

# Sviluppo di una DGCNN per link prediction in una social network su Twitter

Corso di Analisi di Social networks

Corso di laurea magistrale in Ingegneria Informatica –  
Indirizzo Artificial Intelligence & Machine Learning

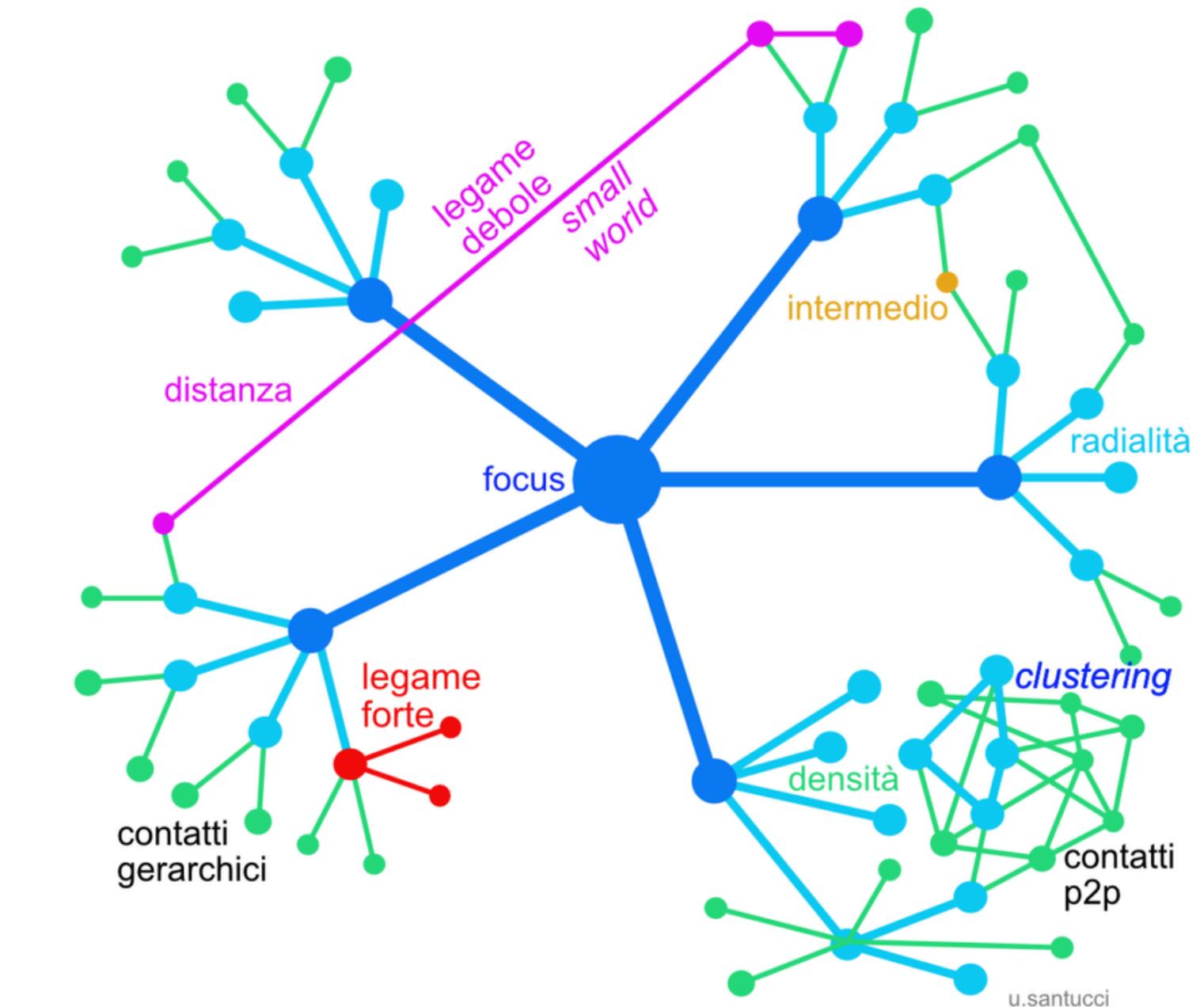
**Gaia Assunta Bertolino**

Mat. 242590



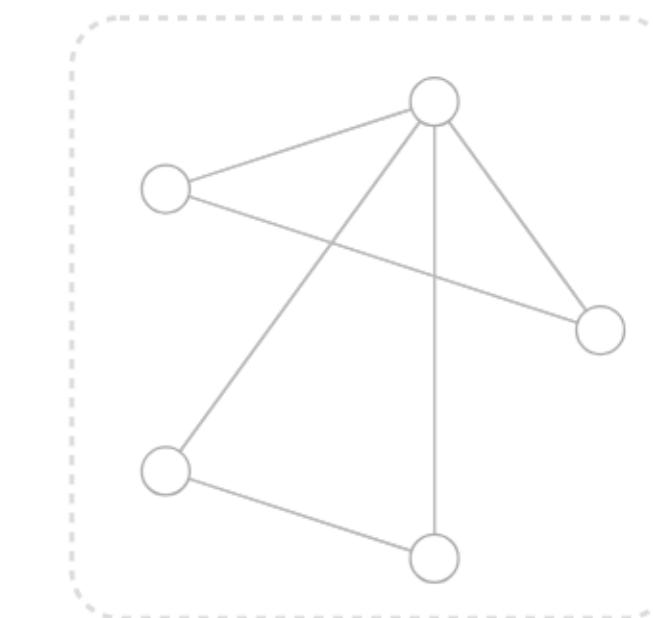
# Introduzione

- L'analisi delle reti sociali è uno strumento potente per **comprendere le relazioni** tra individui, organizzazioni, o entità, e il loro impatto su fenomeni complessi
- Nelle reti sociali è interessante identificare le **dinamiche sottostanti**: chi è centrale, chi influenza, chi connette
- Le reti sociali sono fondamentali per comprendere come le informazioni, le idee o le malattie si **diffondono**



# Graph Neural Networks

- Molte reti neurali tradizionali sono progettate senza considerare le informazioni topologiche. A tale scopo, sono state introdotte le **Graph Neural Networks**, progettate per operare su dati rappresentandoli **sottoforma di grafi**
- Un **grafo** è un insieme di elementi detti **nodi** o vertici che possono essere collegati fra loro da linee chiamate **archi** o lati o spigoli. Più formalmente, si dice grafo una coppia ordinata  $G = (V, E)$
- Se  $E$  è una relazione simmetrica, allora si dice che il grafo è **non orientato** (o indiretto), altrimenti si dice che è **orientato** (o diretto).

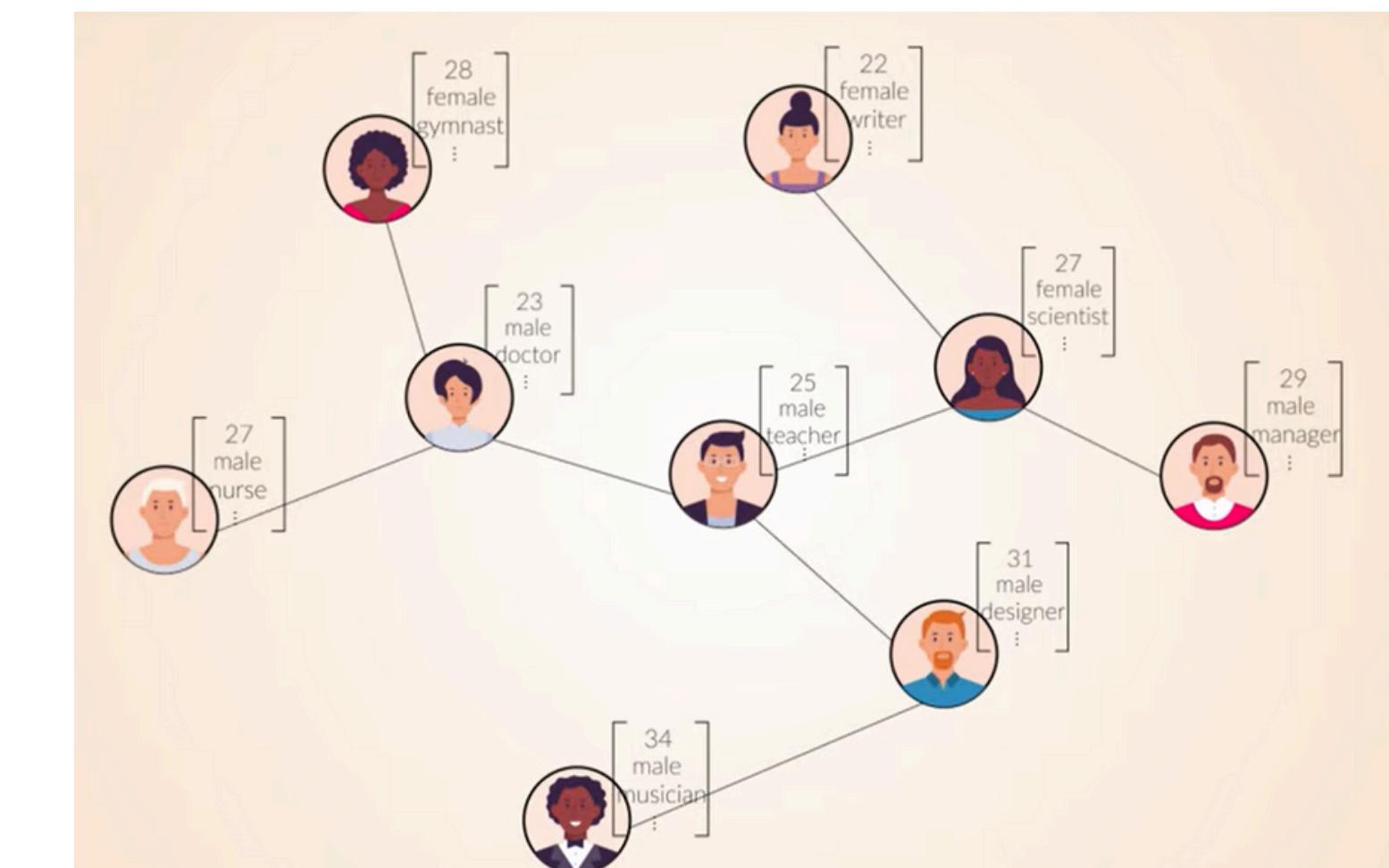


**V** Vertex (or node) attributes  
e.g., node identity, number of neighbors

**E** Edge (or link) attributes and directions  
e.g., edge identity, edge weight

**U** Global (or master node) attributes  
e.g., number of nodes, longest path

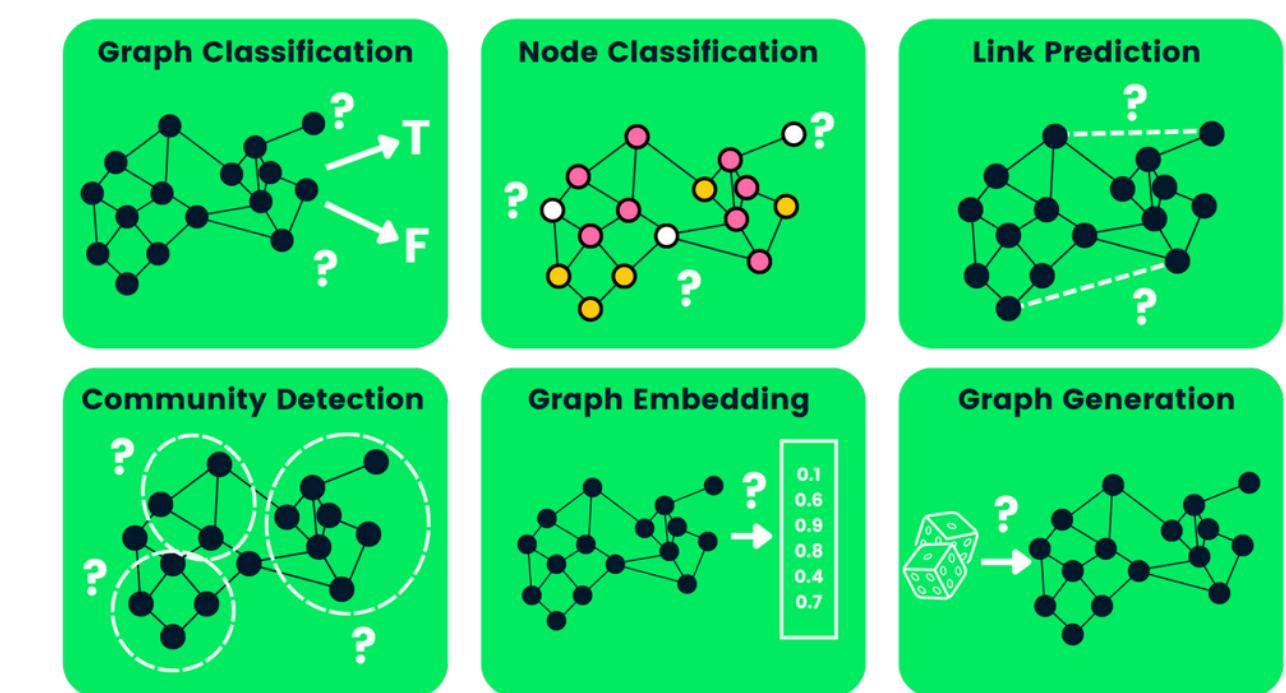
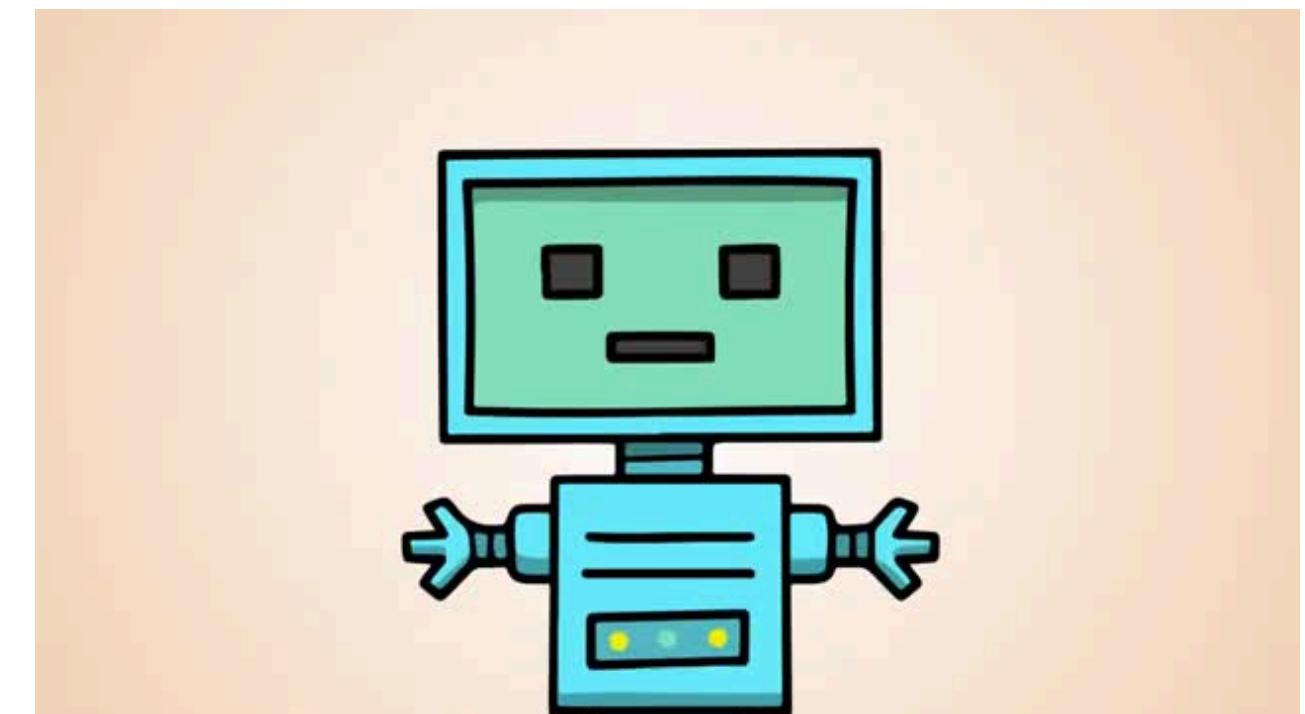
- Uno dei punti di forza delle GNN risiede nella loro capacità di gestire **dati che non sono strutturati o ordinati** ma che presentano una **topologia variabile** da contesto a contesto
- Ad esempio:
  - nell'analisi delle reti sociali, le GNN sono impiegate per **predire il comportamento degli utenti**, identificare nodi influenti e rilevare cluster all'interno della rete
  - Il sistema AlphaFold è noto per la precisione con cui predice le strutture tridimensionali delle proteine.
- Le GNN la scelta preferita per compiti che richiedono la **comprendione delle relazioni**



# Un primo modello

- L'architettura della GNN proposta da Franco Scarselli et al. nel loro articolo dal titolo "The Graph Neural Network Model" del 2009 è pensata per apprendere e aggiornare le informazioni sui nodi in un grafo attraverso un processo di **propagazione delle informazioni (pairwise message passing)** lungo gli archi del grafo
- Più nel dettaglio:
  - Ogni nodo del grafo è associato a uno **stato**, che viene rappresentato da un **vettore di caratteristiche**
  - Ogni nodo aggiorna il proprio stato prendendo in considerazione le **informazioni provenienti dai suoi vicini**
- L'idea è che i nodi nel grafo possano "**comunicare**" tra di loro e condividere informazioni con lo scopo finale di avere una rappresentazione che permetta di **risolvere un compito**
- L'architettura proposta è **ricorsiva** in quanto l'aggiornamento dello stato dei nodi viene ripetuto più volte, in un numero fisso di passi o fino a convergenza

- Ogni passaggio successivo consente ai nodi di raccogliere informazioni da una **regione sempre più ampia** del grafo
- Una volta che il processo di propagazione è stato completato, si ottiene una **rappresentazione dei nodi** tale che nodi vicini rappresentano nodi vicini "simili" nel grafo. Non a caso, nell'ambito delle GNN si parla anche di **graph representation learning**
- Questo spazio ottenuto può essere utilizzato per una **previsione** per ciascun nodo o per l'intero grafo
- Il grafo viene rappresentato sottoforma di una **matrice di adiacenza** dove ogni cella riporta un valore associato al peso dell'arco fra i nodi che individuano la cella stessa. Questa è una delle potenziali **componenti learnable** della rete



# La rete DGCNN

- La **Graph Convolutional Neural Network (GCNN)** è un'estensione della CNN in grado di gestire segnali in domini spaziali discreti, come quelli rappresentati da grafi. Questo tipo di rete sfrutta la struttura topologica dei grafi per definire operazioni di convoluzione, adattandosi così alla natura irregolare dei **dati non euclidei** (i grafi)
- Nell'architettura proposta nel paper "EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks" Song et al., i nodi rappresentano i **canali EEG**, dove col termine canali ci si riferisce agli **elettrodi** posti sulla superficie del cuoio capelluto.
- Tuttavia, una vicinanza fisica fra gli elettrodi non implica necessariamente un qualche legame significativo ai fini dell'analisi delle emozioni; l'approccio proposto prevede dunque di non predeterminare le connessioni del grafo ma rendere **dinamico l'apprendimento**

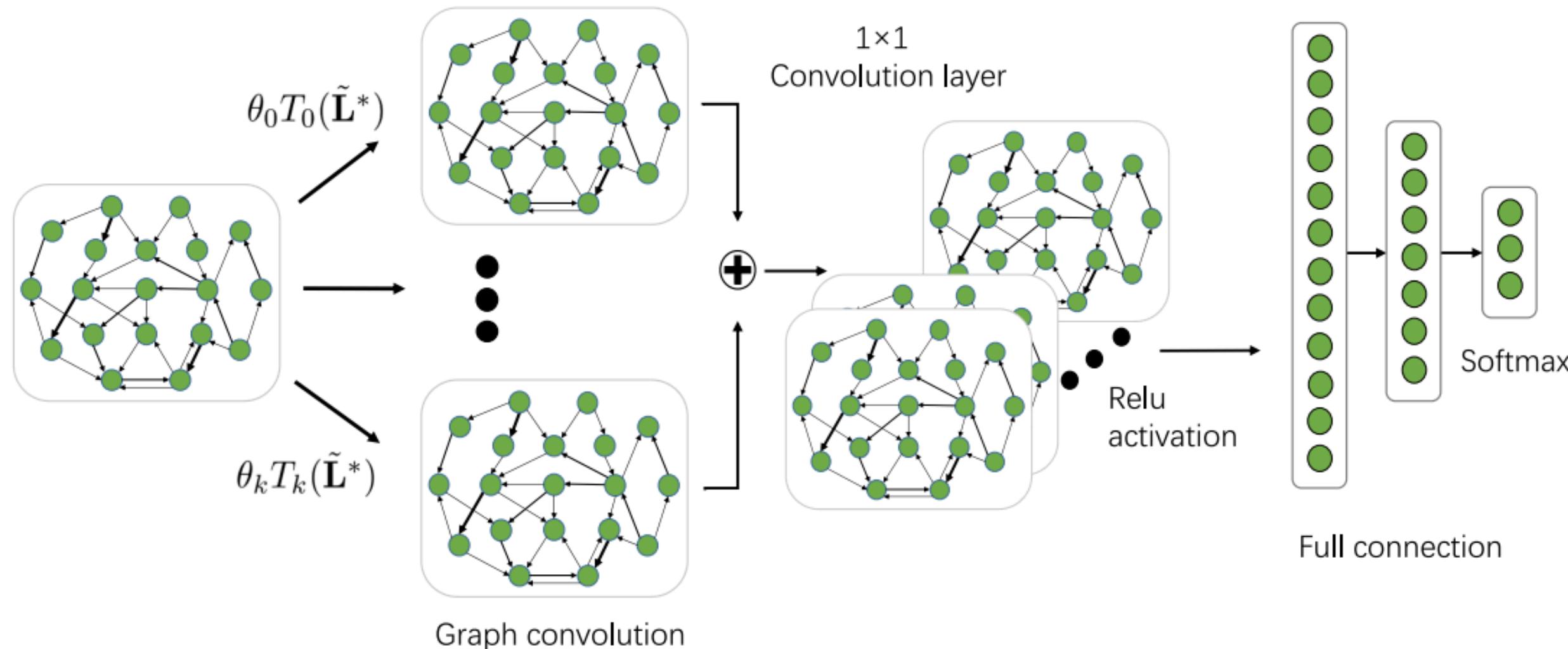


Fig. 2. The framework of the DGCNN model for EEG emotion recognition, which consists of the dynamical graph convolutional operation via the learned graph connections, convolution layer with  $1 \times 1$  kernel, Relu activation and the full connection. The inputs of the model are the EEG features extracted from multiple frequency bands, e.g., five frequency bands ( $\delta$  band,  $\theta$  band,  $\alpha$  band,  $\beta$  band, and  $\gamma$  band), in which each EEG channel is represented as a node of the graph. The outputs are the predicted labels through softmax.

- **Input Layer:** Riceve le features (le diverse bande di frequenza per ciascun elettrodo (nodo))
- **Dynamical Graph Convolution Layer:**
  - Impara dinamicamente una matrice di adiacenza ottimale che cattura le connessioni intrinseche (funzionali) tra i canali EEG.
  - Utilizza polinomi di Chebyshev per approssimare le operazioni di convoluzione spettrale sul grafo
- **1×1 Convolution Layer:**
  - Agisce individualmente su ogni nodo. È equivalente a una combinazione lineare delle feature del nodo stesso attraverso pesi appresi
- **ReLU Activation Layer:** Introduce non-linearità, garantendo valori non negativi nelle feature
- **Fully Connected Layer + Softmax:**
  - Classifica le emozioni basandosi sulle feature apprese.
  - Predice le etichette utilizzando una funzione di attivazione Softmax.

# Il dataset di followers

- Il dataset considerato riporta alcuni dati di utenti di Twitter. In particolare:
  - **avatar**: URL dell'immagine di profilo
  - **followerCount**: numero di followers
  - **friendsCount**: numero di persone che questo profilo segue
  - **friendName**: nome pubblico
  - **id**: user ID
  - **friends**: user IDs dei followed
  - **lang**: lingua dichiarata dall'utente
  - **lastSeen**: time stamp dell'ultimo collegamento
  - **tags**: hashtag usati dall'utente
  - **tweetID**: id dell'ultimo Tweet dell'utente

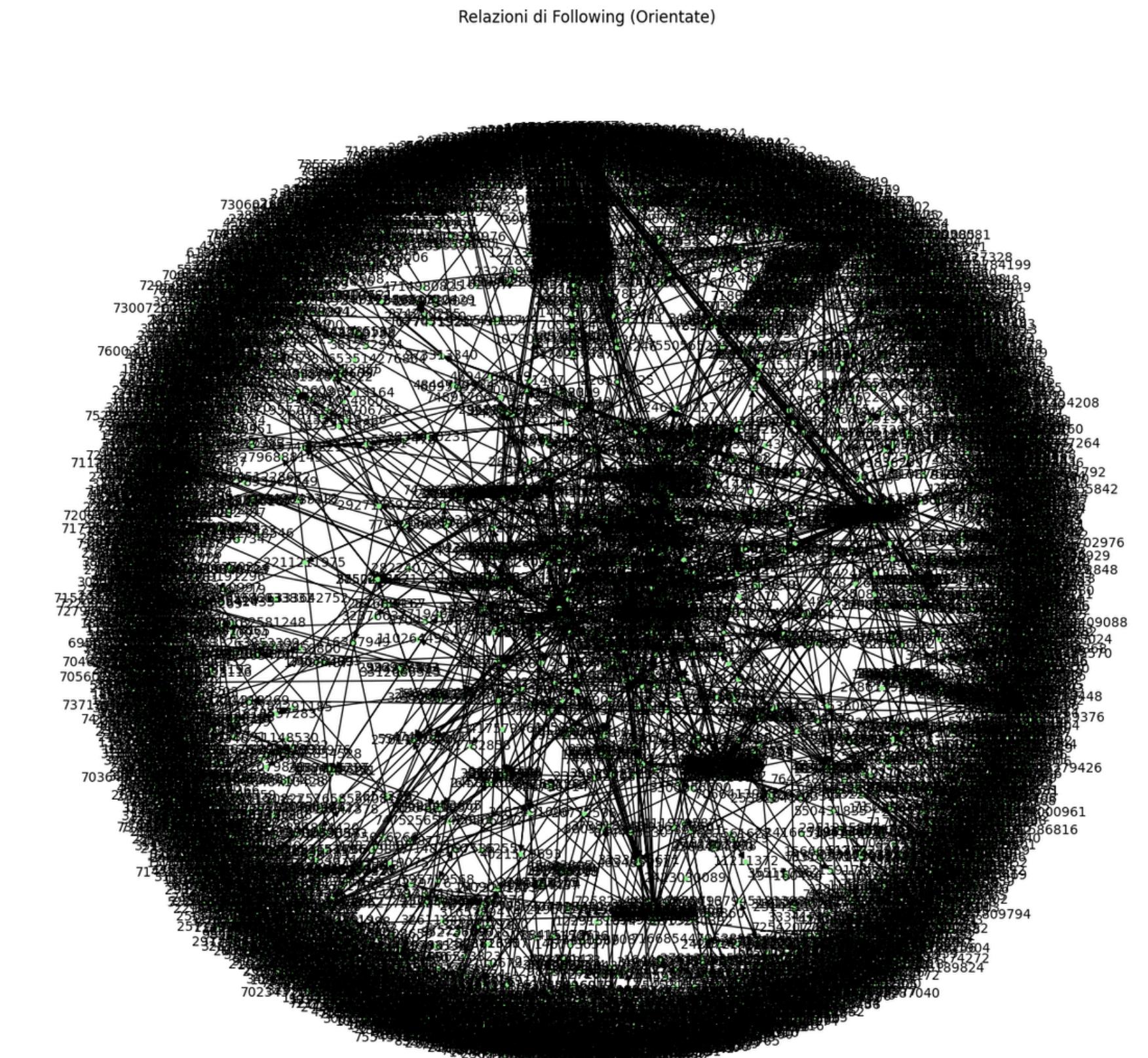
<https://www.kaggle.com/datasets/hwassner/TwitterFriends/data>

- Si tratta di **40.000 utenti** che hanno twittato su argomenti di tendenza, che hanno **almeno 100 follower** e seguono almeno altri 100 account per filtrare lo spam e gli account non informativi/vuoti
- Sono stati presi a **campione 5000 account** e tra le feature disponibili sono state considerate
  - **friendsCount**: numero di persone che questo profilo segue
  - **id**: user ID
  - **friends**: user IDs dei followed (per la matrice di adiacenza)
  - **lang**: lingua dichiarata dall'utente
  - **tags**: hashtag usati dall'utente

	<b>id</b>	<b>tags</b>	<b>friendsCount</b>	<b>lang</b>	<b>friends</b>
0	1969527638	[ "#nationaldogday" ]	112	"en"	[1969574754, 1969295556, 1969284056, 196961221...
1	51878493	[ "#nationaldogday" ]	115	"en"	[60789485, 2420931980, 2899776756, 127410795, ...]
2	1393409100	[ "#narcos" ]	107	"en"	[86868062, 19697415, 2998836604, 456295047, 74...
3	232891415	[ "#gloryoutnow" ]	325	"en"	[361335082, 1405248468, 24626354, 725675895965...
4	710130422907207680	[ "#nationaldogday" ]	218	"en"	[1571896093, 768938323612008448, 2548665930, 3...

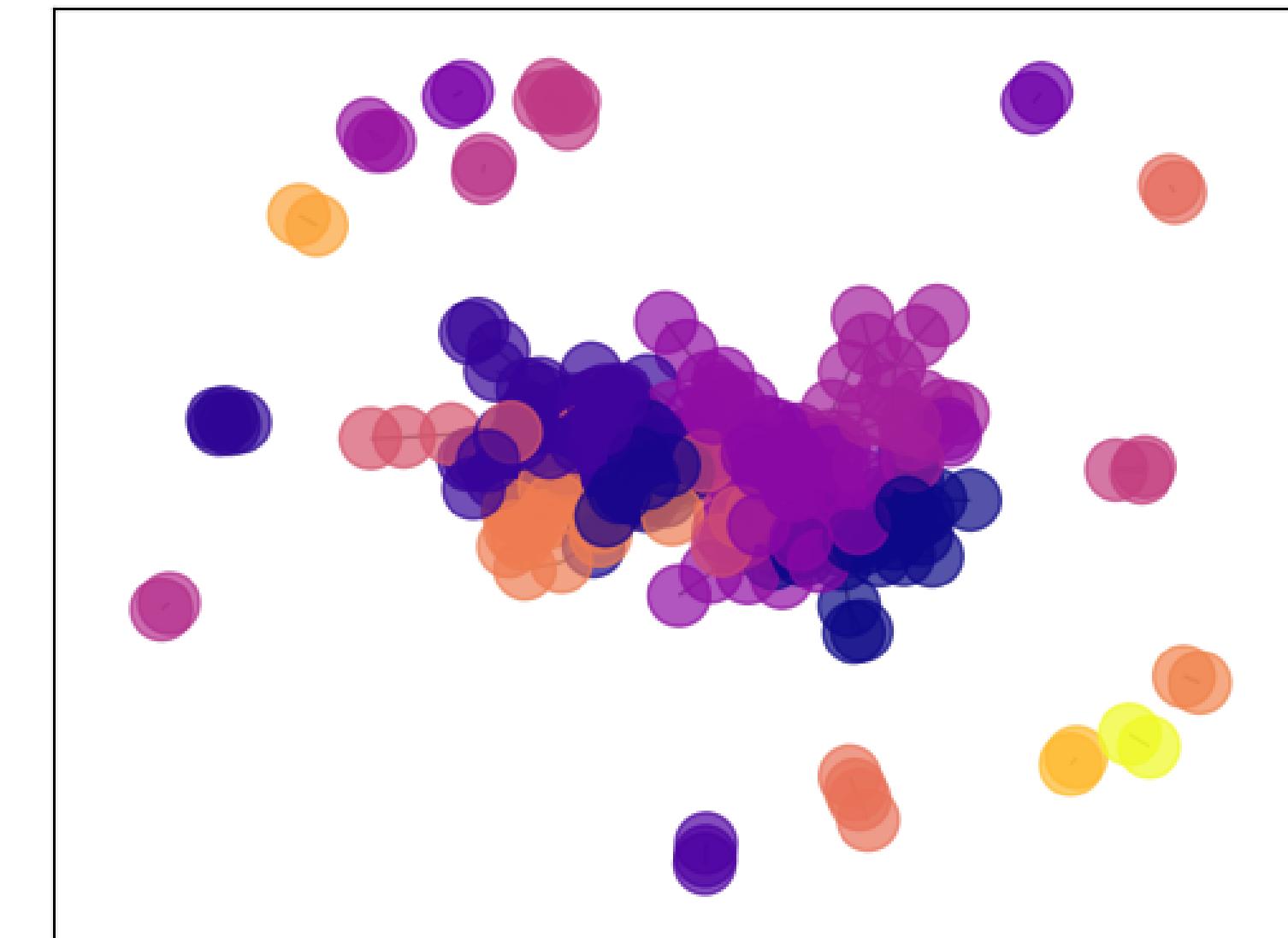
# Analisi dei dati

- Il grafo è composto da **2785 relazioni**
- Alcune informazioni utili estrapolate:
  - Il nodo con **maggiore degree centrality** è id 189283341, valore 0.0464
  - I nodi con id 3253427959 e 41668670 sono quelli che hanno più followers (entrambi 14.273)
  - La coppia di nodi con **più amici in comune** è composta da 3076725085, e 731954451448102912 che hanno in comune 8 amici
  - Il nodo con **più alta betweenness centrality** ha id 791958464, valore 0.18044227666945534

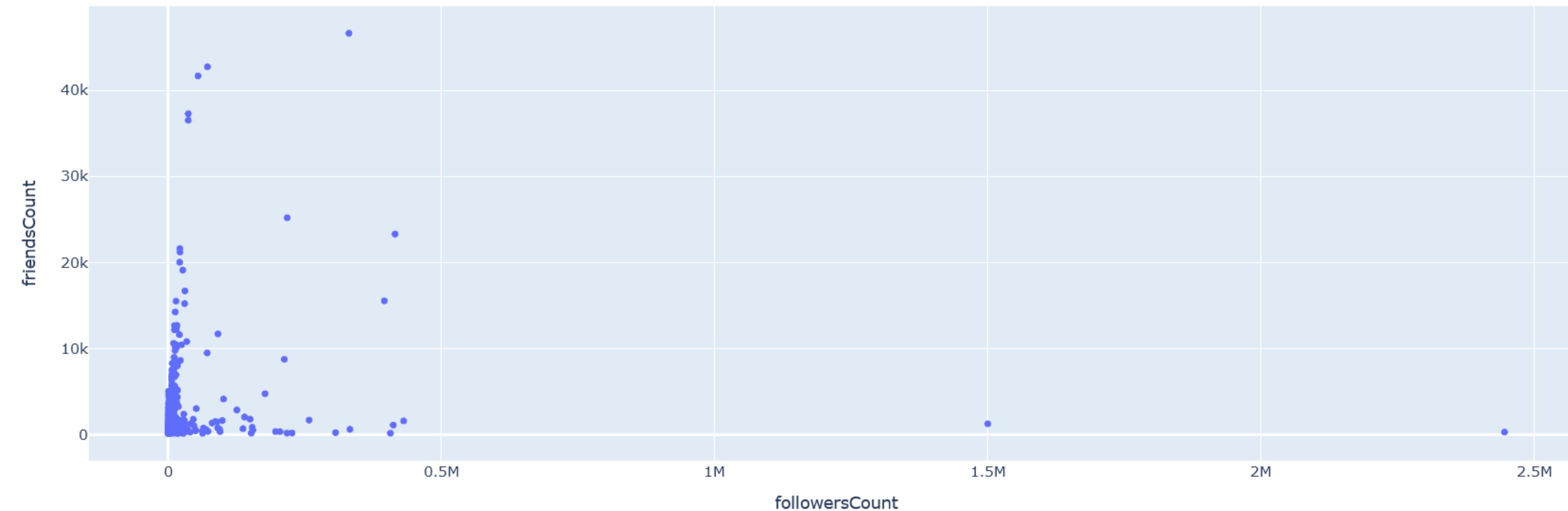


- Estrapolando il sottografo dei nodi con un **degree maggiore di tre**, otteniamo un sottografo composto da **360 nodi e 882 archi**, le cui communities sono rappresentate nella seguente immagine:

- Community 0: 31 nodes, Central Node: 2298043189
- Community 1: 47 nodes, Central Node: 731954451448102912
- Community 36: 23 nodes, Central Node: 721925849956630528
- Community 3: 5 nodes, Central Node: 761981319328870400
- Community 4: 28 nodes, Central Node: 1025128844
- Community 5: 42 nodes, Central Node: 4819969125
- Community 7: 3 nodes, Central Node: 3260744162
- Community 16: 25 nodes, Central Node: 29514951
- Community 15: 57 nodes, Central Node: 19772559
- Community 13: 2 nodes, Central Node: 220821568
- Community 17: 3 nodes, Central Node: 1428826266
- Community 19: 17 nodes, Central Node: 264470292
- Community 22: 2 nodes, Central Node: 1187746027
- Community 23: 2 nodes, Central Node: 3207427365
- Community 24: 6 nodes, Central Node: 513200857
- Community 25: 3 nodes, Central Node: 735742410
- Community 27: 11 nodes, Central Node: 370404328
- Community 29: 5 nodes, Central Node: 3094168105
- Community 33: 2 nodes, Central Node: 1302180632
- Community 34: 4 nodes, Central Node: 33234853
- Community 37: 2 nodes, Central Node: 244826894
- Community 42: 2 nodes, Central Node: 2888687387
- Community 45: 2 nodes, Central Node: 95982570
- Community 53: 2 nodes, Central Node: 1646251393
- Community 11: 2 nodes, Central Node: 717535199173734400

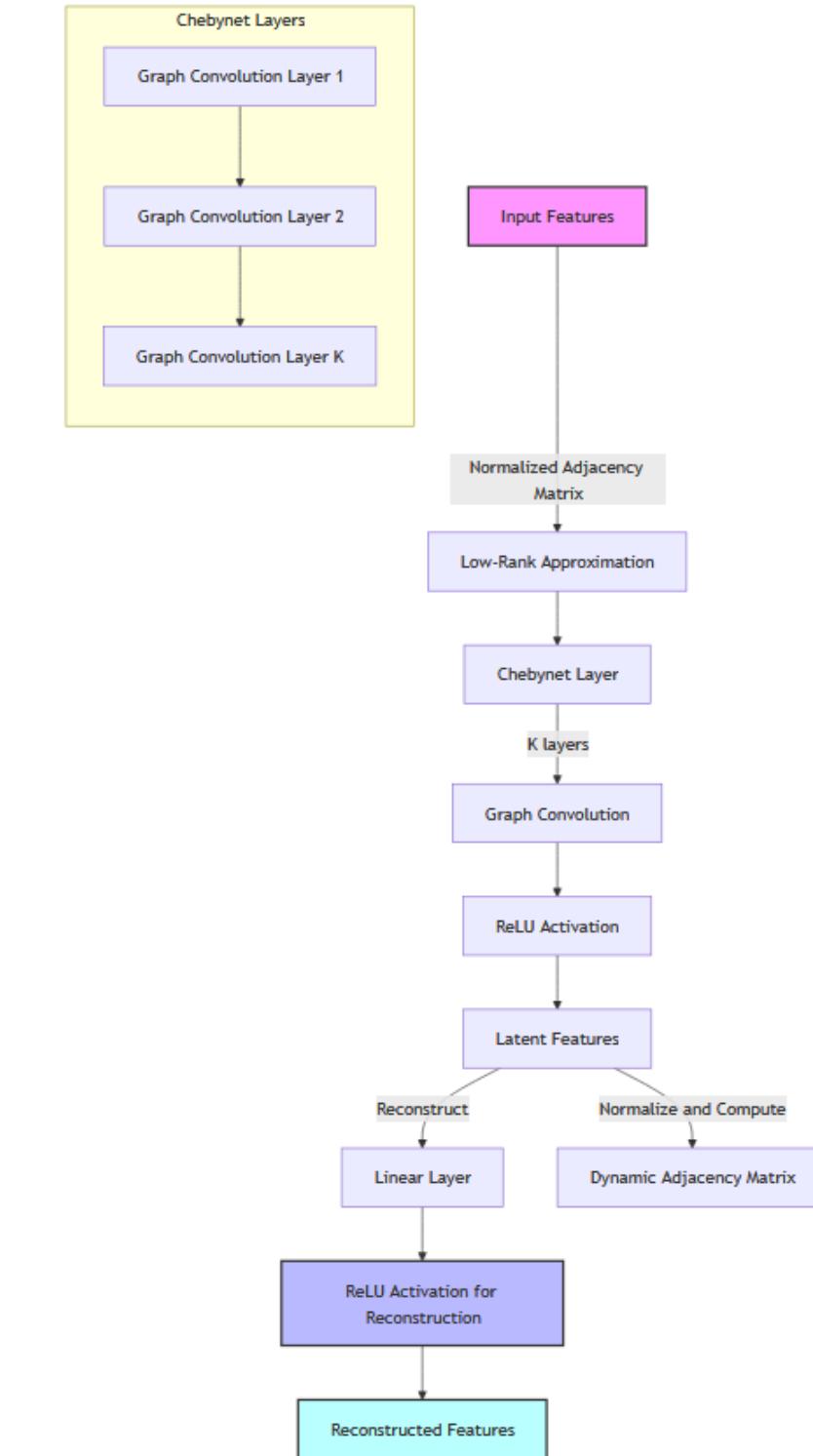


- Le persone con molti followers hanno generalmente pochi amici e viceversa



# Una rete DGCNN custom

- Il task prevede di allenare una rete DGCNN ad apprendere
  - la **ricostruzione** dei dati in ingresso (funzione di autoencoder)
  - la **matrice di adiacenza orientata** (non simmetrica)
- Questo modello implementa una rete basata su convoluzioni grafiche per l'elaborazione e la ricostruzione dei dati nei grafi. Le principali componenti includono:
  - **Approssimazione a basso rango**
  - **Graph Convolution**
  - **Chebynet**
  - **DGCNN con Ricostruzione**



```
# Funzione per l'approssimazione a basso rango
def low_rank_approximation(A, rank):
    epsilon = 1e-3
    # Stabilizzazione della matrice
    A_reg = A + epsilon * torch.eye(A.shape[0], device=A.device)

    try:
        # SVD stabilizzato
        U, S, V = torch.linalg.svd(A_reg, full_matrices=False)
    except RuntimeError as e:
        print(f"Errore durante l'SVD: {e}")
        return A # Se l'SVD fallisce, restituisce la matrice originale

    # Mantiene solo i primi 'rank' valori singolari
    U = U[:, :rank]
    S = S[:rank]
    V = V[:, :rank]

    # Costruisce l'approssimazione a basso rango
    A_approx = torch.matmul(U, torch.matmul(torch.diag(S), V.transpose(0, 1)))

    return A_approx
```

La **Singular Value Decomposition (SVD)** è una tecnica dell'algebra lineare che scomponete una matrice  $A$  in tre componenti fondamentali:

$$A = U\Sigma V^T$$

- $U$ : Matrice ortogonale ( $U^T U = I$ ), le cui colonne sono gli autovettori associati a  $AA^T$  (spazio delle righe).
- $\Sigma$ : Matrice diagonale contenente i valori singolari  $\sigma_i$ , che rappresentano la "potenza" o l'importanza delle componenti di  $A$ .
- $V$ : Matrice ortogonale ( $V^T V = I$ ), le cui colonne sono gli autovettori di  $A^T A$  (spazio delle colonne).

## La funzione definita

- Aggiunge una **piccola quantità  $\epsilon$**  alla diagonale della matrice di adiacenza, rendendola meglio condizionata e meno incline a errori numerici
- Conserva solo i primi **rank** di valori singolari più grandi e le corrispondenti colonne di  $U$  e  $V$  in modo da eliminare le componenti meno significative
- Ricostruisce un'**approssimazione a basso rango** usando i valori singolari e le componenti  $U$  e  $V$

- La riduzione della dimensionalità è funzionale per diversi motivi:
  - L'approssimazione a basso rango può **filtrare situazioni di rumore**, concentrandosi sulle relazioni più significative
  - I valori singolari più grandi catturano **pattern globali e relazioni forti** tra i nodi, che sono spesso utili per predire nuovi collegamenti
  - **Riduce il numero di parametri** necessari per rappresentare la matrice
  - La rappresentazione a basso rango può evitare l'**overfitting**
- Tuttavia, ha anche delle limitazioni:
  - Nei **grafi molto sparsi**, dove il numero di collegamenti è già basso, l'approssimazione potrebbe peggiorare la qualità dei dati
  - La **scelta del rank** è cruciale. Se troppo basso, potrebbero essere ignorate relazioni importanti; se troppo alto, il vantaggio in termini di riduzione di rumore potrebbe perdere

```
class GraphConvolution(nn.Module):
    def __init__(self, in_channels, out_channels, bias=False):
        super(GraphConvolution, self).__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.weight = nn.Parameter(torch.FloatTensor(in_channels, out_channels))
        nn.init.xavier_normal_(self.weight)
        self.bias = None
        if bias:
            self.bias = nn.Parameter(torch.FloatTensor(out_channels))
            nn.init.zeros_(self.bias)

    def forward(self, x, adj):
        # matmul tra matrice di adiacenza e caratteristiche dei nodi
        support = torch.matmul(adj, x)
        # moltiplicazione del supporto con il peso
        output = torch.matmul(support, self.weight)

        if self.bias is not None:
            return output + self.bias
        else:
            return output
```

## La classe GraphConvolution

- implementa un **layer di convoluzione** per grafi, progettato per **trasformare le feature** dei nodi in un grafo, rispettando le connessioni definite dalla matrice di adiacenza

```
class Linear(nn.Module):
    def __init__(self, in_channels, out_channels, bias=True):
        super(Linear, self).__init__()
        self.linear = nn.Linear(in_channels, out_channels, bias=bias)
        nn.init.xavier_normal_(self.linear.weight)
        if bias:
            nn.init.zeros_(self.linear.bias)

    def forward(self, inputs):
        return self.linear(inputs)
```

## La classe Linear

- implementa un **layer lineare standard**

```
class Chebynet(nn.Module):
    def __init__(self, in_channels, K, out_channels):
        super(Chebynet, self).__init__()
        self.K = K
        self.gc = nn.ModuleList()
        for i in range(K):
            self.gc.append(GraphConvolution(in_channels, out_channels))

    def forward(self, x, L):
        adj = generate_cheby_adj(L, self.K)
        result = None
        for i, gc_layer in enumerate(self.gc):
            if i == 0:
                # Passa l'intero tensore x attraverso il layer
                result = gc_layer(x, adj[i])
            else:
                result += gc_layer(x, adj[i])
        result = F.relu(result) # Applicare ReLU
        return result
```

## La classe Chebynet

- implementa una sottorete basata sui **polinomi di Chebyshev** per convoluzioni sui grafi
- memorizza una lista di **K layer di GraphConvolution**, ognuno indipendente
- implementa una **variante della convoluzione spettrale** per grafi generando matrici di adiacenza multiple basate su potenze del Laplaciano normalizzato, catturando così **informazioni locali e non locali**

- Riassumendo:
  - **GraphConvolution** implementa la convoluzione di base su grafi
  - **Linear** fornisce una trasformazione lineare semplice
  - **Chebynet** cattura relazioni multi-hop tra i nodi

```
class DGCNN_Reconstruction(nn.Module):
    def __init__(self, in_channels, num_nodes, k_adj, out_channels, dropout_prob=0.5, rank=50):
        super(DGCNN_Reconstruction, self).__init__()
        self.K = k_adj
        self.num_nodes = num_nodes
        self.layer1 = Chebnet(in_channels, k_adj, out_channels)
        self.rank = rank # Rango per l'approssimazione a basso rango

        # Layers per ricostruire le features
        self.reconstruct_features = nn.Sequential(
            nn.Linear(out_channels, in_channels),
            nn.ReLU()
        )

        # Learnable adjacency matrix
        self.A = nn.Parameter(torch.FloatTensor(num_nodes, num_nodes))
        nn.init.kaiming_normal_(self.A, a=0, mode='fan_in', nonlinearity='relu')
        self.A.data = torch.abs(self.A.data)

    def forward(self, x, batch_indices=None, epoch=None):
        # Normalizzazione della matrice di adiacenza
        if batch_indices is not None:
            L = normalize_A_batch(self.A, batch_indices) # Normalizza solo per il batch

        # Approssimazione a basso rango della matrice di adiacenza
        L_approx = low_rank_approximation(L, self.rank) # Approssimazione a basso rango

        # Passa il batch di nodi attraverso i livelli
        latent_features = self.layer1(x, L_approx)

        # Ricostruzione
        reconstructed_features = self.reconstruct_features(latent_features)

        # Normalizza le caratteristiche latenti
        norm_latent_features = F.normalize(latent_features, p=2, dim=-1)
        dynamic_adj = F.relu(torch.matmul(norm_latent_features,
                                         norm_latent_features.transpose(0, 1)))

    return reconstructed_features, dynamic_adj
```

## La classe DGCNN\_Reconstruction

- implementa una rete neurale per la **ricostruzione delle caratteristiche** e **l'apprendimento dinamico** della matrice di adiacenza in un grafo
- utilizza una combinazione di **convoluzioni sui grafî** (utilizzando polinomi di Chebyshev) e tecniche di **approssimazione a basso rango** per ricostruire la struttura e le caratteristiche del grafo
- restituisce in output
  - le **feature ricostruite** per ogni nodo
  - la **matrice di adiacenza dinamica**, calcolata usando le feature latenti normalizzate

# Fase di training

- **Embedding:**
  - Le features sono state trasformate in un embedding di dimensione 205
  - General features tensor shape: [5000, 205]
- **Funzioni di perdita:**
  - SmoothL1Loss per la ricostruzione delle caratteristiche
  - BCEWithLogitsLoss per l'apprendimento della matrice di adiacenza dinamica
  - Le funzioni sono combinate nel seguente modo:  $\text{loss} = \text{loss\_adj} + 0.2 * \text{loss\_reconstruction}$
- **Configurazione dell'ottimizzatore:**
  - AdamW con learning rate 0.0005
- **Training:**
  - 500 epoch
  - batch size pari a 512
  - L'output di embedding della rete è stato fissato a 50

- Per ogni batch avviene:
  - **Forward pass:** Ricostruzione delle caratteristiche dei nodi e generazione di una matrice di adiacenza dinamica basata sulle caratteristiche normalizzate
  - **Calcolo delle perdite:** Perdita di ricostruzione per le caratteristiche dei nodi e perdita di confronto tra la matrice di adiacenza dinamica e quella reale del batch
  - **Backward pass:** Calcolo del gradiente e aggiornamento dei pesi del modello e aggiornamento della matrice di adiacenza globale
- Nella matrice di adiacenza vengono considerati solo i valori delle celle **superiori ad una soglia fissata a 0.9**

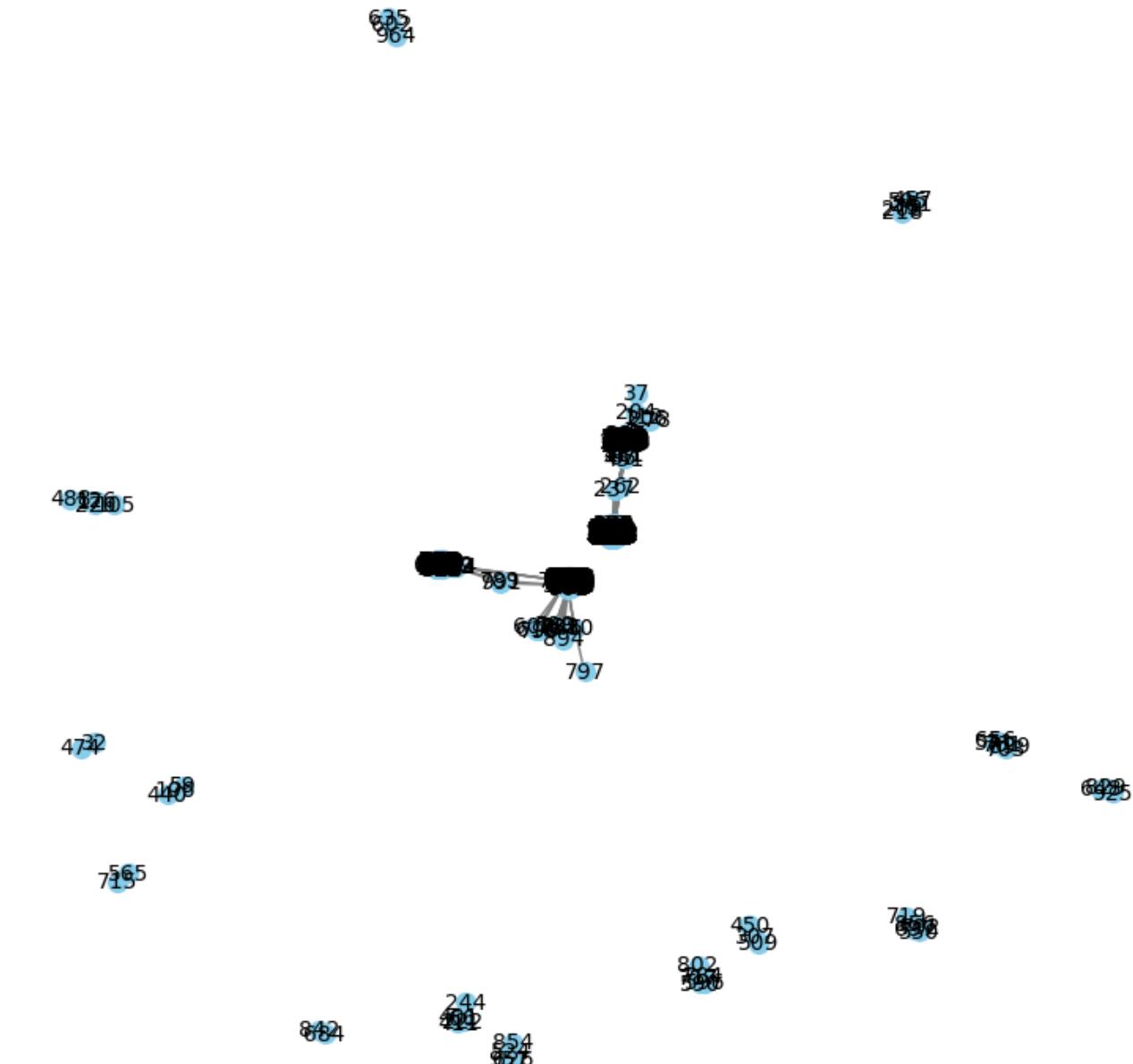
# L'evaluation

- Ogni **5 epoch**, il modello è validato utilizzando il set di validazione:
  - Generazione di una matrice di adiacenza dinamica per il set di validazione
  - Valutazione delle prestazioni rispetto alla matrice di adiacenza reale
  - Se le performance migliorano, salvataggio del modello e della matrice di adiacenza
- Obiettivi finali:
  - **Minimizzare la perdita combinata** (ricostruzione delle caratteristiche + matrice di adiacenza).
  - **Generare una matrice di adiacenza dinamica** che rispecchi accuratamente la struttura del grafo originale

# I risultati

## Rappresentazione del Grafo (Solo Nodi Connessi)

- Link prediction:
    - Sono stati rilevate **625 nuove connessioni**
    - **31 connessioni su 85** sono state rilevate correttamente (circa il 36%)
  - Aspetti rilevanti:
    - Un **batch size** più grande aiuta a stabilizzare la predizione
    - Valori più **alti di k** per la Chebyshev influenzano negativamente la predizione
    - Si formano molte regioni isolate fra di loro (**no bridges**)



- Esempio su alcune coppie di nodi:

Nuovi nodi connessi: 263939125 e 2639575060

Tags: [ "#nationaldogday" ] - [ "#happybirthdaydylanobrien" ]

Followers: 10054 vs 1135

Friends: 932 vs 1039

Linguaggi: "en" vs "en"

- **Amici in comune: 10**

Nuovi nodi connessi: 263939125 e 740366871803920384

Tags: [ "#nationaldogday" ] - [ "#nationaldogday" ]

Followers: 10054 vs 455

Friends: 932 vs 938

Linguaggi: "en" vs "en"

- **Amici in comune: 7**

- Esempio su alcune coppie di nodi:

Nuovi nodi connessi: 4754899462 e 1122222090

Tags: [ "#nationaldogday" ] - [ "#nationaldogday" ]

Followers: 179 vs 200

Friends: 156 vs 178

Linguaggi: "en" vs "en"

- **Amici in comune: 1**

Nuovi nodi connessi: 263939125 e 523652545

Tags: [ "#nationaldogday" ] - [ "#nationaldogday" ]

Followers: 10054 vs 1459

Friends: 932 vs 2

Linguaggi: "en" vs "en"

- **Amici in comune: 2**

# Conclusioni

- Adattare la DGCNN ideata per gli ECG è risultato essere un **task sfidante**, soprattutto a causa della natura della matrice di adiacenza che rischia di divenire un parametro troppo pesante per l'analisi nelle reti sociali
- Utilizzare **ulteriori tecniche di riduzione** della dimensionalità/PCA potrebbero migliorare le prestazioni
- La scelta della **metrica per l'evaluation** influenza il tipo di risultato ottenuto
- Sono molti i **parametri** che possono essere **customizzati**
- Risultati migliori si potrebbero avere su **dataset più significativi**