UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

PREDICTIVE PROCESS MONITORING CON RETI NEURALI A GRAFO

Ottimizzazione e analisi delle performance attraverso combinazioni di iperparametri della rete

Autore

Sbattella Mattia

ANNO ACCADEMICO 2023-2024

Indice

Elenco delle figure

CAPITOLO 1

Introduzione

In questo capitolo introduttivo verrà presentato il contesto dell'elaborato, seguito da una breve illustrazione dei temi trattati e degli obiettivi finali che si cercherà di raggiungere.

1.1 Contesto

Il Predictive Process Monitoring (PPM) è un'area di ricerca emergente che si occupa di prevedere l'evoluzione di un processo durante la sua esecuzione. Analizzando i log di eventi generati dai processi aziendali, il PPM fornisce informazioni predittive utili per migliorare la gestione e l'efficienza delle operazioni. Ad esempio, le tecniche di PPM consentono di stimare il tempo residuo per il completamento di un processo o di valutare la probabilità che vengano violati determinati vincoli operativi.

Uno degli obiettivi principali del PPM è la previsione della prossima attività: determinare quale sarà la prossima azione in base allo stato corrente del processo. Questa capacità è fondamentale per ottimizzare la gestione delle risorse, prevenire inefficienze e individuare eventuali deviazioni dai comportamenti attesi. Tuttavia, i processi reali presentano spesso strutture complesse, caratterizzate da costrutti come concorrenza, cicli e scelte condizionali, che rendono difficile la costruzione di modelli predittivi efficaci.

Negli ultimi anni, tecniche di deep learning sono state proposte per affrontare queste sfide. Architetture come le reti LSTM (Long Short-Term Memory), inizialmente sviluppate per l'elaborazione del linguaggio naturale, sono state applicate per catturare la natura sequenziale dei log di eventi. Altri studi hanno esplorato l'uso delle reti neurali convoluzionali (CNN), proponendo rappresentazioni multidimensionali dei log. Nonostante questi progressi, molte soluzioni trascurano la struttura intrinseca del processo,

limitandosi a rappresentazioni sequenziali che non catturano pienamente relazioni parallele e caratteristiche temporali.

1.2 Temi trattati

Nei seguenti capitoli verranno illustrati e confrontati due modelli di reti neurali a grafo, la Deep Graph Convolutional Neural Network (DGCNN) e la Message Passing Graph Neural Network (MPGNN). Saranno approfonditi gli aspetti teorici alla base della loro costruzione, con un focus particolare sulle loro architetture e sulle modalità con cui ciascuna di esse affronta la sfida del Predictive Process Monitoring (PPM). In particolare, verranno descritti i meccanismi di convoluzione grafica utilizzati per estrarre e sfruttare le relazioni strutturali tra le attività di un processo, nonché l'integrazione dei dati multidimensionali, come quelli temporali, nelle previsioni.

Successivamente, si passerà alla descrizione dell'implementazione della MPGNN in codice Python. Verrà presentata sommariamente la struttura del codice consegnato, con enfasi sui problemi riscontrati per il porting e il funzionamento e sulla descrizione degli iperparametri che consentono di trovare le combinazioni migliori e superare i concorrenti. Gli obiettivi del lavoro, descritti nel capitolo 1.3, sono focalizzati sul miglioramento delle prestazioni predittive rispetto ai modelli esistenti.

1.3 Obiettivi

Questo studio si pone i seguenti obiettivi:

- Superare le prestazioni delle DGCNN utilizzando la rete MPGNN, integrando ulteriori ottimizzazioni e affinamenti nei modelli.
- Analizzare i risultati ottenuti in modo dettagliato, considerando metriche specifiche a livello di prefisso e confrontando le prestazioni in base alla lunghezza delle tracce.
- Sperimentare configurazioni ottimali per la MPGNN, utilizzando una grid search per identificare le migliori combinazioni di iperparametri.

CAPITOLO 2

Metodologia

Questo capitolo presenta la metodologia adottata per il processo di next activity prediction, descrivendo i diversi algoritmi e gli strumenti impiegati.

2.1 Descrizione del processo di classificazione

Il processo di classificazione della **next activity prediction** (figura 2.1) inizia con la costruzione e utilizzo dei grafi di istanza attraverso l'input costituito da un log di eventi e un modello di processo. A partire da queste informazioni, entra in gioco l'algoritmo BIG (Building Instance Graph), che permette di creare grafi coerenti anche in presenza di tracce non conformi. Questo approccio include la capacità di riparare i grafi qualora si incontrino anomalie, come attività mancanti o fuori ordine, garantendo così un'analisi robusta e accurata dei dati.

Successivamente, avviene un arricchimento multi-prospettico dei grafi. In questa fase, si generano prefissi di grafo, che consistono nell'estrazione progressiva di sottografi etichettati con l'attività successiva. A ciò si aggiungono prospettive che ampliano la descrizione dei grafi: da un lato, tramite l'integrazione di attributi diretti, come i dati degli eventi stessi; dall'altro, con attributi indiretti, quali intervalli temporali o la posizione relativa nel ciclo settimanale, fornendo così una visione più completa e contestualizzata.

Infine, la predizione viene effettuata utilizzando due reti neurali convoluzionali su grafi con in output una funzione di softmax:

- Deep Graph Convolutional Neural Network (DGCNN)
- Message Passing Graph Neural Network (MPGNN)

Per riassumere:

1. **Costruzione dei grafi di istanza:**
    - Input: log di eventi e modello di processo.
    - Uso dell'algoritmo BIG (Building Instance Graph):
        - IG: creazione di grafi coerenti anche per tracce non conformi.
        - IG-Repair: riparazione dei grafi in presenza di anomalie (es. attività mancanti o fuori ordine).
2. **Arricchimento multi-prospettico dei grafi (Data encoding):**
    - Generazione dei prefissi di grafo:
        - Estrazione di sottografi progressivi etichettati con l'attività successiva.
    - Aggiunta di prospettive:
        - Attributi diretti (dati provenienti dai file dei Log).
        - Attributi indiretti (es. intervalli temporali, posizione nel ciclo settimanale).
3. **Addestramento del classificatore basato su reti neurali convoluzionali su grafi:**
    - DGCNN
        - Layer convoluzionali su grafi: aggregazione di informazioni locali per ogni nodo.
        - SortPooling Layer: riduzione della dimensionalità selezionando i nodi più rilevanti.
        - Layer convoluzionali 1-D e fully connected: predizione finale tramite funzione softmax.
    - MPGNN
        - EdgeConv Layer: calcola le features dei nodi considerando le relazioni con i loro vicini
        - Global Max Pooling e layer fully connected: predizione finale tramite funzione softmax.
4. **Predizione**

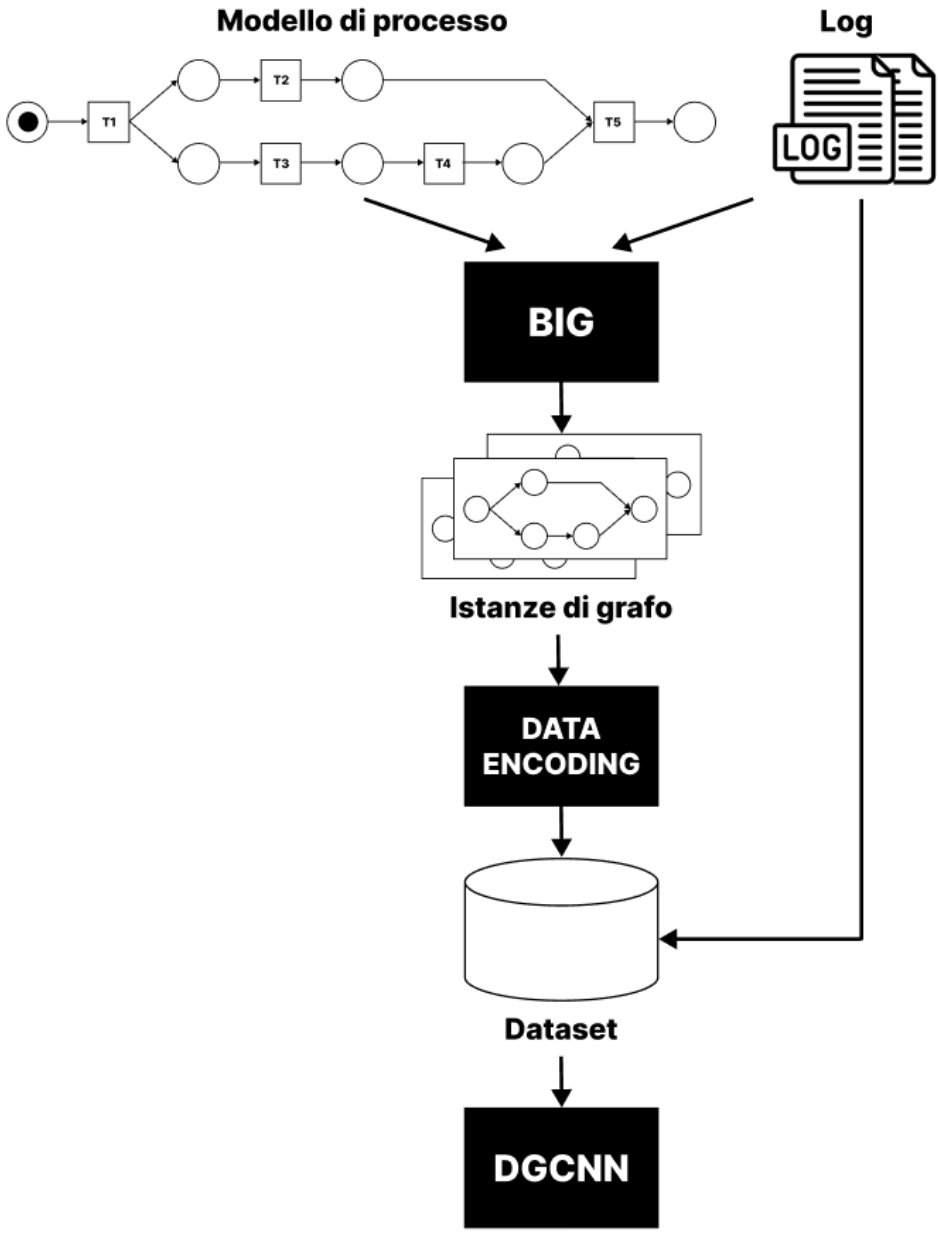

Figura 2.1: Processo di next activity prediction.

2.2 Building Instance Graph (BIG)

A partire dal modello di processo, rappresentato come rete di Petri e dai file di log, è possibile ricostruire i grafi di istanza. L'algoritmo utilizzato è il BIG (Building Instance Graph) illustrato nell'articolo di Diamantini C. [2016]. Questi grafi rappresentano istanze specifiche del processo, fondamentali nelle attività di next activity prediction.

La costruzione dei grafi è divisa in due parti.

2.2.1 IG

In primo luogo, viene costruito un Instance Graph (IG) per ciascuna traccia, utilizzando l'insieme delle relazioni causali estratte dal modello di processo fornito in input.

2.2.2 IG-Repair

Successivamente, viene applicata la procedura di riparazione degli IG. Questa procedura ha lo scopo di trasformare gli IG di tracce anomale in grafi che siano in grado di rappresentare correttamente le anomalie senza però generalizzare eccessivamente. La generalizzazione eccessiva può portare a una rappresentazione dei comportamenti del processo troppo ampia, includendo potenzialmente sequenze di eventi che non sono realmente presenti nei log o nel modello originale. Da un lato, vogliamo che il grafo riparato sia abbastanza specifico da rappresentare fedelmente l'anomalia osservata, evitando l'introduzione di comportamenti non realistici che non corrispondono ad alcuna traccia reale. Dall'altro lato, è necessario preservare le relazioni di concorrenza per mantenere la struttura originale del processo. Le relazioni di concorrenza rappresentano comportamenti paralleli che, se trascurati o alterati durante la riparazione, potrebbero portare a distorsioni nel modello e compromettere la capacità del sistema predittivo di comprendere le dipendenze temporali e strutturali tra le attività, danneggiando la precisione della predizione, poiché il modello predittivo verrebbe addestrato su dati non accurati.

2.3 Data Encoding

In questa sezione, viene descritto il processo di generazione e arricchimento del dataset di prefix-IGs, organizzato in due fasi principali: la generazione dei prefix-IG e il loro arricchimento con prospettive multi-dimensionali derivate dai dati disponibili nel log degli eventi.

2.3.1 Prefix-IG Generation

Dato un insieme di n Instance Graphs (IG), l'obiettivo è costruire il dataset $S = \{(p_i(g_j), a_l)\}$, dove $p_i(g_j) = (E_p, W_p)$ rappresenta un prefix-IG di lunghezza i di un gra-

fo $g_j = (E_j, W_j)$, dove $i \in [1, |E_j| - 1]$, e a_l è l'attività successiva alla parziale esecuzione descritta da $p_i(g_j)$. Da ogni IG si producono $N - 1$ coppie per il dataset S. La costruzione del set di prefix-IG avviene seguendo l'ordine totale degli eventi nella traccia. Poiché ogni nodo in un IG corrisponde a un evento della traccia, ogni nodo può essere associato a un indice progressivo che rappresenta la posizione dell'evento nella traccia. Questo indice determina l'ordine dei nodi, utilizzato per costruire progressivamente i prefix-IG.

Si consideri un processo aziendale composto dalle attività A, B, C e D, registrate in una traccia organizzata come segue:

$$\sigma = [A \to B \to C \to D]$$

Un Instance Graph (IG) rappresenta questa traccia come un grafo in cui:

- i **nodi** corrispondono alle attività eseguite (A, B, C, D);
- gli **archi** rappresentano le relazioni di dipendenza causale tra le attività.

Il grafo IG per questa traccia è definito come segue:

- **Nodi**: $\{(1 : A), (2 : B), (3 : C), (4 : D)\}$;
- **Archi**: $\{(A \to B), (B \to C), (C \to D)\}$.

Per analizzare i prefissi, si costruiscono i *prefix-IG*[1] come segue:

1. Il *prefix-IG* $p_2(g)$ include i primi due nodi e l'arco tra essi:
   - **Nodi**: $\{(1 : A), (2 : B)\}$;
   - **Archi**: $\{(A \to B)\}$;
   - **Etichetta**: C (l'attività successiva nella traccia).
2. Estendendo $p_2(g)$ con il nodo 3 e l'arco associato, si ottiene il *prefix-IG* $p_3(g)$:
   - **Nodi**: $\{(1 : A), (2 : B), (3 : C)\}$;
   - **Archi**: $\{(A \to B), (B \to C)\}$;
   - **Etichetta**: D (l'attività successiva nella traccia).

Questo processo continua fino a quando l'ultima attività viene selezionata come etichetta finale.

[1] Sono stati omessi i passaggi rappresentanti i prefissi in posizione $p_0(g)$ e $p_1(g)$unicamente a scopo illustrativo.

Tabella 2.1: Esempio di prefix-IG di lunghezza 2 e 3 estratti da un IG.

	Nodi	*Archi*	Label(next activity)
$p_2(g)$	$\{(1:A),(2:B)\}$	$\{(A \rightarrow B)\}$	C
$p_3(g)$	$\{(1:A),(2:B),(3:C)\}$	$\{(A \rightarrow B),(B \rightarrow C)\}$	D

2.3.2 Multi-Perspective Prefix-IG Enrichment

I prefix-IG generati nella fase precedente rappresentano solo il nome dell'attività e le relazioni causali corrispondenti per ciascun evento di una traccia. In questa fase vengono incorporate prospettive aggiuntive derivanti da attributi diretti e indiretti presenti nel log degli eventi.

Siano M l'insieme delle prospettive (feature) scelte per la predizione, G l'insieme dei valori delle feature, e $Val_M : M \rightarrow 2^G$ la funzione che definisce i valori ammissibili per ciascuna feature. L'obiettivo è costruire il dataset $S^* = \{(p_i^*(g_j), a_l)\}$, dove $p_i^*(g_j) = (E_p, W_p, Val_M)$ rappresenta un prefix-IG arricchito multi-prospettiva di lunghezza i.

Le feature considerate includono:

- **Feature dirette**: attributi memorizzati direttamente nel log degli eventi, come ad esempio il timestamp.
- **Feature indirette**: informazioni temporali derivate dall'ordine sequenziale degli eventi nella traccia. In particolare:
  - Δt_{n_i}: il tempo tra un evento corrente e il suo predecessore causale.
  - $t_d(n_i)$: il tempo trascorso dall'inizio del processo.
  - $t_w(n_i)$: il tempo dell'evento rispetto all'inizio della settimana lavorativa (da mezzanotte della domenica precedente).

Le funzioni temporali sono normalizzate per migliorare le prestazioni della rete, e calcolate come:

$$t_d(n_i) = \frac{time(n_i) - t_0}{\Delta t_{max}}$$
$$t_w(n_i) = \frac{time(n_i) - t_{w_0}}{\Delta t_w}$$

dove t_0 è il timestamp iniziale del processo, t_{w_0} è il timestamp della mezzanotte della domenica precedente, Δt_{max} è la durata massima della traccia, e Δt_w è la durata massima di una settimana lavorativa. Riprendendo l'esempio precedente:

Tabella 2.2: Esempio di prefix-IG enrichment di lunghezza 2 e 3 estratti da un IG.

	Nodi	*Archi*	Feature indirette	Label
$p_2(g)$	$\{(1:A),(2:B)\}$	$\{(A \rightarrow B)\}$	{(1,{Δt_{n_i}=n, $t_d(n_i)$=n, $t_w(n_i)$=n)}) (2,{Δt_{n_i}=n, $t_d(n_i)$=n, $t_w(n_i)$=n)})}	C

2.4 Deep Graph Convolutional Neural Network (DGCNN)

La Deep Graph Convolutional Neural Network è un modello che analizza grafi per compiti come classificazione e predizione, utilizzando convoluzioni per aggregare informazioni tra nodi connessi. Integra un livello di SortPooling per selezionare i nodi più importanti, riducendo la dimensionalità mantenendo le caratteristiche chiave. Dopo l'elaborazione dei grafi, utilizza strati convoluzionali 1-D e completamente connessi per produrre la predizione finale.

2.4.1 Layer di convoluzione su grafi

La convoluzione su grafi viene definita come:

$$Z = f\left(\tilde{D}^{-1}\tilde{A}XW\right) \tag{2.4.1}$$

dove:

- $\tilde{A} = A + I$: matrice di adiacenza con self-loops. Se un nodo non si collegasse a sé stesso, l'aggregazione considererebbe solo i nodi vicini, perdendo le informazioni uniche del nodo corrente (le sue caratteristiche). Collegandosi a sé stesso, il nodo riesce a combinare le proprie informazioni con quelle dei vicini, rendendo l'embedding più rappresentativo.
- $\tilde{D}$: matrice diagonale dei gradi, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.
- $X \in \mathbb{R}^{n \times c}$: matrice delle informazioni dei nodi.
- $W \in \mathbb{R}^{c \times c'}$: matrice dei pesi addestrabili.
- f: funzione di attivazione non lineare.

La convoluzione multi-strato è definita come:

$$Z^{(k+1)} = f\left(\tilde{D}^{-1}\tilde{A}Z^{(k)}W^{(k)}\right),\ Z^{(0)} = X \tag{2.4.2}$$

con:

- $Z^{(k)}$ è la rappresentazione dei nodi al layer k;
- $Z^{(0)}$: le feature iniziali dei nodi.

Come "aggregatore" dei vari stati convoluzionari è stato utilizzato il GraphSAGE descritto nella sezione 2.7.1.

2.4.2 SortPooling Layer

Il SortPooling Layer ordina i nodi in base all'output dell'ultimo strato convoluzionale $Z^{(h)}$ e seleziona i primi m nodi:

$$Z_{sort} \in \mathbb{R}^{m \times \sum_{k=1}^{h} c_k} \tag{2.4.3}$$

dove m è il numero di nodi selezionati.

Si applicano 4 passaggi fondamentali:

1. Strati convoluzionali e canali: se si hanno n strati convoluzionali, ogni strato genera delle caratteristiche (features) che corrispondono a un canale. Per ogni nodo, esistono delle caratteristiche che vengono estratte da ogni layer della rete.
2. Creazione delle liste: per ogni nodo, viene creata una lista, in cui:
   - Il primo elemento è la norma (un valore che rappresenta la forza o l'importanza) delle caratteristiche estratte nel primo layer.
   - L'ennesimo elemento è la norma delle caratteristiche estratte nell'ultimo layer.
3. Selezione dei nodi: l'obiettivo è selezionare gli m nodi più importanti. Per farlo, si confrontano tutte le liste per ogni nodo. I nodi con il valore maggiore (ovvero, con la norma più alta) vengono selezionati come i nodi più rilevanti.
4. Risoluzione dei conflitti: nel caso due nodi avessero lo stesso valore nel primo livello (due nodi hanno la stessa norma delle caratteristiche nel primo layer), si guarda il secondo layer per decidere quale nodo scegliere. Se ancora ci sono conflitti, si scende ai layer successivi finché non si risolve il conflitto.

In sostanza, la tecnica seleziona i nodi più importanti in base alla loro "norma" in ogni layer, utilizzando una scansione che risolve i conflitti man mano che si scende nei layer della rete.

2.4.3 Layer convoluzionale 1-D

L'articolo analizzato in origine proponeva un layer convoluzionale seguito da più MaxPooling layer (semplificazione in figura 2.2), ma l'architettura è stata semplificata lasciando un unico layer convoluzionale 1-D che prende in ingresso Z_{sort}, output del passaggio 2.4.2 precedente.

$$\text{Conv1D}(Z_{sort}) = \text{ReLU}\left(W_{\text{conv}} * Z_{sort} + b\right) \tag{2.4.4}$$

dove:

- Z_{sort} è la rappresentazione in input,

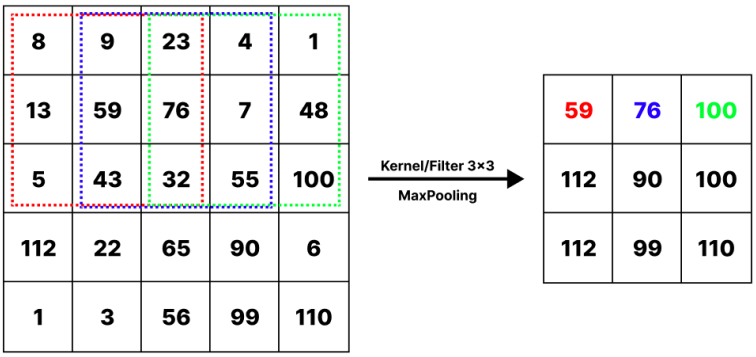

Figura 2.2: MaxPooling con kernel(filtro) 3x3.

- W_{conv} rappresenta i pesi del filtro convoluzionale,
- $*$ denota l'operazione di convoluzione,
- $\text{ReLU}(x) = \max(0, x)$ è la funzione di attivazione ReLU. Una ReLU (Rectified Linear Unit) non è nient'altro che un rettificatore che trasforma un input numerico annullando i valori negativi e lasciando invariati quelli positivi.
- $b(bias)$: rappresenta un valore che viene aggiunto al risultato della convoluzione per introdurre flessibilità nel modello. Il bias aiuta la rete a spostare la funzione di attivazione, migliorando la capacità del modello di adattarsi ai dati.

2.4.4 Dense Layer con Softmax

L'output del layer convoluzionale viene successivamente "appiattito" tramite un'operazione di *flatten* e passato a uno layer denso (dove ogni nodo è connesso a tutti gli altri nodi del grafo), che produce la predizione finale tramite la funzione softmax (processo completo in figura 2.3):

$$\hat{y} = \text{softmax}\left(W_{\text{dense}} \cdot \text{flatten}(\text{Conv1D}(Z_{\text{sort}})) + b_{\text{dense}}\right), \tag{2.4.5}$$

dove:

- $\text{Conv1D}(Z_{\text{sort}})$ è l'output dello strato convoluzionale.
- flatten la funzione flatten prende la matrice risultante dalla convoluzione e la trasforma in un vettore unidimensionale. Questo perché la rete densa (fully connected) lavora con vettori, non matrici.
- W_{dense} sono i pesi dello strato denso. In uno strato denso, ogni nodo è connesso a tutti i nodi del layer precedente, e W_{dense} è la matrice che contiene questi pesi.
- b_{dense} è un bias associato allo strato denso, simile al bias nel layer convoluzionale.

- softmax descritta nella sezione 2.7.2.

La predizione finale $\hat{y}$ rappresenta un vettore di probabilità, in cui ciascun elemento indica la probabilità assegnata alla rispettiva classe, ovvero la **predizione del modello**.

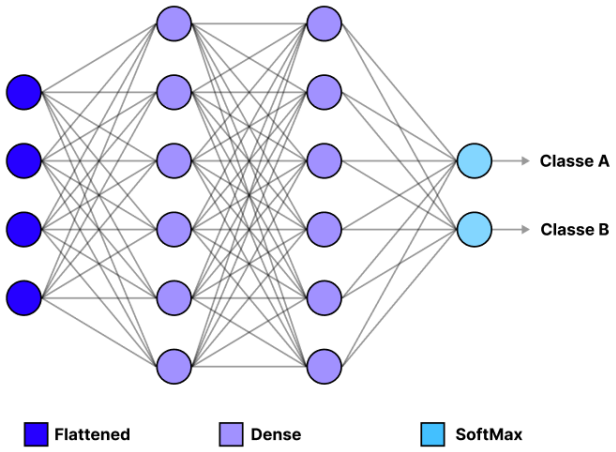

Figura 2.3: Flatten layer, Dense layer, SoftMax layer

2.5 Message Passing Convolutional Neural Network

La *Message Passing Convolutional Neural Network* (MPCNN) è un'architettura progettata per analizzare grafi, con particolare applicazione alle nuvole di punti 3D. La sua principale innovazione risiede nella costruzione dinamica del grafo a ogni livello della rete, consentendo di apprendere relazioni locali e globali adattive.

2.5.1 EdgeConv layer

Il layer *EdgeConv* rappresenta il nucleo della MPCNN. Esso calcola le features dei nodi considerando le relazioni con i loro vicini, definite attraverso il grafo locale. L'operazione EdgeConv è definita come:

$$x_i^{(k+1)} = \max_{j \in \mathcal{N}(i)} \text{ReLU}\left(\Theta \cdot (x_j - x_i) + \Phi \cdot x_i\right), \tag{2.5.1}$$

dove:

- $\mathbf{x_i^{(k+1)}}$: rappresenta la nuova caratterizzazione del nodo i al layer successivo $k+1$. Questo embedding è una combinazione delle informazioni relative ai vicini di i (x_j) e delle sue caratteristiche originali (x_i). L'aggiornamento iterativo consente al modello di arricchire la rappresentazione del nodo con contesto locale, strato dopo strato.

- $\mathcal{N}(i)$: è l'insieme dei nodi vicini a i nel grafo locale. Questo insieme definisce il contesto locale del nodo e rappresenta l'area di "influenza" delle sue connessioni.
- $x_j - x_i$: è la differenza tra le rappresentazioni del nodo vicino j e del nodo centrale i. Questa differenza cattura informazioni locali relative, come variazioni di caratteristiche o connessioni specifiche nel grafo.
- Θ e Φ: sono parametri apprendibili della rete.
  - Θ: pesa le informazioni relative tra i nodi j e i, enfatizzando o riducendo l'importanza delle connessioni.
  - Φ: pesa le informazioni intrinseche del nodo i, permettendo al modello di bilanciare tra il contributo locale e quello globale.
- **Funzione di attivazione ReLU**: introduce non linearità, aiutando il modello a catturare relazioni complesse tra i nodi.
- **Operazione di aggregazione Max**: seleziona il valore massimo tra tutti i nodi vicini per ogni caratteristica del nodo centrale. Questo garantisce che l'operazione sia *invariante rispetto alla permutazione* dei nodi, una proprietà cruciale per l'analisi di grafi dove l'ordine dei nodi non è significativo.

Dynamic graph update

A differenza delle architetture statiche, il grafo nella MPCNN è dinamico e viene aggiornato a ogni livello della rete. Questo processo consente di adattare continuamente la struttura del grafo, migliorando la capacità del modello di rappresentare relazioni complesse tra i dati.

Ad ogni layer, i nodi possiedono una rappresentazione $x_i^{(l)}$ che descrive le loro caratteristiche attuali. Questo spazio delle caratteristiche evolve progressivamente man mano che il modello apprende, riflettendo informazioni più ricche e contestualizzate. Il grafo $G^{(l)}$ viene ricostruito individuando i k-vicini più prossimi (k-nearest neighbors) di ciascun nodo nel nuovo spazio delle caratteristiche $x^{(l)}$. In questo modo, le connessioni tra i nodi cambiano dinamicamente, adattandosi per rappresentare le relazioni più rilevanti in base alle rappresentazioni apprese fino a quel punto. Una volta aggiornato, il grafo consente di combinare informazioni tra i nodi vicini (nel grafo aggiornato) utilizzando l'operazione EdgeConv o altre tecniche di aggregazione, favorendo la propagazione delle informazioni anche su distanze semantiche significative.

Un esempio intuitivo

Analizzando una rete sociale:

- **Grafo statico:** le connessioni tra le persone sono basate sulla vicinanza geografica iniziale. Tuttavia, questo non riflette necessariamente le somiglianze nelle loro opinioni o interessi.

- **Grafo dinamico:** dopo aver analizzato alcune caratteristiche (es. preferenze musicali), il modello aggiorna il grafo collegando persone con gusti simili, anche se non vivono vicine. Con ogni aggiornamento, il modello raffina queste connessioni per catturare meglio i legami semantici.

2.5.2 Dinamic max pooling e dense layer

L'aggregazione delle caratteristiche avviene attraverso il *Global Max Pooling*[2], che raccoglie le informazioni da tutti i nodi per ottenere una rappresentazione globale del grafo. Successivamente, questa rappresentazione è passata a uno strato completamente connesso per produrre la predizione finale attraverso una funzione softmax:

$$\hat{y} = \text{softmax}\left(W_{\text{dense}} \cdot Z_{\text{global}} + b_{\text{dense}}\right), \tag{2.5.2}$$

dove:

- Z_{global}: output del max pooling globale;
- W_{dense}: pesi dello strato denso;
- b_{dense}: bias associato allo strato denso.

La predizione finale $\hat{y}$ rappresenta un vettore di probabilità, in cui ciascun elemento indica la probabilità assegnata alla rispettiva classe, ovvero la **predizione del modello**.

2.6 Confronto tra DGCNN e MPCNN

I due approci differiscono dal modello descritto nella sezione originale per vari aspetti, come sintetizzato in Tabella 2.3.

Tabella 2.3: Confronto tra DGCNN e MPCNN

Componente	**DGCNN**	**MPCNN**
Convoluzione su grafi	Matrice $\tilde{A}$ statica	Grafo dinamico
Pooling	SortPooling per nodi rilevanti	Max pooling globale
Output	Conv1D →Flatten→ Dense	Max pooling → Dense

- Dinamismo del grafo: la principale innovazione della MPCNN è l'aggiornamento dinamico del grafo, che consente di adattarsi meglio alla struttura dei dati, migliorando la capacità di catturare relazioni semantiche locali e globali. Nella DGCNN, il grafo è statico, sebbene sia arricchito da self-loops.

[2]Differisce da quanto proposto in figura 2.2, in quanto il semplice Max Pooling applica un kernel e fornisce in output multipli valori, al contrario del Global che invece è possibile vedere come un semplice Max Pooling, ma con un filtro della stessa dimensione dell'input, quindi con un solo valore in uscita.

- Pooling: la MPCNN usa un global pooling semplice, mentre la DGCNN ordina e seleziona nodi specifici con il SortPooling.
- Complessità computazionale: la MPCNN può essere più costosa computazionalmente a causa dell'aggiornamento dinamico, mentre la DGCNN risulta essere più leggera.

2.7 Concetti ausiliari

In questa sezione vengono trattati argomenti ausiliari, ma essenziali per la comprensione del processo di predizione delle attività.

2.7.1 GraphSAGE: Graph Sample and Aggregation

GraphSAGE (Graph Sample and Aggregation) è un algoritmo di machine learning introdotto per affrontare il problema della scalabilità nell'apprendimento su grafi di grandi dimensioni, come reti sociali, reti biologiche e sistemi infrastrutturali. William L. Hamilton e Leskovec [2017]

Obiettivo

L'obiettivo di GraphSAGE è quello di generare rappresentazioni vettoriali (embedding) per i nodi di un grafo in modo *generalizzabile*. Ciò significa che il modello non necessita di essere riaddestrato ogni volta che vengono introdotti nuovi nodi nel grafo. Questo approccio è particolarmente utile per applicazioni dinamiche in cui il grafo evolve continuamente.

Caratteristiche principali

Scalabilità GraphSAGE non richiede di elaborare l'intero grafo durante l'addestramento, ma campiona una porzione fissa dei vicini per ogni nodo. Ciò consente al modello di scalare efficientemente su grafi di grandi dimensioni.

Generalizzazione Grazie al suo approccio *induttivo*, GraphSAGE è in grado di produrre embedding anche per nodi mai visti in precedenza durante la fase di addestramento. Questo lo rende ideale per grafi in evoluzione.

Aggregazione dei vicini L'algoritmo apprende funzioni parametrizzate di aggregazione per combinare le informazioni provenienti dai nodi vicini. Tali funzioni includono:

- **Mean aggregator**: aggregazione tramite media dei vettori dei nodi vicini.
- **Pooling**: aggregazione tramite max pooling dopo una trasformazione non lineare.

- **LSTM aggregator**: utilizza una rete neurale ricorrente (LSTM) per analizzare i nodi vicini in un ordine specifico. Questo permette di catturare relazioni più complesse tra i vicini rispetto a una semplice media o pooling. Anche se i vicini non hanno un ordine naturale, l'LSTM riesce comunque a combinare le informazioni in modo efficace.

Metodologia

Il processo di generazione degli embedding tramite GraphSAGE può essere sintetizzato nei seguenti passaggi:

1. **Campionamento**: come illustrato in figura 2.4,per ogni nodo target si campionano un numero fisso di vicini a profondità K (distanza K-hop). Ad esempio, con $K = 2$, vengono inclusi i vicini diretti e i loro vicini, con K molto grande si include l'intero grafo nel nodo.

2. **Aggregazione**: Si aggregano le rappresentazioni dei vicini utilizzando una funzione di aggregazione AGG, come la media, il pooling o un modello LSTM:

$$h_v^{(k)} = \text{AGG}\left(\{h_u^{(k-1)}, \forall u \in N(v)\}\right), \tag{2.7.1}$$

   dove $h_v^{(k)}$ è la rappresentazione del nodo v al livello k e $N(v)$ denota l'insieme dei vicini di v. Notazioni riprese da Marinelli [2024].

3. **Aggiornamento**: L'embedding del nodo target viene aggiornato combinando l'aggregato dei vicini h_N con la rappresentazione corrente del nodo:

$$h_v^{(k)} = f\left(W \cdot \text{CONCAT}(h_v^{(k-1)}, h_N^{(k)})\right), \tag{2.7.2}$$

   dove W è una matrice di peso appresa e f è una funzione di attivazione non lineare.

4. **Addestramento**: Il modello viene ottimizzato con una funzione di perdita appropriata per il task finale, come la classificazione dei nodi o la predizione di collegamenti tra nodi.

2.7.2 Funzione Softmax

Nella sezione 2.4.4, al layer denso viene applicata una funzione **softmax**. La funzione softmax prende in input un vettore $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ di K valori reali e restituisce un vettore $\sigma(\mathbf{z})$ di probabilità, in cui ogni elemento è compreso tra 0 e 1 e la somma totale è uguale a 1:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{per ogni } j = 1, \ldots, K.$$

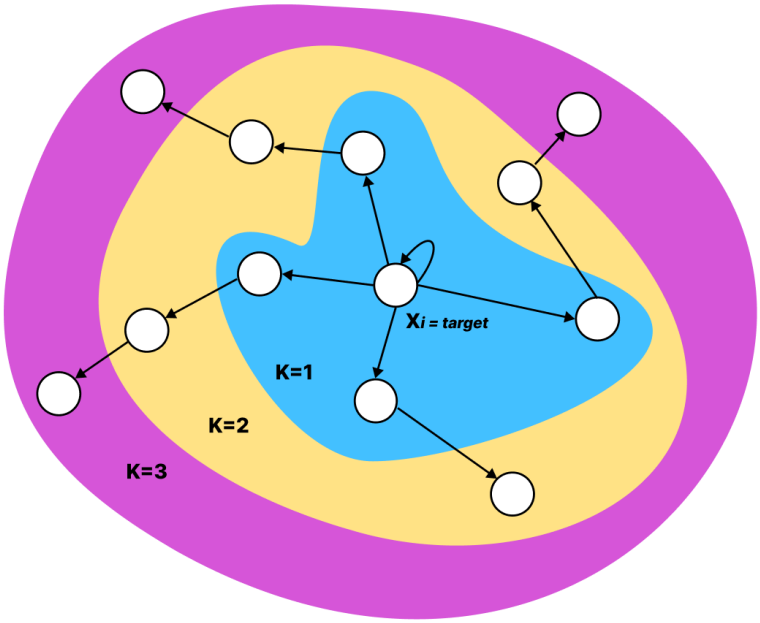

Figura 2.4: Campionamento GraphSAGE.

Ad esempio, dato il vettore di input:

$$\mathbf{z} = [2.0, 1.0, 0.1].$$

Calcolando gli esponenziali di ciascun valore:

$$e^{z_1} = e^{2.0} = 7.389, \quad e^{z_2} = e^{1.0} = 2.718, \quad e^{z_3} = e^{0.1} \approx 1.105.$$

E dividendo successivamente ciascun valore esponenziale per la somma totale dei valori appena calcolati si ottiene:

$$\mathbf{z} = [0.659, 0.242, 0.099].$$

Ovvero il vettore di probabilità di ogni elemento.

CAPITOLO 3

Codice e parametri

In questo capitolo verrà proposto sommariamente il codice, proponendo cambiamenti e migliorie non tanto sul modello, quanto sulla logica delle prove. Successivamente ci si concentrerà sugli iperparametri giustificandone la scelta e mostrando i possibili valori. Verranno poi presentati i dataset sui quali verranno effettuate le prove e gli strumenti utilizzati.

3.1 Codice

Il codice presentato in questa sezione non è stato sviluppato direttamente, ma unicamente utilizzato a scopo di approfondimento. Per questo verrà presentato sommariamente. L'albero delle cartelle è il seguente:

```
+---next_activity_prediction
    |-   completad_combination.txt
    |-   config.py
    |-   MPGNN.py
    |-   requirements.txt
    |-   results_evaluation.py
    |-   train_MPGNN.py
```

Il progetto è organizzato in diversi file principali:

- `train_MPGNN.py` : punto di ingresso principale del programma, dove viene avviato il processo di addestramento.
- `MPGNN.py` : contiene la configurazione della rete neurale MPGNN.
- `config.py` : definisce gli iperparametri del modello e crea le varie combinazioni di parametri da testare durante l'allenamento.

- `results_evaluation.py` : calcola le metriche di valutazione come precision, F1-score e genera matrici di confusione per valutare le prestazioni del modello.
- `requirements.txt` : specifica le librerie necessarie per eseguire il progetto, in particolare:
  - `matplotlib==3.7.2` : libreria per la creazione di grafici e visualizzazioni.
  - `networkx==3.2.1` : libreria per la creazione, la manipolazione e lo studio di strutture complesse come grafi e reti.
  - `numpy==1.25.2` : libreria fondamentale per il calcolo scientifico con Python, supporta array multidimensionali e operazioni matematiche veloci.
  - `pandas==2.0.3` : libreria per la manipolazione e l'analisi dei dati, fornendo strutture dati come DataFrame.
  - `scikit-learn==1.3.0` : libreria per l'apprendimento automatico, offre algoritmi per classificazione, regressione e clustering, oltre a metodi di preprocessing e valutazione.
  - `torch-geometric==2.3.1` : estensione di PyTorch per il calcolo su grafi e reti neurali su grafi.
  - `torch==2.3.1` : libreria principale per il calcolo numerico e il deep learning.
  - `pm4py==2.2.16` : libreria per il Process Mining, utile per l'analisi dei processi aziendali a partire dai log degli eventi.
- `completad_combination.txt` : specifica le combinazioni già effettuate, in modo da tener traccia di quelle già effettuate, lasciando però la libertà di scegliere se replicarle o meno.

3.2 Ottimizzazione e debugging

3.2.1 Utilizzo della GPU

Con l'installazione delle librerie standard, il sistema utilizza di default la CPU per l'elaborazione. Per sfruttare la GPU e accelerare i calcoli, è stato necessario installare una versione di `torch` con supporto per CUDA[1]. In particolare, la libreria `torch==2.3.1+cu118` è stata installata, poiché offre il supporto per l'elaborazione su GPU con la versione CUDA 11.8. Questo passaggio ha richiesto l'installazione di `Python 3.10` , poiché la compatibilità di `torch==2.3.1+cu118` con versioni precedenti di Python non era garantita.

[1]CUDA è una piattaforma di calcolo parallelo e un'API di programmazione sviluppata da NVIDIA che consente di utilizzare le GPU per eseguire calcoli generici, accelerando notevolmente i tempi di elaborazione, soprattutto in operazioni complesse come quelle nel deep learning.

3.2.2 Parallelizzazione e utilizzo di workers

Nella fase di caricamento dei dati per l'addestramento e il test, inizialmente erano stati configurati 4 `workers` per la parallelizzazione del caricamento dei dati:

```
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          num_workers=4)
```

Tuttavia, è stato deciso di ridurre il numero di workers a 0, eseguendo tutto in un unico thread, come mostrato qui:

```
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          num_workers=0)
```

(analogamente per il test loader).

Contrariamente a quanto si poteva pensare, la riduzione del numero di workers ha aumentato notevolmente i tempi di elaborazione. Questo probabilmente è dovuto a una gestione inefficiente della parallelizzazione tra la GPU e la CPU da parte della libreria `DataLoader`.

3.2.3 Gestione dei path e salvataggio dei file

Nel salvataggio dei risultati si presentava il seguente problema:

```
FileNotFoundError: [Errno 2] No such file or directory
```

Il seguente codice di errore è molto strano, poiché la cartella veniva creata correttamente, ma il file csv non riusciva ad essere salvato. Sono stati testati vari aspetti, come la validità del nome del file, l'esistenza del percorso, i permessi sulle cartelle, i metodi di salvataggio in CSV con `Pandas` e la creazione di `DataFrame` artificiali per testare il processo di salvataggio.

Dopo una ricerca, si è scoperto che Windows impone un limite alla lunghezza dei path, definito come `MAX_PATH`, che è pari a 260 caratteri. Poiché il percorso superava i 280 caratteri, questo eccedeva il limite e causava l'errore durante il salvataggio.

> *In the Windows API (with some exceptions discussed in the following paragraphs), the maximum length for a path is MAX_PATH, which is defined as 260 characters. A local path is structured in the following order: drive letter, colon, backslash, name components separated by backslashes, and a terminating null character. For example, the maximum path on drive D is "D:\some 256-character path string<NUL>" where "<NUL>" represents the invisible terminating null character for the current system codepage. (The characters < > are used here for visual clarity and cannot be part of a valid path string.)*

Per risolvere questo problema, ho modificato il codice per ridurre la lunghezza del nome del file, utilizzando un hash del nome del dataset per generare un nome breve, come segue (figura 3.1):

```
total_path=join(comb_path, f'results_{hashlib.md5(comb_string.encode()).hexdigest()[:8]}.csv')
results_df.to_csv(total_path, header=True, sep=',', index=False)
```

Figura 3.1: Riduzione con hash.

In questo modo, il nome del file è stato ridotto utilizzando solo i primi 8 caratteri dell'hash MD5, evitando di superare il limite di lunghezza del path.

3.2.4 Formato della Data

Nel tentativo di risolvere eventuali problemi legati al formato della data nel nome del file, è stato modificato la stringa di formato della data, come segue, da:

```
run = datetime.now().strftime("%Y-%m-%d %H:%M:%S").
                    replace(' ', '_').replace(':', '_')}
```

A:

```
run = datetime.now().strftime("%Y-%m-%d_%H_%M_%S")
```

Questo cambiamento ha eliminato i caratteri speciali come i due punti (":") e gli spazi, sostituendoli con caratteri di sottolineatura ("_"), che sono più compatibili con i nomi di file in vari sistemi operativi.

3.2.5 Gestione del device (CPU/GPU)

Nonostante l'impostazione della GPU come dispositivo principale, alcuni dati rimanevano sulla CPU, il che causava conflitti nell'elaborazione. Per risolvere il problema, ogni batch di dati veniva esplicitamente trasferito alla GPU per l'elaborazione, utilizzando il seguente codice:

```
batch = batch.to(device)
```

Dove `device` è impostato su `'cuda'` per utilizzare la GPU o `'cpu'` se non disponibile.

3.2.6 Arresto forzato dell'epoca

Poiché alcune combinazioni di iperparametri sono intrensicamente meno promettenti di altre, è stata implementata una logica per arrestare immediatamente l'epoca (oltre il classico early stop basato sull'assenza di miglioramento). Se le metriche di valutazione, come l'accuratezza (`accuracy`) o il punteggio F1, non raggiungono una soglia minima, l'allenamento viene interrotto anticipatamente per evitare di sprecare tempo computazionale in epoche non produttive.

```
#soppressione se iperparametri non promettenti
if test_metrics.get('acc') < THRESHOLD or test_metrics.get('f1') < THRESHOLD: break
```

Figura 3.2: Threeshold per la soppressione dell'epoca.

3.2.7 Generazione della matrice di confusione nel dataset BPI12

Nel file `result_evaluation.py`, è stato corretto un problema relativo alla generazione della matrice di confusione. Inizialmente, il codice per la creazione della matrice di confusione era il seguente:

```
cm = confusion_matrix(true_labels, predicted_labels)
```

Questo approccio non funzionava correttamente poiché `sklearn` non accetta parametri numerici per la creazione della matrice di confusione. Le etichette nel secondo dataset erano di tipo intero, il che causava un errore. La soluzione è stata quella di convertire le etichette in formato stringa prima di passare alla funzione `confusion_matrix`, come segue:

```
cm = confusion_matrix(true_labels.astype(str),
                      predicted_labels.astype(str))
```

In questo modo, la matrice di confusione è stata generata correttamente.

3.2.8 Ripetizione di combinazioni di parametri già eseguite

Nel codice è stata implementa una logica per garantire il controllo della ripetizione di combinazioni di dataset e parametri, al fine di evitare che vengano ricalcolate o rieseguite elaborazioni già completate. Il controllo delle combinazioni già elaborate riduce il carico computazionale eliminando elaborazioni ridondanti.

```
#Aggiunta del file con le combinazioni effettuate
completed_file = "completed_combinations.txt"
skipped_combinations = 0
already_in=False
if not os.path.exists(completed_file):
    with open(completed_file, 'w') as f:
        pass
else:
    with open(completed_file, 'r') as f:
        completed_combinations = set(line.strip() for line in f.readlines())
```

Figura 3.3: Creazione (se non esiste) e lettura file.

```
#Se la combinazione è stata effettuata salta alla prossima
if (ds_name + "_" + comb_string) in completed_combinations:
    already_in = True
    if NO_REPEAT:
        print(f"Skipping combination {ds_name}_{comb_string} (already completed)")
        skipped_combinations += 1
        actual_comb += 1
        continue
```

Figura 3.4: Skip della combinazione se già elaborata.

```
# Registra la combinazione completata
# NOTA: indipendentemente dal fatto che NO_REPEAT sia attivo
#       o meno la salva, a meno che non sia già presente
if not already_in:
    with open(completed_file, 'a') as f:
        f.write(f"{ds_name}_{comb_string}\n")
```

Figura 3.5: Salvataggio combinazione nel file.

- **Variabile di controllo `already_in` :** La variabile viene inizializzata come `False` prima dell'inizio delle operazioni di verifica. Il suo scopo è monitorare se la combinazione corrente è già presente nel file. Se il valore risulta `True` , questa non verrà salvata di nuovo. La variabile è stata pensata per evitare duplicati.
- **Blocco condizionale basato su `NO_REPEAT` :** La logica del controllo della ripetizione è attivata esclusivamente quando il flag `NO_REPEAT` è impostato su `True` . Ciò consente al sistema di gestire dinamicamente l'esecuzione, con la possibilità di abilitare o disabilitare il controllo a seconda delle necessità.
- **Verifica della combinazione:** La combinazione corrente è identificata concatenando il nome del dataset (`ds_name`) con una rappresentazione univoca dei parametri (`comb_string`).
- **Messaggio diagnostico:** Se la combinazione è già stata elaborata, viene stampato un messaggio informativo. Questo messaggio include il nome della combinazione ed esplicita che essa sarà ignorata, migliorando la tracciabilità del processo.
- **Incremento del contatore `actual_comb` :** Indipendentemente dal fatto che la combinazione venga processata o ignorata, la variabile `actual_comb` è incrementata.

- **Scrittura nel file delle combinazioni completate:** Se la combinazione non è già presente indipentemente se il controllo `NO_REPEAT` è abilitato o meno, essa viene aggiunta al file `completed_file` tramite un'operazione di append. Il file funge da registro permanente delle combinazioni processate, garantendo la persistenza delle informazioni tra diverse sessioni o esecuzioni del codice.

3.3 Descrizione iperparametri e scelta

3.3.1 Training Parameters

- **SEED**:
  - **Descrizione**: valore intero utilizzato per inizializzare i generatori di numeri casuali di PyTorch e di Python.
  - **Scopo**: garantire la riproducibilità del modello.
  - **Criterio di scelta**: utilizza un numero fisso (ad esempio, 42, standard comune) per assicurare coerenza tra esecuzioni.
- **TRAIN_SPLIT**:
  - **Descrizione**: percentuale del dataset utilizzata per il training rispetto al totale.
  - **Scopo**: suddivide il dataset in training e test. Ad esempio, con 0.67, il 67% dei dati sarà usato per il training.
  - **Criterio di scelta**: Dipende dalla dimensione del dataset. Comune è la scelta 70%-30%.
- **PATIENCE**:
  - **Descrizione**: numero massimo di epoche consecutive senza miglioramenti (early stopping).
  - **Scopo**: evitare di continuare il training quando non ci sono miglioramenti significativi.
  - **Criterio di scelta**: è bene usare valori più bassi per modelli più piccoli e più alti per modelli complessi. La velocità di convergenza del modello influenza anche la scelta. Un learning rate più basso porta a convergenza più rapida e quindi ad un early stop.
- **THRESHOLD**:
  - **Descrizione**: soglia minima per metriche come accuratezza o F1 score, al di sotto della quale la combinazione di iperparametri viene interrotta.
  - **Scopo**: evitare di continuare il training di modelli con prestazioni insufficienti.

  - **Criterio di scelta**: impostato ad 1 per richiedere una percentuale minima. Possibile modificare con valori più alti. Valori troppo elevati potrebbero far abortire combinazioni valide.

- **NO_REPEAT**:
  - **Descrizione**: booleano che indica se evitare di ripetere combinazioni già eseguite.
  - **Scopo**: ottimizzare il processo evitando lavoro ridondante (presupponendo il seed rimanga lo stesso).
  - **Criterio di scelta**: *True* o *False* in base alla scelta dell'utente, se vuole ripetere o meno gli esperimenti. Molto utile se si vogliono provare nuove combinazioni, ma tra le presenti ce ne sono alcune già effettuate.

- **EPOCH**:
  - **Descrizione**: numero massimo di epoche di training.
  - **Scopo**: determina quante iterazioni completerà il modello sul dataset.
  - **Criterio di scelta**: valori più alti sono preferibili per dataset complessi. Settata a 200.

- **BATCH_SIZE**:
  - **Descrizione**: il batch size rappresenta il numero di campioni del dataset elaborati contemporaneamente durante un singolo passaggio del modello. Invece di aggiornare i pesi del modello dopo ogni singolo campione utilizza gruppi (batch) di campioni per bilanciare efficienza e accuratezza dell'aggiornamento.
  - **Scopo**:
    1. Migliora la stabilità del processo di training, riducendo le oscillazioni che si verificano con il *stochastic gradient descent*.
    2. Incrementa l'efficienza computazionale sfruttando l'ottimizzazione hardware (GPU), che consente di elaborare dati in parallelo.
    3. Riduce i tempi di calcolo rispetto all'elaborazione di tutto il dataset in una singola iterazione.
  - **Criterio di scelta**:
    1. **Dimensione della memoria hardware**:
       * Il batch size deve essere sufficientemente piccolo da poter essere caricato interamente nella memoria della GPU.
       * Una memoria limitata può richiedere batch size più piccoli, come 32 o 64.
    2. **Dimensione del dataset**:

    * Per dataset di grandi dimensioni, valori più elevati (256 o 1024) possono accelerare il training.
    * Per dataset piccoli, batch size troppo elevati potrebbero non essere vantaggiosi.
  3. **Bilanciamento tra efficienza e accuratezza**:
    * Valori piccoli (ad esempio, 16 o 32) consentono aggiornamenti più frequenti dei pesi, ma sono computazionalmente meno efficienti.
    * Valori grandi (512, 1024 o superiori) riducono la frequenza degli aggiornamenti, ma migliorano la stabilità dell'apprendimento.
- **Scelta**: si inizia con un valore moderato per il batch size, come 64 o 128, e aumentato progressivamente fino a raggiungere il limite della memoria disponibile. Durante il processo, sono state monitorate le prestazioni del modello per valutare l'impatto delle dimensioni del batch. Non riscontrando cambiamenti significativi era stato scelto quello che offre il miglior compromesso tra velocità di training e accuratezza finale (512 o 1024). Come ultimo cambiamento si è deciso di reimpostare il batch size a 64, avendo dataset di dimensione ridotta e per motivi che verranno spiegati nel capitolo successivo

- **LEARNING_RATE**:
  - **Descrizione**: il *learning rate* (η) rappresenta il passo con cui i pesi del modello vengono aggiornati, determinando la velocità con cui l'algoritmo si avvicina al minimo della funzione di perdita.
  - **Scopo:** il learning rate è essenziale per garantire un addestramento stabile ed efficace. Un valore troppo elevato può causare instabilità nell'addestramento, portando il modello a oscillare intorno al minimo senza convergere. Al contrario, un valore troppo basso può rallentare significativamente il processo di ottimizzazione, richiedendo un numero elevato di epoche per ottenere buone prestazioni e aumentando il rischio di rimanere bloccati in minimi locali.
  - **Criterio di scelta:** la scelta del learning rate è spesso il risultato di un compromesso tra velocità di addestramento e stabilità. Valori comunemente utilizzati, per avere una base di riferimento, sono: 10^{-2}, 10^{-3}, 10^{-4} .
  - **Scelta:** è stato scelto di impostare il learning rate su 10^{-3} e aggiustarlo in base ai risultati ottenuti. È stato dunque diminuito gradualmente impostandolo a 10^{-4} per infine arrivare a 10^{-5}.

- **DROPOUT**:
  - **Descrizione**: il dropout è una tecnica di regolarizzazione usata nelle reti neurali per prevenire l'overfitting. Consiste nel disattivare casualmente una parte dei neuroni (insieme alle loro connessioni) durante l'addestramento della rete. Ad ogni passo di addestramento, una percentuale dei neuroni è "spenta", cioè non contribuisce né alla propagazione. Tuttavia, durante la fase di test, tutti i neuroni sono attivi. In pratica:

1. Durante l'addestramento, viene applicata una maschera casuale che azzera l'uscita di alcuni neuroni.
2. Durante l'inferenza, le uscite dei neuroni vengono scalate (normalmente moltiplicate per $1 - p$, dove p è la probabilità di dropout) per riflettere il comportamento medio osservato durante il training.

– **Scopo:** si utilizza il dropout principalmente per:

1. **Riduzione dell'overfitting:** costringendo la rete a non dipendere troppo da specifici neuroni, il dropout favorisce l'apprendimento di rappresentazioni più robuste.
2. **Miglioramento della generalizzazione:** durante il training, la rete impara a distribuire le informazioni su diverse combinazioni di neuroni, migliorando la sua capacità di generalizzare a nuovi dati.
3. **Aumento della robustezza:** poiché il dropout simula come se fossero tanti modelli più piccoli (ogni "sotto-rete" è una possibile configurazione con dropout applicato), il modello finale è più robusto.

– **Criterio di scelta**: la probabilità di dropout (p) indica la percentuale di neuroni che verranno disattivati in ogni passaggio. Valori tipici di p solitamente sono $p = 0.1$ o $p = 0.2$ per i layer di input. Valori superiori ($p > 0.5$) di solito compromettono la capacità di apprendimento della rete. Reti molto grandi o dense beneficiano spesso di dropout più elevato rispetto a reti piccole.

– **Scelta:** la scelta del valore p dipende dal dataset e dalla complessità della rete. In alcuni casi, come con dataset piccoli il dropout può non essere necessario o addirittura dannoso. infatti, il dropout potrebbe introdurre rumore, portando a rallentare la convergenza. Per questo è stato scelto di utilizzare un dropout di $p = 0.1$ o $p = 0.2$.

3.3.2 Model Parameters

- **GRAPH_CONV_LAYERS**:

  – **Descrizione**: numero di layer convoluzionali utilizzati nel modello di apprendimento basato su grafi. Ogni layer convoluzionale consente al modello di aggregare e trasformare informazioni dai nodi vicini nel grafo.

  – **Scopo**: aumentare la capacità del modello di apprendere rappresentazioni più complesse ed espressive dai dati di input.

  – **Criterio di scelta**: il valore dipende dalla complessità del grafo e dal problema specifico. Risulta buona norma utilizzare valori piccoli come tre o cinque layer. Nelle prove effettuate è stato scelto di analizzare questi valori, aumentando gradualmente per osservare i miglioramenti delle prestazioni, fino ad arrivare a 7 layer. Un numero eccessivo di layer potrebbe portare a problemi di overfitting. Nelle prove svolte si è scelto di analizzare i valori di 2, 5 e 7 layer.

- **NUM_NEURONS**:
  - **Descrizione**: numero di neuroni per layer convoluzionale. Determina la capacità di ogni layer di rappresentare caratteristiche complesse dei dati.
  - **Scopo**: controllare il grado di complessità del modello e la sua capacità di apprendere dai dati.
  - **Criterio di scelta**: valori più alti aumentano la capacità del modello ma richiedono più risorse computazionali e possono causare overfitting. Si è deciso di iniziare con valori moderati quali 32 (testato in precedenza) o 64, arrivando a 128.
- **K_VALUE**:
  - **Descrizione**: numero di nodi selezionati dal meccanismo di sort pooling. Questo parametro influenza il numero di nodi mantenuti dopo l'operazione di pooling.
  - **Scopo**: ridurre la dimensione del grafo preservando i nodi più rilevanti per il compito di apprendimento.
  - **Criterio di scelta**: il valore dipende dalla struttura del grafo e dall'importanza relativa dei nodi. È necessario bilanciare la riduzione della dimensionalità con il mantenimento di informazioni critiche. Ad esempio, valori bassi come 3 e 5 potrebbero funzionare bene per grafi semplici, mentre valori più alti come 7, 10 o 30 sono più adatti a grafi complessi. Nelle prove si è scelto di effettuare esperimenti con i valori 3, 7 e 10.

CAPITOLO 4

Risultati ed Analisi

Preambolo

4.1 Descrizione degli esperimenti

4.1.1 Dataset utilizzati

Per gli esperimenti, sono stati selezionati due dataset ampiamente utilizzati nella letteratura di Process Mining: **Helpdesk** e **BPI12**.

Helpdesk (HD)

Il dataset **Helpdesk** (Verenich [2016]) contiene tracce relative a un processo di gestione ticket del help desk di un'azienda italiana di software. Questo dataset include 3804 tracce, per un totale di 13710 eventi, distribuiti su 9 tipi di attività. La lunghezza delle tracce varia da 1 a 14 eventi, con una media di 3 eventi per traccia.

BPIChallenge2012 (BPI12)

Il dataset **BPI12** (Van Dongen [2012]) riguarda un processo di gestione delle richieste di prestito personale in una grande organizzazione finanziaria globale. Il log di eventi è il risultato della fusione di tre sottoprocessi paralleli. Per gli esperimenti, è stato utilizzato il dataset completo BPI12, che comprende 13087 tracce e 262200 eventi, distribuiti su 23 tipi di attività. La lunghezza delle tracce varia da 3 a 175 eventi, con una media di 38 eventi per traccia.

4.1.2 Preprocessing

Per entrambi i dataset, sono stati mantenuti solo gli eventi completati. È stato applicato uno split 67%-33% per la divisione dei dati in train e test, preservando l'ordine cronologico delle tracce.

4.2 Strumenti

Gli esperimenti realizzati nei successivi capitoli sono stati realizzati con una macchina avente sistema operativo Windows11, processore Intel® Core™ i7 i7-13620H, RAM 16 GB DDR5, GPU NVIDIA GeForce RTX 4050.

4.3 Metriche di valutazione

Per confrontare i risultati ottenuti dai classificatori testati, sono state utilizzate metriche ampiamente diffuse nei task di classificazione, ovvero *Accuracy* e *F1-score*, insieme a metriche basate su ranking.

4.3.1 Accuracy

L'accuracy misura la proporzione di campioni classificati correttamente sul totale dei campioni:

$$\text{Accuracy} = \frac{T}{N}$$

dove T rappresenta il numero totale di campioni correttamente classificati e N il numero totale di campioni.

4.3.2 F1-Score

L'F1-score complessivo è calcolato come media ponderata degli F1-score delle singole classi, tenendo conto del numero di campioni per ciascuna classe. Per una classe i, l'F1-score è definito come la media armonica tra precision (P_i) e recall (R_i):

$$F1_i = 2 \cdot \frac{P_i \cdot R_i}{P_i + R_i}$$

dove:

$$P_i = \frac{TP_i}{TP_i + FP_i}$$

è la precision, che rappresenta la proporzione di campioni della classe i correttamente classificati (TP_i) rispetto al totale delle predizioni della classe i, incluse quelle errate (FP_i).

$$R_i = \frac{TP_i}{TP_i + FN_i}$$

è il recall, che misura la proporzione di campioni della classe i correttamente classificati (TP_i) rispetto al totale dei campioni effettivi della classe i, inclusi quelli erroneamente assegnati ad altre classi (FN_i).

4.4 Valutazione degli iperparametri

Nel contesto degli esperimenti svolti, la scelta dei parametri di training e configurazione del modello è stata effettuata con attenzione, basandosi su considerazioni teoriche ed evidenze empiriche, al fine di ottimizzare le prestazioni del modello e adattarlo alle peculiarità del dataset, non senza difficoltà.

Partendo dal **dropout**, sono stati scelti i valori di 0.1 e 0.2. Il valore più basso, 0.1, è ideale per contesti in cui il dataset non presenta una complessità elevata, permettendo al modello di mantenere una buona capacità di apprendimento senza perdere troppe informazioni. Al contrario, 0.2 introduce una regolarizzazione più marcata. Il valore di 0.3 è stato scartato dopo poche prove soprattutto per il dataset HD più piccolo, in quanto risultato controproducente per l'apprendimento.

Un altro aspetto cruciale riguarda il **batch size**, fissato a 64 dopo diverse sperimentazioni. Inizialmente si era considerato di utilizzare batch size più elevati (sono state effettuate numerose prove a riguardo), come 512 e 1024. Questi valori, seppur avendo prodotto valori buoni in minor tempo, sono stati scartati per una motivazione puramente concettuale. L'idea di utilizzare batch size moderati deriva dalla necessità di evitare che la rete processi troppi dati contemporaneamente, un approccio che può rendere l'apprendimento meno efficace, seppur apparentemente non in questo caso. Poiché il nostro dataset, specialmente HD, non è particolarmente grande, valori come 32 o 64 sono generalmente considerati più adeguati. Questo approccio permette al modello di apprendere in modo più graduale e dettagliato, a scapito di una lieve riduzione nella velocità complessiva dell'addestramento.

Per quanto riguarda il **learning rate**, si è optato per i valori di 1e-4 e 1e-3, evitando 1e-5 che, sebbene prudente, si è dimostrato troppo lento, non permettendo una convergenza efficace entro un numero ragionevole di epoche, raggiungendo più volte la soglia delle 200, rendendo gli esperimenti di fatto irrealizzabili. Il valore di 1e-4 rappresenta un compromesso ideale per garantire stabilità e precisione nell'apprendimento, soprattutto in contesti in cui il modello può mostrare oscillazioni. D'altra parte, 1e-3 accelera il processo di apprendimento, risultando utile nei casi in cui il modello non presenti difficoltà significative nella convergenza.

Sul fronte della struttura del modello, sono stati selezionati 2, 5 e 7 come valori per il

numero di **strati convoluzionali** su grafi. Cinque strati si sono rivelati sufficienti per catturare le relazioni locali nei dati, evitando di appesantire eccessivamente la rete. Tuttavia, per analisi più complesse, sette strati consentono di esplorare relazioni più profonde, a costo di un aumento della complessità computazionale. Al contrario, per il dataset meno complesso è stata fatta la scelta di aggiungere anche 2 come valore, coprendo interamente le casistiche.

Il **numero di neuroni** per strato, fissato a 64 e 128. La configurazione con 64 neuroni si adatta bene a dataset di dimensioni modeste, mantenendo un rischio minimo di overfitting. La configurazione con 128 neuroni, invece, offre una maggiore capacità di rappresentazione seppur rappresenti una sorta di azzardo per dataset moderati come quelli proposti.

Infine, è stato scelto di impostare i valori di **K** pari a 3, 7 e 10, che rappresentano il numero di nodi vicini considerati nelle operazioni di pooling e aggregazione sui grafi. K=7 è più conservativo, focalizzandosi su relazioni locali, mentre K=10 espande il raggio d'azione, rendendo il modello più adatto a contesti con strutture di dati più intricate, ma anche qui meno efficenti per dataset piccoli. Per cui è stato aggiunto anche k=3.

4.5 Analisi delle metriche e delle combinazioni

Tra le molte combinazioni di iperparametri analizzate vengono riportate le più rilevanti.

4.5.1 Selezione delle top combinazioni (HD)

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 10_k	**89.39**	80.82	88.06	79.38	0.0046	0.0141
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 7_k	**89.22**	80.02	87.89	78.47	0.0047	0.0137
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 5_k	**89.20**	81.17	87.80	79.76	0.0047	0.0134
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 10_k	**89.17**	80.22	87.71	78.78	0.0047	0.0143
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	**89.13**	80.41	87.76	78.52	0.0047	0.0150

Tabella 4.1: Tabella con top 5 combinazioni di Train Acc per HD.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 10_k	86.98	**82.83**	84.02	80.31	0.0062	0.0117
200_epochs 0.001_lr 64_bs 0.1_dropout 8_conv_layers 128_num_neurons 10_k	85.46	**82.68**	81.47	77.73	0.0069	0.0121
200_epochs 0.001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	86.87	**82.66**	83.97	77.86	0.0062	0.0125
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 5_k	86.34	**82.66**	83.38	77.86	0.0066	0.0117
200_epochs 0.001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 7_k	86.56	**82.66**	83.57	77.89	0.0064	0.0117

Tabella 4.2: Tabella con top 5 combinazioni di Test Accuracy per HD.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 10_k	89.39	80.82	**88.06**	79.38	0.0046	0.0141
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 7_k	89.22	80.02	**87.89**	78.47	0.0047	0.0137
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 5_k	89.20	81.17	**87.80**	79.76	0.0047	0.0134
200_epochs 0.0001_lr 64_bs 0.1_dropout 8_conv_layers 128_num_neurons 10_k	89.08	79.99	**87.77**	78.58	0.0046	0.0148
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	89.13	80.41	**87.76**	78.52	0.0047	0.0150

Tabella 4.3: Tabella con top 5 combinazioni di Train F1 per HD.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 7_k	86.78	82.52	83.95	**80.82**	0.0063	0.0101
200_epochs 0.001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	87.16	82.53	84.41	**80.44**	0.0061	0.0129
200_epochs 0.0001_lr 64_bs 0.1_dropout 8_conv_layers 128_num_neurons 7_k	86.82	82.17	83.81	**80.40**	0.0064	0.0103
200_epochs 0.0001_lr 64_bs 0.2_dropout 8_conv_layers 128_num_neurons 10_k	87.63	82.35	85.12	**80.40**	0.0058	0.0107
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 10_k	86.98	82.83	84.02	**80.31**	0.0062	0.0117

Tabella 4.4: Tabella con top 5 combinazioni di Test F1 per HD.

[1]

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.01_lr 1024_bs 0.2_dropout 5_conv_layers 256_num_neurons 7_k	85.66	81.91	82.43	77.14	**0.000479**	0.000772
200_epochs 0.01_lr 1024_bs 0.2_dropout 5_conv_layers 256_num_neurons 10_k	81.25	81.79	78.36	77.22	**0.000587**	0.000841
200_epochs 0.0001_lr 512_bs 0.3_dropout 7_conv_layers 128_num_neurons 7_k	88.35	80.89	86.27	77.20	**0.000707**	0.001743
200_epochs 0.0001_lr 512_bs 0.2_dropout 8_conv_layers 128_num_neurons 5_k	88.02	79.60	85.70	76.40	**0.000732**	0.001506
200_epochs 0.0001_lr 512_bs 0.3_dropout 8_conv_layers 128_num_neurons 5_k	87.99	79.73	85.84	77.40	**0.000736**	0.001502

Tabella 4.5: Tabella con top 5 combinazioni di Train Loss per HD.

[1]Solo in questo caso sono state lasciate combinazioni di iperparametri fatte prima delle considerazioni della sezione 4.4. Dunque sono riportati iperparametri come 1024 di bach size, prima che questo valore venisse scartato per le motivazioni sopra riportate.

4.5.2 Selezione delle top combinazioni (BPI12)

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 10_k	**91.66**	73.68	91.47	74.03	0.0033	0.0171
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 10_k	**91.18**	71.95	90.95	72.98	0.0037	0.0193
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	**91.08**	74.04	90.88	73.35	0.0036	0.0147
200_epochs 0.0001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 10_k	**90.94**	75.24	90.69	74.72	0.0037	0.0154
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 7_k	**90.90**	76.10	90.65	75.05	0.0038	0.0181

Tabella 4.6: Tabella con top 5 combinazioni di Train Acc per BPI12.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.001_lr 64_bs 0.2_dropout 5_conv_layers 64_num_neurons 10_k	84.26	**81.24**	82.79	78.13	0.0069	0.0091
200_epochs 0.001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	84.23	**80.72**	82.61	77.54	0.0069	0.0091
200_epochs 0.001_lr 64_bs 0.2_dropout 7_conv_layers 64_num_neurons 7_k	84.54	**80.16**	83.07	76.76	0.0070	0.0095
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 64_num_neurons 10_k	84.43	**80.14**	82.95	78.35	0.0072	0.0093
200_epochs 0.001_lr 64_bs 0.2_dropout 5_conv_layers 64_num_neurons 7_k	82.52	**80.02**	80.56	76.35	0.0079	0.0097

Tabella 4.7: Tabella con top 5 combinazioni di Test Acc per BPI12.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 10_k	91.66	73.68	**91.47**	74.03	0.0033	0.0171
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 10_k	91.18	71.95	**90.95**	72.98	0.0037	0.0193
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	91.08	74.04	**90.88**	73.35	0.0036	0.0147
200_epochs 0.0001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 10_k	90.94	75.24	**90.69**	74.72	0.0037	0.0154
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 7_k	90.90	76.10	**90.65**	75.05	0.0038	0.0181

Tabella 4.8: Tabella con top 5 combinazioni di Train F1 per BPI12.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 64_num_neurons 7_k	86.37	79.88	85.68	**78.56**	0.0062	0.0097
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 64_num_neurons 10_k	84.43	80.14	82.95	**78.35**	0.0072	0.0093
200_epochs 0.001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	82.52	80.17	81.04	**78.18**	0.0077	0.0092
200_epochs 0.001_lr 64_bs 0.2_dropout 5_conv_layers 64_num_neurons 10_k	84.26	81.24	82.79	**78.13**	0.0069	0.0091
200_epochs 0.0001_lr 64_bs 0.2_dropout 5_conv_layers 64_num_neurons 10_k	82.92	79.74	80.96	**77.64**	0.0079	0.0097

Tabella 4.9: Tabella con top 5 combinazioni di Test F1 per BPI12.

Combination	Train Acc	Test Acc	Train F1	Test F1	Train Loss	Test Loss
200_epochs 0.0001_lr 64_bs 0.1_dropout 7_conv_layers 128_num_neurons 10_k	91.66	73.68	91.47	74.03	**0.003267**	0.017072
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 7_k	91.08	74.04	90.88	73.35	**0.003593**	0.014697
200_epochs 0.0001_lr 64_bs 0.2_dropout 7_conv_layers 128_num_neurons 10_k	91.18	71.95	90.95	72.98	**0.003655**	0.019307
200_epochs 0.0001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 10_k	90.94	75.24	90.69	74.72	**0.003703**	0.015428
200_epochs 0.001_lr 64_bs 0.1_dropout 5_conv_layers 128_num_neurons 7_k	90.85	77.41	90.58	75.59	**0.003764**	0.017809

Tabella 4.10: Tabella con top 5 combinazioni di Train Loss per BPI12

4.5.3 Analisi degli iperparametri di Test più efficaci (HD)

Di seguito si analizzeranno gli iperparametri di test che hanno portato ai valori migliori di accuracy ed f1 per dataset HD.

Migliore combinazione per Test Accuracy

- **Configurazione:** 200 epochs, 0.001 lr, 64 batch size, 0.1 dropout, 5 conv layers, 128 neurons, 10k.
- **Metriche:**
  - Train Accuracy: 86.98%
  - Test Accuracy: **82.83**% **(Migliore valore)**
  - Train F1-score: 84.02%
  - Test F1-score: 80.31%
  - Train Loss: 0.0062
  - Test Loss: 0.0117

Migliore combinazione per Test F1-score

- **Configurazione:** 200 epochs, 0.0001 lr, 64 batch size, 0.2 dropout, 8 conv layers, 128 neurons, 7k.
- **Metriche:**
  - Train Accuracy: 86.78%
  - Test Accuracy: 82.52%
  - Train F1-score: 83.95%
  - Test F1-score: **80.82**% **(Migliore valore)**
  - Train Loss: 0.0063
  - Test Loss: 0.0101

Interpretazione dei risultati

- **Effetto del Learning Rate (*lr*)** Un learning rate più basso (0.0001) ha favorito un miglioramento del **Test F1-score**, suggerendo un apprendimento più graduale e una maggiore capacità di generalizzazione. D'altra parte, un valore più alto (0.001) ha favorito una **migliore Test Accuracy**, permettendo al modello di apprendere più rapidamente pattern rilevanti.
- **Effetto del Numero di Convolutional Layers** Il valore 5 ha ottenuto la **migliore Test Accuracy** (82.83%), indicando che troppi strati convoluzionali potrebbero portare a overfitting. Al contrario, 8 strati hanno favorito il **miglior Test F1-score** (80.82%), migliorando la capacità del modello di catturare dettagli complessi.
- **Effetto del Dropout** Un dropout di 0.2 ha contribuito al **miglior Test F1-score**, riducendo il rischio di overfitting. Un dropout di 0.1, invece, ha permesso di ottenere la **migliore Test Accuracy**, mantenendo più informazioni durante l'addestramento.

- **Effetto del Numero di Campioni (*k*)** - 10*k* campioni ha favorito la migliore **Test Accuracy**. - 7*k* campioni ha portato alla migliore **Test F1-score**, suggerendo che una riduzione dei dati di training può migliorare la generalizzazione su classi meno rappresentate.

4.5.4 Analisi degli iperparametri di Test più efficaci (BPI12)

Di seguito si analizzeranno gli iperparametri di test che hanno portato ai valori migliori di accuracy ed f1 per dataset BPI12.

Migliore combinazione per Test Accuracy

- **Configurazione:** 200 epochs, 0.001 lr, 64 batch size, 0.2 dropout, 5 conv layers, 64 neurons, 10*k*.
- **Metriche:**
  - Train Accuracy: 84.26%
  - Test Accuracy: **81.24**% **(Migliore valore)**
  - Train F1-score: 82.79%
  - Test F1-score: 78.13%
  - Train Loss: 0.0069
  - Test Loss: 0.0091

Migliore combinazione per Test F1-score

- **Configurazione:** 200 epochs, 0.0001 lr, 64 batch size, 0.2 dropout, 7 conv layers, 64 neurons, 7*k*.
- **Metriche:**
  - Train Accuracy: 86.37%
  - Test Accuracy: 79.88%
  - Train F1-score: 85.68%
  - Test F1-score: **78.56**% **(Migliore valore)**
  - Train Loss: 0.0062
  - Test Loss: 0.0097

Interpretazione dei risultati

- **Effetto del Learning Rate (*lr*)** Un learning rate più basso (0.0001) ha favorito un miglioramento del **Test F1-score**, suggerendo che un apprendimento più lento può aiutare il modello a generalizzare meglio, evitando sovradattamenti locali. Un valore più alto (0.001) ha invece favorito una **migliore Test Accuracy**, suggerendo una convergenza più rapida e un apprendimento più deciso dei pattern principali.
- **Effetto del Numero di Convolutional Layers** Il valore 5 ha ottenuto la **migliore Test Accuracy** (81.24%), indicando che una rete più semplice può favorire una migliore generalizzazione. Al contrario, 7 strati hanno favorito il **miglior Test F1-score** (78.56%), suggerendo che una maggiore profondità può migliorare la capacità del modello di distinguere le classi più difficili.
- **Effetto del Dropout** Un dropout di 0.2 è presente in entrambe le migliori configurazioni, confermando il suo ruolo nel migliorare la capacità di generalizzazione senza perdere troppa informazione durante l'addestramento.
- **Effetto del Numero di Neuroni e Dati (*k*)** - 10*k* campioni ha favorito la migliore **Test Accuracy**, indicando che un dataset più grande aiuta il modello a generalizzare meglio. - 7*k* campioni ha portato alla migliore **Test F1-score**, suggerendo che una quantità inferiore di dati può favorire un migliore bilanciamento tra precision e recall.

4.5.5 Osservazioni

L'analisi delle migliori configurazioni nei dataset BPI12 e HD ha fattto emergere alcuni aspetti interessanti e altri controintuitivi. A partire dalla dimensione e complessità dei dataset, il BPI12 è significativamente più grande e presenta tracce più lunghe e complesse rispetto all'HD. Questo dovrebbe riflettersi nelle scelte degli iperparametri, ma l'analisi dei risultati suggerisce delle discrepanze rispetto alle aspettative teoriche.

In primo luogo, ci si potrebbe aspettare che un dataset più grande e complesso come il BPI12 necessiti di una rete più profonda (più convolutional layers) o con più neuroni per catturare al meglio le dipendenze a lungo termine. Tuttavia, si osserva che per la migliore combinazione in termini di Test Accuracy, il numero di conv layers è 5 sia per BPI12 che per HD, mentre per il Test F1-score il BPI12 arriva a 7 conv layers, solo leggermente inferiore agli 8 layers dell'HD. Questo è un risultato controintuitivo: ci si sarebbe aspettati che l'HD, essendo più semplice e con tracce più corte, potesse funzionare meglio con una rete meno profonda, mentre il BPI12, più complesso, beneficiasse di una rete più profonda.

Un'altra differenza notevole riguarda il numero di neuroni per layer, un altro fattore chiave per la capacità di apprendimento della rete. Nel caso del Test Accuracy, l'HD utilizza 128 neuroni per layer, il doppio dei 64 neuroni per layer del BPI12. Anche per

il Test F1-score, l'HD mantiene 128 neuroni, mentre il BPI12 resta fermo a 64. Di nuovo, questo va in controtendenza rispetto all'ipotesi iniziale che un dataset più grande richieda una capacità di rappresentazione maggiore. Un possibile motivo di questa configurazione potrebbe essere nella natura delle attività nel dataset HD, che richiedono una maggiore capacità espressiva per la classificazione.

Per quanto riguarda l'impostazione del learning rate, notiamo che per entrambi i dataset la configurazione ottimale per il Test Accuracy utilizza un learning rate più alto (0.001), mentre per il Test F1-score è preferibile un valore più basso (0.0001). Questo comportamento suggerisce che la convergenza dell'accuratezza sia più rapida con un apprendimento più aggressivo, mentre l'ottimizzazione dell'F1-score, che dipende da un miglior bilanciamento tra precision e recall, potrebbe beneficiare di un apprendimento più graduale e fine. Tuttavia, non si nota una differenza nel learning rate tra BPI12 e HD: ci si sarebbe aspettati che l'HD, essendo più piccolo, fosse più incline a funzionare bene con un learning rate più alto per evitare il rischio di overfitting, mentre il BPI12 necessitasse di un tuning più delicato.

Il dropout segue un pattern coerente tra i due dataset: nelle configurazioni che massimizzano il Test Accuracy, il BPI12 usa un valore 0.2, mentre l'HD scende a 0.1, indicando un minore bisogno di regolarizzazione. Questo potrebbe essere giustificato dal fatto che il dataset HD, essendo più piccolo, soffra maggiormente della perdita di informazione introdotta dal dropout. Per il Test F1-score, entrambi i dataset adottano 0.2, suggerendo che per ottimizzare questo parametro sia necessaria una regolarizzazione più forte. Questo valore risulta in linea con i motivi delle scelte del parametro effettuate.

In sintesi, i risultati evidenziano alcune scelte che confermano le aspettative teoriche, come la necessità di più conv layers nel BPI12 rispetto all'HD, ma allo stesso tempo mettono in discussione altre ipotesi, come il numero di neuroni superiore nell'HD o l'assenza di una differenza chiara nel learning rate. In previsione di miglioramenti futuri, si consiglia di analizzare più attentamente questi valori.

4.5.6 Analisi di Train loss e Test loss

Nel valutare la qualità dell'apprendimento di un modello, la Train Loss rappresenta la metrica più affidabile per comprendere la capacità del modello di adattarsi ai dati di addestramento. La sua importanza si basa su due fattori chiave:

- **Apprendimento efficace:** un modello con una Train Loss più bassa indica che ha appreso meglio la distribuzione dei dati, riuscendo a minimizzare l'errore sui campioni forniti durante l'addestramento. Questo suggerisce che il modello ha trovato una rappresentazione significativa dei pattern nei dati e ha ottimizzato i suoi parametri in modo efficace.

- **Assenza di overfitting:** se un modello avesse una Train Loss bassa e una Test Loss molto alta, si potrebbe concludere che ha sovradattato i dati di addestramento, ovvero ha imparato dettagli e rumore specifici piuttosto che pattern generali.

4.5.7 Analisi degli iperparametri di Train Loss più efficaci (HD)

- **Configurazione:** 200 epochs, 0.01 lr, 1024 batch size, 0.2 dropout, 5 conv layers, 256 neurons, 7k.
- **Metriche:**
  - Train Accuracy: 85.66%
  - Test Accuracy: 81.91%
  - Train F1-score: 82.43%
  - Test F1-score: 77.14%
  - Train Loss: **0.000479 (Migliore valore)**
  - Test Loss: **0.000772**

Interpretazione dei risultati

- **Effetto del Learning Rate (lr)** Un learning rate elevato (0.01) ha portato al miglior valore di **Train Loss**, indicando un apprendimento rapido ed efficace del modello. Tuttavia, la Test Loss non ha ottenuto il miglior valore, suggerendo un potenziale overfitting. Infatti si può notare come la terza misurazione, seppur abbia una train loss più alta, abbia anche una test loss più alta, che suggerisce una migliore generalizzazione.
- **Effetto del Numero di Convolutional Layers** L'architettura con 5 strati convoluzionali ha raggiunto la **migliore Train Loss**, suggerendo che una profondità moderata facilita l'apprendimento senza rendere il modello eccessivamente complesso.
- **Effetto del Dropout** Un dropout di 0.2 sembra aver bilanciato il training, evitando di perdere troppa informazione mentre preveniva l'overfitting. Tuttavia, dropout più alti in altre configurazioni hanno portato a una peggiore Train Loss.
- **Effetto del Numero di Campioni (k)** 7k campioni hanno favorito il miglior valore di **Train Loss**, mostrando che con una quantità adeguata di dati il modello è in grado di apprendere meglio i pattern.

4.5.8 Analisi degli iperparametri di Train Loss più efficaci (BPI12)

- **Configurazione:** 200 epochs, 0.0001 lr, 64 batch size, 0.1 dropout, 7 conv layers, 128 neurons, 10k.

- **Metriche:**
  - Train Accuracy: 91.66%
  - Test Accuracy: 73.68%
  - Train F1-score: 91.47%
  - Test F1-score: 74.03%
  - Train Loss: **0.003267 (Migliore valore)**
  - Test Loss: 0.017072

Interpretazione dei risultati

- **Effetto del Learning Rate (*lr*)** Un valore di 0.0001 ha permesso di ottenere il valore minimo di **Train Loss**, suggerendo un apprendimento più gradual. Tuttavia, un valore leggermente più alto (0.001) ha portato a una migliore **Test Accuracy**, evidenziando un compromesso tra stabilità e capacità di generalizzazione.
- **Effetto del Numero di Convolutional Layers** Utilizzando 7 strati convoluzionali si è ottenuta la minima **Train Loss**.
- **Effetto del Dropout** Un dropout di 0.1 ha portato alla minore **Train Loss**, indicando che il modello è stato in grado di apprendere con meno interruzioni nel processo di addestramento. Tuttavia, un valore leggermente più alto (0.2) ha portato a una **migliore Test Loss**, suggerendo che una regolarizzazione maggiore potrebbe prevenire l'overfitting.
- **Effetto del Numero di Campioni (*k*)** $10k$ campioni hanno portato alla **minima Train Loss**, indicando che più dati di addestramento migliorano l'apprendimento del modello.

4.6 Visualizzazione dei risultati

Per semplicità in questa sezione verranno proposte solo le migliori tre combinazioni di Test Accuracy analizzando Accuracy, F1, Loss per Train e Test, mostrando la corrispettiva matrice di confusione .

4.6.1 Visualizzazione dataset HD

1° combinazione

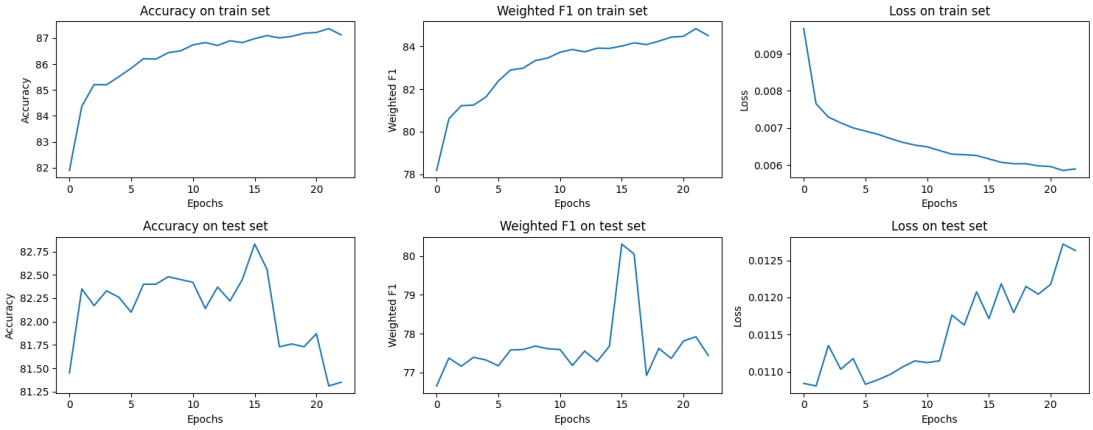

Figura 4.1: Metriche 1° combinazione.

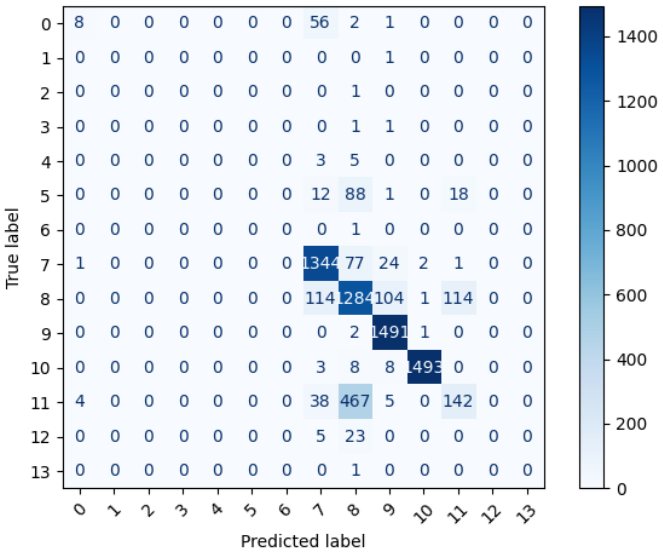

Figura 4.2: Matrice di confusione 1° combinazione.

2° combinazione

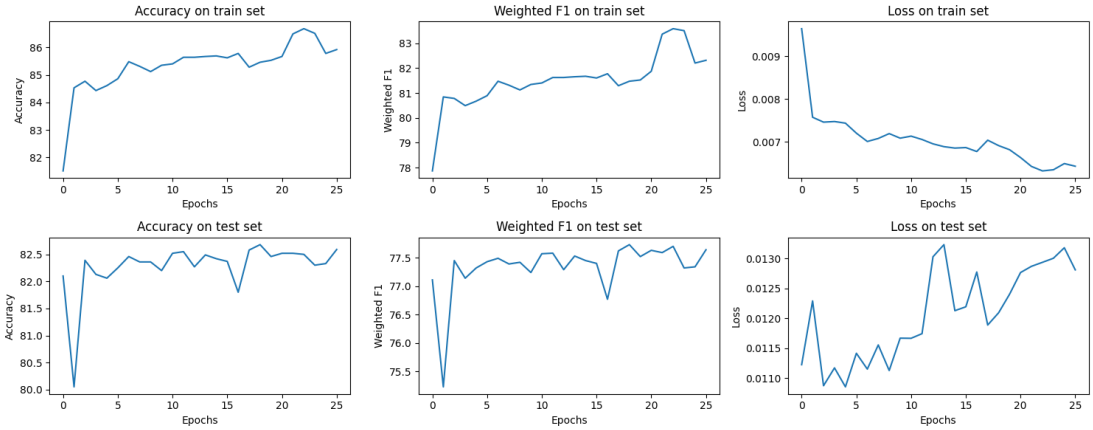

Figura 4.3: Metriche 2° combinazione.

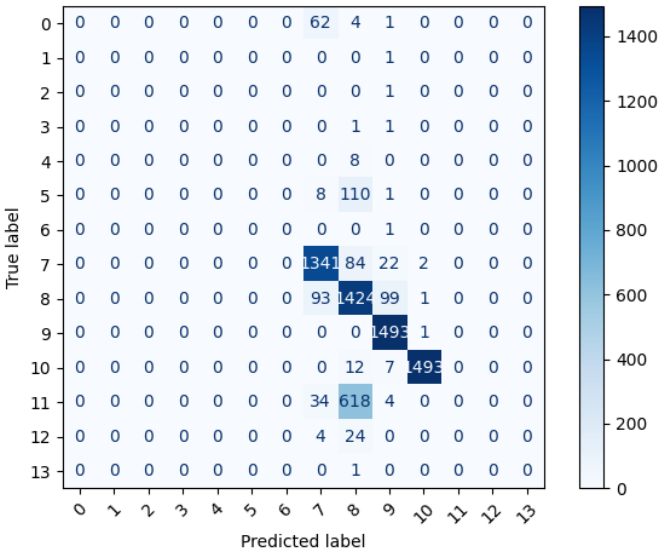

Figura 4.4: Matrice di confusione 2° combinazione.

3° combinazione

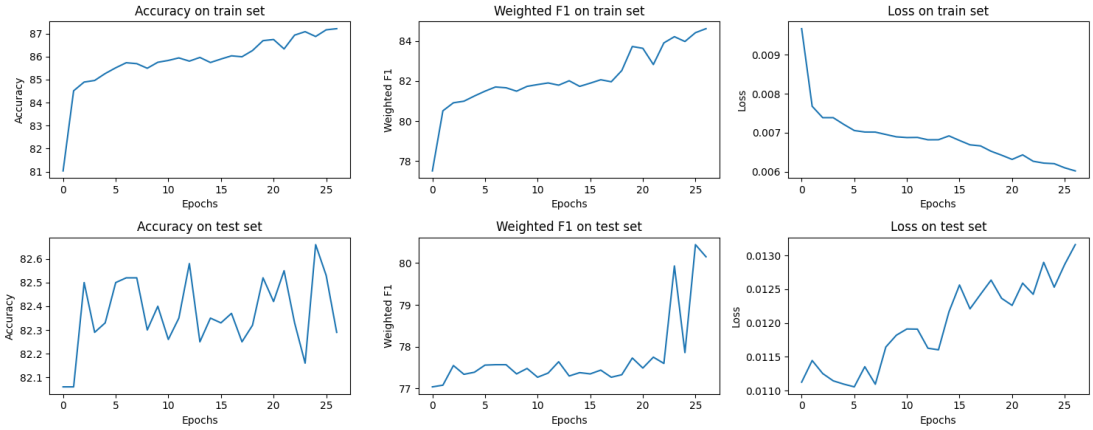

Figura 4.5: Metriche 3° combinazione.

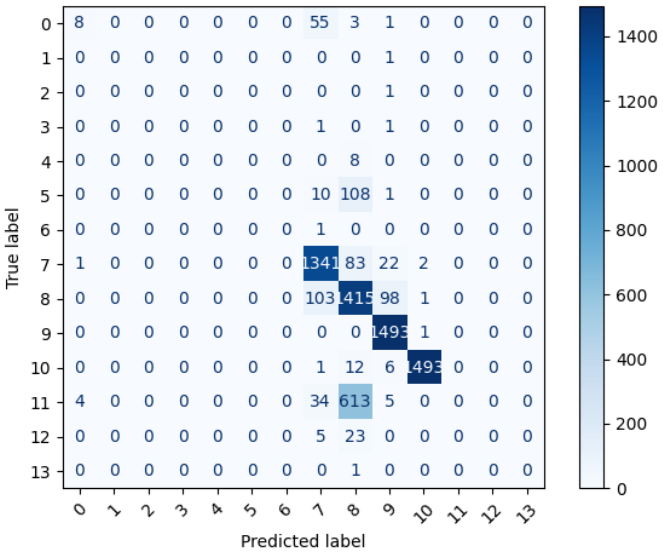

Figura 4.6: Matrice di confusione 3° combinazione.

4.6.2 Visualizzazione datasetBPI12

1° combinazione

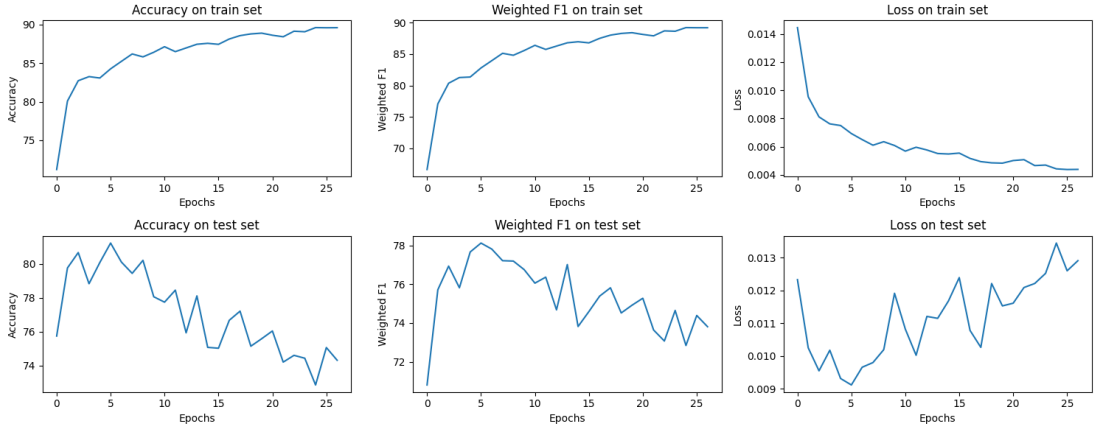

Figura 4.7: Metriche 1° combinazione.

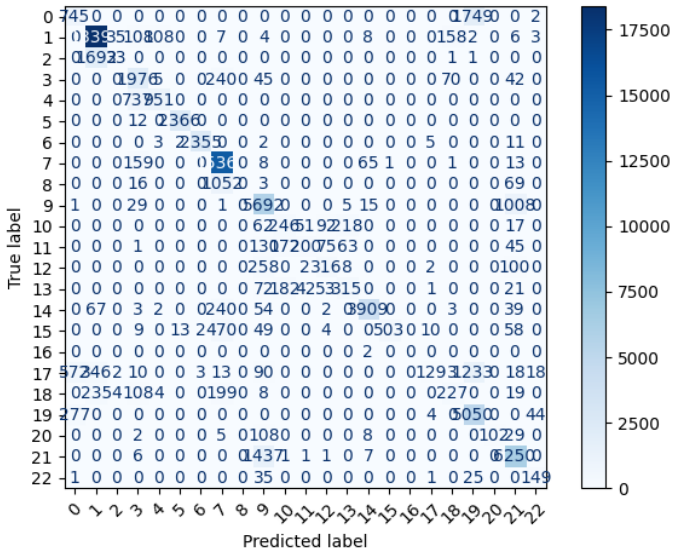

Figura 4.8: Matrice di confusione 1° combinazione.

2° combinazione

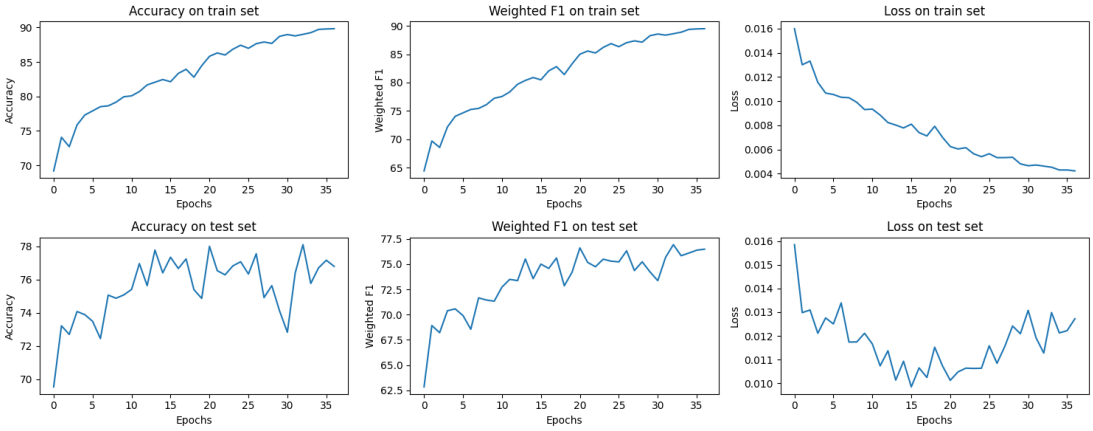

Figura 4.9: Metriche 2° combinazione.

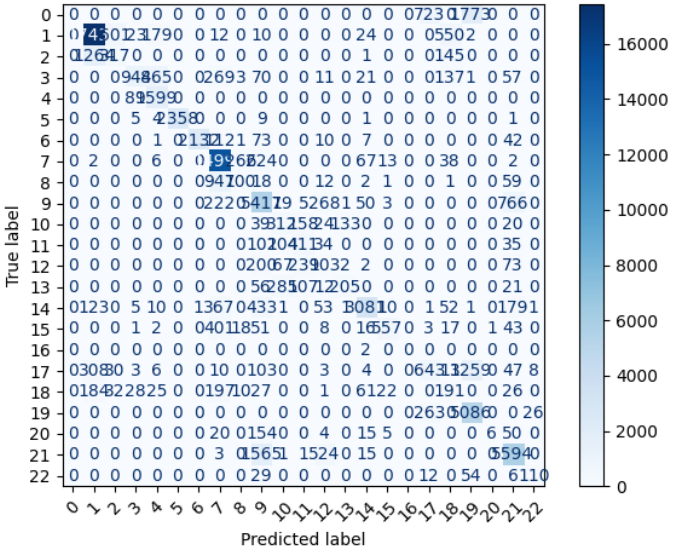

Figura 4.10: Matrice di confusione 2° combinazione.

3° combinazione

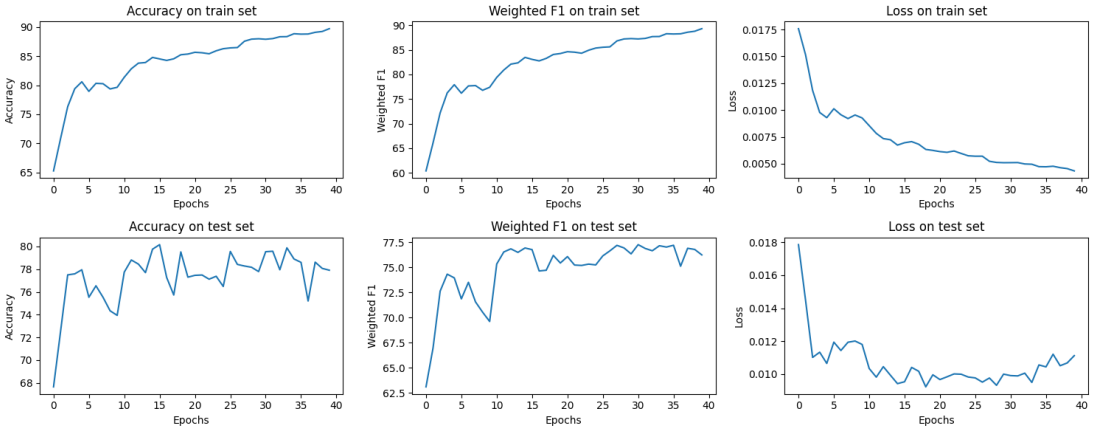

Figura 4.11: Metriche 3° combinazione.

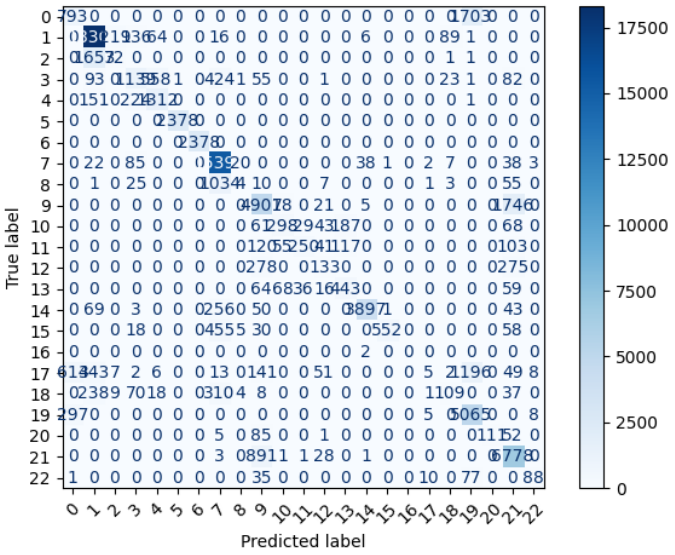

Figura 4.12: Matrice di confusione 3° combinazione.

4.7 Analisi al variare della lunghezza del prefisso

4.7.1 Lunghezza di prefisso

Il prefisso, come descritto precedentemente nella sezione 2.3, rappresenta un'osservazione parziale della sequenza completa di attività di un processo e viene utilizzato per prevedere quale sarà l'attività successiva o altre informazioni sul processo. La lunghezza del prefisso indica quanti eventi iniziali della traccia sono stati osservati fino a un dato momento. Formalmente, un prefisso di lunghezza k di una traccia $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ è una sottosequenza $p_k(\sigma) = \langle e_1, e_2, \ldots, e_k \rangle$, dove $k \leq n$.

4.7.2 Visualizzazione delle metriche per prefisso (HD)

L'analisi dei grafici (di cui uno riportato in figura 4.13) mostra che l'accuratezza e l'F1-score presentano un andamento con lievi oscillazioni, con una fase di stabilizzazione nelle tracce finali, che evidenzia la capacità del modello di adattarsi alle specificità delle lunghezze intermedie. Questo comportamento può essere compreso considerando la struttura del dataset, che include 3804 tracce distribuite su 9 tipi di attività, con lunghezze variabili da 1 a 14 eventi, ma una media di 3 eventi per traccia. Con attività limitate ed eventi relativamente brevi il modello riesce ad avere misure di accuratezza ed F1 elevate (siamo sopra l'80% nella maggior parte delle lunghezze di prefisso).

Con tracce che contengono dai 5 ai 6 eventi, il modello sembra raggiungere un punto di ottimizzazione, dove riesce a rappresentare in modo più adeguato i dati. Questo è probabilmente dovuto al fatto che la maggior parte delle tracce risulta avere una lunghezza stimabile a 5 o 6 eventi.

Quando si arriva a tracce più lunghe, in particolare quelle con eventi pari a 7 o maggiori di 9, il modello inizia a mostrare difficoltà a causa della probabile mancanza di tracce con lunghezze comprese tra 11 e 13 eventi, le quali non sono rappresentate nel dataset, risultando in un'accuratezza pari a 0 per queste lunghezze.

Il miglioramento delle prestazioni nelle tracce finali suggerisce che, quando il modello ha accesso a un contesto più ampio (ovvero le tracce con un numero maggiore di eventi), riesce a cogliere meglio le relazioni sequenziali e causali tra gli eventi, migliorando così la sua predittività. Le attività finali tendono a essere più regolari e meno suscettibili alla variabilità rispetto alle fasi intermedie.

4.7.3 Visualizzazione delle metriche per prefisso (BPI12)

L'andamento dei grafici proposti (di cui uno riportato in figura 4.14) evidenzia che accuratezza e F1 seguono un trend generalmente decrescente con l'aumentare della lunghezza del prefisso, per poi risalire verso il prefisso completo. Questo andamento, accompagnato da oscillazioni significative, riflette dinamiche complesse tipiche del

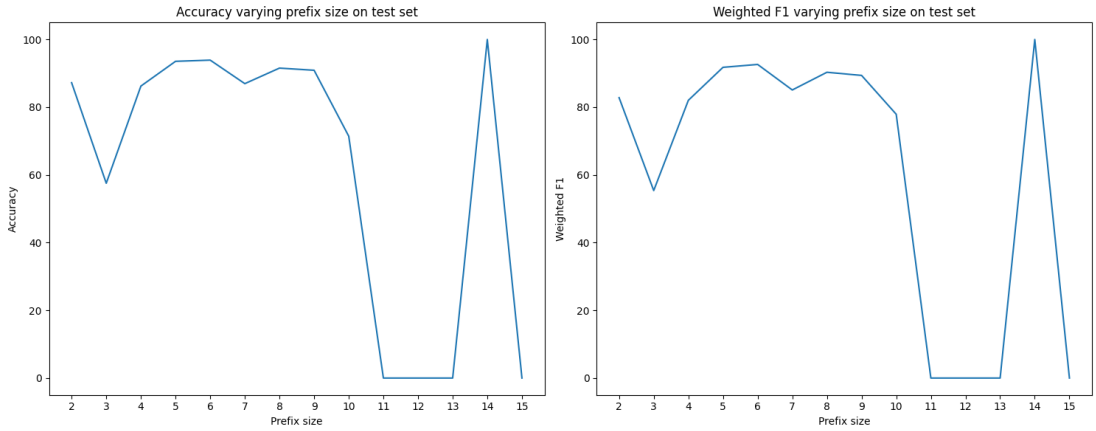

Figura 4.13: Accuracy e F1 di Test per dataset HD.

comportamento di un modello predittivo applicato a dataset di process mining. Va ricordato che il dataset in esame presenta tracce con lunghezze variabili da 3 a 175 eventi, con una media di 38 eventi per traccia.

Nei prefissi brevi, il modello sembra beneficiare della presenza di pattern ricorrenti e comuni, caratteristici delle fasi iniziali di molti processi. Queste prime fasi tendono a essere più regolari e strutturate, rendendo relativamente semplice al modello produrre previsioni accurate basandosi su un numero limitato di eventi. Tuttavia, con l'aumentare della lunghezza del prefisso, si entra in una zona del processo più complessa, in cui le tracce divergono e i percorsi seguiti diventano meno prevedibili. In queste fasi intermedie, il modello può incontrare difficoltà di generalizzazione, specialmente se la distribuzione dei dati non rappresenta adeguatamente i prefissi di lunghezza media. Ad esempio, in un processo di prenotazione, le attività iniziali legate alla raccolta dei dati personali seguono schemi ripetitivi. Con l'avanzare del processo, però, le sequenze si differenziano in base alle operazioni specifiche richieste, aumentando la variabilità e rompendo i pattern comuni. Questi punti di biforcazione, insieme a una possibile scarsità di campioni per alcune lunghezze di prefisso, possono amplificare l'instabilità predittiva del modello.

Il miglioramento delle prestazioni verso il prefisso completo, invece, indica che, quando il modello dispone di un contesto più ampio, è in grado di cogliere informazioni più dettagliate e significative sul processo. Nei prefissi vicini alla traccia completa, il modello riesce a riconoscere relazioni causali e sequenziali che erano difficili da individuare nei prefissi intermedi. Questo si traduce in una capacità predittiva più robusta nelle fasi finali, dove le attività tendono a seguire schemi più prevedibili, analoghi a quelli delle fasi iniziali.

Questa dinamica complessiva suggerisce che il modello eccelle nel catturare le relazioni

locali nei prefissi brevi e quelle globali nei prefissi completi, ma fatica a generalizzare efficacemente nei prefissi intermedi. Tale comportamento riflette in parte la natura intrinseca dei dati, che includono 13.087 tracce distribuite su 23 tipi di attività: un numero significativo ma non particolarmente elevato. Questo può spiegare le difficoltà del modello nell'affrontare scenari complessi o scarsamente rappresentati nei dati.

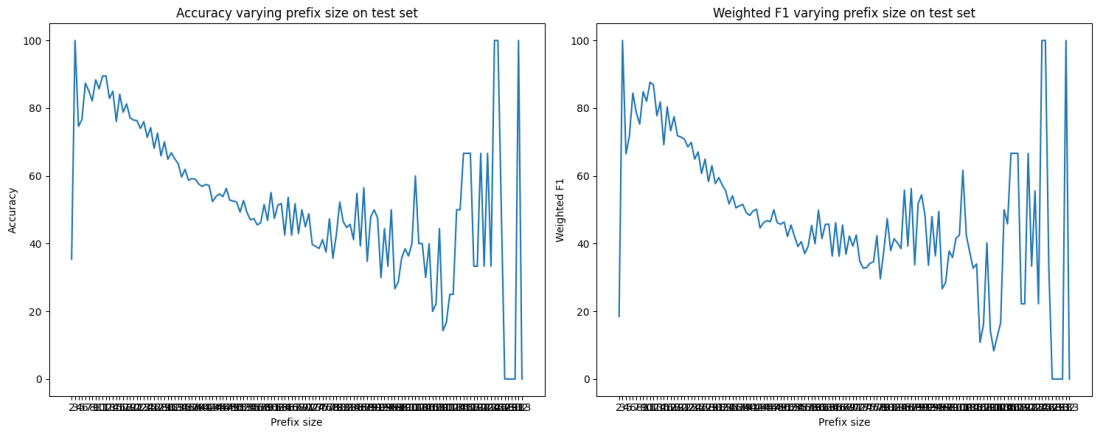

Figura 4.14: Accuracy e F1 di Test per dataset BPI12.

4.8 Osservazioni

Prendendo in considerazione, ad esempio, il database HD, è importante notare che il dataset presenta una lunghezza media di tracce pari a 3 eventi, con un massimo di 14 eventi per traccia. In base a queste caratteristiche, ci si potrebbe aspettare che il modello raggiunga performance migliori in termini di accuratezza e F1 per tracce la cui lunghezza si avvicina al valore medio, senza mai superare la lunghezza massima del prefisso. Tuttavia, questo non è quanto osservato nel comportamento del modello.

Una possibile spiegazione di questa discrepanza risiede nel fatto che la lunghezza media delle tracce, come descritta nel dataset, non tiene conto degli eventi fittizi aggiunti all'inizio e alla fine delle tracce durante il processo di IG, né delle trasformazioni applicate in fase di IG repair. Questi eventi supplementari influenzano la lunghezza complessiva del prefisso, facendo sì che essa possa superare il limite massimo precedentemente definito per il dataset. Di conseguenza, la lunghezza del prefisso effettivo risulta maggiore rispetto a quella descritta nella documentazione del dataset, con impatti anche sui valori medi di lunghezza delle tracce che potrebbero non rispecchiare correttamente la distribuzione originale dei dati.descritta nella sezione dei dataset. Inoltre questo fa cambiare anche valori medi descritti precedentemente.

4.9 Conclusioni

Approach		Dataset	
		Helpdesk	BPI12
Multi-BIG-DGCNN	Acc	**86.15%**	76.09%
	F1	**83.19%**	71.12%
BIG-DGCNN	Acc	85.18%	72.90%
	F1	82.93%	68.31%
GCNN	Acc	80.42%	60.92%
	F1	76.73%	58.95%
MLP	Acc	82.16%	71.80%
	F1	77.45%	66.07%
CNN	Acc	85.02%	78.45%
	F1	82.13%	75.92%
LSTM	Acc	74.49%	**79.06%**
	F1	72.13%	**75.48%**

Figura 4.15: Valori di Accuracy e F1 da superare per dataset.

Per quanto riguarda il dataset BPI12 i risultati sono stati ottimi, riuscendo a superare di gran lunga le combinazioni di iperparametri precedentemente utilizzate. Si è passati da un Accuracy del 76.09% ad una dell'**81.24%**. Anche la F1 risulta notevolmente migliorata passando dal 71.12% al **78.56%**.

I risultati del dataset HD invece risultano essere meno promettenti. L'obiettivo di riuscire a superare i precedenti esperimenti non ha portato i risultati attesi, ma sono stati comunque raggiunti buoni livelli di Accuracy e F1 rispettivamente del 82.83% (rispetto la top combinazione di 86.15%) e 80.82% (rispetto all'83.19%). *Nonostante ciò, poichè le valutazioni di test sono relativamente elvate, ritengo che gli iperparametri utilizzati per gli esperimenti del dataset HD possano aver portato ad una generalizzazione migliore e che dunque possano effettivamente portare a valutazioni migliori in dataset con più attività o con attività inusuali, classificando meglio rispetto ai parametri che erano stati proposti da superare, che al contrario potrebbero aver portato ad un leggero overfitting.*

Per quanto riguarda le tabelle riguardanti le migliori combinazioni di Train e Test Loss ci sono risultati contraddittori. In particolare, si è osservato che una train loss più bassa non ha necessariamente portato a un miglioramento delle performance sui dati di test, con una diminuzione dell'accuratezza e dell'F1 score rispetto ad altre configurazioni. Questo fenomeno potrebbe essere interpretabile come un caso di overfitting, dove il modello, ottimizzandosi troppo sui dati di addestramento, ha finito per apprendere caratteristiche troppo specifiche e poco generalizzabili. Sebbene una riduzione della train loss suggerisca, in linea teorica, un miglioramento nella performance sui dati di

addestramento, tale abbassamento non implica automaticamente una capacità migliore di generalizzare sui dati di test. Infatti, quando un modello riduce la train loss troppo rapidamente, potrebbe finire per memorizzare il dataset di addestramento, perdendo così la sua capacità di adattarsi a nuovi dati. Questo tipo di comportamento è più probabile quando il modello risulta essere eccessivamente complesso rispetto ai dati. Pertanto, nonostante il miglioramento della train loss, il modello potrebbe risultare meno adattabile ai dati di test, con conseguente peggioramento delle metriche di accuracy e F1 score. *Per questo motivo sono state prese come riferimento per le migliori combinazioni quelle con i risultati migliori di Test Accuracy e F1*.

In sviluppi futuri, si prevede di esplorare l'impostazione di un dropout a 0.3, per verificare se una tale configurazione possa meglio bilanciare la riduzione della train loss con una maggiore capacità di generalizzazione. Inatti, seppur inizialmente scartato, si è notato che un dropout del genere possa portare a dei miglioramenti significativi (presumibilmente solo per il dataset HD più complesso) in termini di Train e Test Loss.

Bibliography

D., P. (2024), «Slides rilasciate a lezione», .

DIAMANTINI C., P. D. . E. A., GENGA L. (2016), «Building instance graphs for highly variable processes. Expert Systems with Applications», . (Cited at page 6)

MARINELLI (2024), «Slides di Ricerca Operativa 2», . (Cited at page 16)

VAN DONGEN, B. (2012), «BPIChallenge2012 , https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-7597 6070e91f. https://data.4tu.nl/articles/dataset/BPI Challenge 2012/12689204», . (Cited at page 29)

VERENICH, I. (2016), «Helpdesk , https://doi.org/10.17632/39bp3vv62t.1», . (Cited at page 29)

WILLIAM L. HAMILTON, R. Y. e LESKOVEC, J. (2017), «Inductive Representation Learning on Large Graphs», . (Cited at page 15)