Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Reti neurali convoluzionali per la selezione di dati nell'esperimento HERD

Candidato
Federico Magnolfi

Relatore
Prof. Paolo Frasconi

Correlatori
Dott. Stefano Martina
Dott. Nicola Mori

Anno Accademico 2017/2018

Sommario

L'esperimento *High Energy cosmic-Radiation Detection* (HERD) verrà svolto sulla stazione spaziale cinese con l'obiettivo di misurare il flusso dei raggi cosmici galattici, in modo particolarmente dettagliato alle alte energie. A tal scopo verrà usato uno strumento progettato per rivelare il passaggio di una particella cosmica al suo interno, identificarne la specie e misurarne l'energia con uno dei suoi sottosistemi, detto calorimetro. I dati raccolti devono essere analizzati a terra dai fisici, ma a causa della limitata larghezza di banda disponibile per la trasmissione, non tutti i dati acquisiti possono verosimilmente essere trasmessi a terra. Inoltre alcune particelle che attraversano lo strumento producono dati inutili per l'analisi, che dovranno essere scartati.

In questa tesi si studiano algoritmi di selezione dei dati che, in ottica di essere eseguiti da risorse di calcolo presenti sul rivelatore stesso, consentano di individuare i dati da memorizzare e trasferire a terra, in modo da ottimizzare l'utilizzo delle limitate risorse del sistema.

Uno studio di questo tipo è una novità assoluta in questo ambito: si realizza un'analisi preliminare del problema, facendo delle semplificazioni rispetto al caso reale per ottenere una prima stima della fattibilità e delle prestazioni di tale approccio. Come algoritmi di classificazione si testano delle *reti neurali convoluzionali* (CNN) e si confrontano con algoritmi più semplici. I risultati ottenuti sono molto differenti per i due tipi di particelle trattati, elettroni e protoni: per i primi il problema si dimostra così facile da poter essere risolto agilmente con un algoritmo molto più semplice di una rete neurale; per i secondi invece è necessario l'utilizzo delle CNN per raggiungere una classificazione di qualità.

Indice

1 Introduzione **1**

2 Stima del flusso **5**

2.1 Flusso 5

2.2 Calorimetro 7

2.2.1 Sciame di un evento 8

2.3 Stima 10

2.3.1 Selezione degli eventi 12

2.3.2 Tempo vivo e tempo morto 15

2.3.3 Accuratezza della stima 16

2.4 Studio del sistema di trigger 17

3 Apprendimento supervisionato **19**

3.1 Reti Neurali 21

3.2 Reti Neurali Convoluzionali 26

3.2.1 Operazione di convoluzione 26

3.2.2 Convoluzione all'interno di reti neurali 27

4 Metodi **31**

4.1 Dataset 31

4.2 Dettagli su CNN utilizzate 32

4.3 Base di partenza 33
4.3.1 Algoritmo baseline 33
4.3.2 Variante algoritmo baseline 34
4.4 Criteri di valutazione 35
4.4.1 Curva precision recall 36
4.4.2 Confronti 38

5 Risultati **39**
5.1 Esperimenti 39
5.1.1 Elettroni 40
5.1.2 Protoni 47
5.2 Conclusioni 54

Bibliografia **56**

Capitolo 1

Introduzione

Nel 2020 è previsto il lancio del primo modulo della stazione spaziale cinese, sulla quale nel 2025 verrà installato l'esperimento **HERD** (High Energy cosmic-Radiation Detection) [1].

Lo scopo principale di questo esperimento è di misurare, in funzione dell'energia, l'abbondanza di varie specie di particelle nei raggi cosmici, identificando per la prima volta in maniera diretta specie nucleari (protoni, nuclei di elio ecc.) ad energie dell'ordine del PetaelettronVolt.
La quantità fisica che descrive questa abbondanza è detta **flusso** dei raggi cosmici galattici, ed ha le proprietà di essere costante nel tempo ed isotropa: si osserva sempre lo stesso valore del flusso indipendentemente dalla direzione di osservazione e dal momento in cui si effettua l'osservazione stessa.

Il calorimetro di HERD, simile ad un cubo, è formato da elementi sensibili di forma a loro volta cubica, i quali inducono una particella cosmica che li attraversi a generare uno sciame di particelle (in breve, la particella che attraversa il calorimetro genera nuove particelle a scapito di parte della sua energia iniziale; le particelle create ne generano a loro volta altre, in un processo a cascata). Il passaggio dello sciame genera a sua volta dei segnali

luminosi nei cubi sensibili, che vengono rivelati, convertiti in segnali elettrici ed acquisiti; essi costituiscono la misura dell'energia iniziale della particella che ha dato origine allo sciame.

Questo tipo di calorimetro ha la particolarità di poter misurare l'energia di una particella che entri dalla sua faccia superiore o da una delle quattro facce laterali: solo la faccia inferiore non è sensibile per via della strumentazione elettronica e delle strutture di supporto che si trovano sotto di essa. Rispetto alle precedenti generazioni, che erano sensibili solamente a particelle provenienti dall'alto ma non a quelle dai lati, questo calorimetro sarà in grado di generare una quantità di dati tale da renderne impossibile un completo invio a terra. Da qui la necessità di scegliere in modo intelligente i dati da mandare a terra, selezionando quelli che possono essere usati per la stima del flusso e scartando gli altri. Sempre in un'ottica di ottimizzazione delle prestazioni dello strumento, sarebbe ottimale effettuare tale selezione immediatamente dopo il passaggio della particella, sfruttando i segnali elettrici grezzi generati da tale passaggio per decidere se scartare l'evento ed evitando in tale modo la conversione dei segnali elettronici analogici in segnali digitali e la loro successiva memorizzazione. Questa attività richiede infatti molto tempo (dell'ordine delle decine di microsecondi) durante il quale lo strumento non è sensibile al passaggio di altre particelle. Limitare quindi questo tempo morto aumenta la capacità di rivelazione di particelle dello strumento.

Lo scopo di questa tesi è quello di effettuare uno studio preliminare sulla possibilità di individuare con grande accuratezza i dati di qualità sufficiente per l'analisi fisica, con modalità che possano in un secondo momento essere adattate alla loro applicazione direttamente in orbita. Questo tipo di studio è altamente innovativo nel campo e differisce in modo sostanziale dall'approccio usato dalla attuale generazione di rivelatori, i quali acquisiscono dati

usando criteri di qualità semplicistici e minimali che risultano nell'invio a terra di una grande frazione di dati non utilizzabili. Tale approccio ovviamente non scala con l'aumentare delle dimensioni del rivelatore, e risulterebbe quindi essere un fattore limitante per un esperimento come HERD. Come caso di studio per questa tesi si assume la problematica della selezione dei dati prodotti dal passaggio di elettroni e protoni, i quali costituiscono due delle specie più interessanti presenti nei raggi cosmici. Dal momento che il criterio per la selezione dei dati è diverso a seconda della particella che ha originato l'evento, si suppone che l'identificazione del tipo di particella sia già stata fatta.

Il criterio di classificazione, pur differente per le due specie a causa delle differenze fisiche intrinseche nei due casi, è volto a selezionare quegli eventi in cui lo sciame è ben contenuto all'interno del calorimetro ed è piccola la parte di sciame che si sviluppa al di fuori dello strumento: per questi eventi, di conseguenza, sarà piccola anche la frazione di energia persa, la quale costituisce la maggior sorgente di errore nella misura dell'energia della particella cosmica.

Si effettuerà una classificazione preliminare su dei dataset, ottenuti tramite simulazione Monte Carlo con il toolkit GEANT4 [2], di elettroni e protoni attraversanti il calorimetro. Per loro natura i dati simulati danno accesso a parametri che tipicamente non sono disponibili nei dataset ottenuti col vero strumento, come i dettagli dello sviluppo dello sciame e i parametri veri della particella incidente (energia, direzione di arrivo ecc.), detti verità Monte Carlo. È quindi possibile disporre di una *ground truth* molto accurata sfruttando la conoscenza di tali parametri.

Si utilizzano questi dati supervisionati per valutare le prestazioni di algoritmi che non sfruttino la verità Monte Carlo e che quindi risultino applicabili

anche ai dati reali prodotti dal vero strumento. La scelta di tali algoritmi non è immediata, motivo per cui ne verranno provati diversi.

Nel presente lavoro abbiamo ottenuto risultati sperimentali con delle reti neurali convoluzionali, confrontate con altri algoritmi usati come base di partenza, i quali si basano sull'osservazione dell'energia totale rilasciata nell'evento.

Gli algoritmi provati sono soltanto versioni preliminari di eventuali algoritmi che verranno utilizzati in orbita, che avranno il requisito di dover essere eseguiti in hardware da particolari circuiti integrati, detti *Field Programmable Gate Array* (FPGA) [3].

Capitolo 2

Stima del flusso

2.1 Flusso

Il flusso dei raggi cosmici galattici è una grandezza fisica differenziale, definita nel seguente modo:

$$\Phi = \frac{dN}{dt\ dE\ d\Omega}\,, \tag{2.1}$$

con:

- N: numero di particelle incidenti sul rivelatore;
- t: tempo di osservazione;
- E: energia della particella incidente;
- Ω: angolo solido osservato dal rivelatore.

Questo flusso ha le proprietà di essere isotropo (si osserva sempre lo stesso flusso indipendentemente dalla direzione di puntamento dello strumento di misura) e costante nel tempo (ad energie maggiori delle decine di Gigae-

lettronVolt, GeV, mentre ad energie inferiori esso è influenzato da fenomeni transienti come l'attività solare).

Uno degli scopi dell'esperimento HERD è quello di misurare il flusso in funzione dell'energia per elettroni e nuclei atomici; in prima approssimazione, il flusso delle varie specie ha un andamento con l'energia a legge di potenza:

$$\Phi(E) = A\,E^{-\lambda}\,. \tag{2.2}$$

Questa forma funzionale determina un rapido decrescere del flusso con l'aumentare dell'energia della particella (in pratica osservare una particella di energia E è tanto più raro quanto più grande è E).

Per misurare quindi i flussi ad alte energie è necessario uno strumento molto grande, capace di osservare un angolo solido molto grande, per riuscire a rivelare un numero statisticamente significativo di particelle di alta energia in un tempo ragionevole (circa qualche anno per gli esperimenti spaziali tipici). Gli esperimenti spaziali, che hanno la possibilità di identificare le specie delle particelle e quindi di misurare il flusso di ognuna separatamente invece che limitarsi allo spettro inclusivo di tutte le specie, sono tipicamente limitati in dimensioni per via dei vincoli tipici di massa e consumo elettrico delle missioni spaziali. Questo costituisce il limite principale alla massima energia del flusso che essi possono misurare; HERD è stato concepito per spostare questo limite verso regioni di alta energia tutt'oggi inesplorate da esperimenti spaziali. Benché le alte energie costituiscano l'obiettivo scientifico principale di HERD, il progetto è in fase di definizione per garantire una certa sensibilità alle energie più basse, specie nella regione influenzata dall'attività solare che costituisce un'interessante caso di studio.

2.2 Calorimetro

Il più importante dei sottosistemi di cui è costituito HERD è il calorimetro, il cui scopo è misurare l'energia di una particella che incida sullo strumento. Questo calorimetro, in questo caso di studio semplificato, è un cubo formato da $20 \times 20 \times 20$ cubetti di un materiale scintillante detto *LYSO*, ciascuno di lato 3 cm circa. Nel progetto reale il calorimetro non è un cubo ma un prisma a base ottagonale, ottenuto togliendo i cubetti in corrispondenza degli spigoli verticali, per risparmiare peso; in questo elaborato si considera il cubo intero, visto che si tratta di uno studio preliminare e si vuole usare un modello semplificato rispetto a quello definitivo. Una eventuale applicazione alla geometria reale del calorimetro è demandata a uno studio futuro.

Il passaggio di una particella, detto evento, avvia l'acquisizione dei segnali luminosi prodotti dallo sciame nei singoli cubi e convertiti in segnali elettrici, registrando un valore per ogni cubetto. Questi valori, sotto forma di numeri reali proporzionali all'energia rilasciata dallo sciame in ogni singolo cubetto, verranno utilizzati in questo lavoro per selezionare gli eventi utili alla stima del flusso. Nel caso reale della selezione nello spazio è lecito ipotizzare che questo tipo di dato, che richiede una conversione analogico-digitale dei segnali elettrici prima di poter essere elaborato, non sia disponibile a causa del lungo tempo necessario per ottenerlo, che rallenterebbe eccessivamente il processo di decisione sulla qualità dell'evento compromettendo l'eventuale acquisizione. Sembra quindi più realistico supporre che la classificazione possa essere fatta su un tensore di segnali quantizzati ad un 1 bit, ottenuti comparando velocemente il segnale analogico di ogni singolo cubo con una valore di riferimento. Tuttavia questo caso è verosimilmente più complesso da trattare, in quanto si dispone di minori informazioni rispetto al caso

semplificato. Inoltre, è impensabile poter analizzare direttamente i singoli valori del tensore, ma essi dovranno essere in qualche modo aggregati per semplificare l'elettronica che li elabori.

Per questi motivi, è stato deciso di lavorare inizialmente con segnali sotto forma di numeri reali, ed eseguire successivamente esperimenti con dati più realistici a valori aggregati e/o ad 1 bit, così da poter comparare i vari casi.

2.2.1 Sciame di un evento

Quando una particella passa all'interno del calorimetro, il rilascio di energia non è sempre uguale, ma dipende dalle leggi della meccanica quantistica, che sono probabilistiche. I casi estremi prevedono che una particella attraversi completamente il calorimetro rilasciando la minima quantità di energia concessa dalle leggi fisiche, oppure che depositi nel calorimetro tutta la sua energia. Nel primo caso, l'energia rilasciata è pressoché indipendente dall'energia della particella: non è quindi possibile risalire a quest'ultima misurando l'energia depositata nel calorimetro, e quindi questo tipo di evento viene tipicamente scartato nella misura dei flussi. Nel secondo caso, invece, la particella inizia a perdere energia sia trasferendola al calorimetro sia producendo altre particelle di energia inferiore; anche queste particelle perdono parte della loro energia depositandola nel calorimetro, e parte producendo a loro volta altre particelle di energia ancora inferiore. Questo processo di moltiplicazione delle particelle, ognuna delle quali deposita energia nel calorimetro, aumenta l'efficacia del trasferimento di energia dalla particella originaria al calorimetro, rendendo possibile un deposito completo nel volume limitato occupato dal calorimetro. Data la natura probabilistica delle leggi che ne governano lo sviluppo, lo sciame di particelle ha forma, dimensioni e composizione variabili evento per evento, anche a seconda del tipo di particella che lo genera.

Da queste dipende l'efficacia del trasferimento di energia al calorimetro. Un esempio di uno sciame prodotto da un elettrone nel calorimetro, ottenuto mediante una simulazione Monte Carlo basata su GEANT4, è riportato in Figura 2.1.

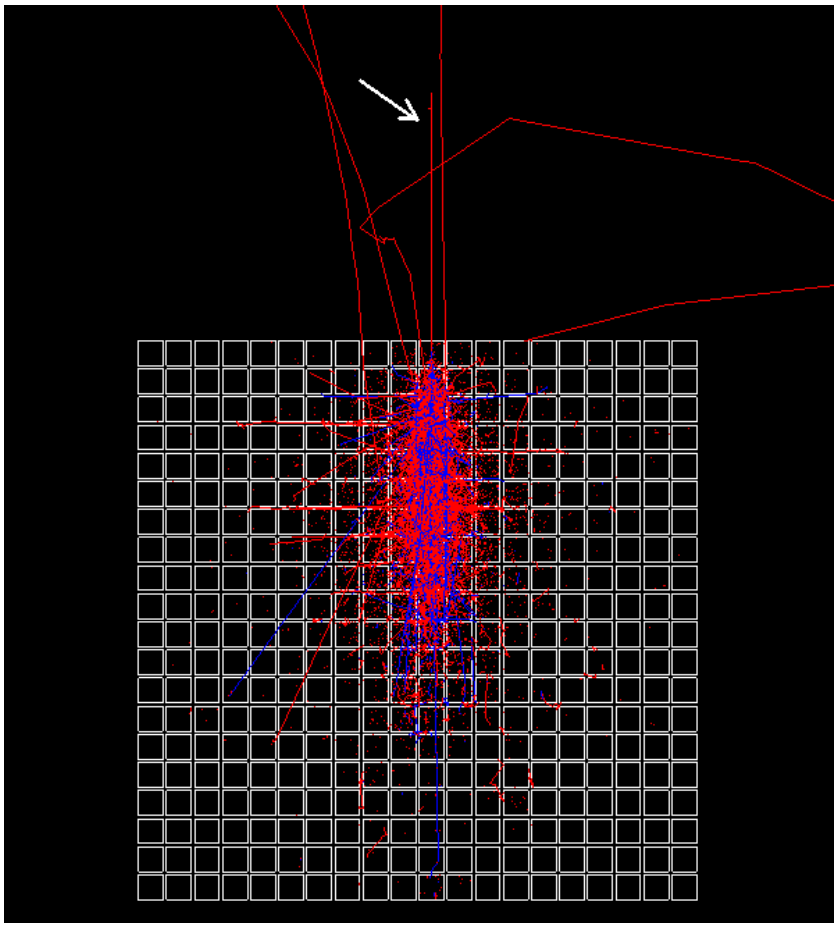

Figura 2.1: Immagine di uno sciame prodotto da un elettrone di energia 100 GeV ottenuto dalla simulazione GEANT4. Lo sciame è composto da particelle con carica elettrica negativa (in rosso, principalmente elettroni) e positiva (in blu, principalmente positroni). Sono visibili l'elettrone in ingresso (indicato dalla freccia bianca) e alcuni elettroni prodotti dallo sciame che fuoriescono dal calorimetro (linee rosse)

Ai fini della misura del flusso sono utili gli eventi che generano sciami ben contenuti nel calorimetro: un'eventuale fuoriuscita dal calorimetro di un

numero consistente di particelle dello sciame produce infatti una misura dell'energia depositata che si scosta molto da quella della particella primaria, introducendo quindi un errore non trascurabile sulla misura di quest'ultima. La quantificazione del contenimento dipende essenzialmente dall'energia e dal tipo di particella iniziale e quindi dal tipo di sciame generato: per gli elettroni un calorimetro come quello considerato in questo studio ha perdite limitatissime che consentono di stimare l'energia iniziale con un errore relativo dell'ordine di qualche percento, mentre per i protoni le perdite sono tali da non consentire errori inferiori al 20% circa.

2.3 Stima

Ogni particella che genera un evento ha una certa energia E_p e il suo passaggio nel calorimetro rilascia in totale un'energia E_m, che è l'energia misurata. L'energia incognita E_p è quella utile per la stima del flusso, e deve essere ricavata dall'energia rilevata E_m. L'evento non è utilizzabile per la stima quando non è possibile ricavare E_p con sufficiente precisione dalla misura di E_m, in tal caso non si può far altro che scartare il dato. Nel caso in cui l'evento sia utilizzabile, conoscendo E_m si può ricavare E_p, infatti esse sono in relazione pressoché lineare finché lo sciame è ben contenuto (quindi non si perde energia) e non intervengono effetti strumentali. In ogni caso, la stima precisa di questa relazione è un compito per i fisici sperimentali e non è rilevante per questo lavoro.

Per ogni evento buono, viene ricostruita l'energia originale $\hat{E}_p$, che sarà idealmente vicina ad E_p. Per stimare il flusso vengono definiti degli intervalli di energia, per l'intervallo i-esimo si conta il numero N_i di eventi che hanno una $\hat{E}_p$ che è compresa nell'intervallo, e si ottiene un istogramma.

L'istogramma deve essere normalizzato, e i fattori di normalizzazione sono quattro:

- ϵ_i: efficienza di selezione ed individuazione della particella all'energia tipica dell'intervallo i-esimo; in altre parole, visto che tipicamente uno strumento reale ha delle prestazioni limitate e quindi non rileva o identifica la totalità dei raggi cosmici che incidono su di esso, questo numero indica la frazione di quelli che lo strumento riesce a rilevare rispetto al totale;
- Δt: intervallo di tempo durante il quale sono state fatte le rilevazioni;
- ΔE_i: ampiezza dell'intervallo di energia i-esimo;
- Ω: angolo solido che riesce a vedere lo strumento.

Quindi, il flusso stimato all'energia E_i, in corrispondenza del valore centrale dell'intervallo i-esimo di energia, sarà:

$$\Phi(E_i) = \Phi_i = \frac{N_i}{\epsilon_i \, \Delta t \, \Delta E_i \, \Omega} \, . \tag{2.3}$$

Ovviamente, ognuna di queste stime è soggetta ad un errore, che viene calcolato trattando opportunamente le varie cause dell'errore stesso. Come risultato finale si ottiene quindi un grafico simile a quello in Figura 2.2.

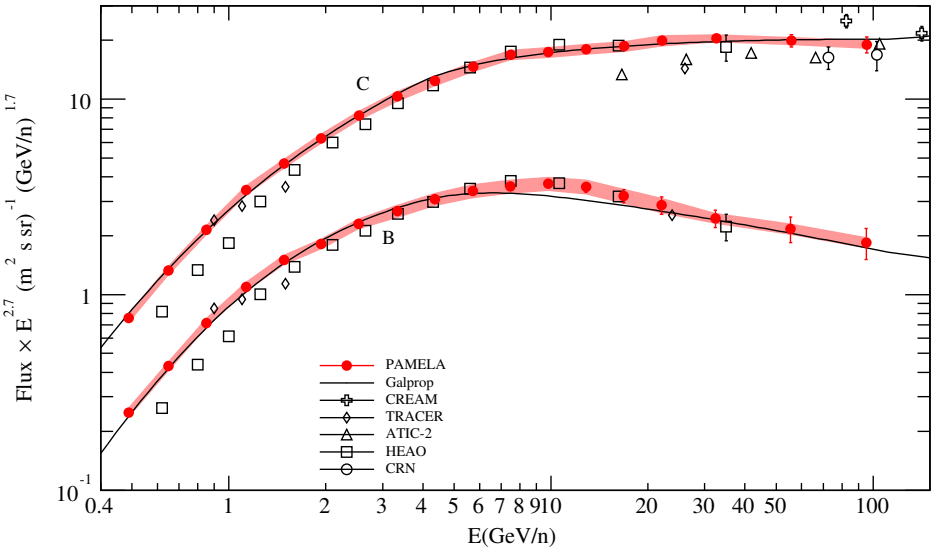

Figura 2.2: Flussi dei nuclei di boro e carbonio misurati dall'esperimento PAMELA, confrontati con misure di altri esperimenti e curve teoriche [4].

2.3.1 Selezione degli eventi

Dire se un evento è "buono" o meno non è sempre facile, nemmeno per un fisico con esperienza che si mette ad osservare lo sciame di un evento: certe volte questo compito è immediato, in altre non si può dare una risposta univoca. Intuitivamente, per quanto concerne la selezione di eventi finalizzata alla misura del flusso, un evento è buono se il suo sciame è fisicamente ben contenuto nel calorimetro, ed in tal caso l'energia osservata è strettamente correlata a quella originale della particella.

Gli esperimenti attuali di raggi cosmici sono basati su una modalità operativa relativamente semplice, che consiste nell'applicare criteri molto inclusivi per decidere se in conseguenza del passaggio di una particella i dati prodotti dal rivelatore debbano essere acquisiti o meno. Questa strategia conservativa consente di acquisire la quasi totalità degli eventi buoni, a prezzo però

di acquisire anche una grande quantità di eventi non buoni. Il dataset così prodotto risulta quindi largamente contaminato da dati inutilizzabili per la misura dei flussi, che devono quindi essere scartati durante la fase di analisi dei dati considerando vari criteri, per esempio: la profondità nel calorimetro alla quale ha iniziato a formarsi lo sciame, la quantità di energia rilasciata vicino alle estremità, e così via. Ovviamente la capacità di tali criteri di discriminare eventi buoni (*positivi*) ed eventi non buoni (*negativi*) non è assoluta; nella terminologia usata dai fisici questa capacità di discriminazione è misurata dall'efficienza di selezione[1] ϵ e dall'analoga quantità ϵ_b, definite come:

$$\epsilon = \frac{TP}{TP + FN}\,, \tag{2.4}$$

$$\epsilon_b = \frac{TN}{TN + FP}\,, \tag{2.5}$$

dove TP sono i veri positivi, TN i veri negativi, FP i falsi positivi, FN i falsi negativi. Tipicamente, al posto di ϵ_b si usa il più significativo potere di reiezione, definito come:

$$R = \frac{\epsilon}{\epsilon_b}\,. \tag{2.6}$$

Dei buoni criteri di selezione risultano quindi in un alto valore sia di ϵ che di R. Per incrementare il valore di questi parametri è ormai comune in fisica l'utilizzo di algoritmi di selezione basati su reti neurali, allenati sia con dati ottenuti da simulazioni Monte Carlo dell'apparato di misura sia con dati acquisiti in condizioni controllate (ad esempio esponendo il rivelatore a fasci di particelle di specie ed energia nota). Tipicamente le prestazioni di reti così allenate sono migliori di quelle ottenibili basandosi sui semplici criteri

[1] L'efficienza di selezione è detta anche *recall* (*recupero*).

sopra descritti, permettendo un'accurata rimozione degli eventi non buoni dal dataset.

Per esperimenti che per loro natura producono dataset di grandi dimensioni e/o contenenti un grande numero di eventi non buoni risulta quindi estremamente interessante anticipare l'applicazione di questi algoritmi di discriminazione, spostandola dalla fase di analisi dei dati alla fase di decisione sull'acquisizione dei dati, detta fase di trigger. Un algoritmo di trigger in grado di riconoscere ed acquisire in larga parte eventi buoni impatta positivamente su vari aspetti di un esperimento:

1. produzione di dataset di minori dimensioni per via dell'eliminazione di eventi non buoni negli stadi preliminari della catena di acquisizione dati, con conseguente riduzione delle richieste di risorse di storage e trasmissione dei dati;

2. funzionamento più efficiente dell'esperimento, grazie alla possibilità di concentrare l'utilizzo delle risorse di acquisizione dati sugli eventi buoni (come spiegato nel prossimo paragrafo).

Molti esperimenti di fisica fondamentale agli acceleratori, per esempio gli esperimenti CMS ed LHCb montati sull'acceleratore LHC del CERN di Ginevra, già usano con profitto dei sistemi di trigger basati, fra gli altri, anche su algoritmi tipo reti neurali. L'applicazione di tali metodologie non è invece mai stata tentata sul trigger di esperimenti di raggi cosmici; tuttavia le crescenti dimensioni di tali esperimenti, dettate dalla spinta verso le regioni di più alte energie e risultante in un incremento considerevole dei dati prodotti, costituiscono un forte stimolo allo studio di questa problematica.

2.3.2 Tempo vivo e tempo morto

Il tempo DT che passa tra il passaggio di una particella e il momento in cui lo strumento è pronto per rilevare il passaggio di un'altra particella, è detto tempo morto, e può essere visto come:

$$DT = \tau_t + \tau_a + \tau_i \,, \tag{2.7}$$

con:

- τ_t, tempo di trigger: tempo che intercorre tra il passaggio della particella e l'invio del segnale di trigger che avvia l'acquisizione; questo tempo è tipicamente piuttosto piccolo rispetto a DT;
- τ_a, tempo di acquisizione: tempo necessario alla lettura dei segnali elettrici prodotti dai sensori e alla conversione analogico-digitale; il rapporto tra questo tempo e DT è di solito una frazione considerevole;
- τ_i, tempo di impacchettamento: tempo necessario all'organizzazione e all'invio dei dati al computer che si occupa della loro trasmissione o memorizzazione.

Durante il tempo morto, lo strumento non è sensibile al passaggio di altre particelle: minimizzare i tempi morti è quindi necessario per massimizzare la probabilità di rilevare gli eventi rari. Se il tempo morto totale risulta essere una larga frazione del tempo totale di funzionamento dell'esperimento, sarà quasi impossibile registrare un evento raro, e di conseguenza non si può stimare il flusso in corrispondenza delle alte energie. Minimizzare il tempo morto, e quindi di conseguenza massimizzare il tempo vivo, è perciò una richiesta comune per i sistemi di trigger degli esperimenti di raggi cosmici.

Come si nota dalla formula (2.7), un metodo efficace per ridurre il tempo morto è evitare l'acquisizione di eventi non buoni. Un evento non buono che venga correttamente identificato come tale dal sistema di trigger e conseguentemente non acquisito comporta un tempo morto che si limita a $DT_b = \tau_t$, che è tipicamente apprezzabilmente inferiore al tempo morto di un evento completamente acquisito. Risulta quindi evidente il beneficio di un sistema di trigger con un'alta capacità di discriminare eventi buoni e non buoni.

2.3.3 Accuratezza della stima

L'equazione (2.3) mette in evidenza che il flusso stimato è una grandezza fisica ottenuta mediante prodotto e divisione di altre misure sperimentali, alle quali è associata un'incertezza sperimentale. Sommando gli errori relativi di tutti i fattori presenti al terzo membro della (2.3), si ottiene quindi l'errore relativo nella misura del flusso, $\frac{\sigma_\Phi}{\Phi}$:

$$\frac{\sigma_\Phi}{\Phi} = \frac{\sigma_{N_i}}{N_i} + \frac{\sigma_{\Delta t}}{\Delta t} + \frac{\sigma_{E_{m_i}}}{E_{m_i}} + \frac{\sigma_\Omega}{\Omega} + \frac{\sigma_{\epsilon_i}}{\epsilon_i}\,, \tag{2.8}$$

dove σ indica l'errore assoluto sulla misura indicata nel pedice. Vediamo come incidono i vari addendi al secondo membro della (2.8) nella stima del flusso:

- $\frac{\sigma_{N_i}}{N_i}$: N_i è una variabile aleatoria con distribuzione di Poisson di parametro N_i; l'errore assoluto di questa misura è quindi lo scarto quadratico medio della variabile aleatoria, ovvero $\sqrt{N_i}$, di conseguenza l'errore relativo è $\frac{1}{\sqrt{N_i}}$; quando N_i è abbastanza grande, questo errore è circa 0 ed è quindi trascurabile;

- $\frac{\sigma_{\Delta t}}{\Delta t}$: Δt è il tempo vivo durante il quale lo strumento è stato in grado di rilevare il passaggio di raggi cosmici; tipicamente i tempi vivi vengo-

no misurati da orologi di bordo che hanno un'accuratezza molto alta, quindi l'errore relativo è trascurabile;

- $\frac{\sigma_{E_{m_i}}}{E_{m_i}}$: questo errore è nullo, in quanto l'ampiezza dell'intervallo è scelta a priori;

- $\frac{\sigma_\Omega}{\Omega}$: il parametro Ω viene in genere stimato, per esempio mediante simulazioni Monte Carlo, con grande accuratezza, risultante in un'incertezza dell'ordine del per mille (circa 0,1%) e quindi trascurabile;

- $\frac{\sigma_{\epsilon_i}}{\epsilon_i}$: questo è il fattore che incide maggiormente nell'errore della stima del flusso, dato che ingloba tutte le incertezze derivanti dalla selezione degli eventi. Attualmente l'obiettivo degli esperimenti di raggi cosmici nello spazio è di misurare i flussi con errori relativi dell'ordine del percento; dato che tipicamente σ_{ϵ_i} è compreso tra 0,01 e 0,1 ciò implica che ϵ_i debba essere il più possibile vicina ad 1.

2.4 Studio del sistema di trigger

La realizzazione di un sistema di trigger descritto nel paragrafo 2.3.2, che operi in uno stadio precedente a quello dell'acquisizione dei dati, implica che esso non può essere basato sui numeri prodotti dai convertitori analogico-digitali del sistema di acquisizione, i quali sono proporzionali ai segnali prodotti dai sensori e quindi in ultima istanza all'energia rilasciata dalla particella. Verosimilmente l'input di questo sistema potrà essere costituito da dei segnali quantizzati a due valori, cioè dei valori binari ad 1 bit associati al superamento di una data tensione di riferimento da parte del segnale prodotto dal sensore montato su un dato cubetto. Questi segnali binari vengono prodotti da componenti elettronici molto veloci detti discriminatori

a soglia, e risultano quindi adatti all'utilizzo entro tempi brevissimi al fine della eventuale generazione del trigger. Uno studio realistico di un algoritmo di trigger basato su reti neurali richiederebbe quindi di essere effettuato usando i segnali discriminati come input per la rete, e possibilmente effettuando anche una sparsificazione e/o un clustering a priori per ridurre la dimensione del dataset di input (e quindi il numero di linee elettriche e di conseguenza il consumo elettrico totale, che è un parametro fortemente vincolato in ambito spaziale). Tuttavia risulta verosimile che l'utilizzo di un dataset realistico (nel senso sopra descritto) riduca le prestazioni dell'algoritmo di discriminazione, oltre a renderne più difficoltosa l'ottimizzazione. In questo studio ci si basa inizialmente sui rilasci di energia prodotti in ogni singolo cubo e codificati come numeri reali, nell'ottica di stabilire dei riferimenti in termini di prestazioni che verosimilmente saranno dei limiti superiori per il caso realistico. Si considereranno poi dati di dimensione ridotta e con valori ad 1 bit che sono più aderenti al caso reale. I dati che verranno utilizzati sono stati prodotti mediante il toolkit GEANT4, un set di librerie scritte in C++ e comunemente usato in ambito fisico per simulare l'effetto del passaggio di particelle nella materia mediante metodi Monte Carlo.

Capitolo 3

Apprendimento supervisionato

Il problema dell'apprendimento supervisionato è il seguente: dato un insieme D detto dataset, formato da N coppie $(\underline{x}_1, \underline{y}_1), (\underline{x}_2, \underline{y}_2), \ldots, (\underline{x}_N, \underline{y}_N)$, e supponendo che queste siano generate da una distribuzione incognita $\mathbb{P}(X, Y)$, si vuole trovare una funzione $f : X \mapsto Y$ che approssimi bene la distribuzione condizionale $\mathbb{P}(Y|X)$.

Data una funzione $f : X \mapsto Y$ si può definire una funzione $L(f(X), Y)$, detta *loss*, che quantifichi la diversità tra $f(X)$ ed Y. L'obiettivo sarebbe quello di minimizzare il valore atteso della loss, ovvero si vorrebbe trovare $f^\star$ tale che:

$$f^\star = \arg\min_f \mathbb{E}[L(f(X), Y)]\,, \tag{3.1}$$

ma visto che non si conosce la distribuzione $\mathbb{P}(X, Y)$, non si può sapere il valore atteso della loss. Il massimo che si può fare è minimizzare la loss empirica, ovvero trovare $f^\star$ tale che:

$$f^\star = \arg\min_f \frac{1}{N} \sum_{(\underline{x},\underline{y})\in D} L(f(\underline{x}), \underline{y})\,. \tag{3.2}$$

Il dataset viene suddiviso in tre sottoinsiemi disgiunti: training set, validation set e test set.

I dati del training set vengono usati dall'algoritmo di apprendimento per adattare i valori dei parametri del modello. Talvolta, il modello è contraddistinto anche da alcuni iperparametri, che sono dei parametri il cui valore non può essere automaticamente appreso durante il normale processo di apprendimento, ma devono essere impostati prima dell'inizio dell'apprendimento.

Il validation set viene usato per avere una valutazione imparziale di un modello allenato sul training set, e può quindi essere usato per la messa a punto degli iperparametri. Si dice che il modello generalizza bene quando la loss calcolata su dati non usati per l'allenamento è simile alla loss calcolata sui dati del training set. Idealmente, si vorrebbe avere elevate capacità di classificazione sia sui dati di allenamento che su quelli di validazione, e che esse siano simili.

Il validation set dà una buona idea della capacità di generalizzare di un modello, ma non può fornire un'indicazione precisa sulle sue prestazioni perché influenza indirettamente il processo di apprendimento. Per una stima veritiera delle qualità di un modello, un insieme di dati separato detto test set viene usato. Nel caso in cui non si riesca a raggiungere una loss abbastanza bassa, si parla di *underfitting*, mentre quando c'è molta differenza tra le loss del test set e del traning set, si parla di *overfitting*.

L'apprendimento supervisionato viene usato per svolgere varie tipologie di compiti, tra cui la *classificazione*: nella classificazione gli input sono divisi in due o più classi e il modello deve essere capace di assegnare la giusta classe ai vari input. Un esempio di classificazione è il filtraggio anti-spam, dove gli input sono le email e le classi sono "spam" e "non spam".

3.1 Reti Neurali

Una rete neurale artificiale è un modello matematico che si ispira al funzionamento del cervello per apprendere a computare la funzione $f^\star$ definita nella 3.2. In un essere vivente, l'attività cerebrale consiste principalmente di attività elettro-chimica in reti di cellule cerebrali dette neuroni. Una rete neurale artificiale è composta da un insieme di unità interconnesse: le proprietà della rete sono determinate dalla sua topologia e dalle proprietà delle unità.

Una semplice esempio di singola unità è il modello matematico di un neurone, schematizzato in Figura 3.1.

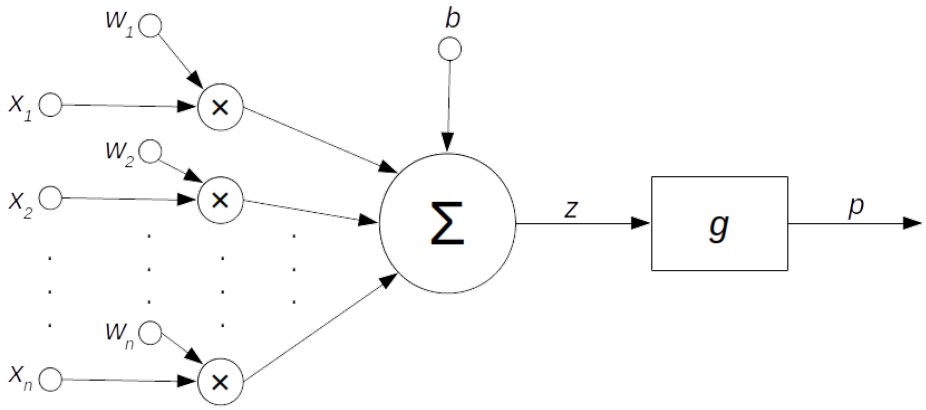

Figura 3.1: Schema del modello matematico del neurone artificiale, unità fondamentale delle reti neurali.

L'uscita p del neurone è data dalla seguente formula:

$$p = f(\underline{x}) = g(\underline{w}^T \underline{x} + b)\,, \tag{3.3}$$

dove:

- $\underline{x}$: vettore degli ingressi;

- $\underline{w}$: vettore dei pesi;
- b: bias;
- g: è la funzione di attivazione, che serve per rendere non lineare il comportamento dell'unità.

L'unità si comporta diversamente al variare di g, $\underline{w}$ e b. La funzione g viene scelta a priori, esistono varie possibilità nel fare questa scelta. Il vettore $\underline{w}$ ed il numero b sono parametri il cui valore viene determinato durante il processo di apprendimento. Si noti che il bias b di un neurone potrebbe essere integrato nel vettore dei pesi $\underline{w}$ del neurone stesso semplicemente associando a b un input fittizio sempre pari ad 1.

Il primo algoritmo basato su una rete neurale è stato il *percettrone* (*perceptron*), proposto da *Frank Rosenblatt* nel 1958 [5]: esso si basa su una sola unità del tipo raffigurato in Figura 3.1. Come funzione di attivazione del percettrone, si usa solitamente la funzione gradino di Heaviside: $g(z)$ vale 1 quando $z \geq 0$, altrimenti vale 0. Si può dire quindi che, a seconda dell'ingresso $\underline{x}$ e dei parametri $\underline{w}$ e b, il percettrone può "accendersi" o rimanere "spento": il processo di apprendimento serve per capire quali valori assegnare a $\underline{w}$ e b in modo che il percettrone si accenda oppurtunamente per distinguere la giusta classe del dato. I parametri vengono inizializzati casualmente ed aggiornati, a seconda dell'implementazione, talvolta con regole ad hoc, talvolta con il metodo più generale che si usa per qualsiasi rete neurale, che verrà descritto a breve.

Rete neurali più complesse si ottengono combinando tra loro più unità, spesso organizzate in layer: in un layer, le unità ricevono in input gli output (tutti o in parte) del layer precedente. Un layer le cui unità ricevono in ingresso tutti gli output del layer precedente è detto *denso* o *completamente*

connesso. Il primo layer della rete è detto di *input*, l'ultimo è detto di *output*, gli altri sono detti *hidden* (*nascosti*). Un esempio di rete neurale più complessa è riportato in Figura 3.2.

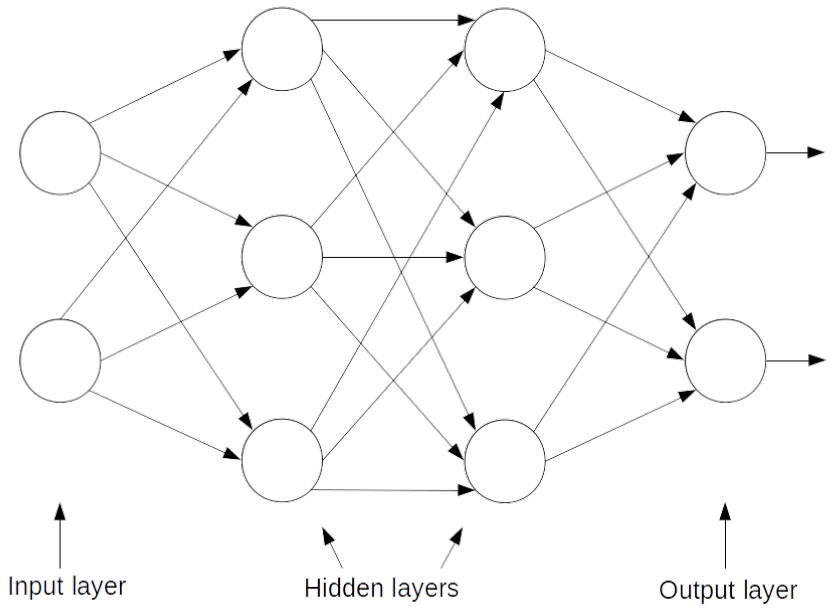

Figura 3.2: Esempio di rete neurale con 2 ingressi, 2 layer nascosti e 2 uscite.

Una rete neurale rappresenta una famiglia di funzioni $\mathcal{F}$ parametrizzata da un vettore $\underline{W}$ che contiene tutti i pesi ed i bias delle unità della rete. La rete, in generale, riceve in ingresso un array $\underline{x}$, computa una certa funzione f e fornisce in uscita un array $\underline{p}$, dato da: $\underline{p} = f(\underline{x}, \underline{W})$. Nel caso di un classificatore, solitamente si fa in modo che le componenti di $\underline{p}$ rappresentino le probabilità di appartenenza dell'input alle diverse classi[1].

[1]Per soddisfare questo intento, se il layer di output è composto da un solo neurone si usa per quest'ultimo la funzione di attivazione sigmoid, se invece il layer finale è composto da più neuroni, per ognuno di essi si usa la funzione di attivazione softmax.

Per una rete neurale si possono scegliere diversi tipi di loss, ad esempio la *cross-entropy loss*, definita nel seguente modo:

$$L(f(\underline{x}, \underline{W}), \underline{y}) = L(\underline{p}, \underline{y}) = -\sum_i y_i \log p_i \,, \tag{3.4}$$

dove $\underline{y}$, l'etichetta associata ad $\underline{x}$, è inteso che sia codificata secondo la codifica one-hot: è un vettore composto da un 1 in corrispondenza della classe giusta, e da 0 altrove. Con la notazione y_i, p_i si indica l'i-esimo elemento rispettivamente di $\underline{y}$ e $\underline{p}$. Si noti che questa loss ha come codominio l'intervallo $[0, +\infty)$: il valore 0 si ha quando il classificatore assegna una probabilità pari ad 1 alla classe corretta, in altre parole quando $\underline{y} = \underline{p}$.

Spesso accade che durante la fase di allenamento di un modello, questo diventi troppo complesso, con pesi di valore assoluto elevato: come conseguenza il modello sarà sensibile al rumore nei dati, presentando il problema dell'overfitting. Per ovviare a questo problema, nel secondo membro della 3.2 si aggiunge alla loss empirica un iperparametro λ moltiplicato per una funzione di regolarizzazione dei pesi, che sarà tanto più grande quanto più grandi saranno i parametri del modello. Per l'equazione 3.2 e per le considerazioni appena fatte, l'obiettivo diventa quello di trovare $\underline{W}^\star$ tale che:

$$\underline{W}^\star = \arg\min_{\underline{W}} \frac{1}{N} \sum_{(\underline{x},\underline{y}) \in D} L(f(\underline{x}, \underline{W}), \underline{y}) + \lambda R(\underline{W}) \,. \tag{3.5}$$

Si tenga di conto che non è necessario che tutti i gli elementi di $\underline{W}$ siano pesati con un unico iperparametro λ, ma possono essere usati dei λ_1, λ_2, ecc. differenti. Il processo di ottimizzazione, che avviene durante l'apprendimento, consiste nel trovare i valori dei parametri che minimizzano la loss. I parametri vengono generati inizialmente in modo casuale, esistono varie tecniche al riguardo. Dopodiché, per aggiornare i parametri si sfrutta

il gradiente della loss rispetto ai pesi $\frac{\partial L}{\partial \underline{W}}$, che viene calcolato con un metodo detto *backpropagation*. L'algoritmo che sfrutta il gradiente per calcolare come aggiornare i pesi in modo da arrivare a $\underline{W}^\star$ è detto *ottimizzatore*. Ci sono diversi tipi ottimizzatori, tutti basati sulla *discesa del gradiente*, che consiste nell'aggiornare i pesi ad ogni passo nel seguente modo:

$$\underline{W} \leftarrow \underline{W} - l_r \frac{\partial L}{\partial \underline{W}}\,, \tag{3.6}$$

dove l_r è un iperparametro detto *learning rate* (tasso di apprendimento): se l_r è troppo basso l'apprendimento risulta molto lento, se l_r è troppo alto non si riesce a raggiungere il minimo della loss.

In generale, è possibile inserire come unità di calcolo nella rete neurale qualsiasi tipo di funzione, a patto che sia differenziabile, condizione necessaria per la propagazione del gradiente nella rete senza la quale non si può applicare la backpropagation e il conseguente aggiornamento dei pesi. Nelle reti odierne si trovano unità di calcolo di svariati tipi, tra cui:

- filtri convoluzionali: sfruttano l'operazione di convoluzione per estrarre caratteristiche da informazioni contenute in zone spazialmente vicine; verranno spiegati nei prossimi paragrafi;
- pooling: riassumono informazioni spazialmente vicine in un unico valore, che può essere la media o il massimo;
- dropout: durante la fase di apprendimento, spegne casualmente alcune parti della rete in modo che essa si irrobustisca; se ne trae un vantaggio in termini di capacità di generalizzare a dati non usati per l'allenamento [6].

3.2 Reti Neurali Convoluzionali

Una rete neurale convoluzionale (CNN, Convolutional Neural Network) è una rete neurale dove una parte delle unità di calcolo svolgono operazioni di convoluzione.

3.2.1 Operazione di convoluzione

Date due funzioni $f, g : \mathbb{R} \to \mathbb{R}$. Si definisce convoluzione di f e g la funzione $h : \mathbb{R} \to \mathbb{R}$ definita nel seguente modo:

$$h(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt = \int_{-\infty}^{\infty} f(x-t)g(t)\,dt\,. \tag{3.7}$$

Nel caso in cui il dominio sia discreto, ovvero $f, g : \mathbb{Z} \to \mathbb{R}$, allora l'integrale degenera in una sommatoria e la convoluzione $h : \mathbb{Z} \to \mathbb{R}$ diventa:

$$h(n) = (f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)\,. \tag{3.8}$$

Si può generalizzare l'operazione di convoluzione, considerando $f, g, h : \mathbb{Z}^k \to \mathbb{R}^k$, ottenendo:

$$\begin{aligned} &h(n_1, n_2, \ldots, n_k) = (f * g)(n_1, n_2, \ldots, n_k) = \\ &= \sum_{m_1=-\infty}^{\infty} \ldots \sum_{m_k=-\infty}^{\infty} f(m_1, \ldots, m_k)g(n_1 - m_1, \ldots, n_k - m_k) = \\ &= \sum_{m_1=-\infty}^{\infty} \ldots \sum_{m_k=-\infty}^{\infty} f(n_1 - m_1, \ldots, n_k - m_k)g(m_1, \ldots, m_k)\,. \end{aligned} \tag{3.9}$$

3.2.2 Convoluzione all'interno di reti neurali

Reti neurali che sfruttano l'operazione di convoluzione sono attualmente gli algoritmi di riferimento in svariate tipologie di problemi, ad esempio: riconoscimento facciale [7], classificazione di video [8], riconoscimento di oggetti in tempo reale [9]. Il motivo del successo della convoluzione sta nel fatto che, se sfruttata opportunamente, questa operazione aiuta a riassumere informazioni spazialmente vicine nei dati, con l'esito di riconoscere determinati pattern.

L'operazione di convoluzione viene sfruttata aggiungendo alla rete dei layer composti da un certo numero di filtri, ognuno con i propri parametri e della stessa dimensione degli altri filtri dello stesso layer: a seconda del valore appreso da questi parametri, ogni filtro diventa capace di rilevare determinate caratteristiche sui dati.

La convoluzione fa leva su delle importanti idee che possono migliorare un algoritmo di apprendimento [10]:

- interazioni sparse: i collegamenti tra gli elementi della rete sono inferiori al massimo possibile, con il risultato di avere meno parametri, quindi meno memoria utilizzata e una migliore efficienza statistica;
- condivisione dei parametri: i parametri sono in comune a più funzioni del modello; ogni parametro del filtro viene usato in quasi ogni posizione dell'input, quindi i parametri memorizzati sono molti meno rispetto al caso in cui avessimo un insieme di parametri diverso per ogni posizione.

Data una convoluzione $h = f * g$ ed interpretando f, g ed h come segnali, se si applicasse la definizione matematica di convoluzione la lunghezza del segnale h sarebbe maggiore della lunghezza del segnale f, con lunghezza intesa come numero di elementi diversi da 0. Questo aumento non è molto

desiderabile, visto che se ad un certo segnale in ingresso venissero applicati diversi filtri in sequenza, si otterrebbe in uscita un segnale molto più lungo rispetto a quello in ingresso.

Esempio 1 *Supponiamo di avere un segnale* $f : \mathbb{Z} \to \mathbb{R}$*, che assume valori diversi da* 0 *soltanto nell'intervallo* $[0, 10]$*, e di avere un segnale* $g : \{0, 1, 2\} \to \mathbb{R}$ *che opera da filtro.*
La convoluzione $h = f * g$*, secondo la definizione matematica, può assumere valori strettamente positivi soltanto nell'intervallo* $[0, 12]$*, che è composto da* 13 *elementi,* 2 *in più degli elementi diversi da* 0 *del segnale originale* f*.*

Considerando che solitamente l'informazione sui "bordi" non è così importante, nelle reti neurali si considerano spesso convoluzioni modificate, che hanno un risultato con stessa lunghezza dell'ingresso, o addirittura inferiore se si considerano soltanto i valori ottenuti da una completa "sovrapposizione" del filtro sull'ingresso; questi due tipi di convoluzione vengono chiamati rispettivamente *same* e *valid*.

In generale, una rete neurale può avere come input anche vettori multidimensionali, ad esempio matrici (2D) e tensori (3D): in tali casi, un layer convoluzionale esegue rispettivamente convoluzioni 2D e 3D. Inoltre, talvolta il dato è suddiviso in vari canali, ad esempio una immagine viene suddivisa nei tre canali RGB. Quindi, nel caso di input bidimensionale, si avrà che $\underline{x} \in \mathbb{R}^{w \times h \times c}$, mentre nel caso di input tridimensionale, si avrà che $\underline{x} \in \mathbb{R}^{w \times h \times d \times c}$.

L'operazione di convoluzione può essere intuitivamente vista come il risultato dello scorrere del filtro lungo il segnale di ingresso, nelle varie direzioni di esso: ad ogni posizionamento del filtro corrisponde un valore di output, calcolato come sommatoria dei prodotti dei singoli elementi del filtro per i

rispettivi valori dell'input. Un esempio di convoluzione bidimensionale, viene mostrato in Figura 3.3. Il caso di convoluzione tridimensionale è molto simile a quello bidimensionale, le differenze sono che l'input ha ovviamente anche una profondità, e i filtri hanno una dimensione in più.

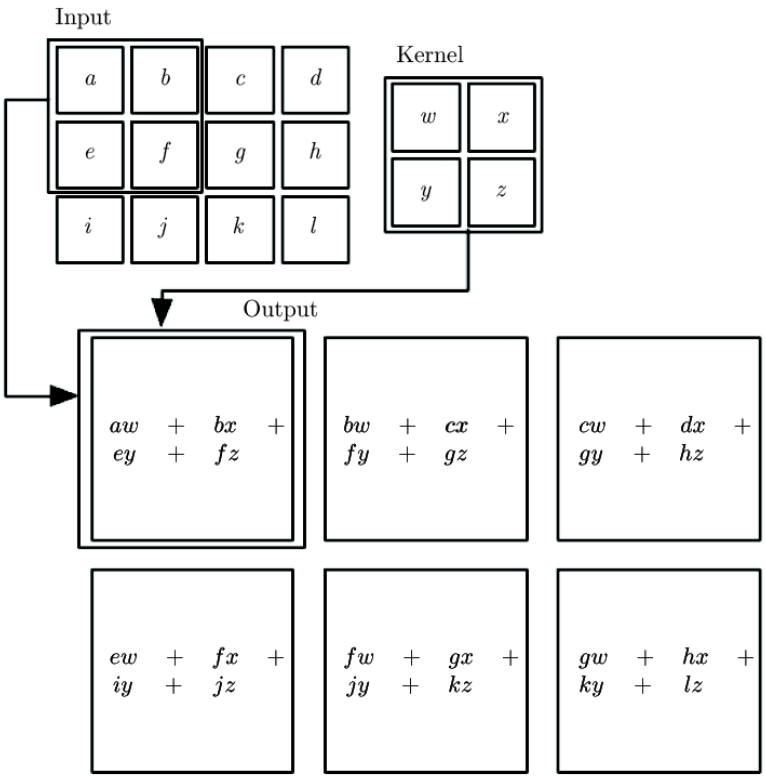

Figura 3.3: Esempio di convoluzione 2D, con il filtro già considerato capovolto (per via del segno meno nella formula della convoluzione). Si limita l'output alle sole posizioni dove il filtro giace interamente nell'immagine, corrispondente alla cosiddetta convoluzione "valida". Si usano caselle con frecce per indicare come l'elemento in alto a sinistra della matrice di uscita è ottenuto applicando il filtro sulla corrispondente regione in alto a sinistra della matrice di input [10]

Quando si definisce un layer convoluzionale, il numero di canali dei filtri è imposto dai layer dell'input, devono infatti essere uguali. Invece, le dimensioni dei filtri ed il numero di essi sono iperparametri; con il numero dei

filtri coincideranno il numero di canali dell'output del layer. La dimensione dell'output dipendono sia dalle dimensioni dei filtri, sia dalla modalità della convoluzione (same/valid), ma anche dallo *stride*, che è un ulteriore iperparametro che indica il passo che intercorre tra una sovrapposizione e l'altra del filtro.

Solitamente, alle uscite dei layer convoluzionali vengono applicate delle funzioni di attivazione, alcuni esempi sono riportati in Figura 3.4. I layer convoluzionali vengono principalmente posti nella parte iniziale della rete, vista la loro capacità di estrarre caratteristiche significative dai dati; nella parte finale della rete invece, si appiattisce prima l'input per renderlo monodimensionale, e successivamente si utilizzano dei livelli densi (completamente connessi), fino ad arrivare alla funzione di attivazione finale che precede l'uscita della rete neurale.

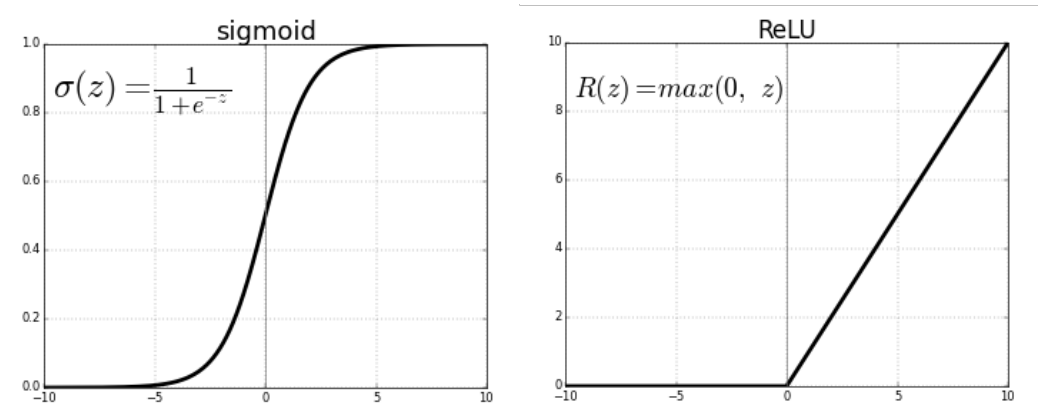

Figura 3.4: Esempio di funzioni di attivazione. La funzione *sigmoid* viene solitamente usata in fondo alla rete neurale in modo da rendere l'output compreso tra 0 ed 1. La funzione *ReLU* [11] è una delle funzioni più comunemente utilizzate in uscita dai layer nascosti.

Capitolo 4

Metodi

4.1 Dataset

Tutti i dati usati in questo studio sono stati creati e forniti dal laboratorio dell'*Istituto Nazionale di Fisica Nucleare* (INFN). I dataset in questione sono due, uno per gli elettroni e uno per i protoni, e sono stati generati mediante il toolkit di simulazione GEANT4. Entrambi i dataset sono formati da coppie $(\underline{x}, y)$ dove $\underline{x} \in \mathbb{R}^{20\times20\times20}$, $y \in \{0, 1\}$. Nella simulazione si ha accesso a dei parametri che non sarebbero disponibili se si creasse il dataset con le rilevazioni del calorimetro reale: grazie a questi parametri aggiuntivi si riesce a generare la supervisione y per ogni tensore $\underline{x}$, ottenendo così il dataset da usare per l'apprendimento supervisionato. Il dataset degli elettroni contiene 1107343 esempi, di cui l'88.4% sono etichettati come positivi; il numero di esempi di eventi di protoni sono invece 414488, di cui il 41.9% è etichettato positivamente. Queste percentuali di dati buoni, per la natura della simulazione che ha originato i dati, sono proprio le percentuali stimate di dati buoni acquisiti da HERD nello spazio. Per questo motivo, i vari algoritmi verranno valutati su dei test set composti da queste percentuali di dati, non viene

eseguito alcun bilanciamento dei dati. Nella prima fase si usano i dataset così come sono stati forniti, ovvero con numeri reali e tensori di dimensione $20 \times 20 \times 20$. Successivamente si provano ad usare dei dati più verosimili a quelli reali: si provano tensori di dimensione ridotta, tensori a valori binari, tensori di dimensione ridotta a valori binari.

4.2 Dettagli su CNN utilizzate

Nel problema trattato in questo studio, i dati a disposizione sono assimilabili ad immagini tridimensionali, si pensa quindi che alcune CNN possano funzionare bene. Vista la natura tridimensionale dei tensori, essi potrebbero essere interpretati da una rete neurale come elementi tridimensionali a canale singolo, oppure come elementi bidimensionali con 20 canali. Scegliere come interpretare i tensori potrebbe portare a risultati differenti: visto che non ci sono evidenti motivi teorici per preferire una delle due scelte, verranno testate reti che interpretano i tensori nei due modi differenti. Nel caso di interpretazione bidimensionale degli elementi, per il motivo che il moto delle particelle avviene prevalentemente lungo l'asse z. Per entrambe le metodologie, negli esperimenti vengono provate varie CNN, che differiscono sia per struttura che per iperparametri.

Per tutte le CNN utilizzate, si usa come loss la funzione *cross entropy* definita nella 3.4. Come funzione di regolarizzazione si usa la sommatoria dei quadrati dei singoli pesi (regolarizzazione L2). Come ottimizzatore si usa *Adam* [12], che combina idee alla base di altri ottimizzatori:

- eseguire un passo (nello spazio dei pesi) nella direzione opposta al gradiente, moltiplicato per l'iperparametro *learning rate*;

- sommare a questo effetto un termine che sia proporzionale al precedente aggiornamento, in analogia con la legge fisica della conservazione della quantità di moto;
- limitare i contributi degli aggiornamenti nelle direzioni in cui il gradiente era stato precedentemente elevato in modulo.

L'allenamento viene eseguito, per entrambi i tipi di particelle, pesando opportunamente gli esempi delle diverse classi, in modo da compensare gli sbilanciamenti dei dataset.

4.3 Base di partenza

Non si conosce la difficoltà di questo problema di classificazione: per questo motivo, come possibili classificatori si provano, oltre alle reti neurali, anche algoritmi più semplici, che operano cercando di sfruttare delle intuizioni originate dalla natura del problema.

4.3.1 Algoritmo baseline

Se un evento è buono, significa che l'energia della particella si è in buona parte depositata all'interno del calorimetro: per questo motivo è ragionevole pensare che, in genere, negli eventi buoni ci sia un maggior rilascio di energia rispetto agli eventi non buoni. L'algoritmo baseline considera esclusivamente l'energia totale ΔE rilasciata da un evento per eseguire la classificazione, che sarà positiva quando il totale dell'energia supera una certa soglia t. Questa soglia chiaramente non si conosce a priori, si vuole vedere come si comporta l'algoritmo al variare della soglia. Il calcolo dell'energia totale rilasciata avviene sommando le energie rilevate nei cubetti:

$$\Delta E = \sum_{i=0}^{19} \sum_{j=0}^{19} \sum_{k=0}^{19} x_{ijk} \,. \tag{4.1}$$

La funzione f computata dall'algoritmo baseline per fare la classificazione sarà quindi:

$$f(\underline{x}, t) = \begin{cases} 1, & \text{se } \Delta E \geq t, \\ 0, & \text{altrimenti.} \end{cases} \tag{4.2}$$

Dove 1 indica che il dato $\underline{x}$ è stato classificato come appartenente alla classe positiva, lo 0 indica che la classificazione è negativa.

4.3.2 Variante algoritmo baseline

Un punto debole dell'algoritmo baseline è che non può distinguere il modo in cui l'energia è stata rilasciata. Per ovviare a questo problema si può sfruttare l'idea che lo sciame deve essere ben contenuto: considerando due eventi di uguale energia, l'evento più interessante tra i due dovrebbe essere quello che ha coinvolto una regione più ampia del calorimetro. Si definisce una variante dell'algoritmo baseline considerando una somma ΔE_{sqrt} delle radice quadrate delle energie rilasciate nei vari cubetti: lo scopo della radice quadrata è quello di ridurre il contributo nella somma degli addendi di modulo elevato. La somma ΔE_{sqrt} è data da:

$$\Delta E_{sqrt} = \sum_{i=0}^{19} \sum_{j=0}^{19} \sum_{k=0}^{19} \sqrt{x_{ijk}} \,. \tag{4.3}$$

La funzione f_{sqrt} computata dalla variante dell'algoritmo baseline per fare la classificazione sarà quindi:

$$f_{sqrt}(\underline{x}, t) = \begin{cases} 1, & \text{se } \Delta E_{sqrt} \geq t, \\ 0, & \text{altrimenti.} \end{cases} \tag{4.4}$$

4.4 Criteri di valutazione

Si indicano come positivi i dati buoni, e come negativi i dati non buoni. In base alla vera classe del dato ed alla classificazione fatta dall'algoritmo, si distinguono quattro gruppi di dati: veri positivi (TP), veri negativi (TF), falsi positivi (FP), falsi negativi (FN).

Una misura molto usata per la valutazione di un classificatore è l'accuratezza, definita come il numero di dati classificati correttamente diviso il numero dei dati in totale:

$$accuratezza = \frac{TP + TN}{TP + TN + FP + FN}\,. \tag{4.5}$$

Per gli scopi di questo studio, gli esperti suggeriscono che per valutare la qualità dei classificatori che verranno provati, sono più interessanti le misure di *precision* e *recall*.
Si definisce precisione (precision) il numero di dati realmente positivi classificati correttamente diviso il numero totale di dati classificati positivamente:

$$precision = \frac{TP}{TP + FP}\,. \tag{4.6}$$

Si definisce recupero (recall) il numero di dati realmente positivi classificati correttamente diviso il numero totale di dati realmente positivi:

$$recall = \frac{TP}{TP + FN}\,. \tag{4.7}$$

Si noti che la *recall* definita in (4.7) coincide con l'efficienza di selezione ϵ definita in (2.4).

Si noti anche che precision e recall sono in tradeoff, è sempre possibile aumentare la recall a discapito della precision.

Visto che in questo studio si tratta una versione semplificata del problema reale, è importante raggiungere performance elevate nella classificazione: gli esperti suggeriscono che le prestazioni possono essere considerate sufficienti se si raggiunge, sia $precision \geq 90\%$ che $recall \geq 90\%$. Non avvicinarsi a questi due valori, significherebbe che non ci sono margini per ottenere in futuro un buon classificatore che operi in real-time; se si riuscisse a superare queste soglie, si preferirà avere una recall più alta possibile, in modo da rendere maggiormente accurata la stima del flusso, considerando quanto detto nel paragrafo 2.3.3. Per gli elettroni in particolare, vista la sproporzione nei dati, si vorrebbe avere una recall prossima al 100%, ed una precision ben più alta del 90%.

4.4.1 Curva precision recall

In fase di test, dato un esempio in input ad un classificatore, esso restituisce un numero reale compreso tra 0 ed 1, da intendere come probabilità che l'esempio appartenga alla classe dei dati buoni. Nelle reti neurali un output di questo tipo si ottiene ponendo la funzione di attivazione sigmoid prima dell'uscita, mentre nelle baseline si ottiene normalizzando la somma dividendo per il valore più grande che la somma può avere. In questo problema di selezione dati, è necessario decidere l'azione da intraprendere in seguito ad un evento, che può alternativamente essere il salvataggio del rilascio di energia, oppure lo scarto di esso. L'output di un classificatore, essendo un numero reale compreso tra 0 ed 1, deve quindi essere trasformato in un numero bina-

rio, applicando ad esso una soglia. La scelta della soglia è importante perché al variare di essa il classificatore ha prestazioni diverse, misurate in termini di precision e recall. Al variare della soglia si ottengono una serie di coppie (precision, recall) che possono essere usate per ottenere una curva precision recall: stabilire il punto di lavoro lungo la curva consiste nello scegliere la soglia corrispondente ai valori di precision e recall desiderati.

Una volta calcolati i punti che andranno a formare la curva precision recall, solitamente la curva viene sommarizzata con un unico valore che indica l'area sotto la curva (AUC, Area Under Curve), che sarà tanto più alto quanto in generale è migliore il modello: verrà calcolato e mostrato questo valore per ogni algoritmo testato. In questa tesi si è particolarmente interessati ad avere sia precision che recall maggiori del 90%, e ad avere una recall più alta possibile: per questi motivi verrà considerato come indicatore della bontà di un classificatore, il punto di lavoro (OP, Operating Point) sulla curva precision recall che soddisfa i vincoli imposti e minimizza la distanza d definita dalla seguente formula:

$$d = 3 - 2recall - precision\,. \tag{4.8}$$

Si noti che nella 4.8, la *recall* incide maggiormente della *precision* nella diminuzione della distanza. Se non esiste un OP che soddisfi i requisiti, verrà comunque riportato l'OP che minimizza la distanza definita in 4.8. Chiaramente, in ogni caso, tale ipotetico punto di lavoro del classificatore è puramente indicativo: vengono riportati su un file tutte le possibili triplette (soglia, precision, recall), e la decisione del punto di lavoro del reale classificatore verrà fatta dai fisici nel modo che ritengono più opportuno.

4.4.2 Confronti

Verranno confrontate le curve precision-recall, sia per elettroni che per protoni, dei seguenti algoritmi: baseline, baseline con variante, perceptron, rete neurale con convoluzioni in 2D, rete neurale con convoluzioni in 3D. Si prova anche perceptron perché la difficoltà del problema non si conosce e può darsi che i dati siano linearmente separabili[1]: in tal caso, perceptron è un algoritmo molto semplice capace di apprendere un iperpiano che separi i dati. Vista la varietà infinita di reti neurali che sarebbe possibile definire, per i confronti vengono scelte le reti neurali che meglio si sono comportate sul validation set, non vengono invece riportate le reti neurali che ottenevano risultati peggiori.

Per quanto riguarda i due algoritmi baseline, visto che la loro classificazione si basa su un singolo numero, è possibile rappresentare facilmente la distribuzione delle somme (con e senza radici quadrate) con un grafico, dal quale se ne può dedurre le capacità di classificazione: si riporteranno quindi queste distribuzioni.

Successivamente, si esegue un confronto delle curve dei cinque algoritmi elencati in precedenza, usando dataset con tensori a dimensione ridotta $10 \times 10 \times 10$ ottenuti applicando un average pooling.

Inoltre, un confronto analogo si effettua su dati binarizzati, sia a dimensione completa che a dimensione ridotta: la soglia di binarizzazione, che non è detto sia la stessa per tutti gli algoritmi, viene prima scelta valutando l'area sotto la curva PR al variare del valore di soglia; per la valutazione della soglia, le reti neurali vengono allenate velocemente.

[1] Dati degli esempi suddivisi in due classi ed uno spazio a cui essi appartengono, questi esempi sono linearmente separabili se esiste un iperpiano che divide gli esempi di una classe da quelli dell'altra.

Capitolo 5

Risultati

5.1 Esperimenti

Gli algoritmi vengono testati con il linguaggio di programmazione Python, versione 3: la conversione in hardware di uno qualsiasi di questi algoritmi, necessaria per l'esecuzione a bordo dell'esperimento HERD, avverrà successivamente. Per quanto riguarda le reti neurali, si utilizza la libreria open source *Keras* [13], che astrae ad alto livello il sottostante framework di machine learning *TensorFlow* [14], anch'esso open source. Si riportano di seguito i risultati degli esperimenti, mostrando prima tutti quelli degli elettroni e successivamente tutti quelli dei protoni.

5.1.1 Elettroni

Dati non modificati

Si riportano in Figura 5.1 i risultati dei cinque diversi algoritmi, ottenuti sul dataset non modificato.

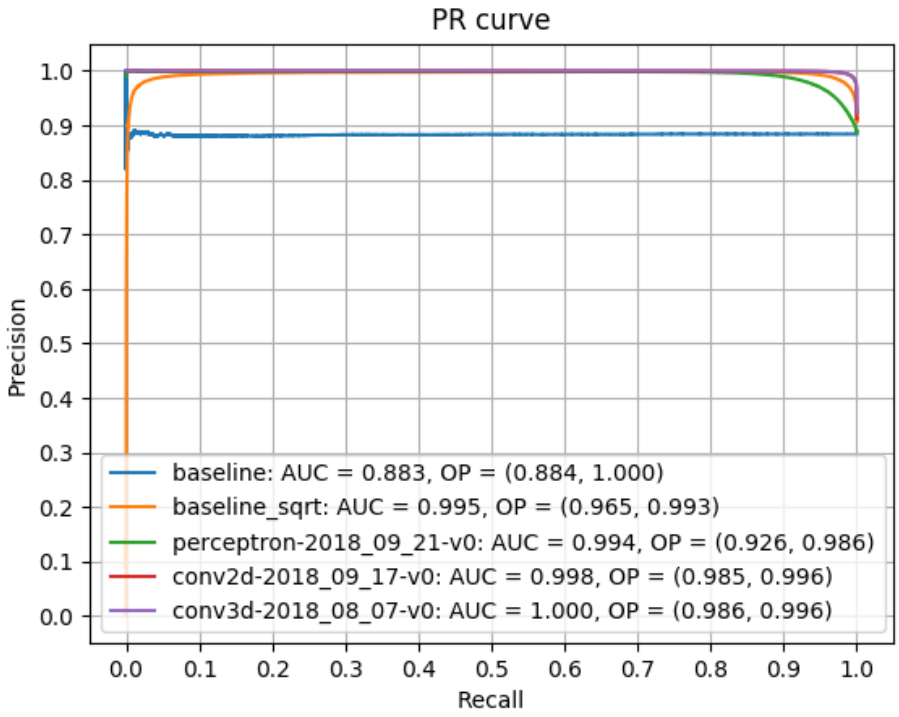

Figura 5.1: Confronto delle curve precision-recall dei 5 diversi algoritmi, calcolate su tensori $20 \times 20 \times 20$ a valori reali. Considerando che la percentuale di esempi buoni è dell'88.4%, l'algoritmo baseline non riesce minimamente ad eseguire una classificazione interessante. Gli altri algoritmi ottengono tutti dei risultati migliori, nel seguente ordine decrescente: CNN 3D, CNN 2D, variante baseline, perceptron.

Distribuzioni di probabilità delle somme

Si riportano in Figura 5.2 le distribuzioni di probabilità delle somme per gli elettroni.

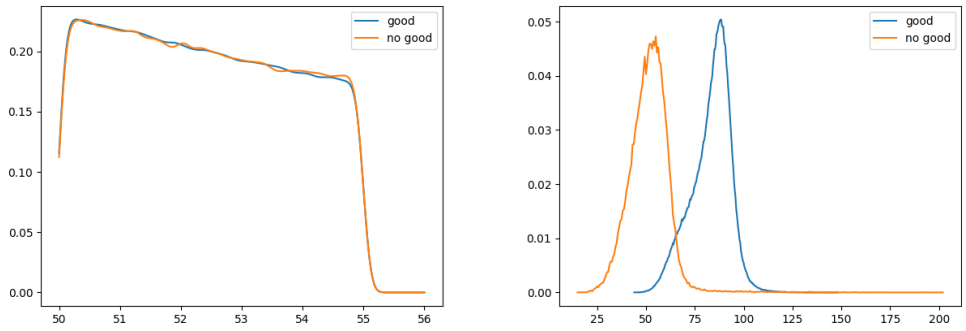

Figura 5.2: Distribuzioni di probabilità delle somme per gli elettroni. A sinistra la distribuzione delle somme degli elementi, a destra la distribuzione delle somme delle radici quadrate degli elementi, in blu la distribuzione di elementi buoni, in giallo quella degli elementi non buoni. La classificazione delle due baseline consiste nell'individuare un valore della somma oltre il quale l'esempio viene considerato buono: dalle distribuzioni in questa figura si evince il motivo per cui soltanto la variante della baseline riesce ad eseguire una buona classificazione.

Dati ridimensionati con average pooling

Si riportano in Figura 5.3 i risultati dei cinque diversi algoritmi calcolati sul dataset con esempi di dimensione ridotta, ottenuto applicando un average pooling agli esempi originali.

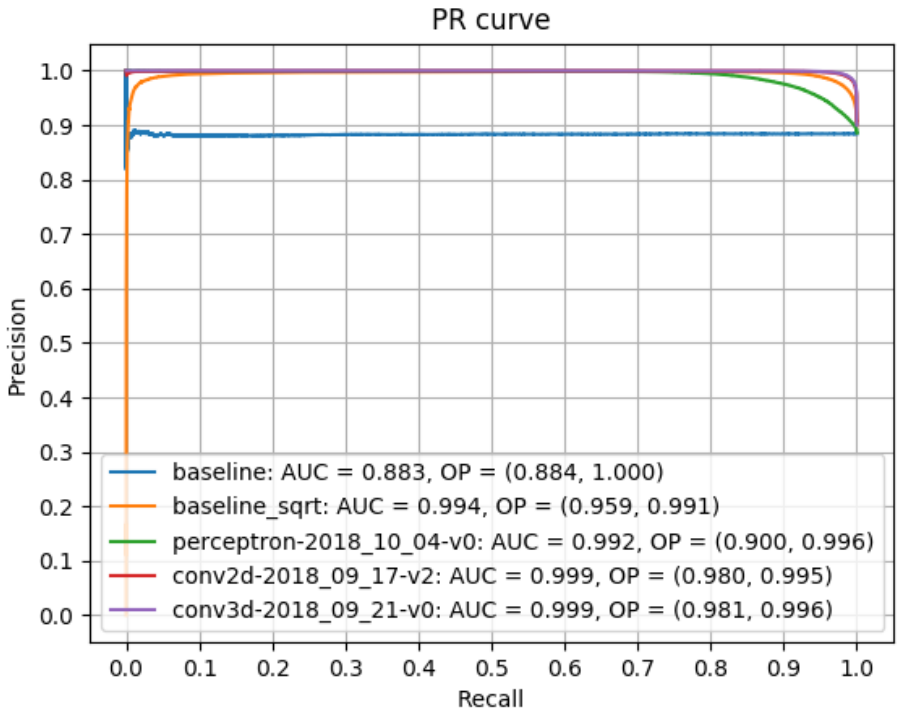

Figura 5.3: Confronto delle curve precision-recall dei 5 diversi algoritmi, calcolate su tensori $10 \times 10 \times 10$ a valori reali, ottenuti mediante un average pooling a blocchi $2 \times 2 \times 2$. Si nota che queste curve sono praticamente identiche a quelle ottenute sui tensori di dimensione $20 \times 20 \times 20$, riportate in Figura 5.1.

Soglie per binarizzazione

Si riportano in Figura 5.4 gli andamenti delle aree sotto la curva precision recall, al variare delle soglie di binarizzazione, per i vari algoritmi (la variante della baseline viene esclusa in quanto eseguire la radice quadrata di uno 0 o di un 1 non cambia il valore).

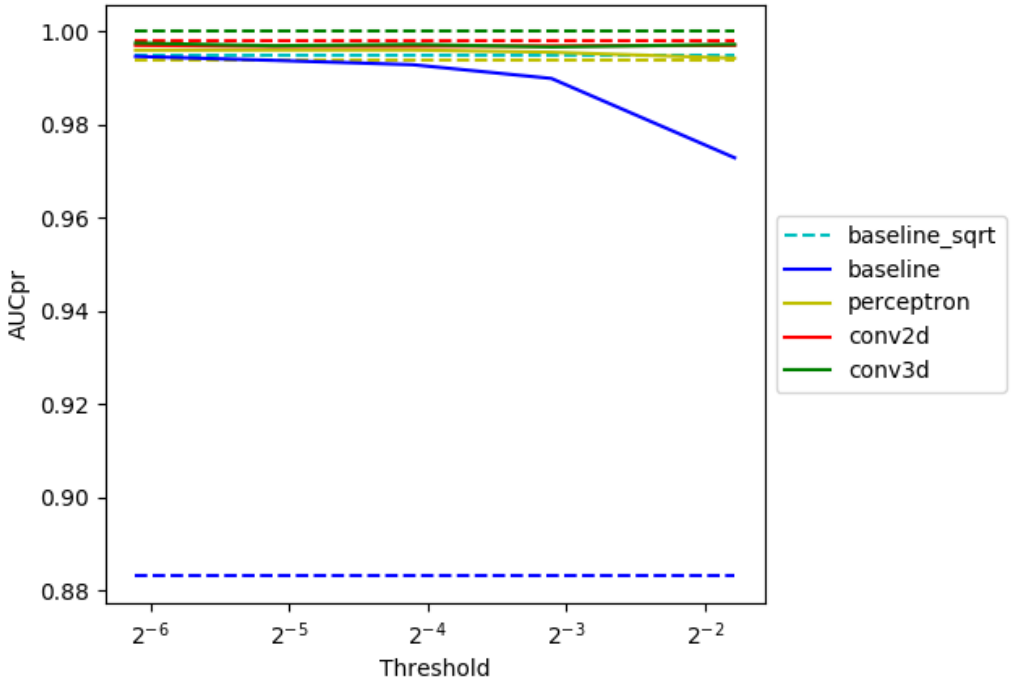

Figura 5.4: Area sotto la curva precision recall per i diversi algoritmi, al variare della soglia di binarizzazione, per gli elettroni. Le linee tratteggiate indicano il valore della metrica con dati non binarizzati. Le CNN registrano con la binarizzazione un lieve calo indipendente dalla soglia; perceptron con la soglia migliora rispetto al riferimento, con soglie grandi inizia a peggiorare. La baseline migliora moltissimo rispetto al riferimento, calando lievemente all'aumentare della soglia: questo miglioramento non è del tutto inaspettato, in quanto con la giusta binarizzazione si ottiene un effetto simile al fare le radici quadrate agli elementi.

Dati binarizzati

In base ai valori dell'area sotto la curva in funzione della soglia, viene scelto il valore 0.058 come soglia da applicare ai dati. Si riportano in Figura 5.5 le curve precision-recall relative ai 4 algoritmi, calcolate sui dati binarizzati.

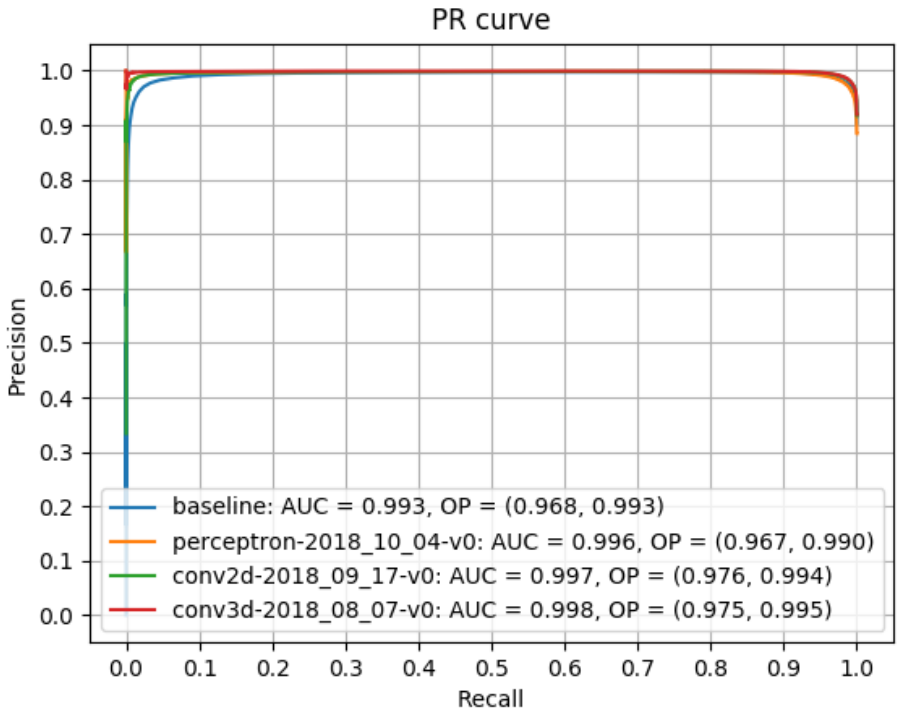

Figura 5.5: Confronto delle curve precision-recall dei 4 diversi algoritmi, ottenute su tensori $20 \times 20 \times 20$ a valori binari. Tutti gli algoritmi ottengono dei risultati eccellenti.

Soglie per binarizzazione dopo average pooling

Si riportano in Figura 5.6 gli andamenti delle aree sotto la curva precision recall, al variare delle soglie di binarizzazione, per i vari algoritmi (la variante della baseline viene esclusa in quanto eseguire la radice quadrata di uno 0 o di un 1 non cambia il valore). Ai dati da binarizzare è stato prima applicato un average pooling.

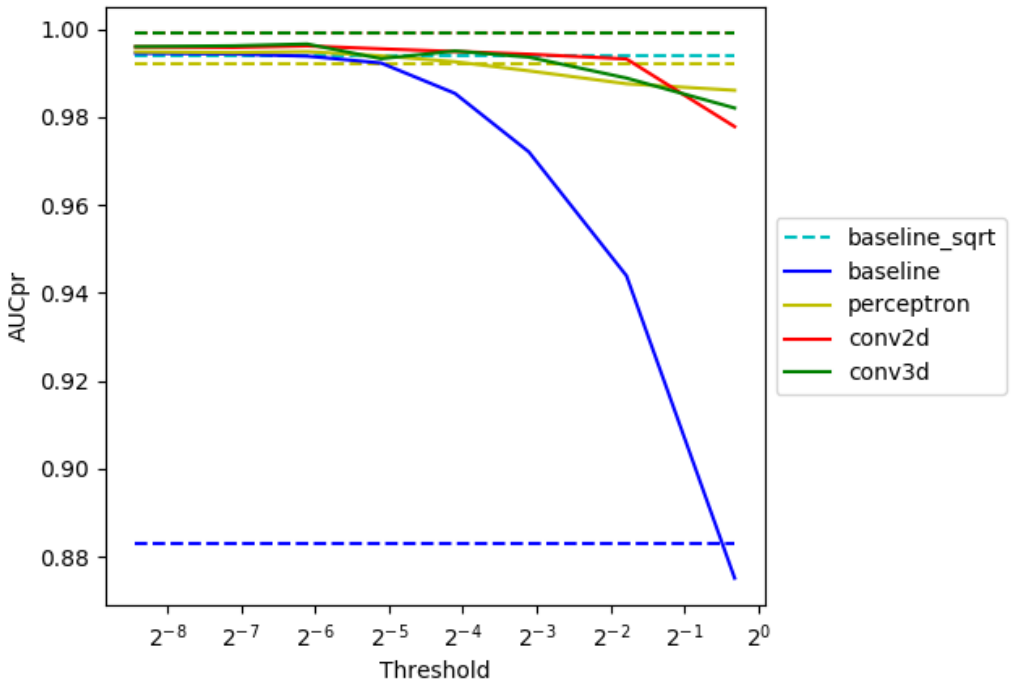

Figura 5.6: Area sotto la curva precision recall per i diversi algoritmi, al variare della soglia di binarizzazione, per gli elettroni, su tensori $10 \times 10 \times 10$ ottenuti con un average pooling a blocchi $2 \times 2 \times 2$. Le linee tratteggiate indicano il valore della metrica con dati non binarizzati. Per valori piccoli della soglia, tutti gli algoritmi si comportano bene e molto similmente; oltre una certa soglia tutti gli algoritmi iniziano a comportarsi via via sempre peggio.

Dati di dimensione ridotta binarizzati

In base ai valori dell'area sotto la curva in funzione della soglia, viene scelto il valore 0.00725 come soglia da applicare ai dati usati da CNN 2D e baseline; i dati usati da CNN 3D e perceptron sono ottenuti usando 0.0145 come soglia. Si riportano in Figura 5.7 le curve precision-recall relative ai 4 algoritmi, calcolate sui dati a cui è stato prima applicato un average pooling e poi binarizzati.

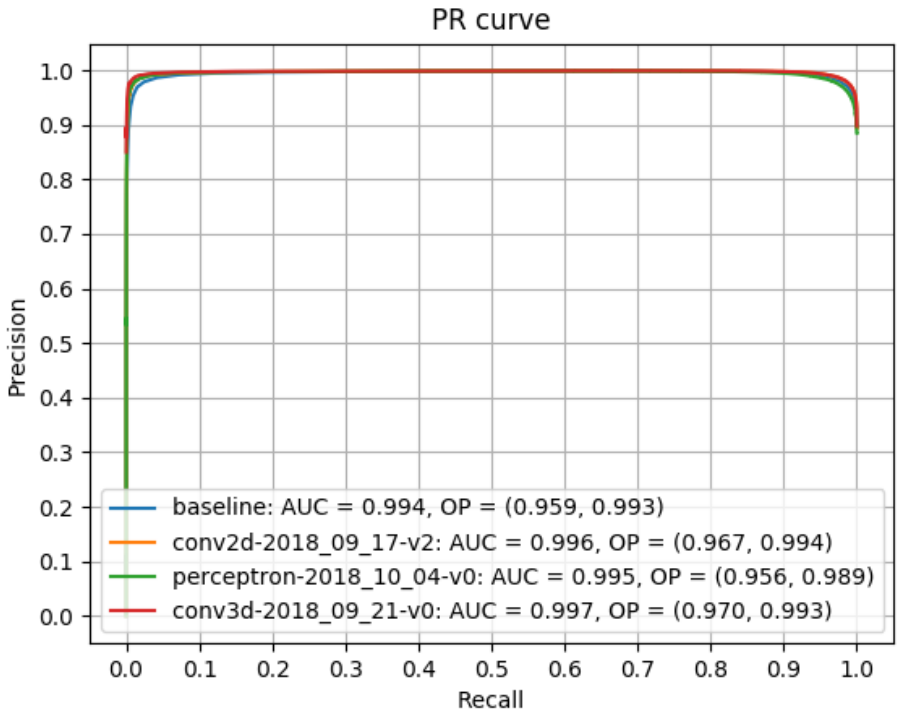

Figura 5.7: Confronto delle curve precision-recall dei 4 diversi algoritmi, ottenute su tensori $10 \times 10 \times 10$ a valori binari. Tutti gli algoritmi si comportano in modo eccellente.

5.1.2 Protoni

Dati non modificati

Si riportano in Figura 5.8 i risultati dei cinque diversi algoritmi, ottenuti sul dataset non modificato.

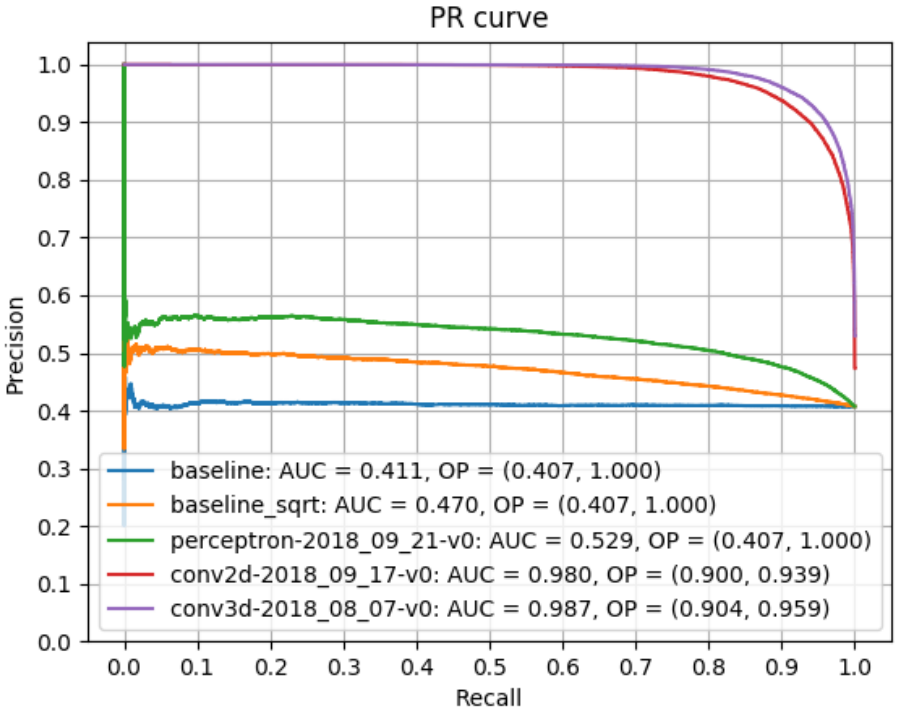

Figura 5.8: Confronto delle curve precision-recall dei 5 diversi algoritmi, calcolate su tensori $20 \times 20 \times 20$ a valori reali. Considerando che la percentuale di esempi buoni è del 41.9%, l'algoritmo baseline non riesce minimamente ad eseguire una classificazione interessante; anche la variante della baseline e perceptron ottengono dei risultati completamente insufficienti. Le CNN invece ottengono entrambe dei risultati soddisfacenti, la versione 3D è leggermente migliore.

Distribuzioni di probabilità delle somme

Si riportano in Figura 5.9 le distribuzioni di probabilità delle somme per i protoni.

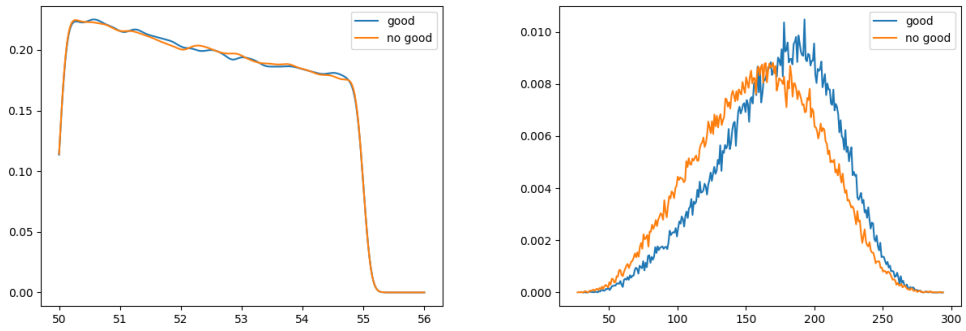

Figura 5.9: Distribuzioni di probabilità delle somme per i protoni. A sinistra la distribuzione delle somme degli elementi, a destra la distribuzione delle somme delle radici quadrate degli elementi, in blu la distribuzione di elementi buoni, in giallo quella degli elementi non buoni. La classificazione delle due baseline consiste nell'individuare un valore della somma oltre il quale l'esempio viene considerato buono: dalle distribuzioni in questa figura è evidente il motivo per cui nessuna delle due baseline riesce ad eseguire una classificazione apprezzabile.

Dati ridimensionati con average pooling

Si riportano in Figura 5.10 i risultati dei cinque diversi algoritmi calcolati sul dataset con esempi di dimensione ridotta, ottenuto applicando un average pooling agli esempi originali.

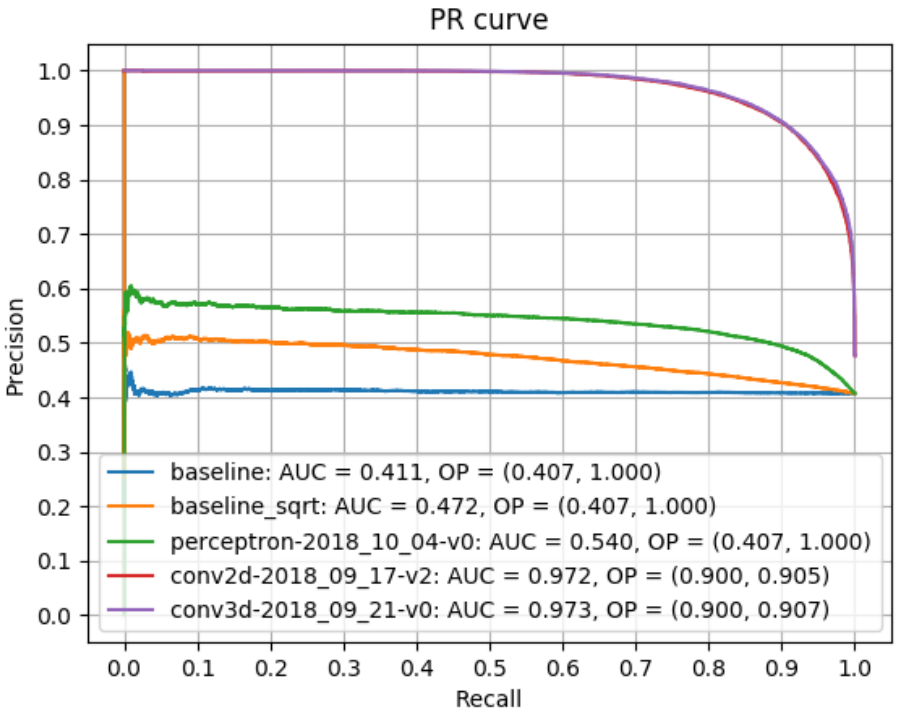

Figura 5.10: Confronto delle curve precision-recall dei 5 diversi algoritmi, calcolate su tensori $10 \times 10 \times 10$ a valori reali, ottenuti mediante un average pooling a blocchi $2 \times 2 \times 2$. Le CNN hanno dei risultati sufficienti considerando l'obiettivo prefissato. Gli altri algoritmi hanno una capacità di classificare completamente insufficiente.

Soglie per binarizzazione

Si riportano in Figura 5.11 gli andamenti delle aree sotto la curva precision recall, al variare delle soglie di binarizzazione, per i vari algoritmi (la variante della baseline viene esclusa in quanto eseguire la radice quadrata di uno 0 o di un 1 non cambia il valore).

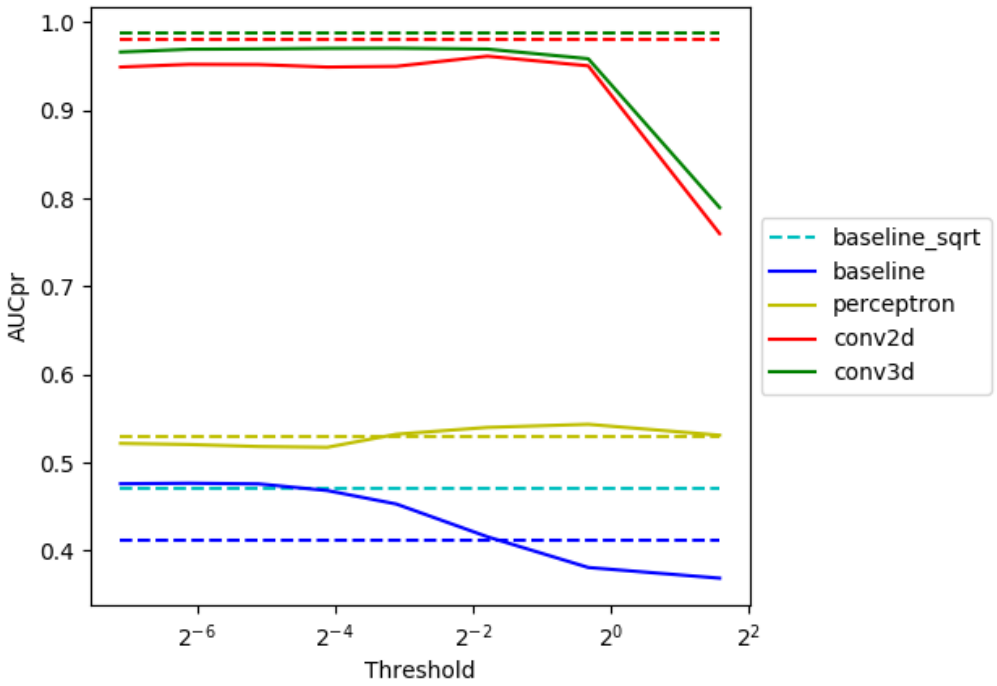

Figura 5.11: Area sotto la curva precision recall per i diversi algoritmi, al variare della soglia di binarizzazione, per i protoni. Le linee tratteggiate indicano il valore della metrica con dati non binarizzati. Le CNN registrano un miglioramento lieve e quasi indipendente dalla soglia finché la soglia è piccola, ma dopo una certa soglia subiscono un brusco peggioramento; perceptron raggiunge il riferimento in corrispondenza delle soglie 0.29 e 0.8, ma con altre soglie i risultati peggiorano; la baseline ha un massimo, superiore al riferimento, in corrispondeza della soglia 0.116.

Dati binarizzati

In base ai valori dell'area sotto la curva in funzione della soglia, viene scelto il valore 0.29 come soglia da applicare ai dati usati da CNN e perceptron; i dati usati dall'algoritmo baseline sono ottenuti usando 0.116 come soglia. Si riportano in Figura 5.12 le curve precision-recall relative ai 4 algoritmi, calcolate sui dati binarizzati.

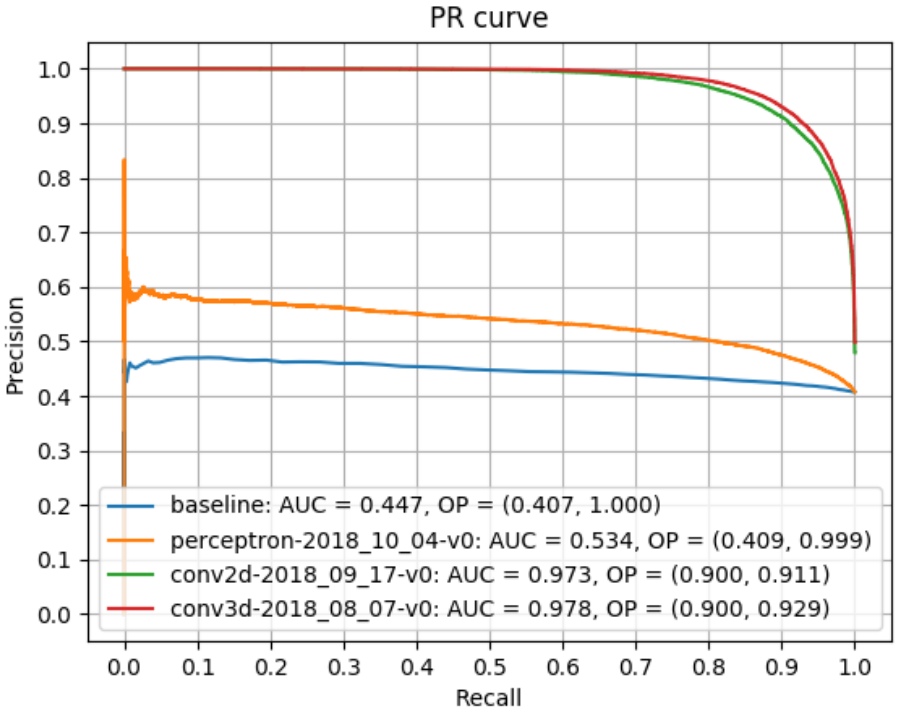

Figura 5.12: Confronto delle curve precision-recall dei 4 diversi algoritmi, ottenute su tensori a valori binari. Le CNN hanno dei risultati sufficienti considerando l'obiettivo prefissato. Gli altri algoritmi hanno una capacità di classificare completamente insufficiente.

Soglie per binarizzazione dopo average pooling

Si riportano in Figura 5.13 gli andamenti delle aree sotto la curva precision recall, al variare delle soglie di binarizzazione, per i vari algoritmi (la variante della baseline viene esclusa in quanto eseguire la radice quadrata di uno 0 o di un 1 non cambia il valore). Ai dati da binarizzare è stato prima applicato un average pooling.

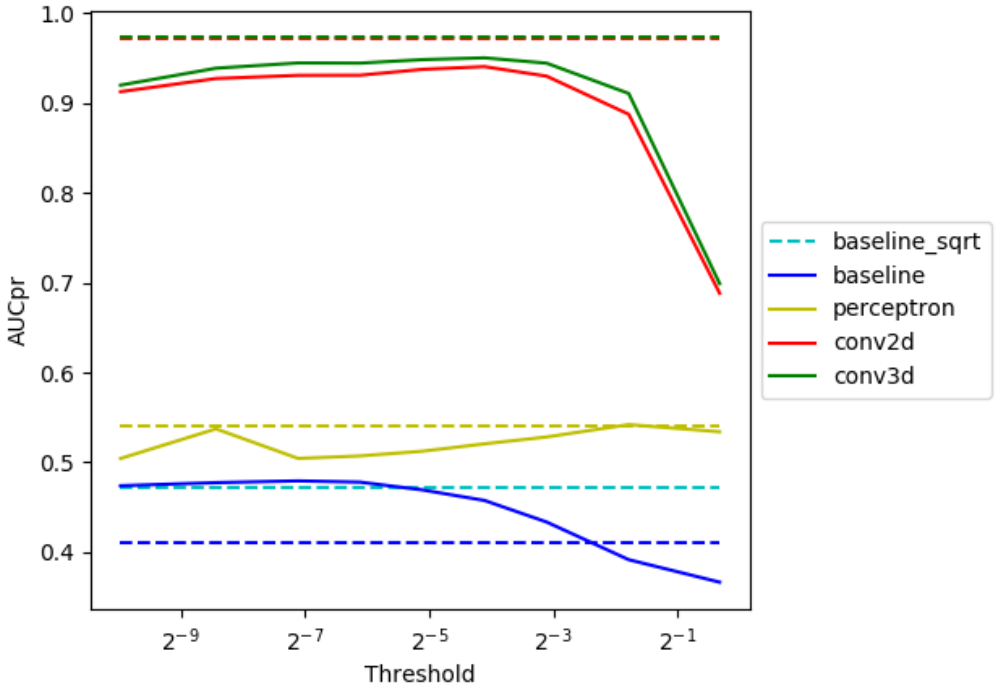

Figura 5.13: Area sotto la curva precision recall per i diversi algoritmi, al variare della soglia di binarizzazione, per i protoni, su tensori $10 \times 10 \times 10$ ottenuti con un average pooling a blocchi $2 \times 2 \times 2$. Le linee tratteggiate indicano il valore della metrica con dati non binarizzati. Gli algoritmi rispondono in modo anche molto differente alle varie soglie.

Dati di dimensione ridotta binarizzati

In base ai valori dell'area sotto la curva in funzione della soglia, viene scelto il valore 0.058 come soglia da applicare ai dati usati da CNN 2D e CNN 3D; invece per perceptron e baseline le soglie scelte sono rispettivamente 0.29 e 0.00725. Si riportano in Figura 5.14 le curve precision-recall relative ai 4 algoritmi, calcolate sui dati binarizzati.

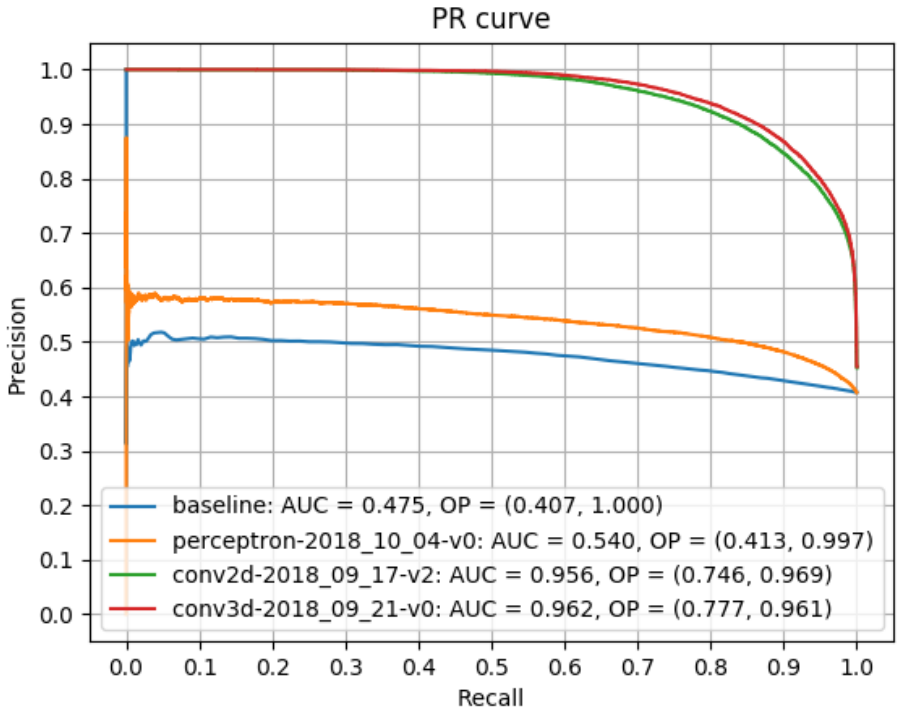

Figura 5.14: Confronto delle curve precision-recall dei 4 diversi algoritmi, ottenute su tensori $10 \times 10 \times 10$ a valori binari. Entrambi i tipi di rete neurale si avvicinano molto all'obiettivo prefissato, gli altri algoritmi sono ampiamente insufficienti.

5.2 Conclusioni

Indubbiamente i due problemi affrontati presentano difficoltà di tipo diverso. Da una parte, gli eventi buoni degli elettroni sono facilmente riconoscibili per via dello sciame in genere contenuto, ma si presentano già in percentuali molto alte, motivo per cui i margini per ridurre la banda usata dal satellite sono ridotti, e il classificatore deve essere quasi impeccabile. Invece, i protoni, i quali possono incidere maggiormente in una riduzione della banda utilizzata, sono molto più complessi da analizzare per via del loro sciame che è molto variabile e difficile da comprendere.

Per entrambi i problemi, i risultati ottenuti nel caso più semplificato, dove i tensori sono a valori reali e non ridotti ($20 \times 20 \times 20$), superano abbondantemente l'obiettivo prefissato in termini di *precision* e *recall*. Per gli elettroni, l'algoritmo baseline con variante della radici quadrate, ha raggiunto dei risultati sorprendenti, non molto lontani dalle reti neurali convoluzionali: in ottica di un utilizzo su HERD, questo significa che probabilmente sarà possibile riuscire a trovare un algoritmo soddisfacente per le analisi che devono essere effettuate, ma anche abbastanza semplice da consumare pochissima energia, che sarà un fattore determinante nella scelta dell'algoritmo da usare viste le risorse ed energia limitate di cui si dispone in orbita. Per i protoni, con gli algoritmi baseline non si riesce a fare una classificazione nemmeno lontanamente sufficiente, così come con perceptron, a conferma della maggiore complessità del problema: l'unica strada percorribile sembra essere quella delle reti neurali. È interessante notare come non ci sia grossa differenza tra i due tipi di CNN, seppure quella 3D ottiene dei risultati leggermente migliori: nel caso reale potrebbe essere scelta quella più semplice da implementare in elettronica e meno avara di risorse.

Cercare di rendere i dati più realistici, applicando un average pooling e/o

binarizzando i dati, per i protoni ha avuto l'effetto di ridurre leggermente la qualità della classificazione, ma sempre rimanendo vicino o al di sopra dell'obiettivo prefissato: solo nel caso più realistico tra quelli provati, con tensori $10 \times 10 \times 10$ a valori binari, non si raggiunge pienamente il target, ma le CNN sono vicine ad esso a tal punto da far sperare che in lavori futuri si riesca ad arrivarci. Quanto agli elettroni, con dati di dimensione ridotta a valori reali, entrambi i tipi di CNN e la variante della baseline si sono comportati egregiamente; con dati binarizzati addirittura, per entrambe le dimensioni testate, tutti gli algoritmi hanno raggiunto risultati eccellenti: questo significa che non solo è importante scegliere una buona soglia, ma anche che la binarizzazione non è detto che porti sicuramente un peggioramento, anzi può aiutare alcuni algoritmi a selezionare meglio i dati.

Bibliografia

[1] Shuang-Nan Zhang et al. Introduction to the High Energy cosmic-Radiation Detection (HERD) Facility onboard China's Future Space Station. *PoS*, ICRC2017:1077, 2018.

[2] J. Allison et al. Recent developments in geant4. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 835:186 – 225, 2016.

[3] J. Duarte et al. Fast inference of deep neural networks in fpgas for particle physics. *Journal of Instrumentation*, 13(07):P07027, 2018.

[4] O. Adriani et al. Measurement of boron and carbon fluxes in cosmic rays with the pamela experiment. *The Astrophysical Journal*, 791(2):93, 2014.

[5] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[7] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, Jan 1997.

[8] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[9] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Sept 2015.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] François Chollet et al. Keras. `https://keras.io`, 2015.

[14] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, third edition, 2010.

[16] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Cs231n: Convolutional neural networks for visual recognition 2016. 2016.

Ringraziamenti

Per concludere vorrei ringraziare il Professor Paolo Frasconi, relatore di questa tesi di laurea, e il Dottor Stefano Martina, correlatore, che mi hanno aiutato a capire e impostare la parte prettamente informatica di questo progetto. Inoltre vorrei ringraziare il Dottor Nicola Mori, correlatore, e gli altri fisici dell'Istituto Nazionale di Fisica Nucleare, da cui è partita l'idea che ha originato questo studio; Nicola in particolare mi ha fornito le conoscenze necessarie per comprendere il problema fisico che sta alla base di questa tesi.

Un ultimo ringraziamento lo faccio ai miei genitori che mi hanno permesso di affrontare gli studi nel migliore dei modi, senza farmi mancare niente, e alla mia ragazza, Giulia, che mi ha sempre supportato ed è stata la mia motivazione in ogni momento.