UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Graph Convolutional Neural Network: uno studio di applicabilità per il Predictive Process Mining

Graph Convolutional Neural Network: a feasibility study for Predictive Process Mining

Relatore:

Prof. **Claudia Diamantini**

Laureando:

Matteo Vallese

A.A. 2020/2021

Sommario

INTRODUZIONE 1

GRAPH NEURAL NETWORK 2

Grafi 6

Cosa si può ottenere dalle GNN? 6

Tipi di GNN 7

Prima GNN 7

Convolutional GNN 10

DGCNN 15

Layer di convoluzione sui grafi 16

Layer di SortPooling 17

Remaining layers 18

NEXT ACTIVITY PREDICTION 19

DATASET E RETE NEURALE 19

RISULTATI 25

CONCLUSIONI 44

Biobiografia 46

INTRODUZIONE

Utilizzare i dati è diventato il modus operandi dello sviluppo aziendale. Tuttavia, fino a poco tempo fa, nelle operazioni quotidiane, le aziende si basavano su limitate disponibilità di capacità di elaborare dati per stimare semplici parametri di business. Perciò, spesso, queste analisi sono state stimate in modo approssimativo. Oggigiorno, le aziende stanno sostituendo le decisioni basate sull'istinto con decisioni basate sui risultati ottenuti dai dati e con miglioramenti dell'analisi dati, metodi automatici possono estendere intuizioni aziendali ben oltre ciò che la mente umana potrebbe mai sperare di ottenere. Il predictive process mining fornisce delle tecniche che permettono di ottenere informazioni basate sui dati. Questi dati sono relative ai processi aziendali, che è un insieme di attività interellate, svolte all'interno dell'azienda nell'ambito della gestione operative delle sue funzioni aziendali, che creano valore trasformando delle risorse in un prodotto finale a valore aggiunto, destinato ad un soggetto interno o esterno all'azienda. Le tecniche messe a disposizione dal predictive process mining permetto di effettuare previsioni, come ad esempio la netx activity prediction (permette di predire la prossima attività da svolgere), o la predizione del tempo di inizio o di fine della successiva attività. Poiché al giorno d'oggi c'è una grande disponibilità di dati registrati dai sistemi informativi a supporto delle operazioni quotidiane delle aziende, e stanno rapidamente crescendo maturi e rapidi approcci nel campo del predictive process mining, queste opportunità stanno diventando sempre più utili. Nel seguente elaborato si è andati a vedere l'applicabilità dell'utilizzo delle graph convolutional neural network per effettuare predictive process mining. In particolare si è andati a vedere il comportamento di un tipo di graph convolutional neural network, la DGCNN, per effettuare task di next activity prediction. Sono state scelte le graph convolution neural network perché i dati contenuti all'interno dei log sono stati trasformati in dati

nella forma di grafi, ciò è stato possible perchè l'esecuzione di un processo aziendale presenta, oltre a sequenze, anche rami alternativi o paralelli, ed ha quindi una naturale struttura a grafo, per questo motive si è voluto cercare di sfruttare la potenza di queste reti neurali. Nella prima parte dell'elaborato verrà data una spiegazione teorica di cosa siano le graph neural network e cosa permettono di fare. Successivamente si mostrerà la rete neurale utilizzata per fare next activity prediction ed infine verrà mostrato come si è deciso di affrontare il problema ed i risultati ottenuti dalla rete neurale.

GRAPH NEURAL NETWORK

Le graph neural network sono dei modelli di deep learning che operano direttamente sui grafi. I grafi sono un tipo di struttura dati che modellano un insieme di oggetti (nodi) e le loro relazioni (archi).

Ci sono diversi principi alla base delle GNN. In particolare, alcuni di questi principi di funzionamento derivano dalle convolutional neural network (CNN). Le CNN sono reti neurali progettate per elaborare dati che hanno la forma di tensori, per esempio un immagine a colori è composta da tre tensori bidimensionali che contengono l'intensità dei pixel per ogni canale del colore (RGB). La tipica architettura di una CNN è formata da layer convoluzionali, layer di pooling e layer fully connected. I layer convoluzionali si occupano di effettuare l'operazione di convoluzione. Questa operazione estrae informazioni dai dati passati in input al layer. Supponendo come esempio di avere come input del layer convoluzionale un'immagine, l'operazione di convoluzione fa scorrere dei filtri, detti anche kernel, sopra l'immagine. In base alla struttura che questi filtri possiedono, estraggono determinate informazioni dall' immagine, denominate "feature map". Con un funzionamento simile ai layer convoluzionali, i layer di pooling vanno a ridurre la dimensione delle feature map, questa riduzione della dimensione porta ad una diminuzione della potenza computazionale richiesta per processare i dati.

L'output che viene prodotto dai layer convoluzionali e dai layer di pooling rappresentano le high-level features estratte dall'input della rete. Per ultimo nelle reti convoluzionali si trova il layer fully connected, che utilizza queste high-level features per andare a classificare il dato di input sulla base delle classi dell'insieme di dati di addestramento [1].

Andando più a fondo nelle CNN e nei grafi si possono ritrovare i principi fondamentali alla base delle CNN, questi sono le connessioni locali, i pesi condivisi e l'uso di più layer. Queste fondamenta sono anche di grande importanza per risolvere i problemi sui grafi, perché i grafi sono la più tipica struttura localmente connessa. I pesi condivisi portano una riduzione del costo computazionale se lo si mette a confronto con quello della classica teoria spettrale sui grafi. E strutture con più layer sono la chiave per risolvere i problemi con pattern gerarchici, in quanto possono catturare l'informazione su più dimensioni. Comunque, le CNN possono solamente operare su regolari dati euclidei come immagini (griglie 2D) e testi (sequenze 1D). Però questi dati possono anche essere considerati come istanze di grafi. Quindi potrebbe essere semplice pensare di trovare una generalizzazione dalle CNN ai grafi. Ma è difficile definire dei filtri di convoluzione localizzati e operatori di aggregazione sui grafi, questi infatti ostacolano la trasformazione delle CNN dal dominio euclideo al dominio non euclideo.

Altri principi di funzionamento delle GNN derivano dal graph embedding, in cui le proprietà di un grafo sono trasformate in un vettore o in un insieme di vettori. L'embedding deve prendere la struttura del grafo, le relazioni tra i vertici e le altre informazioni rilevanti dal grafo. L'embedding si può suddividere in due gruppi: vertex embedding e graph embedding. Il vertex embedding codifica ogni vertice (nodo) con un vettore, e si usa quando si vuole eseguire la visualizzazione dei vertici o si vuole fare predizione sui vertici (per esempio, si vuole fare la visualizzazione dei vertici su

un piano 2D o predire delle nuove connessioni basate sulla similarità dei vertici). Nel graph emebedding si rappresenta l'intero grafo con un singolo vettore. Questo embedding è usato quando si vuole fare predizione sui grafi e quando si vogliono comparare dei grafi tra loro o visualizzare più grafi (per esempio si usa per comparare le strutture chimiche)[2]. Nel campo dell'analisi dei grafi, i tradizionali approcci di machine learning si affidano a funzioni fatte ad-hoc, le quali sono limitate dall'inflessibilità e da un alto costo computazionale. Successivamente, l'idea del representation learning ed il successo del word embedding con il metodo del DeepWalk, che è ritenuto il primo metodo di graph embeddings basato sul representation learning, applica l'architettura SkipGram su cammini generati in maniera random. SkipGram è un modello che ha lo scopo di predire le parole di contesto a partire dal token corrente. L'input consiste quindi nella rappresentazione one-hot encoding del token corrente, mentre il layer nascosto corrisponde alla rappresentazione word embedding del token corrente. Questo lo fa massimizzando la probabilità della co-occorrenza tra i termini che appaiono in un testo. Simili approcci come il node2vec, LINE e TAWD hanno portato dei miglioramenti, ma tutti questi metodi soffrono di due gravi svantaggi. Il primo di questi è che non ci sono parametri condivisi all'interno dei nodi nell'encoder, questo porta ad avere inefficienza computazionale, visto che il numero dei parametri cresce linearmente con il numero dei nodi. Il secondo svantaggio è che questi embeddings method mancano della capacità di generalizzazione, questo porta al non riuscire a lavorare con grafi dinamici o generalizzare su nuovi grafi e ciò è dovuto al fatto che sono intrinsecamente trasduttivi[14].

Basate sulle CNN e il graph embeddings, le graph neural network sono state proposte per aggregare informazioni sui grafi, permettendo di modellare input e/o output composti dagli elementi e dalle dipendenze del grafo.

I motivi che si hanno per studiare o utilizzare le graph neural network sono diversi. Il primo motivo è che le classiche neural network come le CNN e le RNN non possono gestire in modo appropriato i grafi come input della rete, perché hanno bisogno di assegnare un ordinamento ai nodi. Infatti, per i grafi non esiste un ordine naturale dei nodi, quindi per illustrare in maniera completa un grafo si dovrebbero utilizzare tutti i possibili ordinamenti dei nodi come input del modello CNN o RNN; quindi diventa molto ridondante eseguire la computazione. Per risolvere questo problema, le GNN propagano l'informazione su ogni nodo del grafo, ignorando l'ordine di input dei nodi. In altre parole, l'output delle GNN è invariante rispetto all'ordinamento che presentano i nodi nell'input. Il secondo motivo è che nei grafi un arco rappresenta l'informazione di dipendenza tra due nodi. Nelle classiche reti neurali, questa informazione di dipendenza è considerata all'interno delle informazioni dei nodi. Al contrario, nelle GNN si può fare propagazione guidata direttamente attraverso la struttura del grafo, piuttosto che utilizzare le informazioni degli archi come parte delle informazioni dei nodi. Generalmente, le GNN aggiornano lo stato nascosto dei nodi attraverso una somma pesata dello stato dei nodi vicini. Infine, il terzo motivo è che il ragionamento è un importante obiettivo di ricerca per intelligenze artificiali di alto livello e il processo di ragionamento del cervello umano si basa praticamente sui grafi che si ricavano dalle esperienze giornaliere. Le classiche reti neurali hanno mostrato l'abilità di generare immagini e documenti sintetici tramite l'apprendimento da distribuzioni di dati, ma non possono comunque apprendere grafi di ragionamento da dati sperimentali.

Tuttavia, le GNN studiano come generare il grafo da dati non strutturati, questo può essere un potente modello per ulteriori intelligenze artificiali di alto livello [3].

Grafi

Prima di vedere le GNN diamo una definizione di grafo. Un grafo diretto è un dato strutturato che si compone di due elementi, vertici ed archi. Un grafo viene rappresentato come una coppia $G = (V, E)$ dove $V = \{v_1, v_2, \ldots, v_n\}$ è l'insieme di vertici o nodi, ed $E \subseteq V \times V$ è l'insieme degli archi. $e_{ij} = (v_i, v_j) \in E$ rappresenta un arco che punta da v_i a v_j. I vicini di un nodo v sono definiti come $N(v) = \{u \in V \mid (v, u) \in E\}$. La matrice di adiacenza A è una matrice $n \times n$ con $A_{ij} = 1$ se $e_{ij} \in E$ e $A_{ij} = 0$ se $e_{ij} \notin E$. Un grafo può avere degli attributi dei nodi X, dove $X \in R^{n \times d}$ è la matrice delle features dei nodi, $x_v \in R^d$ rappresenta il vettore delle features di un nodo v. Inoltre, un grafo può avere attributi sugli archi X_e, dove $X_e \in R^{m \times e}$ è una matrice delle features degli archi e $x_{v,u}^e \in R^c$ è il vettore delle features di un arco (u, v). Un grafo può essere anche non diretto. Un grafo non diretto è considerato come uno speciale caso di grafo diretto in cui c'è una coppia di archi con direzione inversa se due nodi sono connessi tra loro. Quindi un grafo è non diretto se e solo se la matrice di adiacenza è simmetrica [4].

Cosa si può ottenere dalle GNN?

Con la struttura e il contenuto informativo dei nodi come input, l'output delle GNN può concentrarsi su differenti tipi di analisi del grafo utilizzando uno dei seguenti meccanismi:

- Node-level: Tramite le GNN si possono ottenere rappresentazioni di alto livello dei nodi attraverso la propagazione dell'informazione attraverso i nodi o

tramite la convoluzione sul grafo. Applicando poi un layer multi-perceptron o un layer di softmax come layer di output, attraverso la GNN si è in grado di compiere processi come la regressione dei nodi o la classificazione dei nodi in maniera end-to-end.

- Edge-level: Data la rappresentazione nascosta dei nodi della GNN come input, una funzione di similarità o una rete neurale possono essere utilizzate per predire la label o la forza di connessione di un arco.

Graph-level: Le GNN sono spesso combinate con operazioni di pooling o di readout per ottenere una rappresentazione compatta del grafo che permette di effettuare la classificazione dell'intero grafo.

Tipi di GNN

Prima GNN

La prima GNN è stata proposta da Scarselli et al.[10], la quale estende le esistenti reti neurali per processare dati appartenenti al dominio dei grafi. In un grafo ogni nodo è definito dalle sue features e dai collegamenti con gli altri nodi del grafo. L'obiettivo delle GNN è quello di addestrare uno stato nascosto $h_v \in R^s$ che contiene l'informazione dei nodi vicini di ogni nodo. Lo stato nascosto h_v è un vettore di dimensione s per ogni nodo v e può essere usato per produrre un output o_v, come può essere l'assegnazione di una label a un nodo. Viene definita una funzione parametrica f, chiamata funzione di transizione locale, condivisa da tutti i nodi. Questa funzione aggiorna lo stato dei nodi in accordo all'input, che è l'informazione dei nodi vicini. Viene inoltre definita una funzione g, funzione di output locale, questa descrive come l'output viene prodotto. Infine, h_v e o_v sono definiti come segue:

$$h_v = f\left(x_v, x^e_{(v,u)}, x_u, h_u\right) \tag{1}$$

$$o_v = g(h_v, x_v) \tag{2}$$

dove x_v è il vettore delle features del nodo v, $x^e_{(v,u)}$ sono le features degli archi del nodo v, x_u sono le features dei nodi vicini di v ed h_u sono gli stati dei nodi vicini di v.

Siano H, O, X e X_N i tensori costruiti impilando rispettivamente tutti gli stati, tutti gli output, tutte le features e tutte le features dei nodi; possiamo ottenere una forma compatta delle espressioni precedenti:

$$H = F(H, X) \tag{3}$$

$$O = G(H, X_N) \tag{4}$$

dove F è la funzione di transizione globale e G è la funzione di output globale, che sono le versioni di f e g che lavorano direttamente con tutti i nodi del grafo. Il valore di H è il punto fisso [5] dell'eq. 3 ed è definito unicamente attraverso l'assunzione che F sia una mappa di contrazione[11]. Successivamente con le ipotesi del teorema Banach fixed point[12], la GNN usa un classico schema iterativo per calcolare lo stato successivo:

$$H^{t+1} = F(H^t, X) \tag{5}$$

dove H^t indica la t-esima iterazione di H. Il sistema dinamico dell'eq.5 converge esponenzialmente alla soluzione dell'eq.3 per qualsiasi valore iniziale $H(0)$. Si può anche notare che il calcolo descritto nelle funzioni f e g può essere interpretato come una rete neurale feedforward. Dopo che si ha la struttura della GNN, la successiva questione è come addestrare i parametri di f e g. Con l'informazione obiettivo (t_v per uno specifico nodo) per la supervisione, la funzione di loss può essere scritta come:

$$loss = \sum_{i=1}^{p}(t_i - o_i) \quad (6)$$

dove p è il numero di nodi di cui si conosce la label da predire. L'algoritmo di apprendimento è basato sulla strategia dei gradienti discendenti ed è composto dai seguenti step:

- Lo stato h_v^t è iterativamente aggiornato dall'eq.1 fino ad un tempo T. Questo approccio converge alla soluzione dei punti fissi dell'eq.3: $H(T) \approx H$.
- Il gradiente dei pesi W è calcolato dalla funzione di loss.
- I pesi W sono aggiornati in base al gradiente calcolato all'ultimo passaggio.

Questa architettura di GNN ha però molte limitazioni. In primis, è inefficiente l'aggiornamento iterativo dello stato nascosto dei nodi per il punto fisso. Se si rilassa l'assunzione del punto fisso, si può progettare una GNN multi-layer che permette di avere una rappresentazione stabile dei nodi e dei loro vicini. Secondo, la GNN usa gli stessi parametri nelle varie iterazioni mentre le più popolari reti neurali usano parametri differenti in differenti layer; ciò serve per estrarre delle features gerarchiche. Terzo, ci sono anche molte features sugli archi che non possono essere effettivamente modellate nell'originale GNN. Per esempio, gli archi in un grafo di conoscenza potrebbero essere di diversi tipi e in base al loro tipo potrebbero esprimere relazioni diverse e propagare l'informazione in maniera differente. Inoltre, apprendere lo stato nascosto dei nodi è un altro importante problema. Infine, è inopportuno utilizzare i punti fissi se nella rete ci si focalizza sulla rappresentazione dei nodi invece che sulla rappresentazione dei grafi, perché questa distribuzione della rappresentazione delle informazioni in un punto fisso sarà molto regolare, quindi si avrà meno informazione per distinguere i nodi tra loro [3].

Convolutional GNN

Le GNN convolutive sono strettamente legate alle prime GNN. Invece di aggiornare lo stato dei nodi iterando sempre sullo stesso layer, le GNN convolutive aggiornano lo stato dei nodi utilizzando un numero fisso di layer convoluzionali con pesi differenti su ogni layer ().

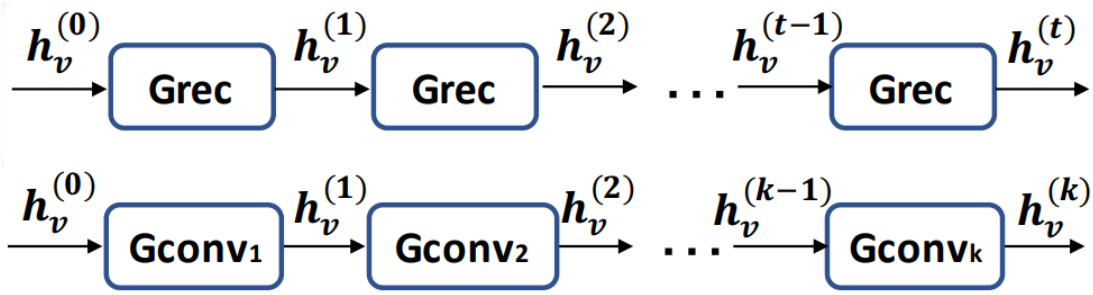

Figura 1: In alto la prima GNN che itera in maniera ricorrente sempre sullo stesso layer per aggiornare la rappresentazione dei nodi, in basso una GNN convolutiva che usa layer differenti per aggiornare la rappresentazione dei nodi. Immagine tratta da "A comprehensive survey on graph neural networks"[4]

Le GNN convolutive si dividono in due categorie, una categoria si basa sulla convoluzione nello spettro mentre l'altra si basa sulla convoluzione nello spazio. Le GNN basate sulla convoluzione spettrale definiscono la convoluzione dei grafi andando ad introdurre filtri che si basano sul calcolo dei segnali dove le operazioni di convoluzione sul grafo sono viste come la rimozione del rumore dai segnali. Invece i metodi basati sugli approcci spaziali prendono spunto dalla classica GNN andando a definire la convoluzione come propagazione dell'informazione attraverso la struttura del grafo.

Metodi nello spettro

I metodi basati sulla convoluzione nello spettro hanno solide basi matematiche nel processamento dei segnali. Questi metodi assumono che il grafo sia non diretto. La matrice laplaciana normalizzata del grafo è la rappresentazione matematica di un grafo non diretto, definito come $L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, dove D è la matrice diagonale del grado

gr dei nodi: $D_{ii} = gr(v_i) = \sum_j (A_{i,j})$. La matrice laplaciana normalizzata può essere fattorizzata come $L = U\Lambda U^T$ dove $U = [u_0, u_1, \dots, u_{n-1}] \in R^{n\times n}$ è la matrice degli auto-vettori ordinati in base agli autovalori e Λ è la matrice diagonale degli autovalori (spettro), $\Lambda_{ii} = \lambda_i$. Gli auto-vettori della matrice laplaciana normalizzata formano uno spazio ortonormale, che in termini matematici è $U^T U = I$. Nell'elaborazione dei segnali, un segnale $x \in R^n$ è un vettore delle features di tutti i nodi di un grafo dove x_i è il valore dell'i-esimo nodo. La trasformata di Fourier su un segnale x è definita come $\mathcal{F}(x) = U^T x$, e l'inverso della trasformata di Fourier è definita $\mathcal{F}^{-1}(\hat{x}) = U\hat{x}$, dove $\hat{x}$ rappresenta il segnale risultante dalla trasformata di Fourier. La trasformata di Fourier proietta il segnale di input nello spazio ortonormale, dove la base è formata dagli auto-vettori della matrice laplaciana normalizza del grafo. Gli elementi del segnale trasformato $\hat{x}$ sono le coordinate del segnale nel nuovo spazio, così che il segnale di input può essere rappresentato come $x = \sum_i \hat{x}_i u_i$, che è esattamente la trasformata di Fourier inversa. Ora la convoluzione sul grafo del segnale di input x con un filtro $g \in R^n$ viene definita come

$$x *_G g = \mathcal{F}^{-1}\big(\mathcal{F}(x) \odot \mathcal{F}(g)\big) = U(U^T x \odot U^T g) \tag{7}$$

Dove $\odot$ è un prodotto elemento per elemento. Se si indica un filtro come $g_\theta = diag(U^T g)$, la convoluzione nello spettro del grafo può essere semplificata come

$$x *_G g_0 = U g_0 U^T x \tag{8}$$

I metodi di GNN che applicano la convoluzione nello spettro si affidano tutti a questa definizione. La differenza chiave si trova nella scelta del filtro g_θ.

La Spectral Convolutional Neural Network assume il filtro $g_\theta = \Theta_{i,j}^{(k)}$, che è un insieme di parametri addestrabili e considera i segnali come multi-canali. Il layer di convoluzione del grafo della Spectral CNN è definito come

$$H_{i,j}^{(k)} = \sigma(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)} U^T H_{:,i}^{(k-1)}) \quad (j = 1, 2, \dots, f_k) \tag{9}$$

dove k è l'indice del layer, $H^{(k-1)} \in R^{n \times f_{k-1}}$ è il segnale di input, $H^{(0)} = X$, f_{k-1} è il numero dei canali di input, f_k è il numero dei canali di output e $\Theta_{j,i}^{(k)}$ è una matrice diagonale composta dai parametri addestrabili. La Spectral CNN mostra tre limitazioni. Queste limitazioni sono dovute alla decomposizione spettrale della matrice laplaciana. Primo, qualsiasi perturbazione al grafo porta un cambio nella base degli auto-vettori. Secondo, i filtri addestrabili sono dipendenti dal dominio, significa che non possono essere applicati a grafi con differenti strutture. Terzo, la decomposizione degli autovalori ha una complessità computazionale di $O(n^3)$.

La Chebyschev Spectral CNN (ChebNet) approssima il filtro g_θ dai polinomi di Chebychev della matrice diagonale degli autovalori, es. $g_\theta = \Sigma_{i=0}^{K} \theta_i T_i(\widetilde{\Lambda})$, dove $\widetilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_n$, e il valore di Λ sta in $[-1, 1]$. I polinomi di Chebyshev sono definiti ricorsivamente da $T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x)$ con $T_0(x) = 1$ e $T_1(x) = x$. Come risultato, la convoluzione di un segnale x con il predefinito filtro g_θ è

$$x *_G g_\theta = U\left(\sum_{i=0}^{K} \theta_i T_i(\widetilde{\Lambda})\right) U^T x \tag{10}$$

dove $\widetilde{L} = \frac{2L}{\lambda_{max}} - I_n$. Così $T_i(\widetilde{L}) = UT_i(\widetilde{\Lambda})U^T$, questo può essere dimostrato tramite induzione per i, ChebNet quindi prende la forma,

$$x *_G g_\theta = \sum_{i=0}^{K} \theta_i T_i(\widetilde{L}) x \tag{11}$$

Il filtro definito dalla ChebNet è considerato un miglioramento della Spectral CNN, questo filtro è localizzato nello spazio, cioè significa che il filtro può estrarre features locali indipendentemente dalla grandezza del grafo. Lo spettro della ChebNet è mappato linearmente su $[-1, 1]$.

La Graph Convolutional Network (GCN) introduce un'approssimazione del primo ordine alle ChebNet. Assumono $K = 1$ e $\lambda_{max} = 2$. L'equazione 10 è semplificata così

$$x *_G g_\theta = \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \tag{12}$$

Per limitare il numero dei parametri ed evitare over-fitting, la GCN inoltre assume $\theta = \theta_0 = -\theta_1$, che porta alla seguente definizione di convoluzione sul grafo,

$$x *_G g_\theta = \theta \left(I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \tag{13}$$

Per consentire input e output multi-canale, GCN modifica l'equazione 11 in un layer composizionale, definito

$$H = X *_G g_\theta = f(\bar{A} X \Theta) \tag{14}$$

dove $\bar{A} = I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ e $f(\cdot)$ è una funzione di attivazione. Però usando direttamente $I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ si vanno a creare problemi di instabilità numerica alla GCN, per risolvere questo problema viene applicato uno stratagemma, in pratica si sostituisce $\bar{A} = I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ con $\bar{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, dove $\tilde{A} = A + I_n$ e $\tilde{D}_{ii} = \Sigma_j \tilde{A}_{ij}$. GCN è un metodo di convoluzione nello spettro ma può essere considerato anche come un metodo di convoluzione nello spazio. Dalla prospettiva di convoluzione nello spazio, si può considerare che nella GCN si fa aggregazione delle features dai nodi vicini e l'equazione 14 può essere espressa come

$$h_v = f\left(\Theta^T \left(\sum_{u \in \{N(v) \cup v\}} \bar{A}_{v,u} x_u \right) \right) \quad \forall v \in V \tag{15}$$

Convoluzione nello spazio

Le GNN che fanno convoluzione nello spazio sono analoghe all'operazione di convoluzione di una CNN su un immagine. I metodi basati sulla convoluzione nello spazio definiscono una convoluzione sui grafi basata sulle relazioni spaziali dei nodi. Prendendo in considerazione le immagini, queste possono essere considerate come un grafo che ha una forma speciale, dove ogni pixel rappresenta un nodo e ogni pixel è direttamente connesso ai pixel vicini. A questa immagine viene poi applicato un filtro,

che può essere per esempio 3x3, il quale va a prendere la media pesata del valore del pixel centrale e dei valori dei pixel vicini per ogni canale. In maniera simile, la convoluzione sui grafi basata sullo spazio coinvolge la rappresentazione del nodo centrale e la rappresentazione dei nodi vicini, questo per ottenere una rappresentazione aggiornata del nodo centrale ().

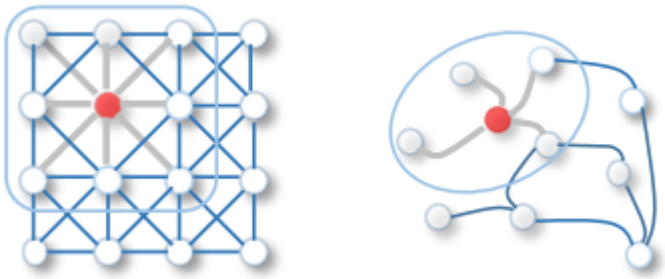

Figura 2: Confronto tra la struttura di un'immagine vista come grafo ed un grafo qualsiasi. Immagine tratta da "A comprehensive survey on graph neural networks"[4]

Viste da un'altra prospettiva, le reti convolutive basate sui metodi spaziali condividono la stessa idea di propagazione/passaggio di messaggi delle informazioni che si ha nella prima GNN. Però, le reti che fanno convoluzione differiscono dalle prime GNN nel modo in cui vanno ad aggiornare lo stato dei nodi. Le prime GNN, per calcolare lo stato, iterano in maniera ricorrente sempre sullo stesso layer. Le reti convolutive, invece, aggiornano lo stato dei nodi andando ad utilizzare diversi layer che hanno diverse matrici dei pesi per ogni layer.

Andando a confrontare i modelli basati sulla convoluzione nello spazio con i modelli che fanno convoluzione nello spettro, si nota che i modelli spettrali hanno fondamenta teoriche nel processamento dei segnali. Dalla costruzione di un nuovo filtro, si possono costruire nuove reti che fanno convoluzione nello spettro sui grafi. Tuttavia, i modelli spaziali sono da preferire ai modelli spettrali a causa dei problemi di efficienza, generalità e flessibilità di questi ultimi. Primo, i modelli spettrali sono meno efficienti dei modelli spaziali e hanno bisogno di calcolare gli auto-vettori o di gestire l'intero

grafo nello stesso momento. I modelli spaziali sono più scalabili su grafi di grandi dimensioni, dato che calcolano la convoluzione direttamente nel dominio del grafo tramite la propagazione dell'informazione. Il calcolo può essere effettuato su un campione di nodi invece che sull'intero grafo. Secondo, i modelli spettrali che fanno affidamento alle regole di Fourier generalizzano male su nuovi grafi, poiché assumono una dimensione fissa per il grafo. Quindi ogni perturbazione a un grafo risulterà in un cambiamento della base degli auto-vettori. I modelli spaziali, al contrario, calcolano la convoluzione del grafo localmente su ogni nodo, dove il peso può essere facilmente condiviso attraverso differenti posizioni e strutture. Terzo, i modelli spettrali sono limitati ad operare su grafi non diretti, mentre i metodi spaziali sono più flessibili e riescono a gestire anche input di grafi di sorgenti diverse così come input di archi, grafi diretti e grafi eterogenei, questo perché gli input del grafo possono essere incorporati facilmente nelle funzioni di aggregazione [4].

DGCNN

La DGCNN[6], () (Deep Graph Convolutional Neural Network) è un modello di graph neural network specializzato per la classificazioni di grafi. Questo modello si può dividere in tre parti: nella prima parte vengono sfruttati dei layer di convoluzione spaziali che effettuano la propagazione dell'informazione tra i nodi del grafo; nella seconda parte si prende l'informazione dei nodi e la si elabora ordinandola tramite un layer di pooling; nella terza parte l'informazione viene passata ad una classica CNN.

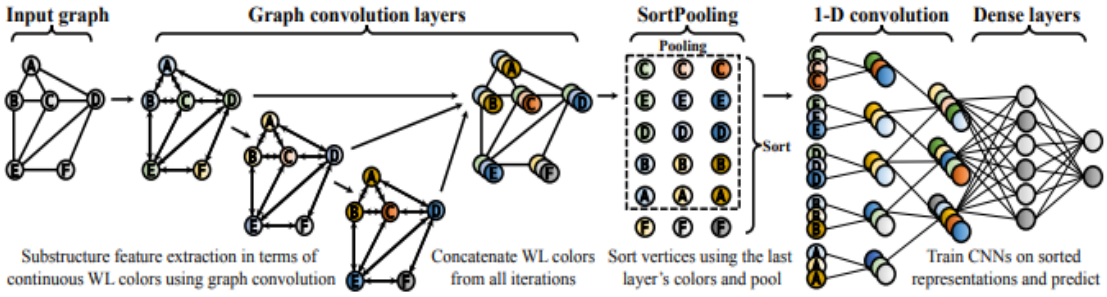

Figura 3: Struttura generale di una DGCNN. Un grafo di input di struttura arbitraria è prima passato attraverso diversi layer di convoluzione sui grafi, dove l'informazione dei nodi è propagata tra i nodi vicini. Poi le feature dei nodi sono ordinate e raggruppate con un layer di SortPooling e infine passate ad una tradizionale struttura CNN per allenare il modello.
Immagine tratta da *"An End-to-End Deep Learning Architecture for Graph Classification"* [6]

Layer di convoluzione sui grafi

Dato un grafo con una matrice di adiacenza A e con una matrice delle features dei nodi $X \in R^{n \times c}$, il layer di convoluzione assume la seguente forma:

$$Z = f\left(\widetilde{D}^{-1}\tilde{A}XW\right) \tag{16}$$

dove $\tilde{A} = A + I$ è la matrice di adiacenza del grafo con l'aggiunta dei self-loop, $\widetilde{D}$ è la matrice diagonale del grado dei nodi dove $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $W \in R^{c \times c'}$ è la matrice dei parametri addestrabili del layer di convoluzione, f è la funzione di attivazione non lineare e $Z \in R^{n \times c'}$ è la matrice di output della funzione di attivazione. La convoluzione sul grafo può essere divisa in quattro fasi. La prima consiste in una trasformazione lineare applicata alla matrice dell'informazione dei nodi da XW, mappa i c canali delle features nei c' canali del layer successivo. La matrice dei pesi è condivisa da tutti i nodi. La seconda fase, $\tilde{A}Y$ dove $Y = XW$, propaga l'informazione di un nodo sui nodi vicini e su sé stesso. La terza fase normalizza tramite la moltiplicazione per $\widetilde{D}^{-1}$. Nell'ultima fase si applica una funzione di attivazione non lineare e l'output ottenuto è il risultato della convoluzione.

La convoluzione sui grafi aggrega l'informazione di un nodo e quella dei suoi vicini per ottenere una sotto-struttura dell'informazione locale. Per ottenere una sotto-

struttura dell'informazione su più livelli, vengono concatenati più layer di convoluzione () come segue

$$Z^{t+1} = f\left(\widetilde{D}^{-1}\tilde{A}Z^{t}W^{t}\right) \tag{17}$$

dove $Z^0 = X$, $Z^t \in R^{n\times c_t}$ che è l'output del t-esimo layer di convoluzione, c^t è il numero dei canali di output del layer t, e $W^t \in R^{c_t\times c_{t+1}}$mappa c_t canali in c_{t+1}canali. Dopo i layer di convoluzione si trova un layer che concatena orizzontalmente gli output Z^t, dove $t = 1, \dots, h$, per formare l'output concatenato, $Z^{1:h} = [Z^1, \dots, Z^h]$, dove h è il numero di layer di convoluzione e $Z^{1:h} \in R^{n\times\sum_1^h c_t}$. Nell'output concatenato $Z^{1:h}$, ogni riga può essere considerata come un "descrittore delle features" di un nodo. Questi "descrittore delle features" possiedono già un ordimento orizzontale dato dal WL color, in cui ad ogni canale di features viene assegnato un colore in base a quanta informazione viene presa dai nodi vicini. Ed essendo i suddetti concatenati dall'output del primo layer di convoluzione fino all'output dell'ultimo layer di convoluzione, sono già ordinati orizzontalmente (cioè i canali delle features di un nodo sono già ordinati). Questo WL color utilizzato nelle DGCNN è una versione meno stringente del Weisfeiler-Lehman subtree kernel[7].

Layer di SortPooling

Il compito del layer di SortPooling è di ordinare i descrittori delle features, in cui ognuno rappresenta un nodo, in un ordine consistente, prima di passarli ad un tradizionale CNN. Questo layer ordina i descrittori delle features dei nodi tra loro in base al WL color ottenuto dai precedenti layer di convoluzione. Per il layer di SortPooling, l'input è un tensore $Z^{1:h}$di dimensione $n \times \sum_1^h c_t$, dove ogni riga è il descrittore delle features di un nodo mentre ogni colonna è un canale di features. L'output di questo layer è un tensore $k \times \sum_1^h c_t$, dove k è un valore intero definito

dall'utente. Questo layer va ad ordinare in maniera decrescente i nodi, questo ordinamento viene fatto all'inizio in base all'ultimo canale di features calcolato Z^h. Se due nodi o più nodi hanno lo stesso valore nell'ultimo canale, per ordinarli si andrà a tenere conto del canale di features precedente Z^{h-1}. Questo ordinamento è simile all'ordinamento lessicografico, eccetto che per il fatto che si controlla da destra verso sinistra. Il layer di SortPooling dopo aver ordinato i nodi, si occupa anche di uniformare la grandezza del tensore di output. Dopo l'ordinamento dei nodi, va a tagliare o estendere il tensore dalla dimensione n alla dimensione k, questo viene fatto per uniformare la dimensione dei grafi, dato che può essere variabile, ad una dimensione fissa k. In pratica, se $n > k$ allora elimina le ultime $n - k$ righe, se $n < k$ va ad aggiungere $k - n$ righe di zeri.

Questo layer di Sortpooling offre un altro beneficio alla rete: il fatto che il passaggio dei gradienti per l'addestramento dei layer può transitare per il layer di SortPoling ed addestrare i valori dei pesi dei layer di convoluzione sui grafi precedenti. Ciò è possibile perché il layer di SortPooling memorizza l'ordinamento dell'input ricevuto.

Remaining layers

Dopo il SortPooling, si ha un tensore Z^{sp} di dimensione $k \times \sum_1^h c_t$ dove ogni riga rappresenta un nodo e ogni colonna rappresenta un canale di feature. Per addestrare una CNN, viene rimodellato il tensore Z^{sp} nella forma $k(\sum_1^h c_t) \times 1$ vettori riga. Dopo viene aggiunto un layer 1-D convolution con kernel e stride di dimensione $\sum_1^h c_t$. Dopo diversi layer di MaxPooling e layer 1-D convolutional, c'è un layer denso seguito da una layer softmax.

NEXT ACTIVITY PREDICTION

L'attività che si è svolta inizialmente è stata quella di trovare un modo di compiere next activity prediction tramite GNN. Inizialmente si sono visionate le varie tipologie di graph neural network presenti in letteratura, le quali sono state poi divise in base al tipo di predizione che possono compiere. Si è visto che con le GNN è possibile fare diversi tipi di attività, suddivise in classificazione di nodi o di archi, predizione di nuovi archi e classificazione di interi grafi. La classificazione dei nodi, che permette di predire la label di un nodo che non la possiede, è stata scartata perché per poterla effettuare si aveva bisogno di un grafo. Oltre a raffigurare l'istanza di processo incompleta di cui si vuole predire la successiva attività, ha anche bisogno dell'informazione strutturale del nodo di cui si vuole andare a predire la label. In pratica, si ha bisogno anche degli archi che collegano il nodo di cui si vuole predire la label al resto del grafo. Anche la predizione di nuovi archi è stata scartata, perché queste reti neurali sono in grado di predire nuovi archi solo tra i nodi già presenti nel grafo. Quindi, si è scelto di ripotare il problema di predizione dell'attività successiva come un problema di classificazione di grafi. Per poter fare la classificazione dei grafi si è andati ad assegnare ad ogni grafo, che rappresenta un istanza di processo incompleta di cui si vuole predire l'attività successiva, una label che rappresenta la prossima attività che si dovrà svolgere.

DATASET E RETE NEURALE

Il dataset utilizzato inizialmente per l'allenamento della rete, è derivato dal dataset "BPI 2012 complete". Successivamente sono state anche svolte delle prove della rete con il dataset "Helpdesk". Il dataset "Helpdesk" contiene i log che descrivono la gestione di ticket all'interno di un azienda italiana produttrice di software. Il dataset "BPI 2012 complete" contiene i log relativi a dei processi eseguiti all'interno di una

banca tedesca. Il lavoro iniziale è stato svolto su una versione già elaborata di BPI, dataset che contiene le varie istanze di processo, presenti in “BPI 2012 complete”, rappresentate sotto forma di grafi “instance graph”. Anche per “Helpdesk” si lavorerà non con il dataset contenente i log, ma con quello contenente gli instance graph.

Un instance graph è una rappresentazione a grafo di una specifica istanza di un processo, cioè di una sua esecuzione. Poichè ad ogni esecuzione può essere seguito solo un ramo dei vari percorsi alternativi del processo, l'istanza non presenta scelte condizionali. L'istanza può però presentare esecuzioni concorrenti di due o più attività. Un instance graph è quindi una rappresentazione generalizzata di un insieme di tracce del log, in cui i nodi rappresentano le attività svolte nel processo, e un arco tra due nodi rappresenta una relazione di sequenza fra le due attività. La presenza di più archi che partono dallo stesso nodo è quindi interpretata come esecuzione parallela delle attività che seguono il nodo stesso. Per la generazione degli instance graph viene utilizzata in questa tesi la procedura descritta in [13]. Un esempio di instance graph è mostrato in Figure 4.

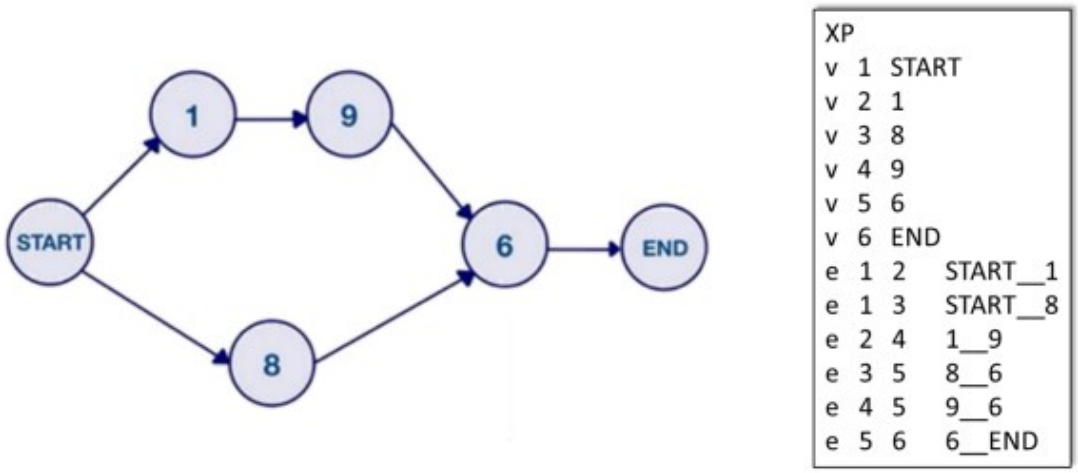

Figura 4: A destra della figura è riportata una porzione del file .g di Helpdesk che descrive un processo sotto forma di instance graph. A sinistra è possibile osservare la traduzione di del processo in grafo orientato.

Agli instance graph, vengono aggiunti due nodi artificiali, uno all'inizio ed uno alla fine di ogni istanza di processo. I nodi aggiunti sono quelli che contengono o l'attività START o l'attività END. Per creare i dataset che si utilizzeranno per l'addestramento della rete neurale, si è deciso di elaborare ulteriormente i dataset forniti. Tramite questa elaborazione, per ogni instance graph verranno creati di una serie di sottografi che rappresentano delle istanze di processo incomplete.

I dataset forniti inizialmente contengono:

	DATASET	
	Helpdesk	BPI12
No. di instance graph	3084	13087
No. di tipi di attività	11	25

Tabella 1

Il dataset iniziale definisce per ogni istance graph i nodi e gli archi che lo compongono. I nodi sono definiti da due informazioni, un id incrementale che fornisce anche l'ordine di esecuzione delle attività all'interno dell'istanza di processo e il tipo di attività che viene svolta. Gli archi contengono solo l'informazione strutturale di collegamento tra i nodi del grafo. Nella si può notare un esempio di come il grafo è salvato all'interno del dataset iniziale.

Per la costruzione dei sottografi sono stati prima costruiti gli istance graph incompleti. Da ogni grafo sono stati creati N-2 sottografi, dove N è il numero di nodi all'interno di un grafo. Questo perché si è scelto che il numero minimo di nodi all'interno di un sottografo deve essere uguale a 2. La creazione dei sottografi è stata fatta in maniera incrementale aggiungendo uno alla volta i nodi del grafo all'interno del sottografo. I nodi da aggiungere sono stati decisi seguendo l'ordinamento dato dall'id dei nodi, il quale rispetta anche l'ordinamento parziale che è definito dalla struttura del grafo. Per esempio, come si evince dalla , si è andati a creare un primo sottografo formato dai nodi 1 e 2 ed il relativo arco di congiungimento tra i nodi, mentre come attività da

predire è stata inserita l'attività del nodo 3. Successivamente si è creato il sottografo contenente i nodi 1-2-3 ed i relativi archi di collegamento tra i nodi, mentre come attività da predire è stata presa quella del nodo 4. Si continua nello stesso modo fino a creare l'ultimo sottografo, che conterrà i nodi da 1 a N-1 con i relativi archi e l'attività da predire sarà quella del nodo N.

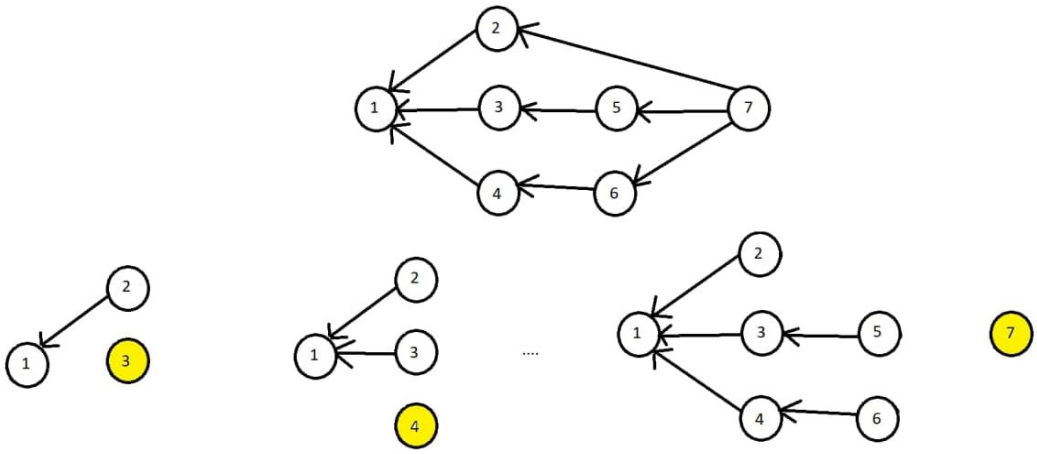

Figura 5: Esempio di creazione dei sottografi per l'allenamento della rete da un istance graph

Da qui in poi non ci si riferirà più agli elementi del dataset creato come sottografi ma direttamente come grafi.

Il tipo di attività, all'interno del dataset creato per l'addestramento della rete, viene codificato in modo differente in base al tipo di informazione che deve andare a rappresentare all'interno del grafo. Quando è utilizzata come informazione collegata ad un nodo del grafo, viene codificata come one-hot-vector, mentre quando si utilizza come informazione dell'attività da predire collegata al grafo, viene codificata tramite una label categorica numerica. I dataset ottenuti dopo l'elaborazione sono costituiti nella seguente maniera (Tabella 2).

	DATASET	
	Helpdesk	BPI12
No. di grafi	17514	164506
Lunghezza media	3,5	12,91
Deviazione standard	1,42	10,29

Tabella 2

Per il dataset "Helpdesk" la lunghezza dei grafi varia da 2 a 15, mentre il numero di nodi contenuti in un grafo di "BPI 2012 complete" varia tra 2 e 97. Il tipo di attività contenuta come informazione all'interno di un nodo per il dataset "Helpdesk" è rappresentata da una delle 10 attività seguenti: START, 1, 2, 3, 4, 5, 6, 7, 8, 9. Come attività da predire per un grafo all'interno del dataset sono presenti 9 classi di attività diverse. Si può vedere quali sono queste attività e la loro distribuzione all'interno del dataset nella Tabella 3.

1	500
2	44
4	14
5	5
6	4148
7	4
8	4230
9	961
END	3804

Tabella 3

Nel dataset "BPI 2012 complete" l'informazione contenuta nei nodi è rappresentata da una delle 24 attività seguenti: START, ASUBMITTED, APARTLYSUBMITTED, APREACCEPTED, AACCEPTED, OSELECTED, AFINALIZED, OCREATED, OSENT, WCompleterenaanvraag, WNabellenoffertes, OSENTBACK, AREGISTERED, AAPPROVED, OACCEPTED, AACTIVATED, WValiderenaanvraag, OCANCELLED, ADECLINED, ACANCELLED, WAfhandelenleads, ODECLINED, WNabellenincompletedossiers, WBeoordelenfraude.

Come attività da predire per un grafo all'interno del dataset sono presenti 23 classi di attività diverse. Si può vedere quali sono queste attività e la loro distribuzione all'interno del dataset nella Tabella 4.

APARTLYSUBMITTED	13087
APREACCEPTED	7367
AACCEPTED	5113
OSELECTED	7030
AFINALIZED	5015
OCREATED	7030
OSENT	7030
WCompleterenaanvraag	23967
WNabellenoffertes	22976
OSENTBACK	3454
AREGISTERED	2246
AAPPROVED	2246
OACCEPTED	2243
AACTIVATED	2246
WValiderenaanvraag	7895
END	13087
OCANCELLED	3655
ADECLINED	7635
ACANCELLED	2807
WAfhandelenleads	5898
ODECLINED	802
WNabellenincompletedossiers	11407
WBeoordelenfraude	270

Tabella 4

La rete neurale utilizzata per le prove è una DGCNN, la cui implementazione è stata presa dal repository github[8] di Pytorch-geometric. Questa implementazione, dopo il layer di Sort Pooling ha un solo layer convoluzionale, che è seguito da un dense layer. I parametri modificabili di questa implementazione che venivano messi a disposizione per ottenere diverse configurazioni della rete, sono:

- il numero di layer di convoluzione sui grafi, che è stato impostato inizialmente uguale a 5 e non è mai stati modificato in seguito;

- “k”, indica il valore di taglio del layer di SortPooling per uniformare l’input per la CNN, è stato sfruttato impostando differenti valori;
- Il valore della funzione di dropout, per quale sono stati provati diversi settaggi.

RISULTATI

Per descrivere i risultati ottenuti dall’allenamento della rete per le varie configurazioni, viene mostrata una tabella riassuntiva. Successivamente verranno presentate delle tabelle e delle immagini, con cui si presenterà un analisi dettagliata dei risultati ottenuti dalla migliore configurazione della rete con ogni dataset. La Tabella 5 e la Tabella 6 mostrano i risultati di accuracy e loss delle fasi di training e test per i dataset “HELPDESK” e “BPI 2012” rispettivamente. Nella prima colonna sono mostrati i valori di configurazione con cui è stata addestrata la rete e per quella determinata configurazione si mostra anche l’epoca in cui si è avuto il miglior valore di loss per il test. La fase di evaluation è stata fatta sul modello della rete recuperato all’epoca migliore.

layer	k	dropout	epoca	Train Accuracy	Test Accuracy	Train Loss	Test Loss
5	5	0,1	270	0,8235	0,8232	0,5363	0,5551
5	5	0,2	159	0,8186	0,8203	0,5576	0,5646
5	3	0,1	68	0,8158	0,8218	0,5519	0,5578
5	3	0,2	348	0,8163	0,8220	0,5563	0,5638
5	30	0,1	99	0,8208	0,8181	0,5484	0,5688
5	30	0,2	183	0,8234	0,8186	0,5418	0,5678

Tabella 5

layer	k	dropout	epoca	Train Accuracy	Test Accuracy	Train Loss	Test Loss
5	5	0,05	4	0,7040	0,7208	0,8576	0,8368
5	7	0,1	3	0,6561	0,6865	1,0296	0,9534
5	7	0,2	3	0,6675	0,6861	0,9859	0,9606
5	13	0,2	6	0,6801	0,6926	0,9297	0,8871
5	13	0,3	3	0,5677	0,6123	1,2876	1,1433
5	13	0,4	5	0,6228	0,6532	1,1314	0,9765
5	13	0,5	4	0,6006	0,6494	1,2266	1,037
5	15	0,2	6	0,6401	0,6647	1,0212	0,9457
5	15	0,3	5	0,6554	0,6804	1,0362	0,9542
5	15	0,4	4	0,6371	0,9506	1,1031	0,9619
5	15	0,5	11	0,5526	0,6053	1,3643	1,1505
5	20	0,3	2	0,6589	0,6862	0,9938	0,9134
5	20	0,4	7	0,6560	0,6886	1,0468	0,9325
5	20	0,5	4	0,6202	0,6760	1,1699	0,9604
5	30	0,2	4	0,6177	0,6518	1,1131	1,0065
5	30	0,3	4	0,6497	0,6893	1,0589	0,9475
5	30	0,4	8	0,6394	0,6664	1,0616	0,9887
5	30	0,5	10	0,6497	0,6893	1,0589	0,9475
5	96	0,5	2	0,5887	0,6296	1,2074	1,0596

Tabella 6

La configurazione della rete, i cui risultati si andranno a vedere in dettaglio per il dataset "HELPDESK", è quella con 5 layer di convoluzione sui grafi, valore di k=5 e dropout=0.1. Mentre per il dataset "BPI 2012 complete" si vedrà in dettaglio la configurazione con 5 layer, k=5 e dropout=0.05.

Nelle e è possibile vedere l'andamento delle curve di accuracy e di loss durante l'addestramento della DGCNN con il dataset "HELPDESK", mentre nella sono presenti i grafici per l'andamento delle curve per il dataset "BPI 2012 complete".

Nelle si osserva la matrice di confusione sul test set dell'epoca migliore, nella quale è stato raggiunto il miglior valore di loss sul test set per il dataset "HELPDESK". Nella si mostra la matrice di confusione nelle stesse condizioni precedenti per il dataset "BPI 2012 complete". In e in si possono vedere le matrici di confusione per l'epoca 48 per il train set e per il test set del dataset "BPI 2012 complete". Mentre in e si possono vedere le matrici di confusione all'epoca 50. Dai grafici dell'andamento di accuracy e

loss per il dataset “BPI 2012 complete”, si nota un miglioramento entro le prime dieci epoche di addestramento. Successivamente si ha un peggioramento dei vallori di accuracy e loss.

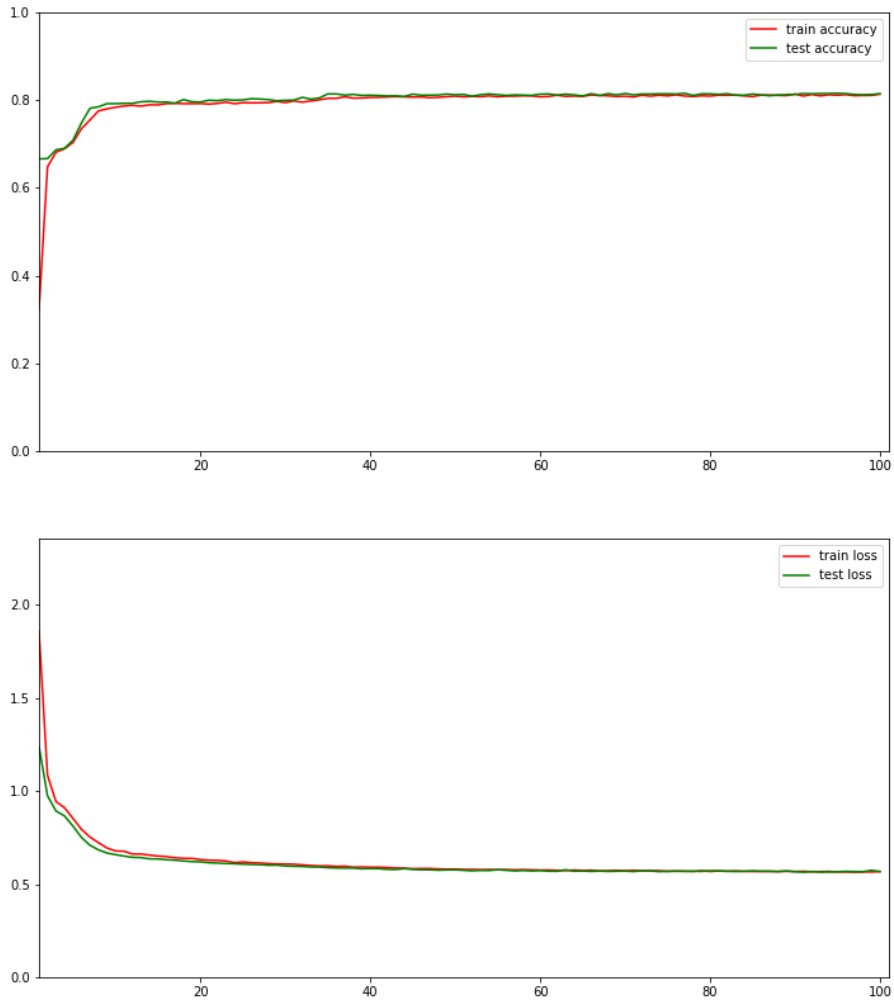

Figura 6: grafici di accuracy (in alto) e loss (in basso) dell’addestramento della DGCNN con il dataset “HELPDESK” per le epoche 1-100

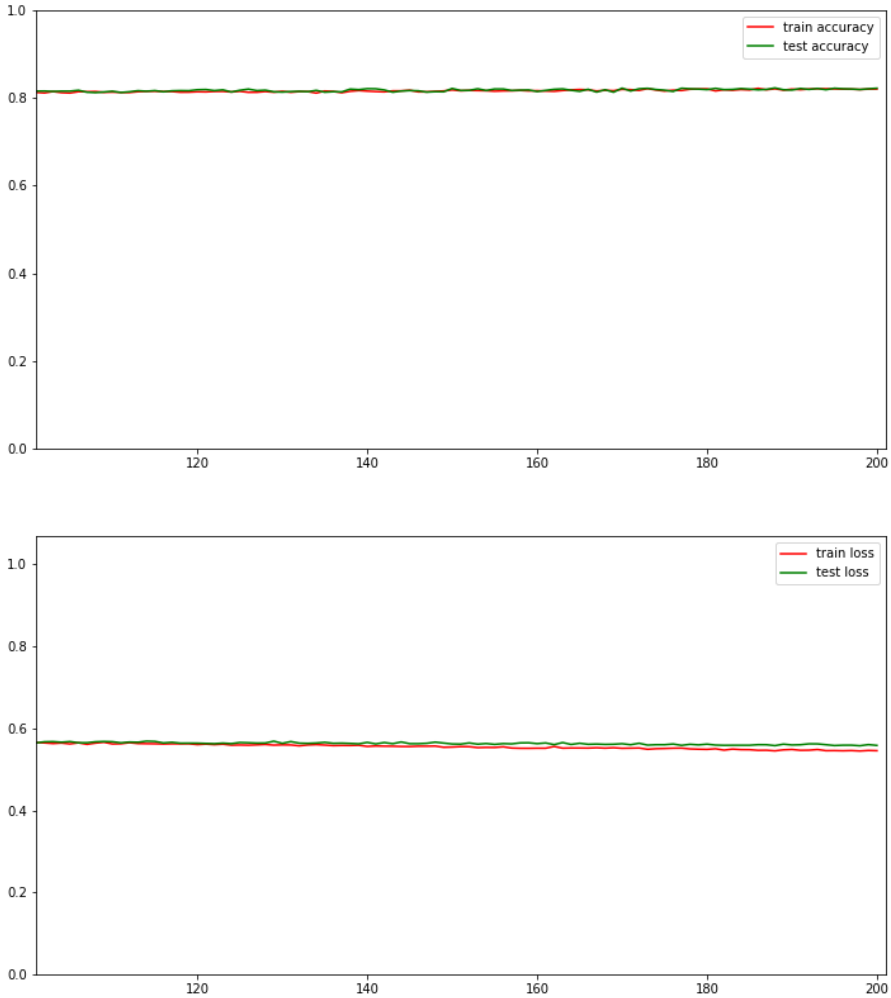

Figura 7: grafici di accuracy (in alto) e loss (in basso) dell'addestramento della DGCNN con il dataset "HELPDESK" per le epoche 101-200

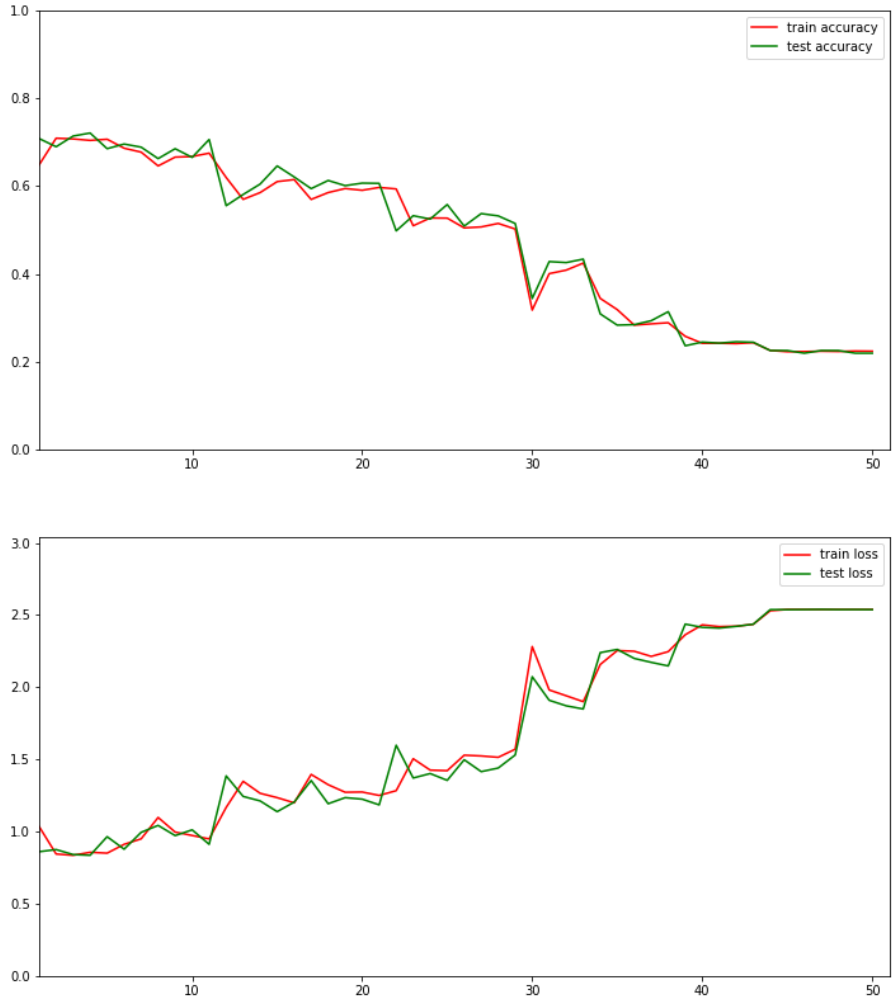

Figura 8 : grafici di accuracy (in alto) e loss (in basso) dell'addestramento della DGCNN con il dataset "BPI 2012" per le epoche 1-50

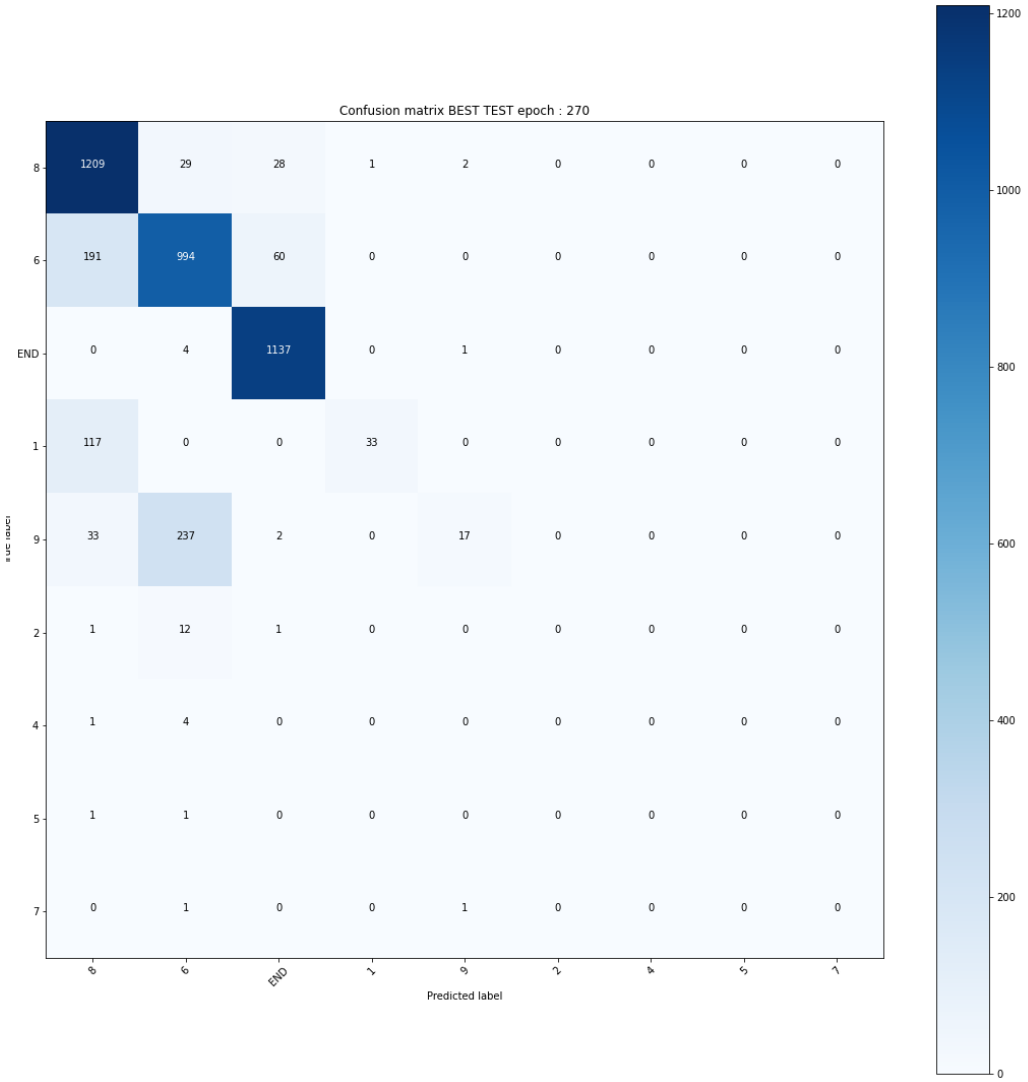

Figura 9: matrice di confusione per del test set per il miglior valore di loss di test per il dataset "HELPDESK"

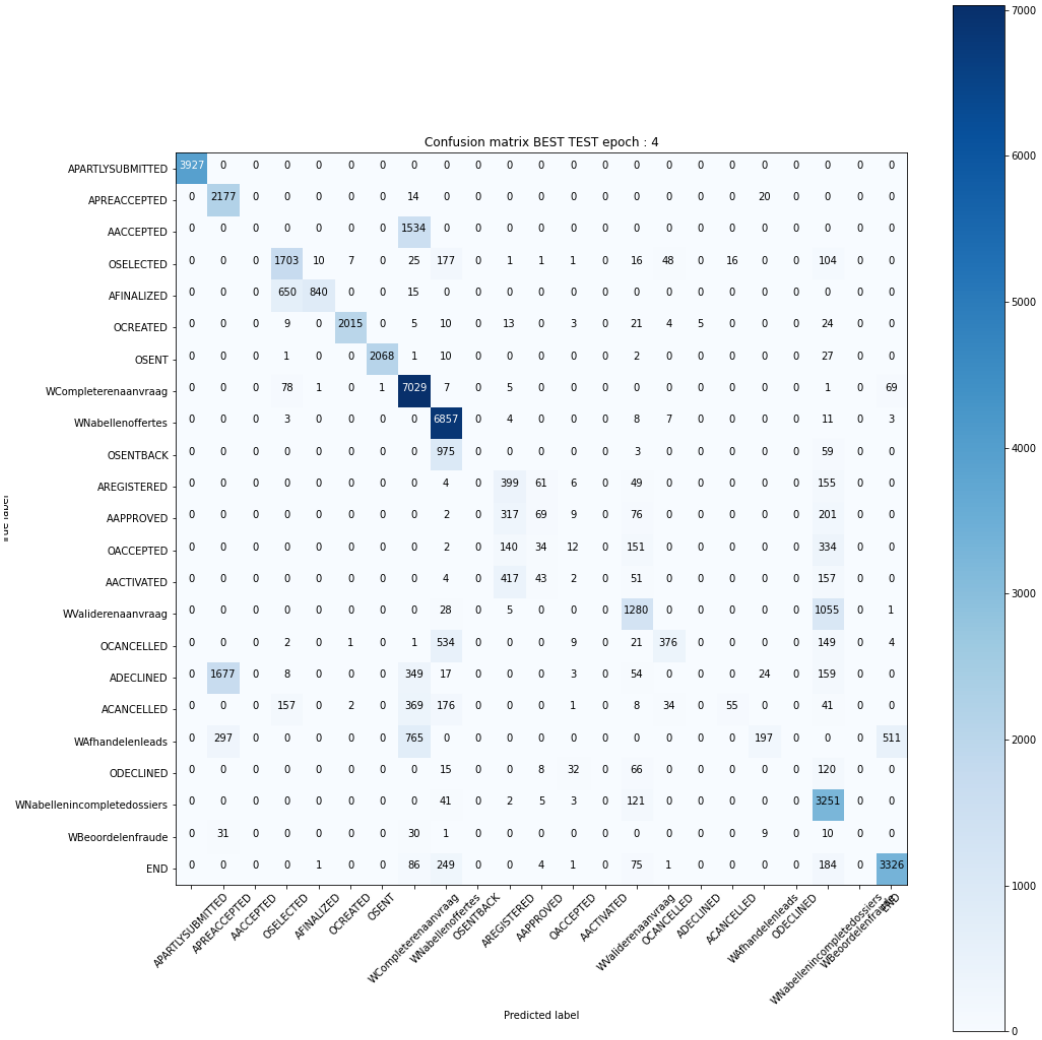

Figura 10: matrice di confusione per del test set per il miglior valore di loss di test per il dataset "BPI 2012"

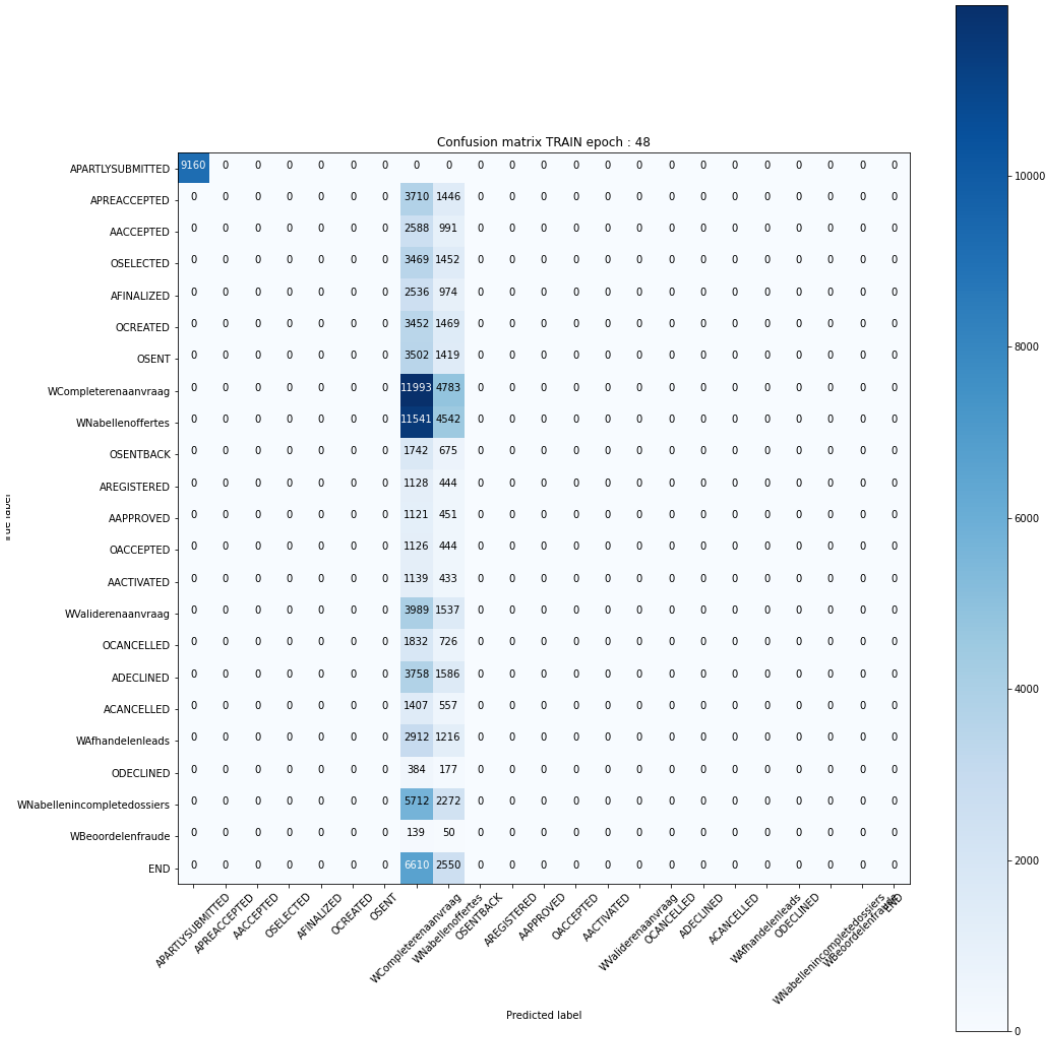

Figura 11: matrice di confusione del train set all'epoca 48 per il dataset "BPI 2012"

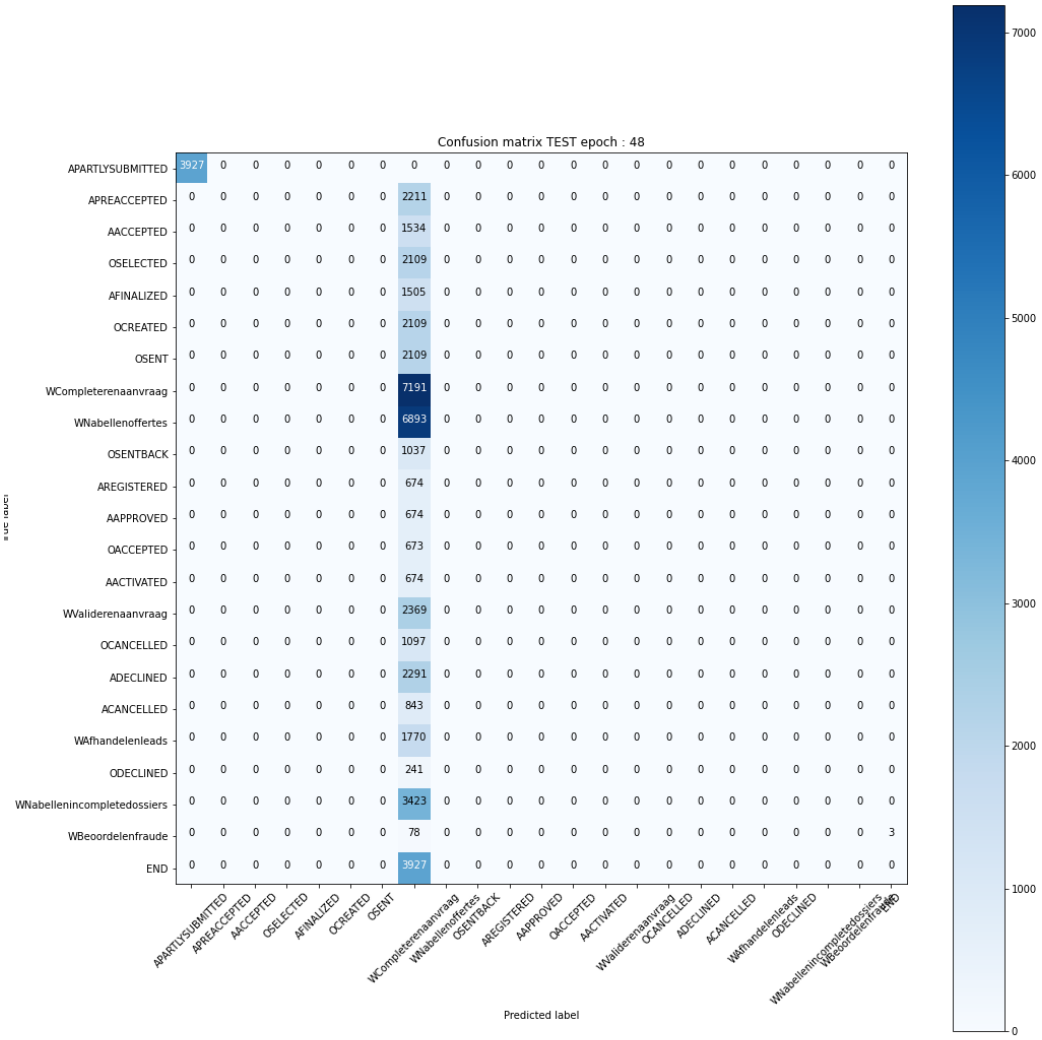

Figura 12: matrice di confusione del test set all'epoca 48 per il dataset "BPI 2012"

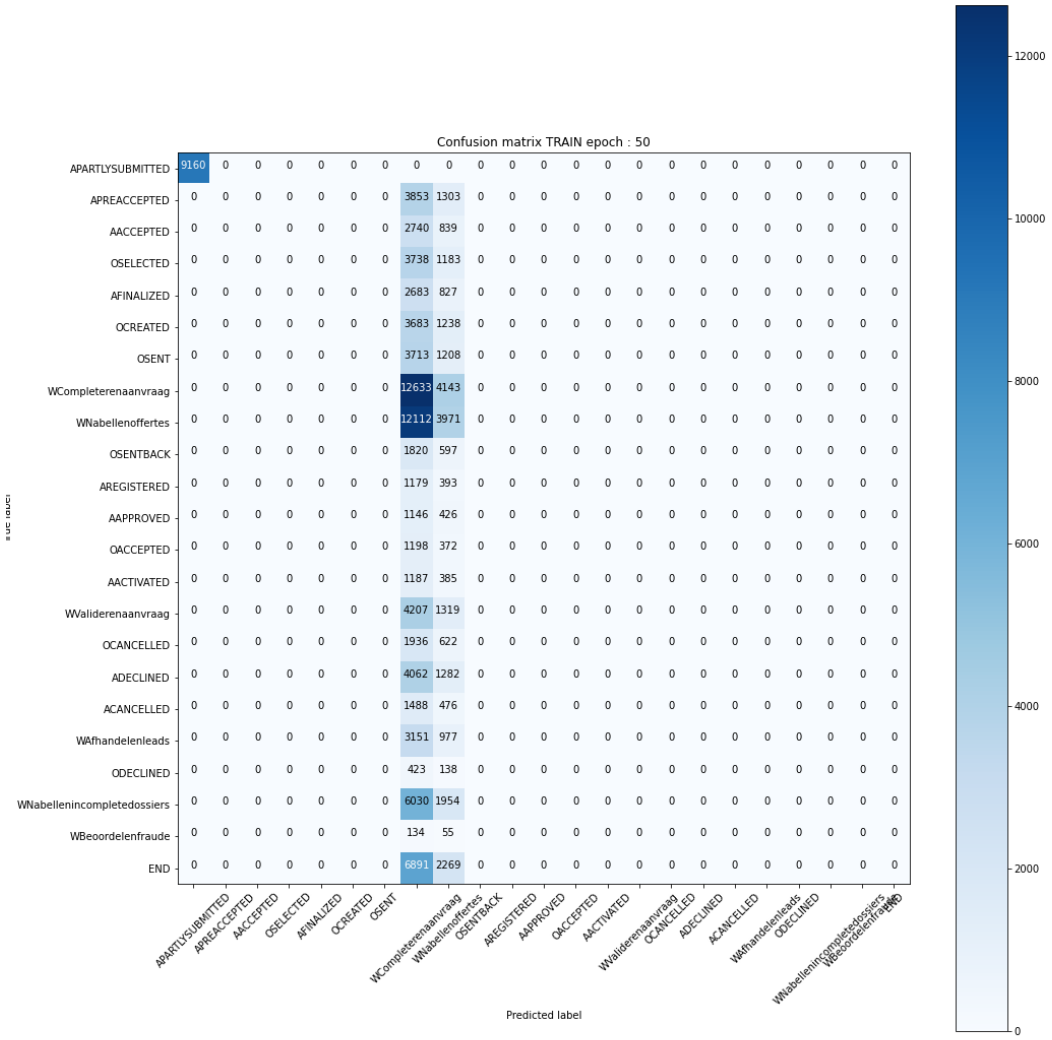

Figura 13: matrice di confusione del train set all'epoca 50 per il dataset "BPI 2012"

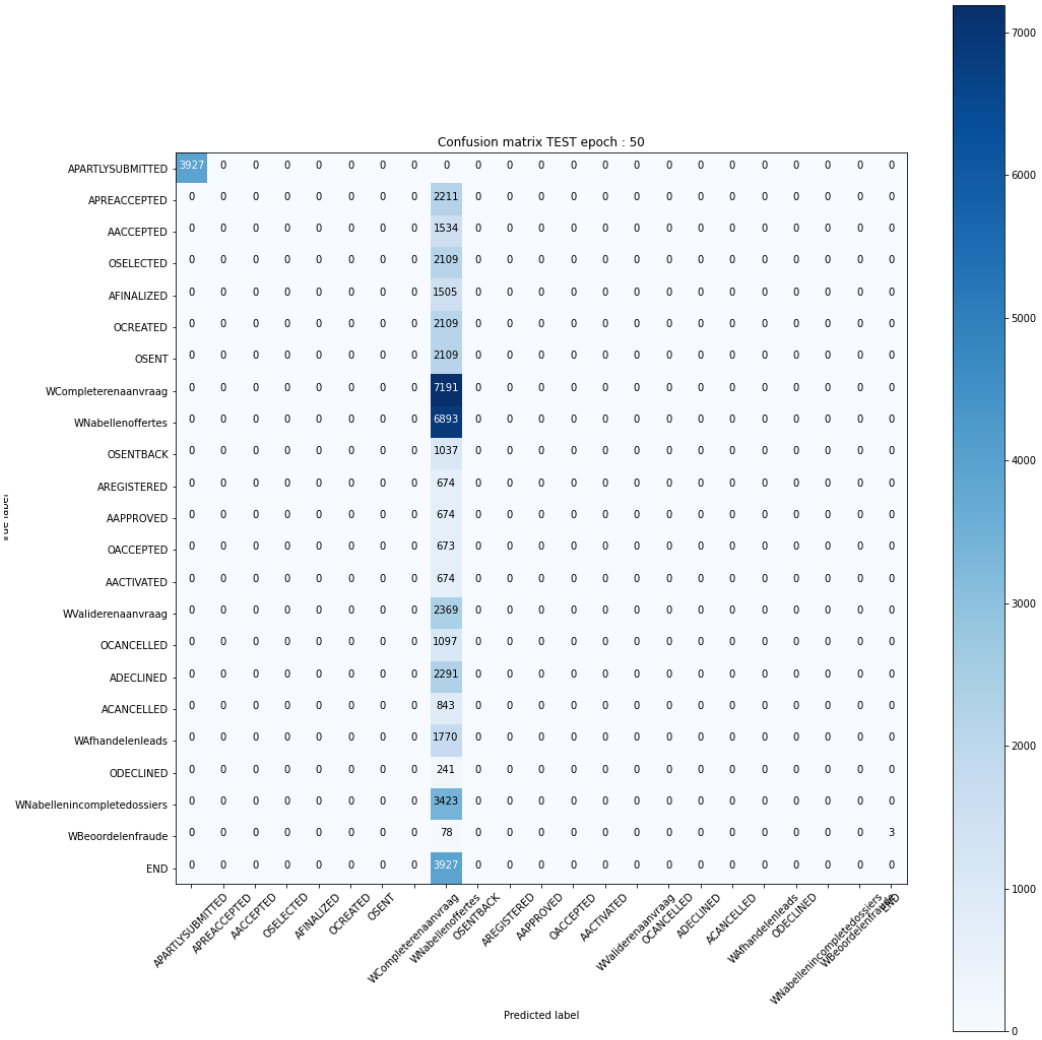

Figura 14: matrice di confusione del test set all'epoca 50 per il dataset "BPI 2012"

Osservando le matrici di confusione per il train set (e) delle epoche dove i valori di accuracy e loss si stabilizzano, si può notare che la rete inizia a predire solo le attività più frequenti all'interno del dataset. Se invece si osservano le matrici di confusione per il test set (e), si nota che la predizione si concentra su un'unica classe alternandosi tra le due più frequenti. Questo comportamento è stato attribuito sia allo sbilanciamento delle classi all'interno del dataset, sia ai principi su cui si basa la propagazione dell'informazione all'interno della rete, che con pochi parametri tende a spargere l'informazione tra i nodi, in particolare questo fenomeno è noto come over-smoothing. L'over-smoothing consiste nel non riuscire più a distinguere la

rappresentazionne dei nodi in differenti classi, questo perchè dopo il passaggio dell'informazione tra un nodo e i suoi vicini, quel nodo riceve parte dell'informazione dei suoi vicini, se questa operazione di diffusione dell'informazione tra i nodi avviene per abbastanza volte si crea un appiattimento dell'informazione contenuta all'interno dei nodi. Nel nostro caso questo problema è molto significativo dato che la dimensione dei grafi con cui si lavora è molto piccola. Si è comunque preso per buono il modello estratto dall'addestramento, perché prendendo i

grafici di loss e accuracy del dataset "HELPDESK" () si vede che la rete neurale riesce ad arrivare a convergenza molto velocemente.

Nell'analisi in dettaglio che si è svolta per il dataset "BPI 2012" si è andati a studiare il tipo di errore che viene commesso nella predizione. E per ogni predizione si è andati anche a vedere la seconda miglior classe predetta. Lo studio che è stato effettuato sul tipo di errore è stato fatto perché l'ordinamento delle attività all'interno dei grafi, è un ordinamento parziale: Questo perché all'interno di un istanza di processo può esserci la presenza di azioni svolte in parallelo, che porta alla presenza di rami paralleli all'interno degli instance graph. Per descrivere il tipo di errore che può esserci tra due attività si è andati a vedere la rete di Petri ricavata dal dataset. Si è scelto di dividere l'errore commesso dalla rete in fase di predizione in cinque tipologie differenti, che sono:

- Parallelo: se l'attività predetta si trova su un ramo parallelo della rete di preti rispetto al ramo in cui si trova la predizione esatta, la divisione dei due rami deve avvenire con uno split che può essere paragonato ad un and;
- Precedente: se l'attività predetta è precedente all'attività che indica la predizione esatta nella rete di Petri;

- Successivo: se l'attività predetta è successiva all'attività che indica la predizione esatta nella rete di Petri;
- Alternativo: se l'attività predetta si trova su un ramo alternativo della rete di preti rispetto al ramo in cui si trova la predizione esatta, la divisione dei due rami deve avvenire con uno split che può essere paragonato ad un or;
- End: se viene predetto l'end al posto della predizione esatta.

Nella rete di Petri è presente anche un ciclo, che dalla posizione precedente all'end permetteva di tornare in una posizione quasi iniziale della rete di Petri. Tuttavia, il ciclo non è stato considerato per determinare la relazione di errore che può intercorrere fra due tipi di attività distinte.

Lo studio sul tipo di errore è stato fatto anche sulla seconda miglior predizione. I risultati in base al tipo di errore sono stati calcolati utilizzando due diverse reti di Petri. Le suddette reti sono state ricavate con il programma ProM, tramite il tool che si chiama "Mine Petri net with Inductive Miner". Questo tool, prima di ricavare la rete di Petri, permette di impostare un valore di noise che è stato fissato una volta a 20 e una volta a 0. Nella e nella sono mostrate le reti di Petri con noise uguale a 20 e con noise uguale a 0, rispettivamente.

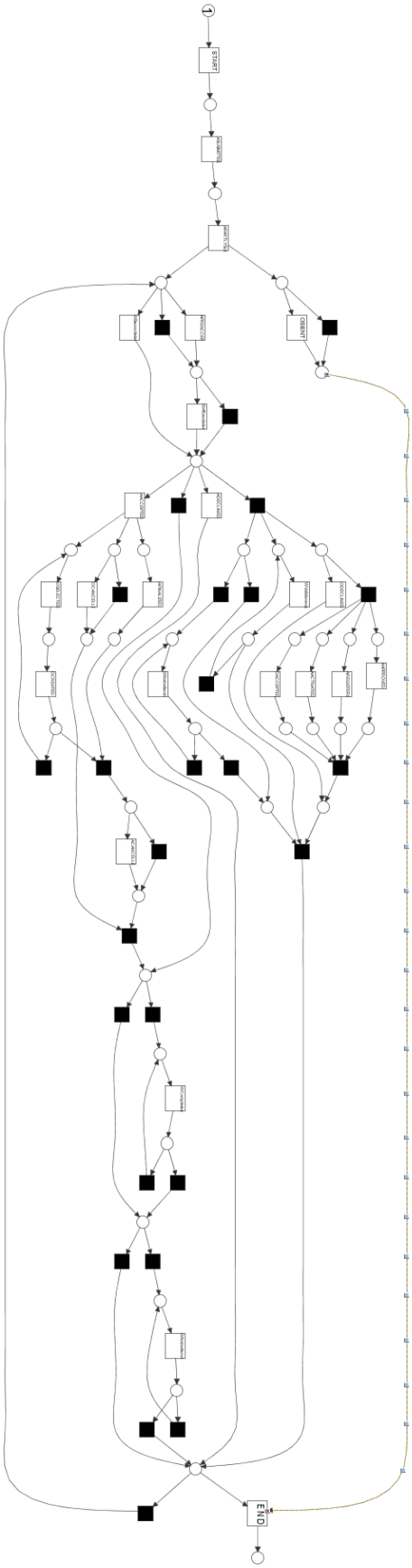

Figura 15: Rete di Petri con noise 20

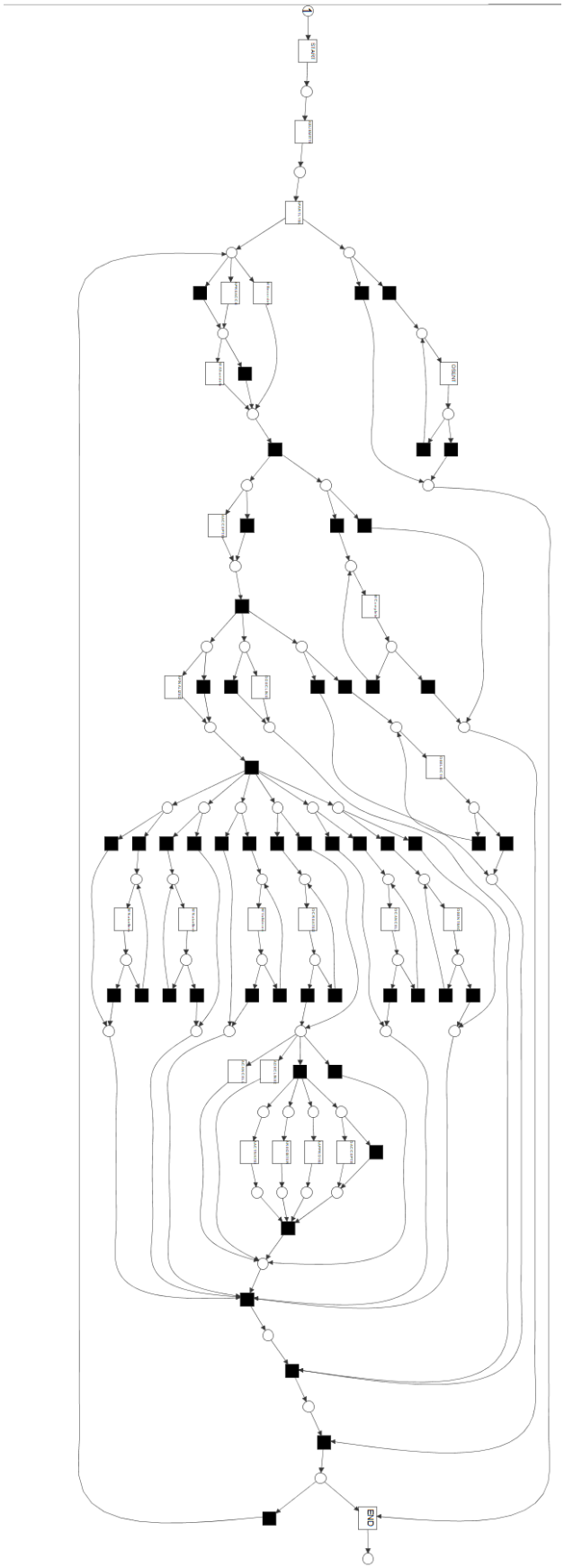

Figura 16 : Rete di Petri con noise a 0

Nella Tabella 7 sono mostrati i risultati delle predizioni e degli errori ricavati dalla rete di Petri con il valore di noise uguale a 20. Mentre nella Tabella 8 sono mostrati i risultati in base alla rete di Petri con noise uguale a 0. Nella rete di Petri creata con noise uguale a 20, l'attività OSENTBACK non è presente. Per questa attività sono state prese le relazioni di errore che si ottengono rispetto alla rete di Petri con noise a 0.

	Giuste	giuste + parallele	errori	parallele	precedenti	successive	alternative	end
1° predizione	72,15%	83,03%	27,85%	39,04%	21,13%	26,88%	8,72%	4,18%
2° predizione	57,04%	74,86%	42,96%	41,47%	30,52%	6,93%	20,95%	0,14%

Tabella 7

	giuste	giuste + parallele	errori	parallele	precedenti	successive	alternative	end
1° predizione	72,11%	91,69%	27,89%	70,22%	18,89%	6,27%	0,31%	4,31%
2° predizione	57,07%	82,85%	42,93%	60,06%	32,14%	7,04%	0,71%	0,05%

Tabella 8

Sono state fatte anche delle tabelle che riportano il comportamento della miglior seconda predizione in base all'errore della prima predizione (Tabella 9 e Tabella 10).

Tipo di errore 1°Predizione	2°Predizione					
	giuste	parallele	precedenti	Successive	alternative	end
Parallele	61,92%	34,53%	0,02%	0,22%	3,32%	0,00%
precedenti	63,81%	0,07%	25,31%	10,50%	0,31%	0,00%
successive	55,89%	14,88%	26,17%	1,33%	1,52%	0,22%
alternative	2,50%	3,59%	8,18%	3,51%	82,22%	0,00%
end	99,30%	0,17%	0,17%	0,17%	0,17%	0,00%

Tabella 9: noise uguale a 20

Tipo di errore 1°Predizione	2°Predizione					
	giuste	parallele	precedenti	Successive	alternative	end
parallele	49,62%	36,58%	13,49%	0,17%	0,11%	0,03%
precedenti	65,90%	0,00%	22,68%	11,42%	0,00%	0,00%
successive	88,43%	0,81%	0,00%	7,29%	3,47%	0,00%
alternative	0,00%	4,65%	0,00%	93,02%	2,33%	0,00%
end	98,31%	0,67%	1,01%	0,00%	0,00%	0,00%

Tabella 10: noise uguale a 0

Dalle Tabelle 3 e 4 si può notare che se per la prima predizione si prendessero per buoni gli errori rispetto ad attività parallele, si otterrebbero per entrambe le reti Petri

dei miglioramenti significativi sulle predizioni giuste, considerando che nel nostro caso il parallelismo è ciò che maggiormente può portare a problemi di classificazione. Dalle Tabelle 5 e 6, possiamo evincere che il secondo miglior valore di predizione per quasi tutti i tipi di errore, tranne che per gli errori su attività alternative, è l'attività corretta.

Lo studio del tipo di errore si è fatto anche in base al tipo di attività che rappresenta la predizione esatta. I risultati sono mostrati nelle Tabella 11 e Tabella 12. Da queste e dalla Tabella 1 si nota che le attività ODECLINED e WBeoordelenFraude non sono mai predette in maniera corretta, per via della bassa numerosità di campioni. Se però si va a guardare in dettaglio ODECLINED, la maggior parte delle predizione errate sono su un'attività parallela in entrambe le reti di Petri. Si nota che per noise a 0 è in parallelo con quasi tutte le attività, ma per la rete di Petri con noise uguale a 20 è in parallelo solo con 4 attività. Andando a guardare in dettaglio anche WBeoordelenFraude si vede che per il 50% circa delle volte l'errore è su un'attività alternativa. Guardano le reti si vede che questa attività ne ha solo due ad essa alternative. Quando l'attività esatta dovrebbe essere OSENTBACK, la rete neurale predice solo attività ad essa parallele. Dalla rete di Petri si osserva che OSENTBACK è posizionata su un ramo che può essere svolto in parallelo alla maggior parte delle altre attività. Per AACCEPTED, nella Tabella 7, si vede che l'errore è solo su attività successive. Osservando la rete di Petri corrispondente a quel determinato noise, si hanno solo sette attività successive. Ma se si va a vedere la Tabella 8 l'errore osservato è su attività parallele, ed esaminando la rete di Petri per quel determinato noise, si nota che l'attività AACCEPTED è in parallelo solo con altre due attività che sono: OSENT e WCompleterenAanvraag. Sebbene OSENT è in parallelo in entrambe le reti di Petri, andando a vedere anche la matrice di confusione (Figura 10), si può notare che al posto

dell'attività AACCEPTED viene sempre predetta l'attività WCompleterenAanvraag. Un discorso simile può essere fatto anche per AACTIVETED, dove per entrambe le tabelle l'errore è soprattutto o totalmente con attività parallele. Ma se per noise a 0 ha molte attività parallele, per la rete con noise a 20 ha solo 7 attività in parallello su 22.

	1°Predizione					
	ok	Par	suc	pre	Alt	End
APARTLYSUBMITTED	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
APREACCEPTED	98,46%	0,00%	1,54%	0,00%	0,00%	0,00%
AACCEPTED	0,00%	0,00%	100,00%	0,00%	0,00%	0,00%
OSELECTED	81,32%	3,08%	9,82%	0,00%	5,78%	0,00%
AFINALIZED	55,88%	42,79%	1,33%	0,00%	0,00%	0,00%
OCREATED	95,54%	0,38%	0,62%	0,43%	3,03%	0,00%
OSENT	98,10%	1,90%	0,00%	0,00%	0,00%	0,00%
WCompleterenAanvraag	97,90%	0,01%	0,11%	1,07%	0,08%	0,82%
WNabellenOffertes	99,36%	0,00%	0,00%	0,25%	0,38%	0,01%
OSENTBACK	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%
AREGISTERED	58,90%	40,65%	0,00%	0,00%	0,45%	0,00%
AAPPROVED	12,02%	87,69%	0,00%	0,00%	0,30%	0,00%
OACCEPTED	2,08%	97,62%	0,00%	0,00%	0,30%	0,00%
AACTIVATED	0,00%	99,41%	0,00%	0,00%	0,59%	0,00%
WValiderenAanvraag	54,16%	44,62%	0,00%	0,00%	1,22%	0,00%
END	84,49%	0,00%	0,00%	15,51%	0,00%	0,00%
OCANCELLED	34,73%	0,27%	48,22%	0,00%	16,50%	0,27%
ADECLINED	0,00%	0,00%	0,00%	74,51%	25,49%	0,00%
ACANCELLED	6,52%	0,00%	64,53%	23,01%	5,93%	0,00%
WAfhandelenLeads	11,13%	0,00%	43,22%	16,78%	0,00%	28,87%
ODECLINED	0,00%	80,91%	0,00%	0,00%	19,09%	0,00%
WNabellenIncompleteDossiers	95,15%	3,65%	0,00%	0,00%	1,20%	0,00%
WBeoordelenFraude	0,00%	0,00%	50,62%	0,00%	49,38%	0,00%

Tabella 11: noise 20

	1°Predizione					
	ok	par	suc	pre	alt	end
APARTLYSUBMITTED	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
APREACCEPTED	98,46%	0,00%	1,54%	0,00%	0,00%	0,00%
AACCEPTED	0,00%	99,87%	0,00%	0,00%	0,00%	0,13%
OSELECTED	80,84%	19,16%	0,00%	0,00%	0,00%	0,00%
AFINALIZED	55,81%	44,19%	0,00%	0,00%	0,00%	0,00%
OCREATED	95,45%	3,41%	1,14%	0,00%	0,00%	0,00%
OSENT	98,01%	1,99%	0,00%	0,00%	0,00%	0,00%
WCompleterenAanvraag	97,73%	1,28%	0,00%	0,00%	0,00%	0,99%
WNabellenOffertes	99,33%	0,59%	0,00%	0,00%	0,00%	0,07%
OSENTBACK	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%
AREGISTERED	59,94%	40,06%	0,00%	0,00%	0,00%	0,00%
AAPPROVED	10,83%	89,17%	0,00%	0,00%	0,00%	0,00%
OACCEPTED	1,93%	98,07%	0,00%	0,00%	0,00%	0,00%
AACTIVATED	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%
WValiderenAanvraag	54,33%	45,63%	0,00%	0,00%	0,00%	0,04%
END	84,64%	0,00%	0,00%	15,36%	0,00%	0,00%
OCANCELLED	34,55%	65,27%	0,00%	0,00%	0,00%	0,18%
ADECLINED	0,00%	25,67%	0,00%	74,25%	0,09%	0,00%
ACANCELLED	6,52%	93,24%	0,00%	0,00%	0,12%	0,12%
WAfhandelenLeads	11,13%	0,00%	43,22%	16,78%	0,00%	28,87%
ODECLINED	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%
WNabellenIncompleteDossiers	95,15%	4,85%	0,00%	0,00%	0,00%	0,00%
WBeoordelenFraude	0,00%	0,00%	50,62%	0,00%	49,38%	0,00%

Tabella 12

CONCLUSIONI

Nel presente elaborato sono stati riportati i risultati degli esperimenti condotti per verificare l'applicabilità delle Convolutional Graph Neural Network al task di next activity prediction. In particolare, si è utilizzato un tipo specifico di rete neurale: la rete DGCNN, Deep Graph Convolutional Neural Network. Dai dati raccolti a seguito degli esperimenti e dalle analisi eseguite sui problemi riscontrati si nota che il problema più rilevante all'interno della rete è quello degli errore di misclassificazione su attività che possono essere svolte in parallelo rispetto alla predizione esatta. Questi errori possono essere riconducibili al fatto che essendo presenti al'interno del dataset solo le informazione strutturali dei grafi e dei tipi di attività, la rete apprende solo i modelli di grafo più frequenti e non è in grado di classificare in maniera esatta quelli meno fraquenti non avendo ulteriori informazioni su cui basarsi. Altro caso che porta a questo tipo di errore su rami paralleli o alternativi è il fatto che si può avere più volte la stessa struttura di grafo all'interno del dataset ma poi cambi solo il tipo di attività da predire. Anche se con tutti i problemi riscontrati alla fine la rete DGCNN nella migliore configurazione trovata, ottiene un valore di accuracy di 82.32% per il dataset "HELPDESK" e di 72,08% per "BPI 2012 complite".

Possibili lavori futuri prevederebbero sicuramente di intervenire sui problemi generati dall'over-smoothing, che nel nostro caso di studio viene accentuato dalla dimensione dei grafi molto piccola. Una possibile strategia potrebbe includere la diminuzione del numero di layer di convoluzione all'interno della DGCNN. L'eliminazione dell'over-smoothing dovrebbe portare ad un miglioramento della capacità della rete nel distinguere l'informazione all'interno dei nodi e quindi migliorare le performance di classificazione della rete. Altri esperimenti da effettuare prevedono di aumentare la quantità di informazioni all'interno del dataset, per esempio aumentando la quantità di

informazione all'interno dei nodi; ovvero aggiungendo al tipo di attività che viene svolta anche altre informazioni per quella specifica attività come chi ha eseguito quell'attività, il tempo di inizio, il tempo di completamento, il tempo in cui quell'attività è stata in esecuzione ed anche altre informazioni relative all'intero processo. Questo aumento delle informazioni può aiutare la rete a distinguere meglio le attività che potrebbero essere in parallelo, dato che la scelta delle attività che possono essere svolte in parallelo può essere determinata da queste informazioni aggiutive. Altro test potrebbe essere quello di inserire le informazioni temporali all'interno degli archi, per effettuare ciò si dovrebbe cambiare tipo di graph neural network andando ad utilizzarne una in grado di gestire la presenza di informazioni all'interno degli archi. Queste reti possono portare miglioramenti visto che riescono a gestire in maniera migliore l'informazione temporale, dato che non c'è bisogno di inserire questa informazione nei nodi, in cui può diventare ridondante.

Biobiografia

[1] Sumit S. - “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way“ https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] Godec P. – “Graph Embeddings — The Summary” https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007

[3] Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Sun M. “Graph neural networks: a review of methods and applications.” 2018. arXiv preprint arXiv:1812.08434.

[4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks” arXiv preprint arXiv:1901.00596v4, 2019.

[5] Fixed point (mathematics) https://en.wikipedia.org/wiki/Fixed_point_(mathematics)

[6] Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). “An End-to-End Deep Learning Architecture for Graph Classification” Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/11782

[7] Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. Journal of Machine Learning Research 12(Sep):2539–2561.

[8] https://github.com/rusty1s/pytorch_geometric/blob/master/benchmark/kernel/sort_pool.py

[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE TNN 2009, vol. 20, no. 1, pp. 61–80, 2009

[11] https://en.wikipedia.org/wiki/Contraction_mapping

[12] https://en.wikipedia.org/wiki/Banach_fixed-point_theorem

[13] Claudia Diamantini, Laura Genga, Domenico Potena, Wil van der Aalst, Building instance graphs for highly variable processes, Expert Systems with Applications, Volume 59, 2016, Pages 101-118.

[14] W. L. Hamilton, R. Ying, and J. leskovec, “Representation learning on graphs: Methods and applications” IEEE Data(base) Engineering Bulletin, vol. 40, pp. 52-74, 2017.