolo verrà descritto in maniera dettagliata il modello di rete neurale convoluzionale a grafi sviluppato per la classificazione supervisionata dei pazienti con cancro al seno, utilizzando il dataset METABRIC. Verranno dettagliate le componenti del modello, i metodi di pre-processamento dei dati, l'architettura della rete e gli algoritmi di addestramento.

4.1 Panoramica del progetto

L'algoritmo utilizza una rete neurale convoluzionale su grafi (GCN) per l'analisi delle espressioni geniche e la previsione della sopravvivenza dei pazienti.
Di seguito si descrivono preliminarmente tutti i passi dell'algoritmo:

1. I dati genici vengono pre-processati per correggere il formato e normalizzare i valori. Dopo aver eliminato le colonne non necessarie e sostituito i caratteri non standard, si crea una matrice di dati che include le espressioni geniche dei pazienti. Questi dati vengono quindi utilizzati per generare campioni casuali basati sulla distribuzione normale, che rappresentano le espressioni geniche dei pazienti.

2. Successivamente si applica l'algoritmo K-Means per effettuare un clustering trasformando e normalizzando i dati per facilitare l'analisi.
   Quindi si esegue la clusterizzazione sui dati pre-processati, identificando il numero ottimale di cluster attraverso il metodo del gomito e il Silhouette Score. I cluster ottenuti vengono poi utilizzati per creare una rete di pazienti, dove i nodi rappresentano i pazienti e gli archi le relazioni di similarità tra di essi.

3. Questa rete è rappresentata tramite la libreria NetworkX, e i nodi vengono colorati in base alla sopravvivenza dei pazienti.

4. Successivamente, viene caricata la rete di pazienti, i dati di espressione genica e le etichette di sopravvivenza. La matrice di adiacenza della rete è normalizzata, e le caratteristiche geniche sono trasformate mediante Z-score normalization. Il dataset è diviso in set di addestramento, validazione e test.

5. Il modello di GCN è definito con diversi strati convoluzionali per estrarre caratteristiche significative dai dati di rete. La rete è addestrata per classificare la sopravvivenza dei pazienti, utilizzando una funzione di perdita cross-entropy e l'ottimizzatore Adam. L'addestramento avviene in diverse epoche, e per ogni epoca, vengono calcolati la perdita e l'accuratezza sia sui set di addestramento che di validazione. Il modello viene valutato periodicamente per verificare il miglioramento delle prestazioni.

6. Infine, dopo l'addestramento, il modello viene testato utilizzando il set di test per valutare la sua capacità di generalizzare a dati non visti.

Prima di addentrarci nella descrizione degli aspetti software diamo uno sguardo al modello in sé.

4.2 L'algoritmo

Il modello è suddiviso in due fasi principali:

1. Costruzione della network di pazienti:
2. Utilizzo della rete di pazienti e classificazione tramite GCN:

4.2.1 Lettura e Preprocessing dei Dati

Il primo passo è la lettura e salvataggio del dataset dei pazienti-gene da un file CSV in strutture dati manipolabili. I dati vengono pre-processati per rimuovere colonne non necessarie e convertire i dati in un formato numerico utilizzabile per le successive operazioni.
In questa sezione di codice si rimuovono le colonne di sopravvivenza non necessarie e si convertono i dati in un array "numpy", spostando in altre strutture gli ID dei pazienti, in modo tale da avere ID, geni e elementi classificanti divisi tra loro e pronti per essere utilizzati.

4.2.2 Generazione delle Distribuzioni Normalizzate

Per ogni paziente e per ogni gene, si costruisce una distribuzione normale sulla base dei valori di espressione genica rilevati dal dataset. In particolare, si calcola la deviazione standard della distribuzione dei valori del gene su tutti i pazienti e si campiona il valore del gene per il paziente considerato, ripetendo l'operazione per 1000 volte. In questo modo, si ottiene una matrice di dimensioni n x k x 1000 per ogni paziente, dove n rappresenta il numero di pazienti e k rappresenta il numero di geni.

Quindi consideriamo il dataset che contiene le espressioni geniche di *n* pazienti e *k* geni, organizzati in una matrice **D**:

$$D = \begin{bmatrix} G_1^{P1} & G_2^{P1} & \cdots & G_k^{P1} \\ G_1^{P2} & G_2^{P2} & \cdots & G_k^{P2} \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot \\ G_1^{Pn} & G_2^{Pn} & \cdots & G_k^{Pn} \end{bmatrix} \tag{2.33}$$

dove G_j^{Pi} rappresenta il valore di espressione del gene *j* per il paziente *i*.

Per ogni paziente P_i e per ogni G_j, costruiamo una distribuzione normale basata su media e deviazione standard.

La deviazione standard σG_j è calcolata come:

$$\sigma_{G_j} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(G_j^{P_i} - \mu_{G_j}\right)^2} \tag{2.34}$$

dove μG_j è la media dei valori di espressione del gene j su tutti i pazienti:

$$\mu_{G_j} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} G_j^{P_i}} \tag{2.35}$$

Per ogni paziente P_i e per ogni gene G_j, consideriamo la distribuzione normale $N(G_jP_i, \sigma G_j)$. Da questa distribuzione campioniamo 1000 valori, generando una matrice per ogni paziente P_i:

$$D = \begin{bmatrix} N(G_1^{Pi}, \sigma_{G_1}) & N(G_1^{Pi}, \sigma_{G_1}) & \cdots & N(G_1^{Pi}, \sigma_{G_1}) \\ N(G_2^{Pi}, \sigma_{G_2}) & N(G_2^{Pi}, \sigma_{G_2}) & \cdots & N(G_2^{Pi}, \sigma_{G_2}) \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot \\ N(G_k^{Pi}, \sigma_{G_k}) & N(G_k^{Pi}, \sigma_{G_k}) & \cdots & N(G_k^{Pi}, \sigma_{G_k}) \end{bmatrix}_{k \times 1000} \tag{2.34}$$

dove ogni elemento della matrice $\mathbf{M}_{Pi}$ rappresenta un valore campionato dalla distribuzione normale corrispondente.

Ripetiamo il processo descritto nel passo precedente per tutti i pazienti P_i (i=1,2,…,n), ottenendo n matrici $\mathbf{M}_{P1}$,$\mathbf{M}_{P2}$,…,$\mathbf{M}_{Pn}$, ciascuna delle dimensioni k×1000.

Tutto questo procedimento viene eseguito per generare una base fittizia di dati e poter procedere all'embedding.

.

4.2.3 Clustering dei Pazienti

Per raggruppare i pazienti per similarità e così creare la network è stato utilizzato il K-Means.

L'algoritmo K-Means è uno dei metodi di clustering più utilizzati e si basa sulla partizione dei dati in k cluster. L'algoritmo inizia selezionando k punti casuali come *centroidi* iniziali. Successivamente, ripete i seguenti passaggi fino a convergenza:

1. Assegnare ciascun punto dati al centroide più vicino, formando k cluster.
2. Calcolare i nuovi centroidi come la media dei punti assegnati a ciascun cluster.
3. Ripetere i passi 1 e 2 fino a quando i centroidi non cambiano più significativamente o il numero massimo di iterazioni è raggiunto.

Il criterio di similarità utilizzato è la distanza euclidea, che misura la "vicinanza" tra i punti dati e i centroidi. La distanza euclidea tra due punti x e y in uno spazio n-dimensionale è data da:

$$|x - y| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.34}$$

Tramite questo algoritmo di clustering, come detto in precedenza costruiamo la rete di pazienti, basandosi su questo criterio, raggruppando punti dati che sono vicini l'uno all'altro rispetto alla distanza euclidea.

Di conseguenza i pazienti vengono raggruppati in cluster basati sulle loro caratteristiche genetiche. Questo permette di identificare gruppi di pazienti con profili genetici simili.
Entrando nei dettagli, viene creato un dizionario che mappa ciascun cluster ai relativi ID dei pazienti fino a creare una rete dove ogni nodo rappresenta un paziente e ogni arco rappresenta una connessione tra pazienti appartenenti allo stesso cluster.
Gli archi tra i nodi sono pesati in base alla frequenza con cui i pazienti condividono lo stesso cluster. Questo peso rappresenta la forza della similarità tra i pazienti.

Il risultato finale è il file:

Network.txt: Esso è la rappresentazione di un grafo bidirezionale, quindi serie di coppie di ID-PAZIENTI, ne è possibile vedere una porzione di esempio nella figura seguente:

1	62	100
2	62	115
3	62	149
4	62	157
5	62	174
6	62	191
7	62	200
8	62	214
9	62	238
10	62	265
...	...	...
3420	3014	4792
3421	3014	4893
3422	3014	4911
3423	3014	5008
3424	3014	5041
3425	3014	5100
3426	3014	5135
3427	3014	5162
3428	3014	5208
3429	3014	5209
3430	3014	5223

Figura 4.1: porzione di rete di pazienti

Mentre il secondo codice rappresenta il cuore del modello GCN, dove risiede la sua implementazione di convoluzione del grafo e tutte le tecniche di addestramento e gestione dei dati necessari per la classificazione.

4.2.4 Inizializzazione dei Dati

Il processo inizia con l'acquisizione dei dati grezzi contenuti nel file CSV, essi rappresentano le informazioni genetiche dei pazienti in forma non elaborata. Successivamente, il codice esegue una trasformazione cruciale su questi dati, ovvero si convertono le informazioni genetiche in un formato matematico più adatto all'analisi computazionale: una matrice di feature sparse.

Questa matrice è una rappresentazione numerica efficiente delle caratteristiche genetiche di ogni paziente.
Il termine "sparse" (sparsa) indica che la maggior parte degli elementi della matrice sono zero, facilitando l'identificazione di pattern e correlazioni significative.

Il processo di preelaborazione continua con la conversione delle etichette in vettori one-hot.
Dato che gli algoritmi di apprendimento automatico elaborano dati numerici con maggiore efficacia, è essenziale trasformare queste variabili categoriche in rappresentazioni numeriche poiché si garantisce che i dati siano nella forma corretta per l'input nel modello GCN.

One-hot encoding è una tecnica che converte variabili categoriche in un formato più adatto agli algoritmi di apprendimento automatico.
Nel nostro caso le variabili categoriche sono le classi che dovrà predire il modello, che rappresentano gli anni di vita rimasti al paziente (>=2 e <2).

Esso rappresenta una variabile categorica come un vettore binario di lunghezza pari al numero di categorie possibili, nel nostro caso 2, dato che la nostra è una classificazione binaria. Ogni elemento del vettore rappresenta una categoria e ha valore 0 o 1, indicando l'assenza o la presenza della categoria corrispondente.

Più precisamente, sia *x* una variabile categorica che assume valori in un insieme di categorie {c1,c2,…,cn}. La funzione di one-hot encoding può essere definita come:

$$one - hot(x) \;=\; [\delta(x, c_1), \delta(x, c_2), \ldots, \delta(x, c_n)] \tag{2.34}$$

dove δ è la funzione delta di Kronecker:

$$\delta(x, c_i) = \begin{cases} 1 \;\; se \;\; x = c_i \\ 0 \;\; se \;\; x \neq c_i \end{cases} \tag{2.10}$$

Questa funzione produce un vettore binario di lunghezza n, in cui solo l'elemento corrispondente alla categoria presente è 1, mentre tutti gli altri elementi sono 0.

4.3 Il modello

4.3.1 Graph Convolutional Network (GCN)

La GCN è il cuore del modello di classificazione ed è composta da due principali classi: la GraphConvolution e la GCN.

La classe GraphConvolution implementa la convoluzione sui grafi, estendendo la classe Module di PyTorch. Questa classe definisce i parametri di convoluzione, inclusi i pesi e i bias, e il metodo forward, che esegue la propagazione in avanti.

Come visto nel paragrafo [2.6], la propagazione in avanti implica la moltiplicazione della matrice delle feature (H) con i pesi (W) e la moltiplicazione del risultato con la matrice di adiacenza normalizzata (N(A)). Questo permette di propagare le informazioni tra i nodi del grafo, incorporando le caratteristiche dei nodi adiacenti nel calcolo. Questa classe riceve in input un tensore rappresentante le caratteristiche dei nodi del grafo e una matrice di adiacenza trasformata, che indica le relazioni tra i nodi considerando anche la loro struttura.

La classe GCN, invece, combina più strati di convoluzione su grafi per costruire un modello di rete neurale profonda. L'architettura di questa rete include un input layer, diversi hidden layers, e un output layer. Ogni hidden layer applica una funzione di attivazione ReLU e un dropout per prevenire l'overfitting. Il metodo forward della classe GCN definisce il flusso dei dati attraverso questi strati, calcolando le probabilità di classificazione per ciascun nodo.

In questo progetto si implementa un modello di rete neurale che consiste in più strati di GraphConvolution, seguiti da un livello di output. Questa architettura è progettata per apprendere rappresentazioni informative dei nodi del grafo e per classificare i nodi stessi in base alle loro caratteristiche.

4.3.2 Architettura della rete

L'architettura del modello GCN utilizzato è composta da due strati principali di convoluzione sui grafi, seguiti da uno strato di output per la classificazione. Di seguito, viene fornita una descrizione dettagliata degli strati:

1. **Strato di Input**: Questo strato riceve le caratteristiche normalizzate dei pazienti e la matrice di adiacenza che rappresenta le relazioni tra i pazienti.
2. **Strati di Convoluzione su Grafi**: Questi strati applicano le operazioni di convoluzione sui grafi per estrarre caratteristiche rilevanti dai nodi e dalle loro connessioni. La convoluzione sui grafi combina le informazioni dei nodi adiacenti, permettendo al modello di catturare la struttura locale del grafo.
3. **Strato di Output**: Lo strato finale produce le predizioni per la classificazione, mappando le rappresentazioni dei nodi apprese negli strati precedenti alle probabilità delle classi target.

L'implementazione del modello GCN in PyTorch è riportata di seguito:

```
class GraphConvolution(Module):

 def __init__(self, in_features, out_features, bias=True):
  super(GraphConvolution, self).__init__()
  # create Weight and Bias trainable parameters
  self.in_features = in_features
  self.out_features = out_features
  self.weight = Parameter(torch.FloatTensor(in_features, out_features))
  if bias:
   self.bias = Parameter(torch.FloatTensor(out_features))
  else:
   self.register_parameter('bias', None)
  self.reset_parameters()

 def reset_parameters(self):
  # standard weight to be uniform
  stdv = 1. / math.sqrt(self.weight.size(1))
  self.weight.data.uniform_(-stdv, stdv)
  if self.bias is not None:
   self.bias.data.uniform_(-stdv, stdv)

 def forward(self, input, adj):
  if adj.is_sparse:
   adj = adj.to_dense()

  # Normalization: D^-1/2 * A_hat * D^-1/2
  identity = torch.eye(adj.size(0)).to(adj.device)
  adj_hat = adj + identity
  D = torch.diag(torch.sum(adj_hat, dim=1))
  D_inv_sqrt = torch.diag(torch.pow(D.diag(), -0.5))
  adj_norm = torch.matmul(torch.matmul(D_inv_sqrt, adj_hat), D_inv_sqrt)
  support = torch.mm(input, self.weight)

  output = torch.mm(adj_norm, support)
  if self.bias is not None:
   return output + self.bias
  else:
   return output

 def __repr__(self):
  return self.__class__.__name__ + ' (' \ + str(self.in_features) + ' -> ' \
      + str(self.out_features) + ')'


class GCN(nn.Module):

 def __init__(self, n_feat, n_hids, n_class, dropout):
  super(GCN, self).__init__()

  # Define the layers of graph convolutional layer
  layers_units = [n_feat] + n_hids
  self.graph_layers = nn.ModuleList(
   [GraphConvolution(layers_units[idx], layers_units[idx + 1]) for idx in
range(len(layers_units) - 1)])
  self.output_layer = GraphConvolution(layers_units[-1], n_class)
  self.dropout = dropout

 def forward(self, x, adj):
  for graph_layer in self.graph_layers:
   x = F.relu(graph_layer(x, adj))
   x = F.dropout(x, self.dropout, training=self.training)

  x = self.output_layer(x, adj)
  return torch.log_softmax(x, dim=1)
```

4.3.3 Processo di inizializzazione e Forward Propagation per la classificazione

Il modello GCN implementato prevede come primo passo l'inizializzazione dei pesi e dei bias all'interno di ciascuno strato di convoluzione grafica (GraphConvolution).

I pesi sono parametri numerici addestrabili che, moltiplicati per gli input del modello, determinano l'importanza relativa di ciascun input. Nel contesto dei GCN, essi definiscono come le informazioni dai nodi e dagli archi nel grafo vengono combinate e propagate attraverso i vari strati della rete, consentendo al modello di estrarre rappresentazioni significative dei dati.

Il bias è una costante aggiunta prima dell'applicazione della funzione di attivazione e aiuta il modello a produrre output non nulli anche quando tutti gli input sono zero, aumentando la flessibilità del modello nell'apprendimento.

Una volta effettuata l'inizializzazione, il processo di forward propagation è definito come segue, per ogni nodo v

- Si aggregano le caratteristiche dei nodi adiacenti a v.
- Ogni caratteristica dei nodi vicini è moltiplicata per un peso specifico.
- Le caratteristiche ponderate sono sommate insieme.
- Si aggiunge il bias.
- Il risultato è passato attraverso una funzione di attivazione, ovvero ReLU (Rectified Linear Unit), per introdurre non linearità nel modello

Il dropout viene utilizzato per ridurre l'overfitting durante l'addestramento del modello.
Esso rimuove casualmente una frazione di connessioni tra i nodi ad ogni passo di addestramento aiutando il modello a essere più robusto e a generalizzare meglio, evitando di dipendere eccessivamente da specifiche connessioni nel grafo.

Al termine della forward propagation, ogni nodo possiede un vettore di valori. Nell'ultimo strato, questi valori vengono trasformati in probabilità di appartenenza alle classi mediante la funzione Softmax. La classe con la probabilità più alta viene scelta come predizione.

Il modello è stato progettato per essere flessibile, consentendo la configurazione del numero di strati nascosti (n_hids), delle dimensioni delle caratteristiche di input (n_feat) .
Ciò permette di adattare il modello a diverse dimensioni e complessità dei dati, mantenendo allo stesso tempo la capacità di apprendere rappresentazioni a grafi significative.

4.3.4 Processo di Addestramento e Valutazione

L'addestramento del modello GCN è un processo iterativo che si svolge attraverso diverse epoche. Ogni epoca comprende tre fasi principali: forward propagation, calcolo della loss e backpropagation.

- Forward Propagation: Come descritto in precedenza, i dati vengono propagati attraverso la rete, generando predizioni per ogni nodo del grafo.

- Calcolo della Loss: Viene calcolata la funzione di perdita (loss function), utilizzando la cross-entropy tra le predizioni del modello e le etichette reali dei nodi. Questa loss quantifica quanto le predizioni del modello si discostano dai valori reali.

- Backpropagation: Questa fase è cruciale per l'apprendimento del modello in quanto si calcola il gradiente della funzione di loss rispetto a ciascun parametro del modello (pesi e bias). Questo gradiente indica la direzione e la grandezza della modifica necessaria per ciascun parametro al fine di ridurre la loss.
  Inoltre, la backpropagation deve tener conto della struttura del grafo. Il gradiente viene propagato all'indietro attraverso gli strati della rete, considerando non solo le connessioni dirette tra i nodi, ma anche l'influenza dei nodi vicini. Questo processo consente al modello di apprendere rappresentazioni che catturano efficacemente la struttura del grafo e le relazioni tra i nodi.

- Aggiornamento dei Parametri: Una volta calcolati i gradienti, l'ottimizzatore Adam viene utilizzato per aggiornare i parametri del modello.

Durante l'addestramento, il modello viene periodicamente valutato su un set di dati di validazione. Questo serve a monitorare le prestazioni del modello su dati non visti e a prevenire l'overfitting, che si verifica quando il modello si adatta troppo ai dati di training perdendo la capacità di generalizzare.

La funzione accuracy viene utilizzata per calcolare l'accuratezza del modello, mentre altre metriche come F1-score, precision e recall vengono calcolate per valutare ulteriormente le performance del modello.

4.3.5 Processo di Test e Valutazione Finale

Dopo l'addestramento, il modello viene testato su un set di dati separato per valutare la sua capacità di generalizzazione. La funzione test esegue questa valutazione, calcolando la perdita e l'accuratezza sul set di test, per visualizzare le performance del modello.
Questo passaggio è cruciale per garantire che il modello operi bene non solo sui dati di addestramento ma anche su dati nuovi di test.

5 Risultati ottenuti

Questo capitolo presenta un'analisi dettagliata del percorso evolutivo del nostro modello di Graph Convolutional Network (GCN) per la stratificazione di pazienti con patologie non trasmissibili. L'obiettivo è offrire una panoramica dettagliata e cronologica dell'intero processo, dalla concezione iniziale all'implementazione definitiva, tramite una visione completa e trasparente del processo di sviluppo, evidenziando come ogni fase di perfezionamento del codice abbia contribuito al miglioramento delle prestazioni.

Verranno evidenziate le sfide incontrate e i progressi incrementali raggiunti in ogni fase. Analizzeremo in dettaglio la conformazione dei dati, il bilanciamento del dataset e come il codice si sia evoluto per affrontare le peculiarità dei dati strutturati a grafo.

5.1 Datasets

Per questo progetto è stato utilizzato il dataset METABRIC (Molecular Taxonomy of Breast Cancer International Consortium), un corpus di dati genomici e clinici ampiamente riconosciuto nel campo della ricerca sul cancro al seno. Esso fornisce una caratterizzazione molecolare completa di oltre 2000 tumori.

Nello specifico, l'analisi si è concentrata sui dati di espressione genica METABRIC, che offrono una rappresentazione quantitativa dell'attività trascrizionale di migliaia di geni in ciascun campione tumorale.

A livello biologico, i geni sono segmenti di DNA che contengono le istruzioni per costruire le proteine, mentre l'espressione genica determina quali proteine vengono prodotte in una cellula, e quindi quali funzioni la cellula può svolgere.
Dato che nel cancro, i pattern di espressione genica sono spesso alterati, mettere a confronto pazienti sani con pazienti non, può aiutare a identificare potenziali bersagli terapeutici.

L'approccio che è stato adottato ha posto le sue fondamenta su una comprensione dettagliata e approfondita dei dataset, prerequisito essenziale per lo sviluppo del codice in maniera canalizzata all’obbiettivo preposto. I primi passi sono stati infatti esaminare minuziosamente la struttura e la composizione dei dataset, la propria natura e la tipologia.
Questa fase preliminare si è rivelata cruciale per acquisire una conoscenza solida sulle informazioni a disposizione e delle loro interconnessioni.

.

5.1.1 Dataset ENTREZ.ID.genomica.metabric.os

Il dataset seguente, come mostrato in Figura 5.1, rappresenta una sintesi integrata di dati genomici e clinici, fornendo una base robusta per l'analisi delle correlazioni tra profili genetici e esiti clinici nei pazienti oncologici, in cui i valori ne rappresentano l'assenza (0) o la presenza (1) del gene in questione.

PATIENT_ID	94	2334	392636	186	79026	113146		7402	677	OS	OS.time	level
MB-0062	0	0	0	0	0	0		0	0	0	12.83	OS>=2
MB-0079	0	0	1	0	0	0		0	0	1	2.38	OS>=2
MB-0100	0	0	0	0	0	0		0	0	1	0.67	OS<2
MB-0115	0	0	0	0	0	0		0	0	1	5.56	OS>=2
MB-0127	0	0	0	0	0	0	...	0	0	0	11.01	OS>=2
MB-0149	0	0	0	0	0	0		0	0	1	4.31	OS>=2
MB-0157	0	0	0	0	0	0		0	0	0	9.56	OS>=2
MB-0164	0	0	0	0	0	0		0	0	0	0.9	OS<2
MB-0174	0	0	0	0	0	0		0	0	0	6.56	OS>=2
MB-0188	0	1	0	0	0	0		0	0	1	2.61	OS>=2
⋮												
MB-5634	0	0	0	0	0	0		0	0	1	5.24	OS>=2
MB-5651	0	0	0	0	0	0		0	0	1	1.75	OS<2
MB-5655	0	0	0	0	0	0		0	0	0	15.98	OS>=2
MB-6143	0	0	0	0	0	2	...	0	0	0	21.3	OS>=2
MB-6223	0	0	0	0	0	0		0	0	0	18.49	OS>=2
MB-6237	0	0	0	0	0	0		0	1	1	8.77	OS>=2
MB-6251	0	0	0	0	0	0		0	0	1	1.22	OS<2

Figura 5.1: Dataset ENTREZ.ID.genomica.metabric.os in formato originale

Struttura:

- Numero di colonne: 156
- Numero di righe: 209
- La prima colonna, "PATIENT_ID", contiene gli identificatori unici (ID) per ogni paziente.
- Le colonne successive (94, 2334, ecc.) contengono dati genomici, rappresentanti con presenza/assenza di specifici geni.
- Le ultime colonne contengono informazioni cliniche:

    - "OS" (Overall Survival): indica se il paziente è sopravvissuto (0) o deceduto (1).
    - "OS.time": il tempo di sopravvivenza in anni.
    - "level": una classificazione basata sul tempo di sopravvivenza (OS>=2 per sopravvivenza di 2 o più anni, OS<2 per meno di 2 anni).

Il parametro "level" costituisce la variabile dipendente per il modello predittivo proposto.

5.1.2 Dataset ENTREZ.ID.gene.expression.metabric.os

PATIENT_ID	X1	X29974	X2	X127550	X53947	X51146		X23140	X26009	OS	OS.time	level
MB-0062	5,412713	5,57146	8,675327	5,372409	6,12862	5,624841		5,912904	9,40027	0	12,83	OS>=2
MB-0079	5,163773	5,412081	10,67962	5,233153	5,723339	5,218649		6,678373	8,959534	1	2,38	OS>=2
MB-0100	5,204053	5,414651	9,744305	5,43954	6,636901	5,788499		7,702676	8,475007	1	0,67	OS<2
MB-0115	5,612779	5,312095	10,88449	5,214144	6,206407	5,401816		7,377194	8,118007	1	5,56	OS>=2
MB-0127	5,458105	5,392807	9,843684	5,077362	6,542352	5,431338		7,334331	8,121314	0	11,01	OS>=2
MB-0149	5,367511	5,509567	9,681886	5,176954	5,825413	5,291021	...	8,396715	8,033341	1	4,31	OS>=2
MB-0157	5,58462	5,051621	10,47546	4,930657	7,189692	5,709059		7,474057	7,913059	0	9,56	OS>=2
MB-0164	5,399935	5,435363	9,596402	5,32333	6,265992	5,368745		7,170153	8,045316	0	0,9	OS<2
MB-0174	5,578841	5,714959	10,45274	5,325357	7,783416	5,716028		8,015598	7,693857	0	6,56	OS>=2
MB-0179	5,372652	5,371565	10,62555	5,254758	7,071885	5,463147		7,558208	7,873357	1	1,49	OS<2
MB-0188	5,551104	5,462855	10,5201	5,403241	6,109512	5,678003		7,380718	8,096481	1	2,61	OS>=2
⋮												
MB-5577	5,560725	5,328761	10,56704	5,329585	6,069552	5,670466		8,610783	8,132073	0	15,19	OS>=2
MB-5616	5,539039	5,450282	9,818055	5,056321	6,346233	5,556551		7,367912	8,267504	0	15,55	OS>=2
MB-5633	5,509044	5,348039	10,29152	5,093135	6,284927	5,563549		8,046165	8,306895	0	14,89	OS>=2
MB-5634	5,507712	5,511096	10,63361	5,348715	5,990919	5,56883		7,947798	8,146131	1	4,62	OS>=2
MB-5651	5,376763	5,263582	9,100757	5,298644	6,44042	5,307602	...	7,737826	8,752925	0	15,05	OS>=2
MB-5655	5,55551	5,384084	10,46553	5,194813	6,493361	5,678821		8,20563	8,40266	1	2,92	OS>=2
MB-6143	5,424203	5,224852	11,60822	5,08454	6,744245	5,40881		7,731763	8,388021	1	5,24	OS>=2
MB-6223	5,366654	5,344031	9,046238	5,258917	6,057556	5,250579		7,662384	8,729026	1	1,75	OS<2
MB-6237	5,633888	5,142537	7,884312	4,935283	7,281366	5,720547		7,436167	8,885672	0	15,98	OS>=2
MB-6251	5,424336	5,234556	7,78792	5,174407	5,946628	5,742796		7,651138	7,565239	0	21,3	OS>=2

Figura 5.2: Dataset ENTREZ.ID.gene.expression.metabric.os in formato originale

Questo dataset contiene dati di espressione genica, anch'esso parte del progetto METABRIC. Come è possibile osservare nella Figura 5.2, questi dati sono relativi ai geni presenti nel tessuto tumorale campionato dai pazienti e sono organizzati in una matrice di espressione genica, dove ogni riga rappresenta un paziente e ogni colonna corrisponde a un gene tumorale distinto.
Questi identificatori, che corrispondono alle colonne nel dataset, indicano di quanto ogni gene è "attivo" o "espresso" nei campioni di tessuto studiati. Quindi, i valori nelle colonne mostrano quanto ciascun gene è espresso nel tessuto del cancro al seno di ogni paziente.

Inoltre, di particolare rilevanza per questo studio è stato l'utilizzo degli identificatori ENTREZ ID, un sistema di nomenclatura standardizzato, impiegato per mappare in modo preciso e coerente i profili di espressione genica ai rispettivi geni, garantendo un'identificazione non ambigua e riproducibile delle entità genomiche oggetto di studio.

Struttura:

- Numero di colonne: 15989
- Numero di righe: 214
- La colonna "PATIENT_ID" contiene identificatori unici per ogni paziente.
- Le colonne successive (X1, X29974, X2, ecc.) rappresentano l'espressione di geni specifici.
- Le ultime colonne contengono informazioni cliniche simili al primo dataset:
    - "OS" (Overall Survival)
    - "OS.time" (tempo di sopravvivenza)
    - "level" (classificazione basata sul tempo di sopravvivenza)

5.2 Fasi iniziali

Una volta chiara la struttura e la natura dei dati, è stato implementato un algoritmo ad hoc che ha permesso la creazione della rete di pazienti, o meglio l'embedding della network, attraverso una serie di passaggi che è possibile vedere nel capitolo [3].

La rete di pazienti (network.txt), ha una struttura a grafo, G=(V,E), dove V è l'insieme dei nodi ed E è l'insieme degli archi.
I nodi del grafo rappresentano i pazienti, gli archi rappresentano la connessione tra di essi. Ogni arco è definito come una coppia di nodi (u,v), dove u e v sono ID di pazienti connessi. Gli archi sono non orientati, il che significa che la connessione tra i pazienti è bidirezionale.

Questa network ottenuta ha come scopo quello di identificare dei sottogruppi di pazienti fortemente connessi tra di loro e verrà utilizzata nel modello GCN, per catturare le informazioni strutturali e relazionali tra i pazienti.
Come abbiamo potuto vedere nel capitolo [2], la GCN sfrutta queste informazioni per migliorare la classificazione delle etichette di sopravvivenza, integrando sia le caratteristiche individuali dei pazienti, che le informazioni relazionali (connessioni nella rete).

Subito dopo è stato implementato il modello GCN, con le relative funzioni di adattamento e normalizzazione dei dati, insieme alla funzione di train, test e metriche per comprendere l'andamento della classificazione.

5.2.1 Risultati con dataset ENTREZ.ID.genomica.metabric.os

Per questa prima fase evolutiva è stato utilizzato il dataset 1, descritto nel paragrafo precedente, ovvero ENTREZ.ID.genomica.metabric.os.

Adesso verranno mostrati i punti chiave del codice e di configurazione con i relativi risultati:

distribuzione dei dati (in tot 207 pazienti):

- 60% per il train set
- 20% per il validation set
- 20% per il test set

Parametri di configurazione del modello GCN:

```
class Args:
 no_cuda = False
 seed = 42
 epochs = 200
 lr = 0.01
 weight_decay = 5e-4
 n_hid = 16
 dropout = 0.5
```

Ovvero:

- Numero di iterazioni (epoche) di addestramento: 200
- Learning rate: 0,01
- Weight decay: $5e^{-4}$
- Hidden layer dimension: 16
- Dropout rate: 0,5

Come classificatore della funzione forward del modello è stato utilizzato il *log_softmax*, come ottimizzatore l'*Adam* e come funzione di loss la *nll_loss*.

```
Epoch: 0200 loss_train: 0.3387 acc_train: 0.8824 loss_val: 0.4018 acc_val: 0.8462 time: 0.0090s
Model training is complete!
Total model training time: 2.2890s
Test set results: loss= 0.4987 accuracy= 0.8085 f1= 0.7229 precision= 0.6537 recall= 0.8085
```

Figura 5.3: Risultati ottenuti con il dataset ENTREZ.ID.genomica.metabric.os

La Figura 5.3 fornisce una rappresentazione visiva dei risultati ottenuti, se ne fa un elenco puntato per una maggiore chiarezza:

Fase di addestramento:

- Loss di Train: 0,3387
- Accuracy di Train: 0,8824
- Loss di Validation: 0,4018
- Accuracy di Validation: 0,8462

Fase di Test:

- Loss: 0,4987
- Accuracy: 0,8085
- F1-Score: 0,7229
- Precision: 0,6537
- Recall: 0,8085

In primis la Loss è una misura di quanto le previsioni del modello si allontanano dai valori reali; quindi, rappresenta l'errore complessivo che il modello commette durante la previsione.

Di conseguenza migliore è il modello più basso sarà il valore di Loss ottenuto.
Come è possibile vedere nella fase di test ha un valore più alto (0.4987), questo squilibrio comunque è comune, in quanto i dati di test non sono mai stati visti dal modello, essendo non presenti nell'addestramento.

L'accuratezza è la proporzione di previsioni corrette (sia positive che negative) sul totale delle previsioni effettuate. Dai valori è possibile notare che acc_val, ovvero l'accuratezza sul set di validazione è di: 0.8462 (84.62%). Essa è leggermente inferiore all'accuratezza di training, il che è un buon segno, in quanto sta a indicare che non c'è overfitting significativo.

La precisione misura la proporzione di previsioni positive corrette rispetto al totale delle previsioni positive. Quindi indica quanto il modello è preciso quando predice un risultato positivo.
Il recall, misura la proporzione di casi positivi reali che sono stati correttamente identificati. Ovvero indica quanto il modello è efficace nell'identificare tutti i casi positivi reali.

Dai risultati si nota che la precisione (0.6537) è inferiore al recall (0.8085), indicando che il modello tende a fare più predizioni positive, alcune delle quali errate.

L'F1 score è la media armonica di precisione e recall, fornendo un singolo valore che bilancia entrambe le metriche. Esso è particolarmente utile quando si cerca un equilibrio tra precisione e recall, specialmente in dataset sbilanciati.
Il punteggio F1 di 0.7229 suggerisce un buon equilibrio tra precisione e recall.

Quindi nel nostro caso, essendo in uno stadio iniziale i risultati ottenuti sono promettenti e suggeriscono che il modello ha una buona capacità predittiva, ma c'è spazio per miglioramenti, in particolare nella riduzione dell'overfitting e nell'aumento della precisione.

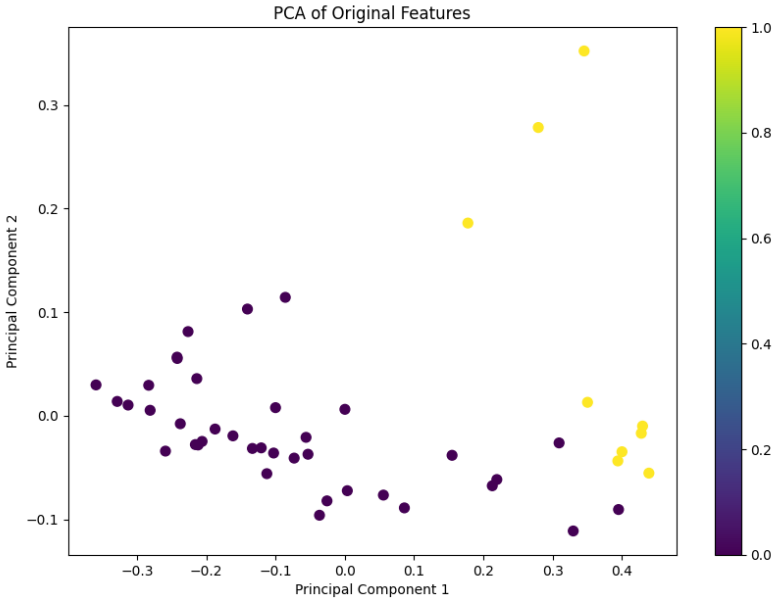


Figura 5.4: PCA sui dati originali

Analizzando il grafico prodotto dalla PCA sui dati originali nella figura 5.4, si nota innanzitutto che sono state riconosciute le due classi del dataset e che i dati sono stati disposti correttamente tra le componenti principali.
Inoltre è possibile appurare che i dati di test sono costituiti per lo più da campioni di classe OS<2, questo è stato uno dei campanelli di allarme che ci ha portato successivamente al bilanciamento del dataset.

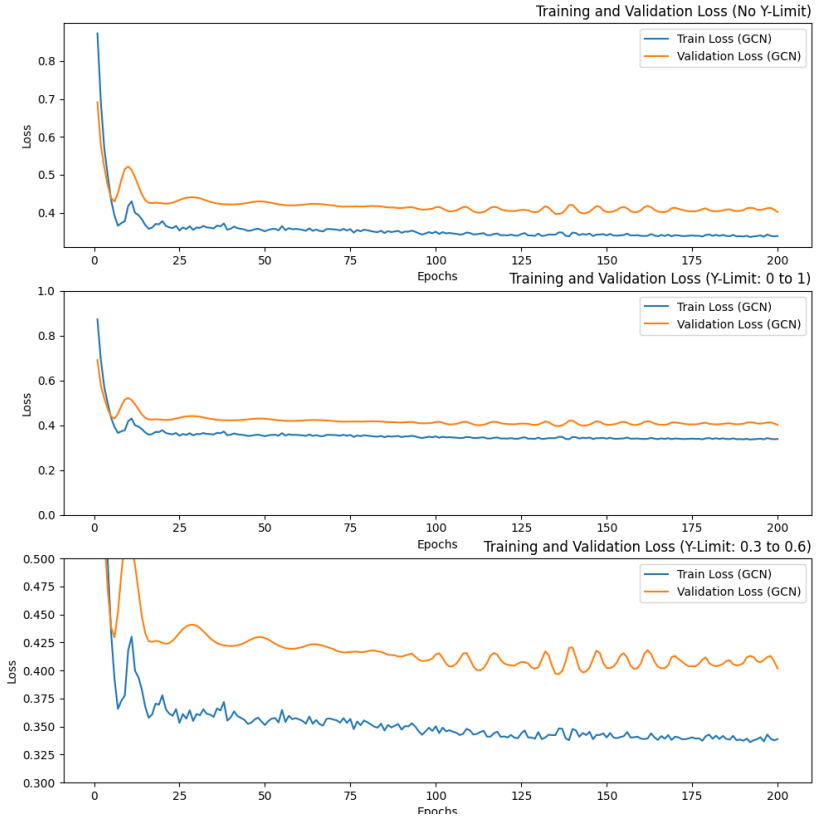


Figura 5.5: Andamento della funzione di Loss durante l'addestramento del modello

Nel grafico in Figura 5.5, possiamo osservare l'andamento della funzione di Loss durante l'addestramento del modello, in particolare nella fase di training e di validazioni.

Nell'asse delle ordinate troviamo la Loss, nell'asse delle ascisse troviamo il numero di epoche.
Sono riportati tre diversi sub-plot, ognuno con un limite diverso per il valore del target per facilitarne la visione.

Come è possibile vedere dalla figura, la Loss inizialmente assume valori alti, mentre dopo circa 50 epoche si stabilizza intorno ai 0,35 per la fase di train e 0.42 per la fase di validation, per poi arrivare rispettivamente a 0.33 e 0.40.

Da qui possiamo capire che il modello impara solamente nella primissima fase, tutto il resto non porta a grossi miglioramenti, dato che i valori rimangono costanti, questo porterà a modifiche di struttura e configurazioni future.

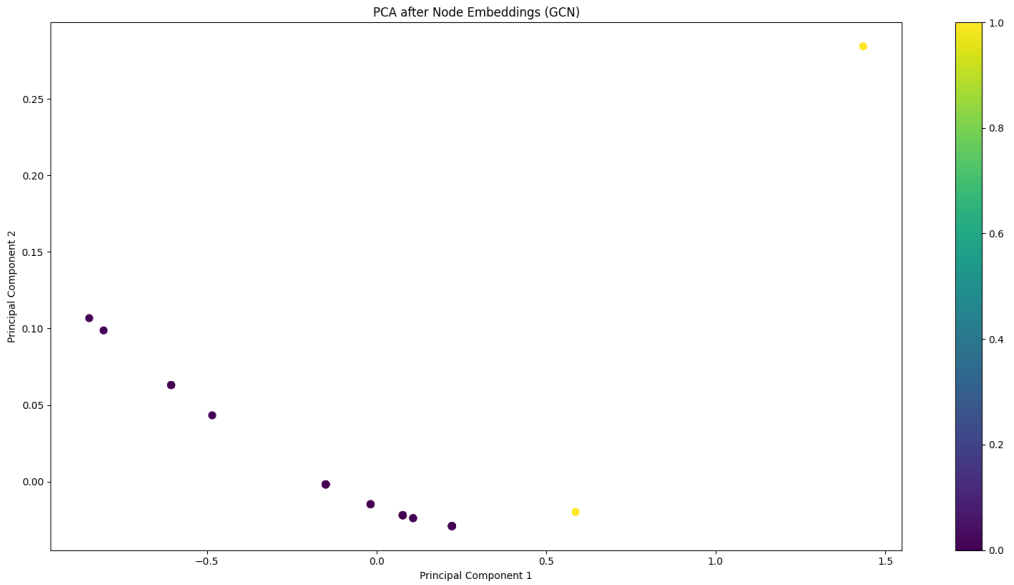


Figura 5.6: PCA dopo la classificazione con GCN

Dall'analisi di PCA dopo aver eseguito la classificazione col modello GCN, come rappresenta la Figura 5.6, possiamo osservare una separazione più netta tra i cluster, ma comunque i punti sono distribuiti in modo sparso e irregolare, con una concentrazione maggiore nell'area inferiore sinistra del grafico. Questa distribuzione non uniforme ci conferma una rappresentazione sbilanciata delle classi nel dataset iniziale.

La separazione tra le classi (rappresentate dai colori giallo e viola) è poco definita, con alcuni punti di classi diverse che si sovrappongono o sono molto vicini tra loro.
Essi occupano un'area limitata del grafico, principalmente concentrata intorno all'origine e leggermente estesa verso destra e hanno densità irregolare, con alcune aree molto dense e altre quasi vuote.

Inoltre, la scala delle componenti principali è limitata, questo suggerisce una minor varianza catturata da queste componenti.

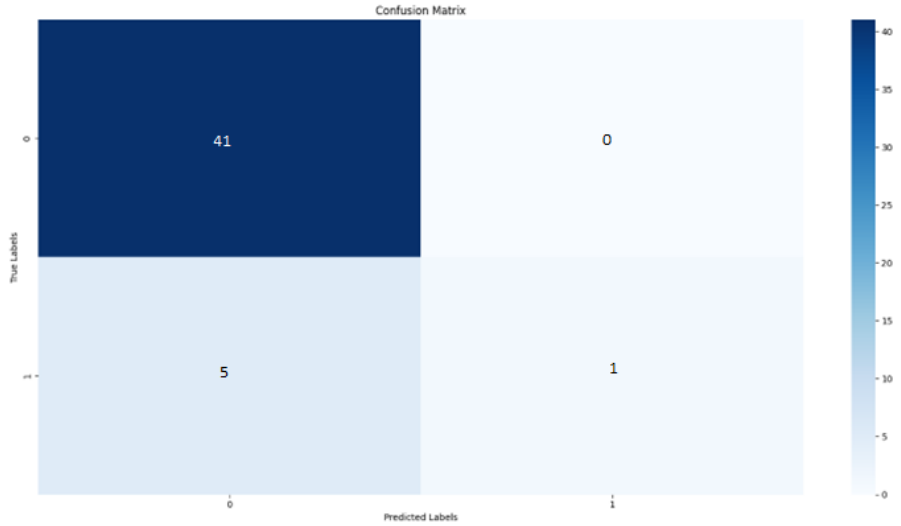


Figura 5.7: Matrice di confusione

Il grafico nella figura 5.7, rappresenta una matrice di confusione (confusion matrix)

Come possiamo osservare dalla figura, il modello ha classificato correttamente 41 istanze della classe 0 (Veri positivi) e 1 istanza della classe 1 (veri negativi).
Non ci sono casi in cui il modello ha erroneamente previsto la classe 0 quando era in realtà 1.(falsi positivi) ma ci sono 5 casi in cui il modello ha erroneamente previsto la classe 1 quando era in realtà 0. (falsi negativi)

Analizzando questi risultati si evince che il modello ha ottimi risultati nel classificare la classe 0 (41/46) mentre scarsi risultati nel classificare la classe 1 (1/6).

Da questo punto era ormai chiaro il forte sbilanciamento del dataset, con molti più esempi della classe 0 rispetto alla classe 1.

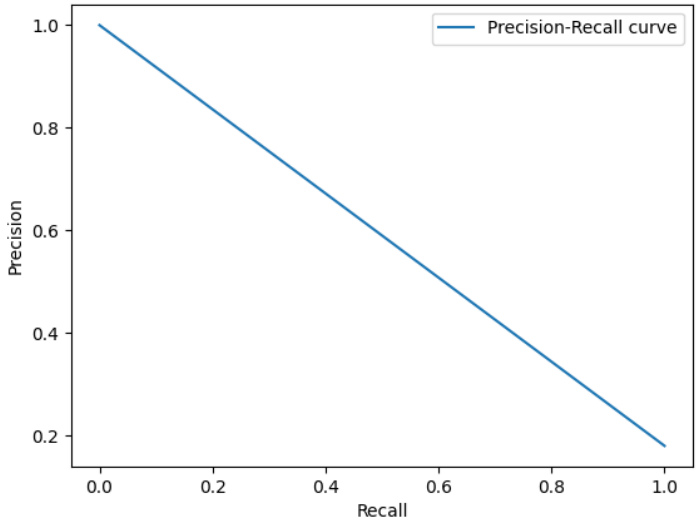


Figura 5.8: Curva Precision-Recall

La Precision-Recall curve mostrata in Figura 5.8, descrive il compromesso tra Precision e Recall.

In questo caso, la curva mostra che il modello ha un'elevata precisione ma un basso recall. Ciò significa che il modello è molto bravo a identificare gli elementi rilevanti, ma non è molto bravo a identificare tutti gli elementi rilevanti. La curva è lineare, il che significa che la precisione e il richiamo sono inversamente proporzionali tra loro. All'aumentare della precisione, il richiamo diminuisce e viceversa.

5.2.2 Risultati con dataset ENTREZ.ID.gene.expression.metabric.os

Nel corso della ricerca, si è deciso di concentrare l'attenzione sul dataset ENTREZ.ID.gene.expression.metabric.os, designato come "dataset 2". Questa scelta è stata motivata da due fattori principali:

- Esso offre una base di dati significativamente più ampia e dettagliata rispetto al precedente. Questa maggiore dimensionalità fornisce un terreno più fertile per analisi approfondite e potenzialmente più significative dal punto di vista statistico.
- Contiene informazioni quantitative sull'espressione genica, anziché limitarsi a indicare la mera presenza o assenza di specifici geni. Questo approccio quantitativo offre una rappresentazione più sfumata e biologicamente rilevante dell'attività genica

Inoltre, per garantire coerenza e comparabilità con le analisi precedenti, il codice e i parametri utilizzati rimarranno invariati rispetto a quanto descritto nel paragrafo precedente.
Di seguito ne sono riportati i risultati:

```
Epoch: 0200 loss_train: 0.4430 acc_train: 0.8403 loss_val: 0.3087 acc_val: 0.9231 time: 0.0430s
Model training is complete!
Total model training time: 7.7110s
Test set results: loss= 0.3889 accuracy= 0.8723 f1= 0.8129 precision= 0.7610 recall= 0.8723
```

Figura 5.9: Risultati ottenuti con il dataset ENTREZ.ID.gene.expression.metabric.os

Fase di addestramento:
- Loss di Train: 0,4430
- Accuracy di Train: 0,8403
- Loss di Validation: 0,3087
- Accuracy di Validation: 0,9231

Fase di Test:
- Loss: 0,3889
- Accuracy: 0,8723
- F1-Score: 0,8129
- Precision: 0,7610
- Recall: 0,8723

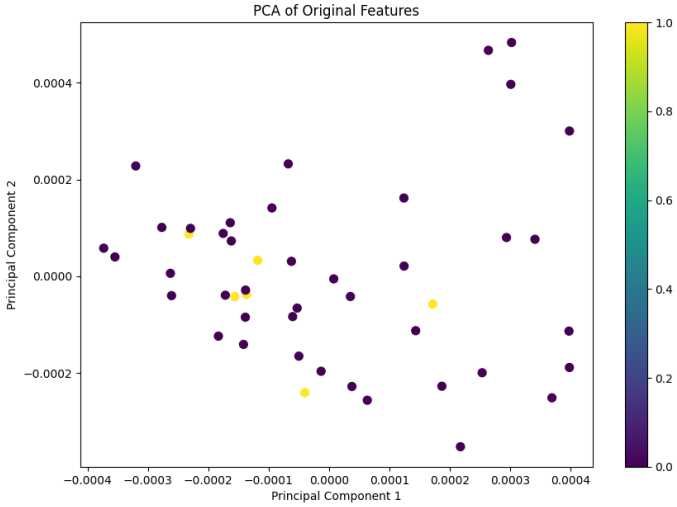


Figura 5.10: PCA sui dati originali

Tramite la PCA visibile nella Figura 5.10, infatti, risulta evidente lo sbilanciamento tra le classi, in quanto la maggior parte dei campioni fanno parte della classe OS<2

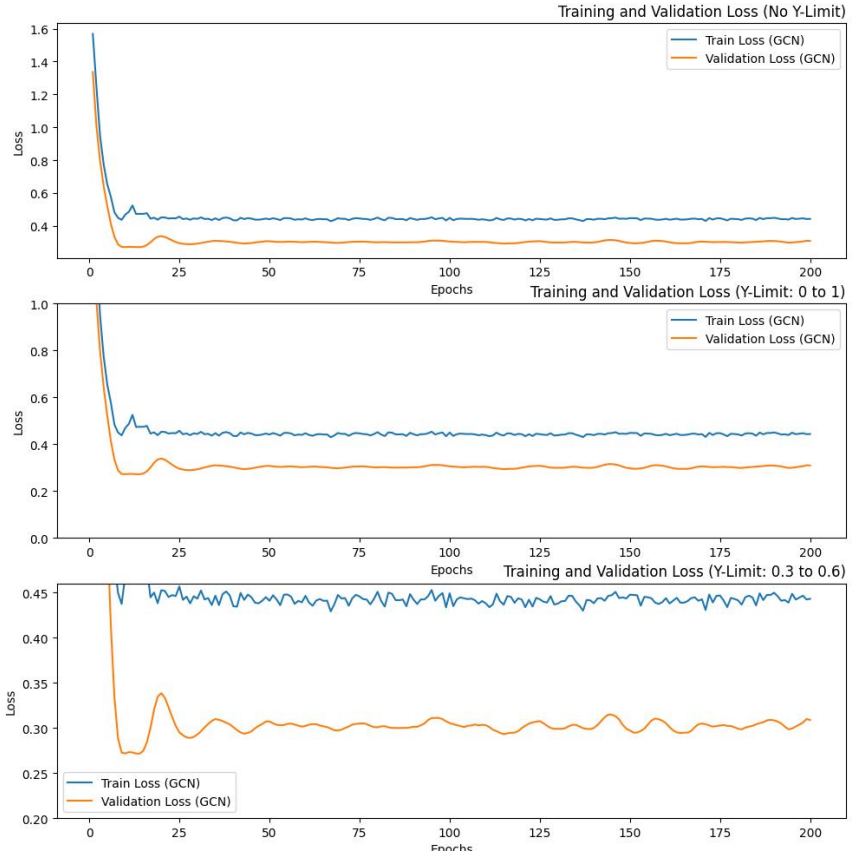


Figura 5.11: Andamento della funzione di Loss durante l'addestramento del modello

La figura 5.11 descrive il comportamento della funzione di Loss durante la fase di addestramento del modello, è possibile vedere che durante le epoche c'è un lieve miglioramento solamente nella prima fase (25 epoche), successivamente il valore rimane costante, il che sta a indicare che non c'è miglioramento.

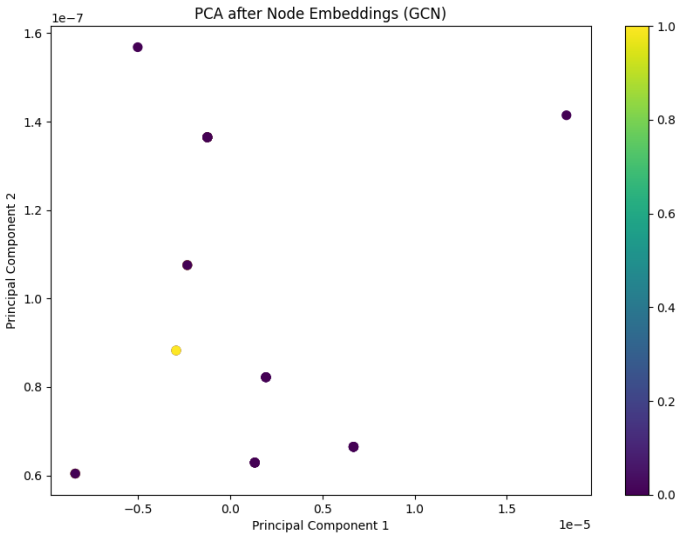


Figura 5.12: PCA dopo la classificazione con GCN

Dal grafico della PCA eseguito dopo la classificazione del modello, visionabile in figura 5.12, notiamo che il modello non ha imparato a classificare i dati, in quanto essi sono sparsi e comunque ancora se ne risalta lo sbilanciamento tra le classi

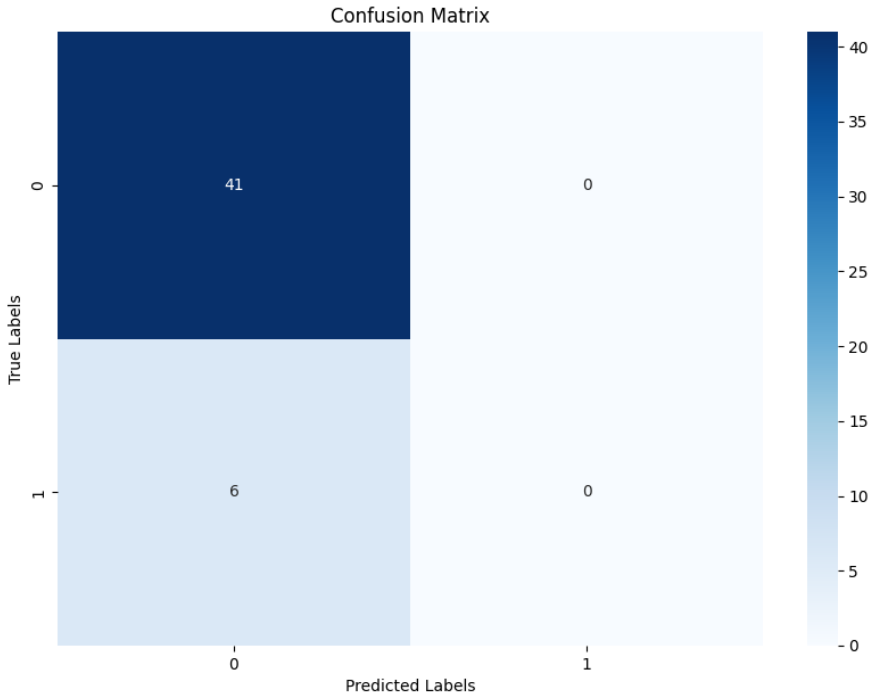


Figura 5.13: Matrice di confusione

Dalla confusion matrix in Figura 5.13 si evince in maniera importante lo sbilanciamento del dataset, o almeno della classe di test, in quanto ci sono solo elementi della classe 0. Nonostante questo, possiamo comunque notare che la classificazione di quella classe è andata molto bene, ovvero 41 elementi classificati correttamente su 47.

5.3 Bilanciamento del dataset

Dopo avere eseguito la PCA e la confusion matrix, ormai era evidente un forte sbilanciamento del dataset, ne sono la prova i seguenti risultati in cui si evidenziano il numero di elementi per ogni classe:

- (classe OS>=2): 183 campioni
- (classe OS<2): 30 campioni

Quindi si è deciso di bilanciarlo, in modo tale da non favorire nessuna classe e di conseguenza evitare di falsare la classificazione.

Dopo un'attenta analisi tra le tecniche utilizzabili, è stato deciso di applicare lo Smote per sovra-campionare la classe minoritaria, considerando che il dataset in questione è relativamente piccolo e fortemente sbilanciato, si è presentata come l'opzione migliore adatta a noi.

I risultati così ottenuti sono stato un aumento della classe OS<2, da 30 campioni a 183, in modo tale da passare in totale da 213 campioni a 366.

Successivamente il dataset è stato stratificato in maniera bilanciata anche tra i gruppi di train (50%), validation (20%) e test (30%).

Questo ha comportato un miglioramento importante alle prestazioni del modello, lasciando invariata la configurazione dello stadio precedente, si sono ottenuti questi risultati:

Epoch: 0200 loss_train: 0.2664 acc_train: 0.9203 loss_val: 0.2390 acc_val: 0.9224 time: 0.0530s

Test set results: loss= 0.2038 accuracy= 0.9345 f1= 0.9345 precision= 0.9347 recall= 0.9345

È possibile notare a colpo d'occhio la Loss ottenuta in tutti e tre le fasi della classificazioni, nettamente inferiore alla fase precedente, ma anche tutte le altre metriche tra cui accuracy precisione ecc.. sono nettamente migliorate.

5.4 Modifiche ai Cluster K-means e Visualizzazione del Grafo

Successivamente, sono stati modificati il numero di cluster dell'algoritmo K-means utilizzato per creare la rete di pazienti, per l'esattezza da 50 a 12.

Questo valore è stato calcolato tramite i metodi grafici Elbow e Silouette Score, entrambi tecniche utilizzate per determinare il numero ottimale di cluster da utilizzare in algoritmi come nel nostro caso il K-means.

Il primo metodo, ovvero Elbow Method o metodo del gomito, ha come idea principale quella di eseguire l'algoritmo di clustering per un range di valori di k (numero di cluster), calcolare una metrica di valutazione per ogni k e individuare il punto in cui la diminuzione di questa metrica rallenta drasticamente, formando un "gomito" nel grafico.

La metrica comunemente utilizzata è la somma delle distanze quadratiche intra-cluster (SSE, Sum of Squared Errors). La SSE è definita come la somma delle distanze quadratiche tra i punti di un cluster e il loro centroide. Dato un cluster Ci con centroide μi , la SSE è data da:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{5.1}$$

Il valore di *k* in corrispondenza del quale la SSE inizia a ridursi lentamente è considerato il numero ottimale di cluster, poiché indica che l'aggiunta di ulteriori cluster non migliora significativamente la qualità della segmentazione, ed esso è circa 12, come è possibile osservare nella Figura 5.14

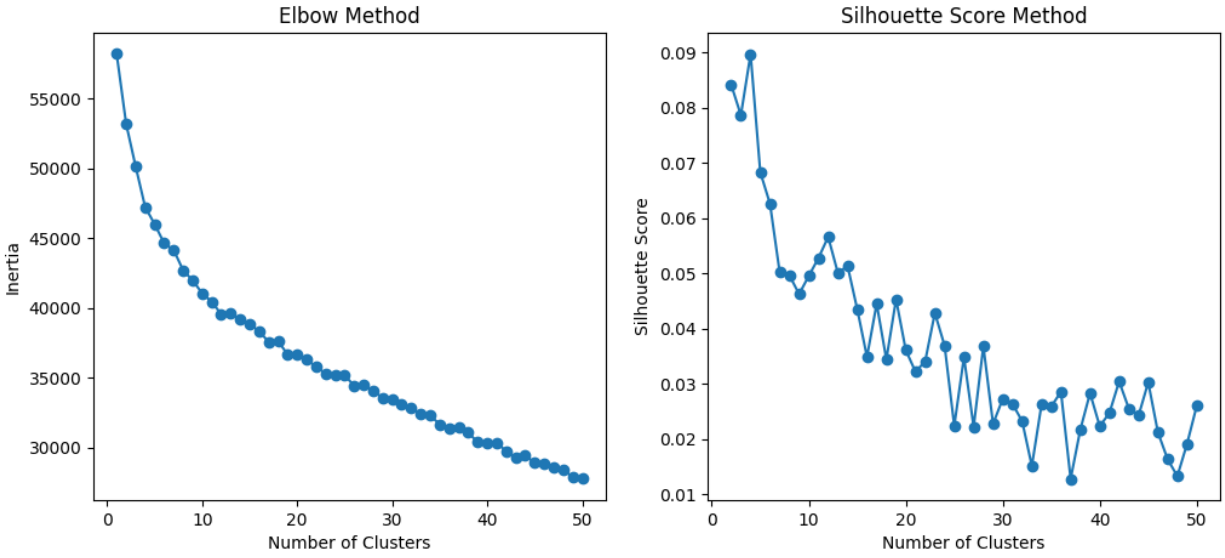


Figura 5.14: Elbow Method e Silhouette Score Method

A differenza del metodo del gomito, il Silhouette Score non si basa solo sulle distanze intra-cluster, ma considera anche le distanze inter-cluster, fornendo una misura di quanto bene un punto dati è raggruppato con i punti del suo cluster rispetto a quelli di altri cluster. Per un punto dati i, il Silhouette Score è definito come:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{5.2}$$

dove *a(i)* è la distanza media tra *i* e tutti gli altri punti del suo cluster, e *b(i)* è la distanza media tra *i* e tutti i punti del cluster più vicino a cui i non appartiene.
Inoltre, il campionamento del dataset è stato ridotto a 500 campioni, permettendo un'analisi più gestibile e una visualizzazione più chiara del grafo. Queste modifiche sono state testate anche su una macchina virtuale più prestante, con l'obbiettivo di utilizzare il maggior numero di campioni del dataset, ma con scarsi miglioramenti.

I risultati ottenuti sono stati una rete di pazienti incrementata da 2200 nodi a 3000, e ciò fa intendere non sono un numero maggiore di connessioni tra i nodi della rete, ma anche raggruppati con un criterio migliore grazie alla modifica del numero di cluster.

5.5 Sperimentazione sugli ottimizzatori

La fase successiva ha visto l'aggiunta di nuove funzionalità e tecniche di ottimizzazione. Il grafo è stato plottato con i nodi colorati in base alla sopravvivenza dei pazienti (>2 anni in verde, <2 anni in rosso). Questo ha permesso di visualizzare l'output del modello in termini di prognosi dei pazienti, come è possibile osservare nella Figura 5.15, anche se la rete troppo densa non permette la visualizzazione in alcuni punti della figura.

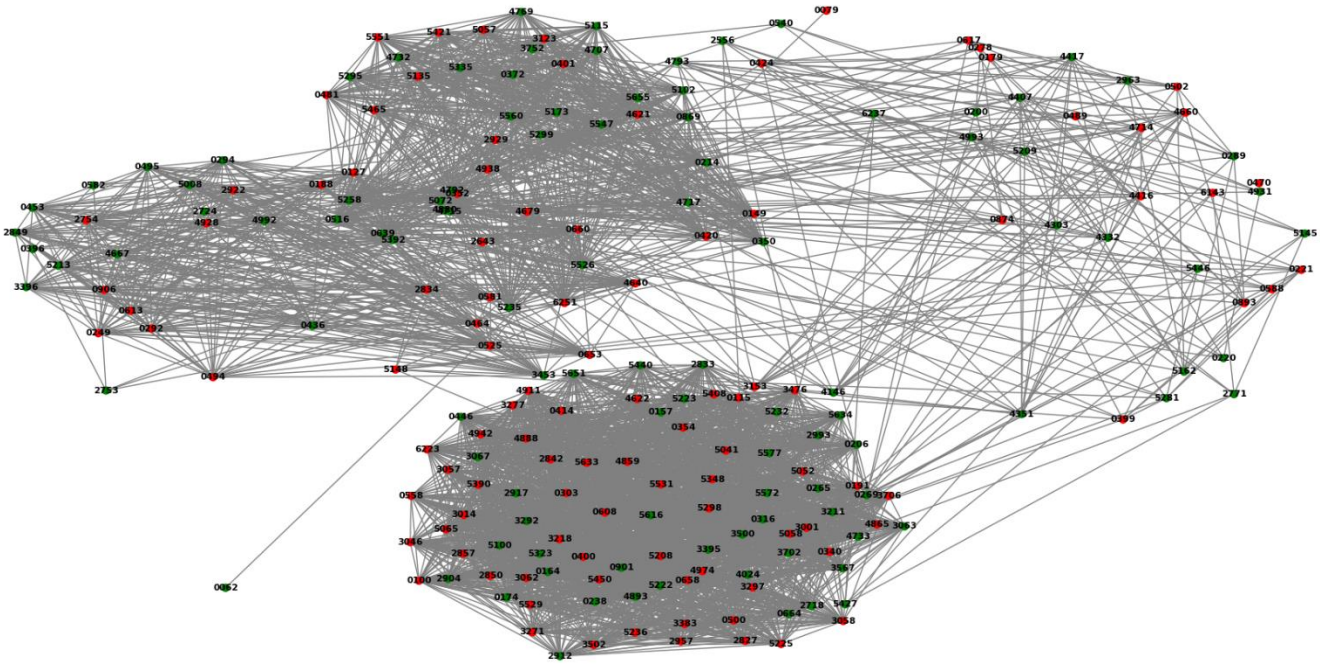


Figura 5.15: Plot della rete di pazienti tramite la libreria networkx

Sono stati sperimentati ottimizzatori alternativi come RMSprop e Adagrad per migliorare la convergenza, oltre a utilizzare AdamW, una variante dell'Adam che include una migliore implementazione della regolarizzazione L2.

Le migliori prestazioni sono state ottenute nella seguente struttura:

Ottimizzatore	*Adam*	*AdamW*	*RMSprop*	*Adagrad*
Epoche	200			
Max train accuracy	90,03 %	88,40 %	87,328 %	91,23 %
Min train loss	0,266	0,257	0,251	0,248
Max valid. accuracy	85,45 %	89,45 %	86,38 %	92,36 %
Min validation loss	0,239	0,248	0,256	0,229
Test accuracy	93,47 %	90,47 %	88,47 %	94,41 %
Test loss	0,203	0,227	0,235	0,191

Tabella 5.1: risultati dei quattro ottimizzatori testati

Come è possibile vedere dalla tabella i risultati migliori, dopo i numerosi test effettuati, sono stati ottenuti tramite l'ottimizzatore Adagrad.

5.6 Sperimentazione sulle funzioni di attivazione

Successivamente sono stati testati delle tecniche di normalizzazione dei dati, tra cui lo z-score, e delle implementazioni per evitare l'overfitting dei dati e tentare di far diminuire la Loss, ovvero early stopping e dell'adaptive learning rate, ma hanno peggiorato le prestazioni, indicando che tali tecniche potrebbero non essere adatte per questo specifico problema.

Lo step seguente di questo progetto è stato quello di sperimentare diverse funzioni di attivazione presenti all'interno della GCN nel metodo forward.

Come sappiamo le funzioni di attivazione sono cruciali per introdurre non-linearità negli strati finali dei modelli di rete neurale, perché permettono di apprendere relazioni complesse nei dati, trasformando l'output grezzo in una forma che può essere interpretata nella classificazione.

Le funzioni di attivazione testate per esaminarne l'efficienza sono state:

- torch.log_softmax
- torch.softmax
- torch.sigmoid
- torch.tanh
- F.softplus

Le migliori prestazioni sono state ottenute nella seguente struttura:

Funzione di attivazione	*Torch.log_softmax*	*Torch.softmax*	*Torch.sigmoid*	*Torch.tanh*	*F.softplus*
Epoche	200				
Max train accuracy	90,03 %	85,40 %	85,328 %	83,23 %	88,03 %
Min train loss	0,266	0,297	0,301	0,338	0,286
Max valid. accuracy	85,45 %	85,67 %	86,38 %	82,12 %	87,47 %
Min validation loss	0,239	0,248	0,256	0,229	0,239
Test accuracy	92,47 %	83,42 %	88,82 %	84,41 %	85,47 %
Test loss	0,216	0,312	0,314	0,357	0,245

Tabella 5.2: risultati delle 5 funzioni di attivazione sperimentate

Dai seguenti risultati visibili nella tabella, si evince che la migliore funzione di attivazione per il nostro caso è stata la *Torch.log_softmax.*

5.7 Aggiunta la normalizzazione della forward

Da questo punto in poi è stata applicata una piccola modifica alla funzione forward della classe Graph Convolutional Network. In particolare, è stata implementata la normalizzazione della matrice di adiacenza nella nostra classe di convoluzione.

Di seguito ne mostro il codice:

```
identity = torch.eye(adj.size(0)).to(adj.device)
adj_hat = adj + identity
D = torch.diag(torch.sum(adj_hat, dim=1))
D_inv_sqrt = torch.diag(torch.pow(D.diag(), -0.5))
adj_norm = torch.matmul(torch.matmul(D_inv_sqrt, adj_hat), D_inv_sqrt)
```

Queste, nello specifico operano nel seguente modo:

identity: Crea una matrice identità di dimensione uguale alla matrice di adiacenza adj.
adj_hat: va a sommare la matrice identità identity alla matrice di adiacenza adj, producendo una nuova matrice adj_hat.
Questo procedimento che aggiunge la matrice identità alla matrice di adiacenza, serve a includere connessioni auto-referenzianti (self-loops) per ogni nodo, il che è utile per mantenere l'informazione del nodo stesso durante la convoluzione.

D: Calcola la matrice delle somme delle righe di adj_hat e poi crea una matrice diagonale D con queste somme.
Essa è una matrice che rappresenta il grado dei nodi nella rete, ovvero il numero di connessioni (inclusi i self-loops) che ogni nodo possiede.

D_inv_sqrt: Calcola l'inverso della radice quadrata della diagonale di D, creando una nuova matrice diagonale D_inv_sqrt.
Essa viene utilizzata per normalizzare adj_hat, bilanciando l'influenza dei nodi in base al loro grado.

adj_norm: Esegue la normalizzazione simmetrica della matrice adj_hat moltiplicandola a sinistra e a destra con D_inv_sqrt.

Tutto questo processo di normalizzazione evita problemi numerici durante il training, come gradienti esplosivi o vanificanti.
Inoltre assicura che le informazioni si propaghino in modo uniforme attraverso il grafo, evitando che i nodi con un alto grado dominino quelli con un basso grado.

Quindi alla fine la capacità del modello di generalizzare su nuovi dati viene migliorata, in quanto le connessioni normalizzate garantiscono una rappresentazione più bilanciata dei nodi.

I risultati di questa fase evidenziano un miglioramento delle prestazioni, nella tabella seguente è possibile vedere un confronto tra il modello senza e con la normalizzazione adesso presentata (lasciando invariate sempre e configurazioni iniziali)

Tipo di modello	*Senza normalizzazione*	*Con normalizzazione*
Epoche	200	
Max train accuracy	90,03 %	94,58 %
Min train loss	0,266	0,149
Max valid. accuracy	85,45 %	84,67 %
Min validation loss	0,239	0,243
Test accuracy	92,47 %	94,55 %
Test loss	0,216	0,149

Tabella 5.3: risultati del modello eseguito con e senza normalizzazione

5.8 Test massivi sulle configurazioni ottimali

Nell'ultima fase, il codice ha subito ulteriori rifinimenti e miglioramenti.
Sono stati fatti interventi di refactoring e ingegnerizzazione del codice per migliorarne la leggibilità e la manutenibilità. La funzione di loss è stata standardizzata utilizzando la cross entropy. Essendo un problema di classificazione binaria è la più indicata.

Inoltre, è stato implementato uno script in pyhton per testare in maniera automatica tutte le configurazioni possibili:

```
param_grid = {
  'epochs': [400, 800, 1200],
  'lr': [0.0001, 0.00001, 0.0005, 0.00005, 0.001, 0.005],
  'weight_decay': [4e-5, 2e-5, 4e-6, 4e-7, 4e-3, 4e-2],
  'n_hid': [[64, 32], [128, 64], [32, 16]],
  'dropout': [0.5, 0.4, 0.3, 0.6, 0.7]
}
```

Questo è un esempio che evidenzia il cluster di possibili opzioni utilizzate, una volta definiti questi insiemi, il codice automatizzato genererà tutte le possibili combinazioni di questi parametri.

Il codice poi eseguirà l'addestramento e la valutazione del modello per ciascuna di queste configurazioni, registrando le prestazioni.
Per l'esattezza, una volta creata la configurazione del test da eseguire, esso viene salvato in un file di testo, ed ha come nome del file i valori delle metriche del ultima epoca, questo per poter individuare già da subito le migliori run, e come contenuto del file tutto l output eseguito in console, quindi i risultati di tutte le epoche per intero.

Questo approccio è stato particolarmente utile perché non si era sicuri dei valori ottimali per i parametri del modello, e nello stesso momento si voleva evitare di trascurare una configurazione potenzialmente migliore.

5.9 Migliori risultati ottenuti

Sono stati eseguiti oltre le due migliaia di test in totale, e la configurazione ottimale è stata la seguente:

```
class Args:
 no_cuda = False
 seed = 42
 epochs = 800
 lr = 0.0001
 weight_decay = 4e-05
 n_hid = [128, 64]
 dropout = 0.3
```

Come classificatore della funzione forward del modello è stato utilizzato il `log_softmax`, come ottimizzatore l'`Adam` e come funzione di loss la `cross_entropy`.

Risultati ottenuti:

```
Epoch: 0800 loss_train: 0.0675 acc_train: 0.9754 loss_val: 0.2239 acc_val:
0.8835 time: 0.0140s
```

È evidente che ci sono dei netti miglioramenti dalla configurazione precedente, ad iniziare dalla fase di addestramento la Loss di train (0.0675) è scesa drasticamente, anche quella di validazione si è abbassata ma con una lieve discrepanza, questo fa intendere che è rimasto un certo grado di overfitting, comunque non elevato.
Nonostante ciò, l'accuratezza di validazione rimane comunque buona, attestandosi all'88.35%.

Le metriche ottenute sul set di test forniscono una valutazione completa delle prestazioni del modello su dati non visti durante l'addestramento:

```
Test set results: loss= 0.0965 accuracy= 0.9636 f1= 0.9636 precision= 0.9643
recall= 0.9636
```

1. Accuratezza: Il modello classifica correttamente il 96.36% delle istanze del set di test, indicando un'elevata capacità predittiva su dati non visti durante l'addestramento.
2. F1-Score: Il valore di 0.9636 rappresenta una media armonica bilanciata tra precisione e recall, confermando le ottime prestazioni del modello in termini di equilibrio tra falsi positivi e falsi negativi.
3. Precisione: Con un valore del 96.43%, il modello dimostra un'alta affidabilità nelle sue previsioni positive, con una bassa percentuale di falsi positivi.
4. Recall: Il 96.36% indica che il modello è in grado di identificare correttamente la gran maggioranza delle istanze positive reali, minimizzando i falsi negativi.

La coerenza tra queste metriche sul set di test ci suggerisce un modello robusto e ben bilanciato, capace di generalizzare efficacemente sui nuovi dati.

È importante notare che la loss sul set di test (0.0965) è significativamente inferiore rispetto alla loss di validazione (0.3139) osservata durante l'addestramento. Questo potrebbe indicare che il modello ha beneficiato dell'addestramento prolungato (800 epoche), riuscendo a

migliorare le sue prestazioni anche oltre il punto in cui si osservava overfitting sul set di validazione.

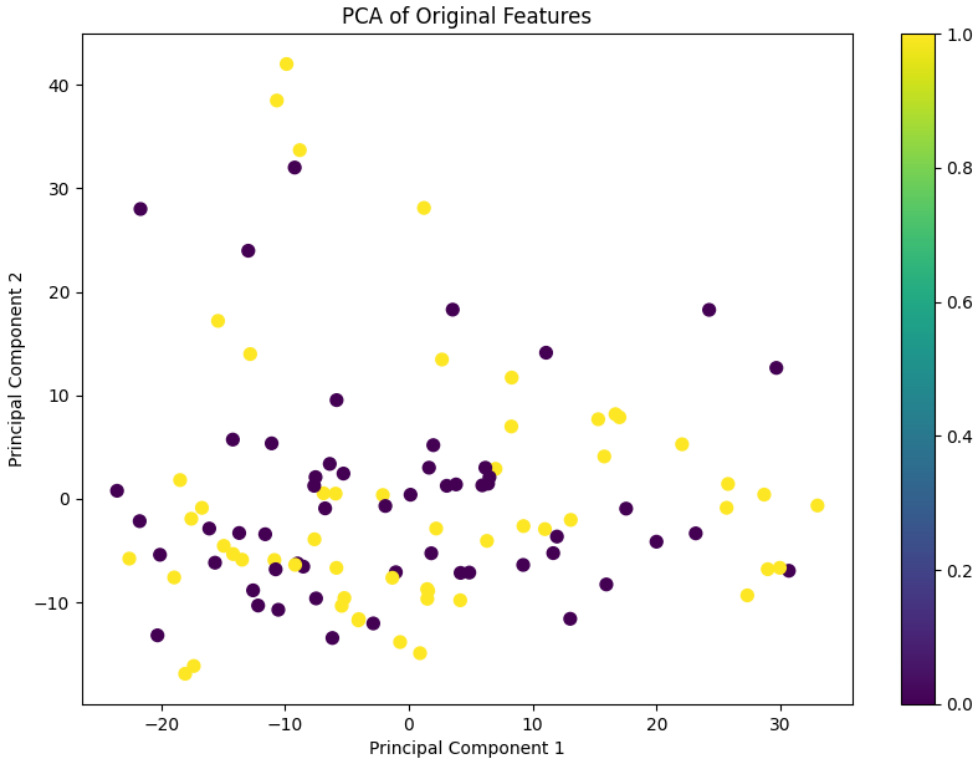


Figura 5.16: PCA sui dati originali dopo il bilanciamento e la stratificazione

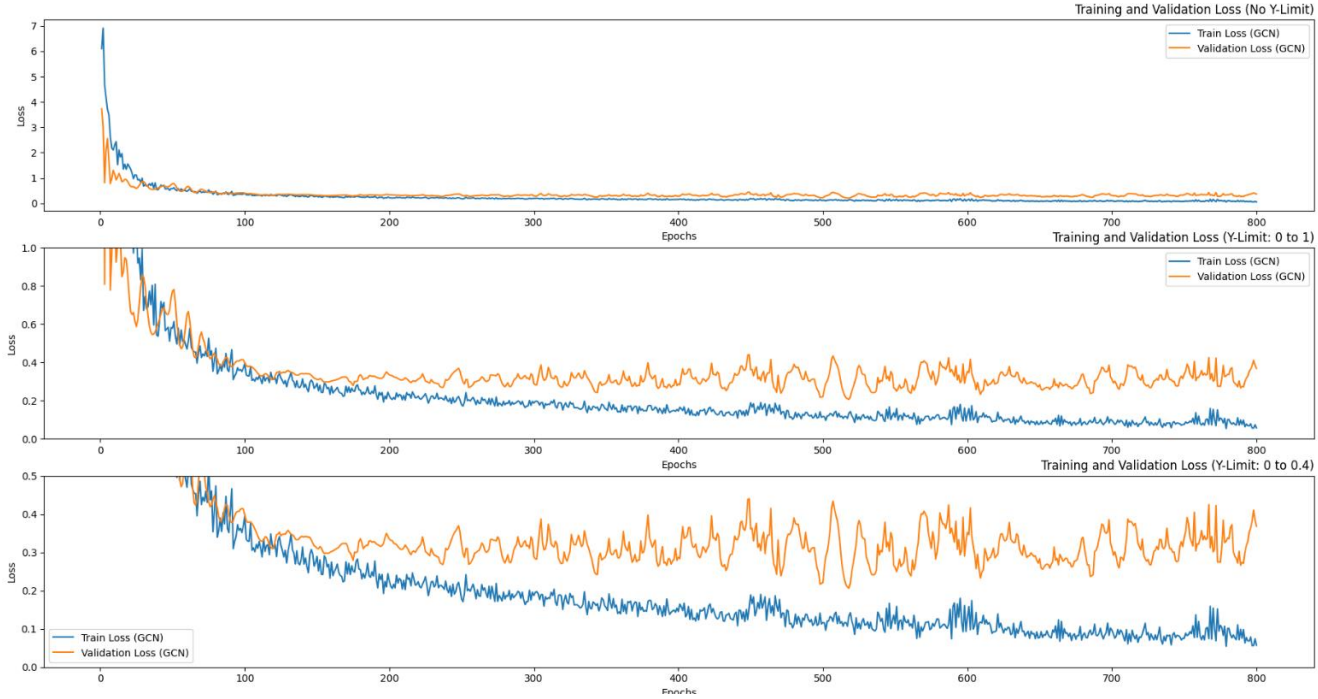


Figura 5.17: Discesa della funzione di Loss durante la fase di addestramento del modello

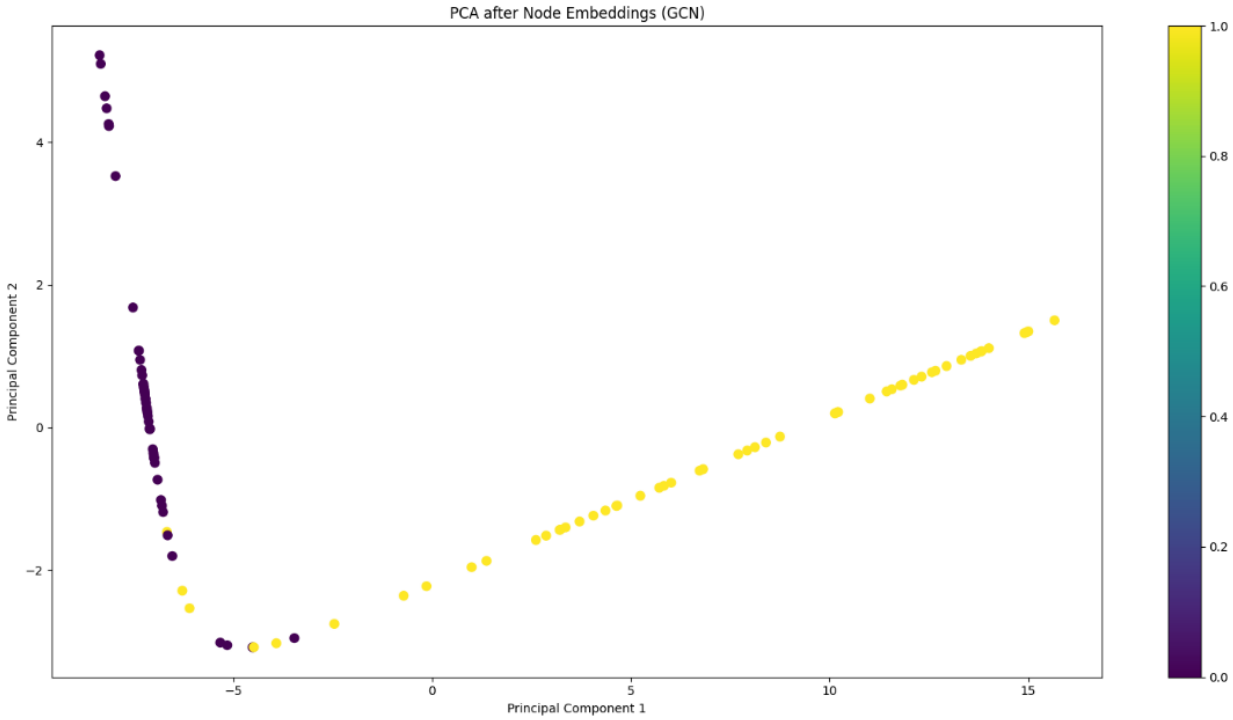


Figura 5.18: PCA dopo la classificazione con GCN

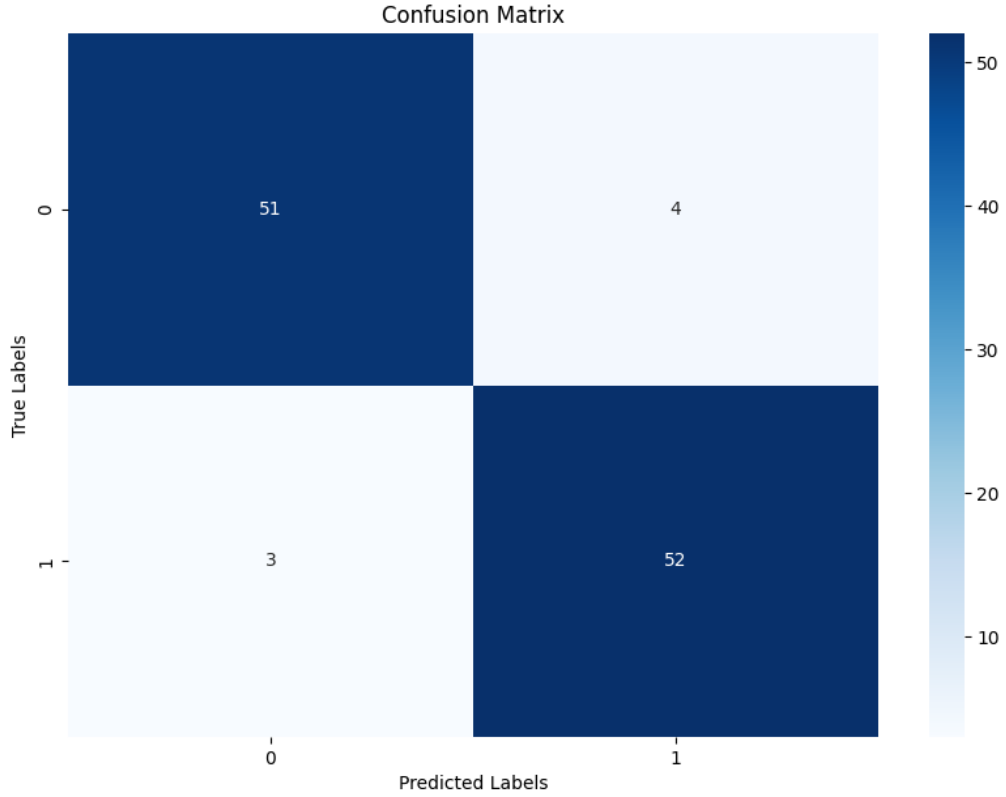


Figura 5.19: Matrice di confusione

L'analisi della figura 5.16 illustra i risultati dell'Analisi delle Componenti Principali (PCA) applicata al dataset bilanciato e stratificato.
In particolare, si osserva una notevole equalizzazione nella rappresentazione delle diverse classi all'interno del dataset e risulta evidente che la nuova configurazione presenta una distribuzione molto più uniforme del numero di campioni per ciascuna classe, migliorando sostanzialmente l'equilibrio complessivo del dataset.

La figura 5.17 illustra l'andamento della funzione di Loss durante le fasi di addestramento e validazione del modello. Si possono osservare due distinti comportamenti:

- Per quanto riguarda la fase di training, la curva mostra una discesa costante e graduale. Questo andamento è indicativo di un processo di apprendimento efficace, dove il modello continua a migliorare la sua capacità di adattarsi ai dati di addestramento con il progredire delle epoche. Tale comportamento suggerisce che il modello sta progressivamente affinando i suoi parametri per minimizzare l'errore sui dati di training.
- D'altra parte, la curva relativa alla fase di validazione presenta un comportamento differente. Dopo circa 200 epoche, il valore della Loss tende a stabilizzarsi, e ciò ci conferma un principio di over fitting notato precedentemente.

Analizzando la figura 5.18 che rappresenta i risultati della PCA applicata alle embeddings dei nodi ottenute con il modello, si osserva innanzitutto una distribuzione dei punti molto più strutturata e bilanciata. Essi formano una curva che si estende attraverso il piano, indicando una rappresentazione più equilibrata e differenziata delle classi.
È visibile una netta separazione tra le due classi principali; infatti, i punti viola sono concentrati principalmente nella parte superiore sinistra, mentre i punti gialli formano una curva che si estende verso la parte inferiore destra del grafico.

I punti coprono un'area molto più ampia del grafico, estendendosi significativamente sia sull'asse della Prima Componente Principale che su quello della Seconda Componente Principale e si osserva una densità più uniforme lungo la curva formata dai punti, questo sta a indicare una rappresentazione più equilibrata di tutte le istanze del dataset.

Questi miglioramenti sono dovuti principalmente alle tecniche di bilanciamento e stratificazione del dataset, hanno migliorato significativamente la capacità del modello di catturare e rappresentare le caratteristiche distintive di ciascuna classe.

5.10 Tre modelli a confronto

Per scopo dimostrativo oltre la GCN, sono stati allenati altri due modelli che operano con i grafi, ovvero GraphSAGE (Graph Sample and Aggregate) e GAT (Graph Attention Networks) in modo tale da poterne confrontare i risultati.

Ecco alcuni dettagli aggiuntivi su questi due modelli:

GraphSAGE si distingue dalla GCN per il suo approccio di campionamento e aggregazione. Mentre la GCN considera tutti i vicini di un nodo, GraphSAGE seleziona un numero fisso di vicini per ogni nodo. Questo rende il modello più efficiente su grafi di grandi dimensioni.
Il processo di aggregazione in GraphSAGE è flessibile e può utilizzare diverse funzioni di aggregazione come media, massimo o LSTM. Questa flessibilità permette al modello di adattarsi a diverse strutture di grafo e tipi di dati.

Un altro vantaggio di GraphSAGE è la sua capacità di generalizzare a nodi non visti durante l'addestramento. Questo lo rende particolarmente utile in scenari dove il grafo è dinamico o in continua evoluzione.

Graph Attention Networks:
GAT introduce un meccanismo di attenzione che permette al modello di assegnare importanza diversa ai vari vicini di un nodo. Questo è particolarmente utile quando non tutti i collegamenti in un grafo hanno la stessa rilevanza per il compito in questione.

Il meccanismo di attenzione in GAT funziona calcolando un coefficiente di attenzione per ogni coppia di nodi connessi. Questi coefficienti sono poi normalizzati usando una funzione softmax, producendo pesi che sommano a 1 per ogni nodo.
GAT può utilizzare più "teste di attenzione", ognuna che impara un diverso set di coefficienti di attenzione. Questo permette al modello di catturare diverse forme di relazioni tra i nodi.

Tuttavia, queste potenziali migliorie vengono al costo di una maggiore complessità computazionale, soprattutto nel caso di GAT.

L'allenamento e il confronto di questi tre modelli permettono di valutare empiricamente quale approccio funziona meglio per il problema specifico in esame, considerando fattori come accuratezza, efficienza computazionale e capacità di generalizzazione.

Sono state adattate le configurazioni in modo tale da poter performare al meglio in tutti e 3 i modelli:

```
class Args:
 no_cuda = False
 seed = 42
 epochs = 1200
 lr = 0.0001
 weight_decay = 0.04
 n_hid = [128, 64]
 dropout = 0.3
```

Di seguito se ne riportano i risultati:

```
Epoch: 1200 | Model: GCN | loss_train: 0.0843 | acc_train: 0.9803 | loss_val:
0.2831 | acc_val: 0.8824 | time: 0.0630s
Epoch: 1200 | Model: GraphSAGE | loss_train: 0.0275 | acc_train: 0.9901 |
loss_val: 0.1697 | acc_val: 0.9020 | time: 0.5050s
Epoch: 1200 | Model: GAT | loss_train: 0.6977 | acc_train: 0.5025 | loss_val:
0.7031 | acc_val: 0.4902 | time: 0.7990s

Total model training time: 933.3294s

Test set results: loss= 0.1131 accuracy= 0.9455 f1= 0.9454 precision= 0.9460
recall= 0.9455
```

Tutti e tre i modelli sono strutturati con layer di convoluzione grafica, seguiti da attivazioni ReLU e dropout.
La funzione train itera su tutti i modelli, addestrando ciascuno separatamente ma in parallelo. Per ogni epoca e per ogni modello:

- Viene calcolato l'output del modello
- Si calcola la loss di training e la accuracy
- Si esegue il backpropagation e l'ottimizzazione
- Si valuta il modello sul set di validazione

La funzione di test valuta le prestazioni dei modelli sul set di test calcolando l'output di ciascun modello e facendone infine una media.

Di seguito sono presenti i grafici delle relative analisi come nelle fasi precedenti:

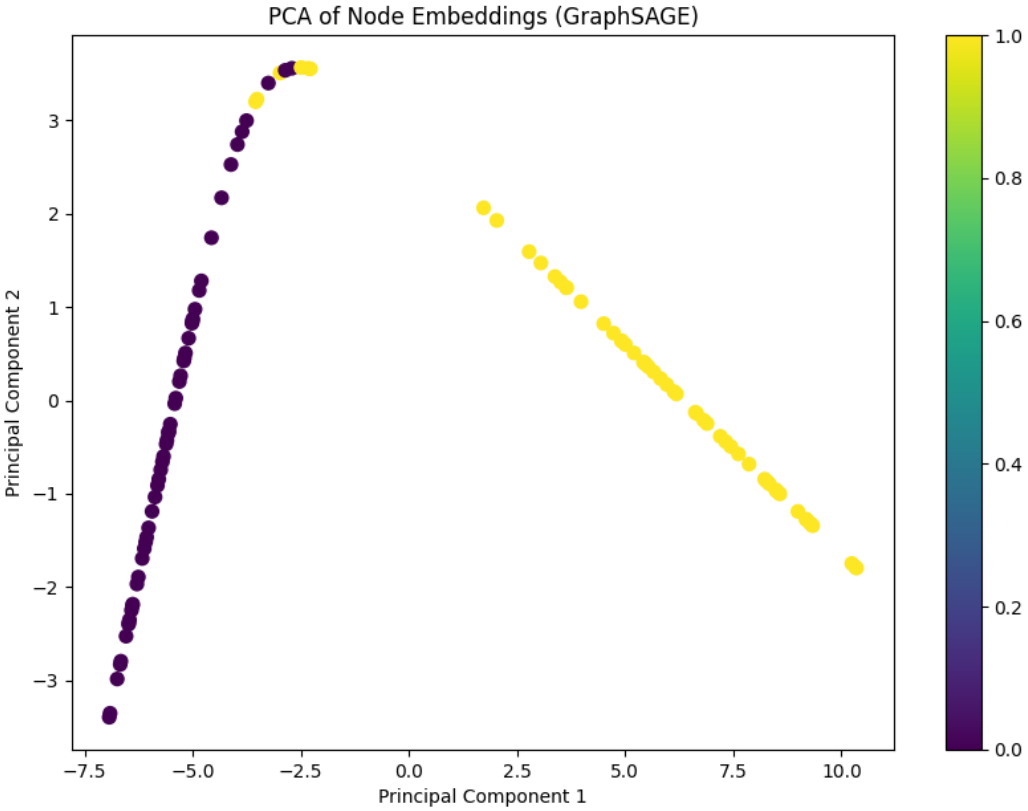


Figura 5.20: PCA dopo la classificazione con il modello GraphSAGE

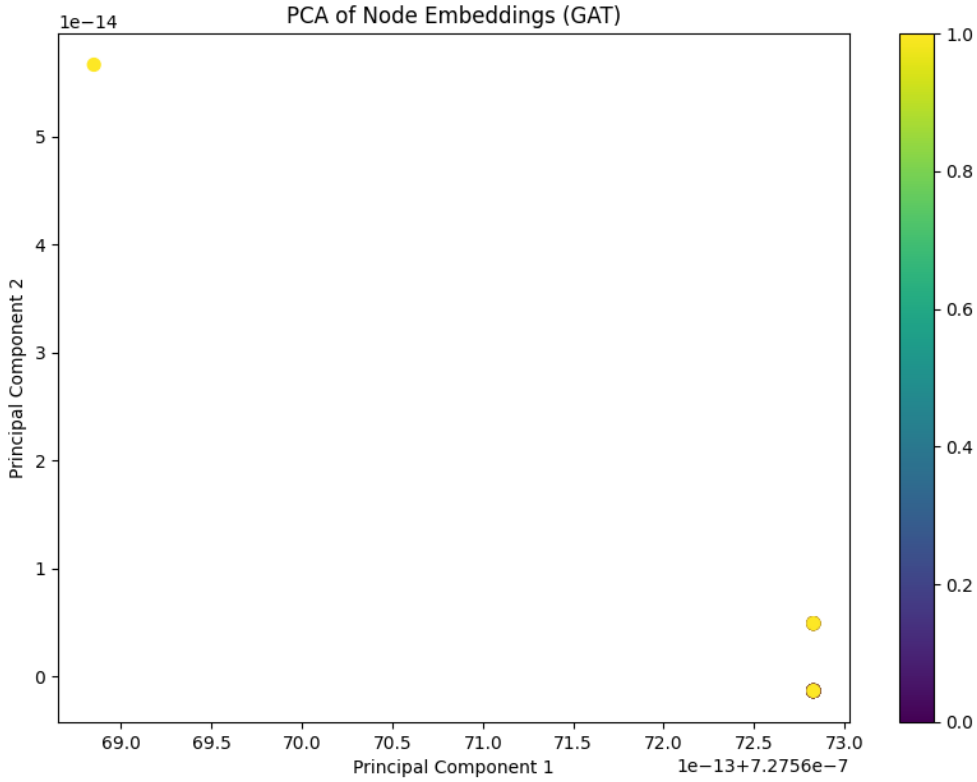


Figura 5.21: PCA dopo la classificazione con il modello GAT

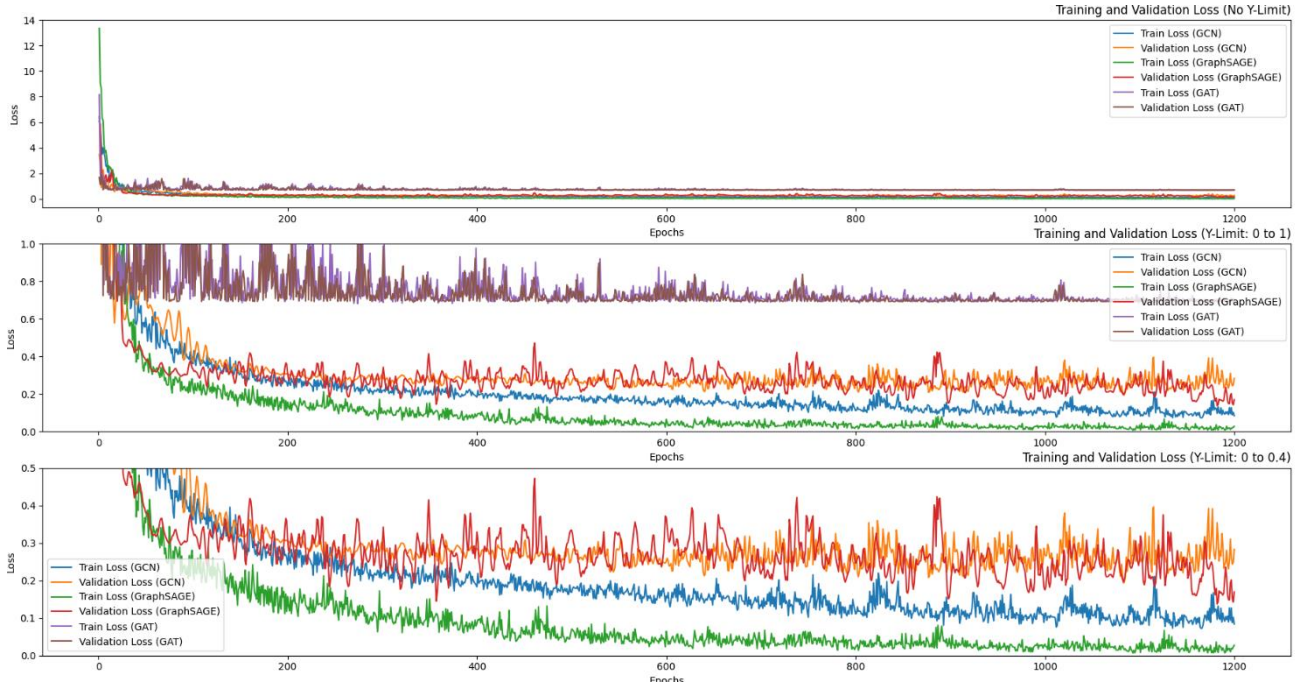


Figura 5.22: Discesa della funzione di Loss durante la fase di addestramento dei tre modelli a confronto

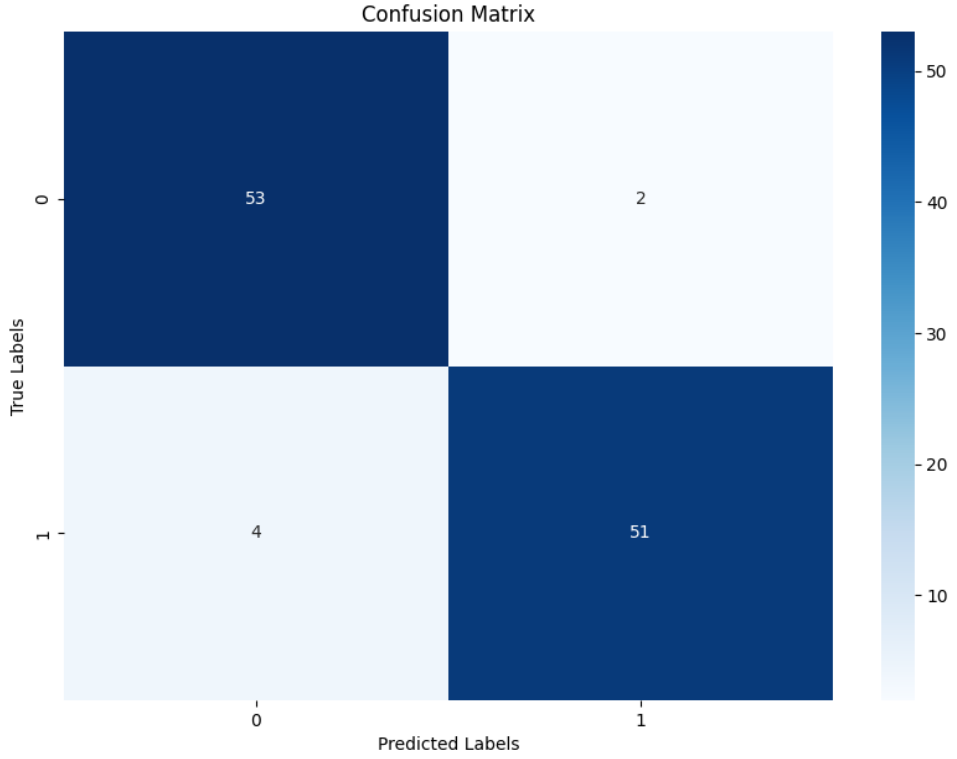


Figura 5.23: Matrice di confusione dei tre modelli nella fase di test

6 Conclusioni e sviluppi futuri

La presente tesi ha esplorato l'efficacia dei modelli di Graph Convolutional Networks (GCN) nell'ambito della stratificazione dei pazienti con patologie non trasmissibili, concentrandosi in particolare su pazienti col cancro al seno, utilizzando dati genomici e clinici dal dataset METABRIC. L'obiettivo principale era quello di sviluppare un modello robusto e preciso in grado di distinguere tra pazienti con prognosi favorevole e sfavorevole, sfruttando la potenza delle reti neurali convoluzionali applicate a dati strutturati a grafo.

Il percorso di sviluppo del modello ha affrontato diverse sfide significative, a partire dalla creazione stessa della rete di pazienti. Questo processo si è rivelato particolarmente impegnativo, richiedendo lo sviluppo di un algoritmo ad hoc per l'embedding della network. La costruzione di questa rete ha comportato la generazione di distribuzioni normalizzate per ogni paziente e per ogni gene, seguita da un processo di clustering per raggruppare i pazienti in base alla loro similarità genetica. Questa fase è stata cruciale per catturare le complesse relazioni tra i pazienti e le loro caratteristiche genomiche.

Un'altra sfida considerevole è stata l'adattamento dei dati al formato richiesto dal modello GCN. Questo ha richiesto una serie di trasformazioni e normalizzazioni dei dati, inclusa la creazione di una matrice di adiacenza normalizzata che rappresentasse la rete di interazioni tra geni. L'implementazione di queste trasformazioni ha richiesto una comprensione approfondita sia della struttura dei dati genomici che dei requisiti specifici delle GCN.

Inizialmente, l'analisi del dataset METABRIC ha rivelato un forte sbilanciamento tra le classi di sopravvivenza, con 183 campioni per la classe OS>=2 e solo 30 per la classe OS<2. Questa disparità ha richiesto l'implementazione di tecniche di bilanciamento, in particolare l'utilizzo dello SMOTE, che ha portato a un aumento dei campioni della classe minoritaria da 30 a 183, creando un dataset più equilibrato di 366 campioni totali.

Un altro aspetto critico è stato l'ottimizzazione della struttura della rete di pazienti. Attraverso l'applicazione dei metodi Elbow e Silhouette Score, il numero ottimale di cluster per l'algoritmo K-means è stato ridotto da 50 a 12, migliorando significativamente la qualità della segmentazione dei pazienti e aumentando il numero di connessioni nella rete da 2200 a 3000 nodi.

La sperimentazione con diversi ottimizzatori e funzioni di attivazione ha portato a risultati notevoli. In particolare, l'ottimizzatore Adagrad ha dimostrato le migliori prestazioni, raggiungendo un'accuratezza del test del 94,41% e una loss di 0,191. Tra le funzioni di attivazione, torch.log_softmax si è rivelata la più efficace, con un'accuratezza del test del 92,47% e una loss di 0,216.

L'introduzione della normalizzazione della matrice di adiacenza nella funzione forward della GCN ha rappresentato un punto di svolta, migliorando significativamente le prestazioni del modello. Questa modifica ha portato l'accuratezza di training dal 90,03% al 94,58% e ha ridotto la loss di training da 0,266 a 0,149.

Nella configurazione finale ottimizzata, il modello ha raggiunto risultati eccellenti sul set di test, con una Loss di 0,0965, un'accuratezza del 96,36%, un F1-score di 0,9636, una precisione del 96,43% e un recall del 96,36%. Questi risultati dimostrano la robustezza e l'efficacia del modello proposto nella stratificazione dei pazienti con cancro al seno.

Il confronto con altri modelli basati su grafi, come GraphSAGE e GAT, ha evidenziato le potenzialità della GCN. In particolare, GraphSAGE ha mostrato prestazioni comparabili, suggerendo che potrebbe essere un'alternativa valida in determinati contesti.

Un altro aspetto significativo riscontrato è l'efficienza computazionale del modello GCN, che, grazie al basso numero di parametri richiesti, risulta adatto anche per dispositivi con risorse limitate, ampliando così le possibili applicazioni pratiche in ambito clinico e di ricerca.

Gli sviluppi futuri di questo lavoro potrebbero concentrarsi su diversi aspetti promettenti.
Ad esempio, esplorare l'incorporazione di altri tipi di dati di sequenziamento del DNA, potrebbe arricchire ulteriormente il modello e migliorare la sua capacità predittiva o applicare il modello ad altre malattie non trasmissibili.

Migliorare l'efficienza computazionale del modello per gestire dataset più grandi e complessi potrebbe aumentarne la scalabilità, in particolare implementare tecniche di early stopping e dell'adaptive learning rate per cercare di abbassare l’overfitting del modello.
Oppure implementare tecniche di regolarizzazione avanzate come DropEdge o il pre-training.

Bibliografia

[1] - Alleviating the Inconsistency Problem of Applying Graph Neural Network to Fraud Detection - https://arxiv.org/abs/2005.00625

[2] - ASA: Adversary Situation Awareness via Heterogeneous Graph Convolutional Networks - https://dl.acm.org/doi/10.1145/3366424.3391266

[3] - A Semi-supervised Graph Attentive Network for Fraud Detection - https://ieeexplore.ieee.org/document/8970829

[4] - Key Player Identification in Underground Forums over Attributed Heterogeneous Information Network Embedding Framework - http://mason.gmu.edu/~lzhao9/materials/papers/lp0110-zhangA.pdf

[5] - Spam Review Detection with Graph Convolutional Networks - https://arxiv.org/abs/1908.10679

[6] - Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics - https://arxiv.org/abs/1908.02591

[7] - Uncovering Insurance Fraud Conspiracy with Network Learning - https://dl.acm.org/citation.cfm?id=3331184.3331372

[8] - FdGars: Fraudster Detection via Graph Convolutional Networks in Online App Review System - https://dl.acm.org/citation.cfm?id=3316586

[9] - GeniePath: Graph Neural Networks with Adaptive Receptive Paths - https://arxiv.org/abs/1802.00910

[10] - Heterogeneous Graph Neural Networks for Malicious Account Detection - https://dl.acm.org/citation.cfm?id=3272010

[11] - GraphRAD: A Graph-based Risky Account Detection System - https://www.mlgworkshop.org/2018/papers/MLG2018_paper_12.pdf

[12] - Graph neural networks learn emergent tissue properties from spatial molecular profiles, David Sebastian Fischer, Mayar Ali, Sabrina Richter, Ali Ertürk [https://www.researchgate.net/publication/366196049_Graph_neural_networks_learn_emergent_tissue_properties_from_spatial_molecular_profiles]

[13] - Semi-Supervised Graph Classification: A Hierarchical Graph Perspective - Jia Li1, Yu Rong, Hong Cheng, Helen Meng, Wenbing Huang, Junzhou Huang, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong [https://arxiv.org/abs/1904.05003]

[14] - Gated Graph Sequence Neural Networks, Li et al.

[15] - M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., volume 2, pages 729–734 vol. 2, July 2005.

[16] - F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, Jan 2009.

[17] - T. N. Kipf and M. Welling, "Variational graph auto-encoders," NIPS Workshop on Bayesian Deep Learning, 2016.

[18] D. V. Tran, A. Sperduti et al., "On filter size in graph convolutional networks," in SSCI. IEEE, 2018, pp. 1534–1541

[19] - Hierarchical Graph Representation Learning with Differentiable Pooling, https://arxiv.org/abs/1806.08804

[20] - A Comprehensive Survey on Graph Neural Networks, https://arxiv.org/1901.00596

[21] - Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li`o, and Yoshua Bengio. Graph attention networks. ArXiv, abs/1710.10903, 2017.

[22] - John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '18, page 1666–1674, New York, NY, USA, 2018. Association for Computing Machinery.

[23] - Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. CoRR, abs/1511.05493, 2016.

[24] - Boris Knyazev, Graham W. Taylor, and Mohammed Abdel Rahman Amer. Understanding attention and generalization in graph neural networks. In NeurIPS, 2019.

[25] - Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In ICML, 2019.

[26] - Matthias Fey, Jan Eric Lenssen - Fast Graph Representation Learning with PyTorch Geometric, https://arxiv.org/abs/1903.02428.