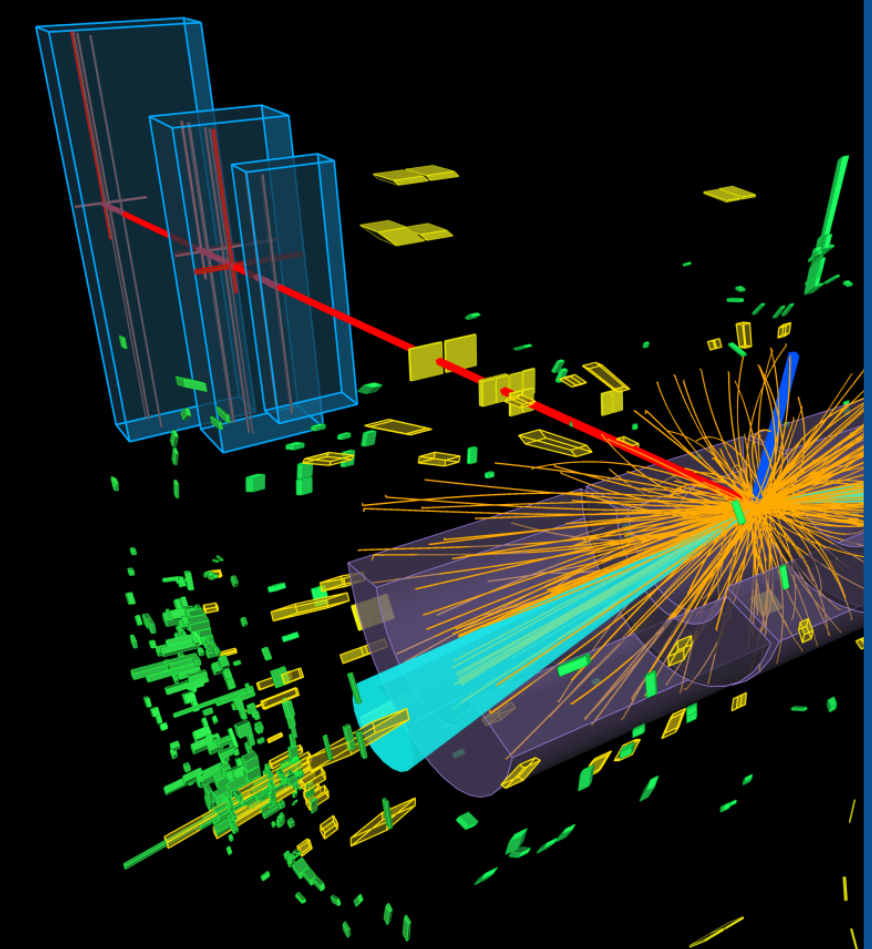


Algoritmi veloci di Deep Learning su processori FPGA per il trigger di muoni di L0 nella fase II dell'esperimento ATLAS a LHC

IFAE2019 - Napoli
9 Aprile 2019

Luigi Sabetta

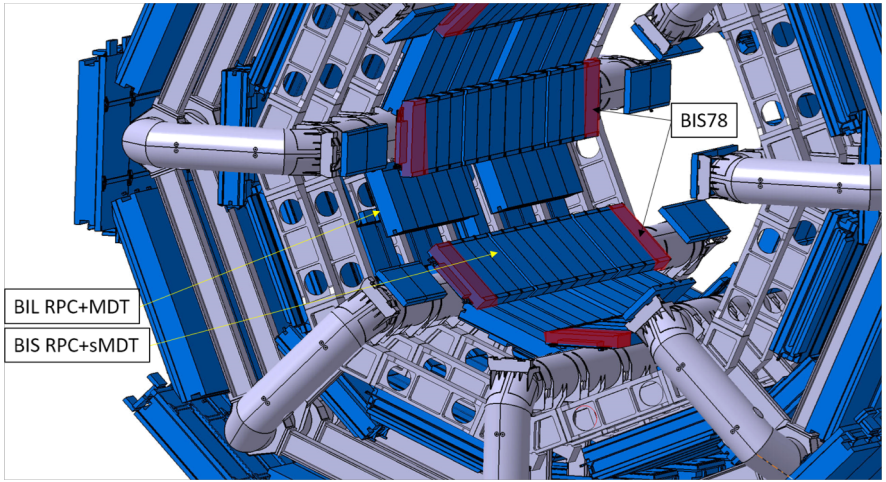
Motivazioni

Il trigger Muonico di livello 0 di ATLAS subirá un upgrade completo per HiLumi-LHC

Parametri della macchina:

- Maggiore pile up: 30-40 → 200
- Luminositá: (2 → 7.5) × 10^{34} $cm^{-2}s^{-1}$

Maggior rate di Hit nel settore muonico:
fino a 600 Hz/cm^2



ATLAS-TDR-026

Potenziamento di ATLAS:

Nuovo processore di trigger

- Sistema basato su FPGA
  - Virtex UltraScale+ XCVU13P
  - Celle logiche (K): 3780
  - Memoria (Mb): 455
  - GTY Transreceivers (32.75 GB/s): 128
  - I/O Pins: 832

Nuova stazione di trigger

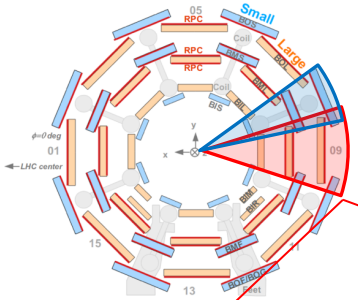
- Nuovi layer RPC

Algoritmo di trigger migliorato

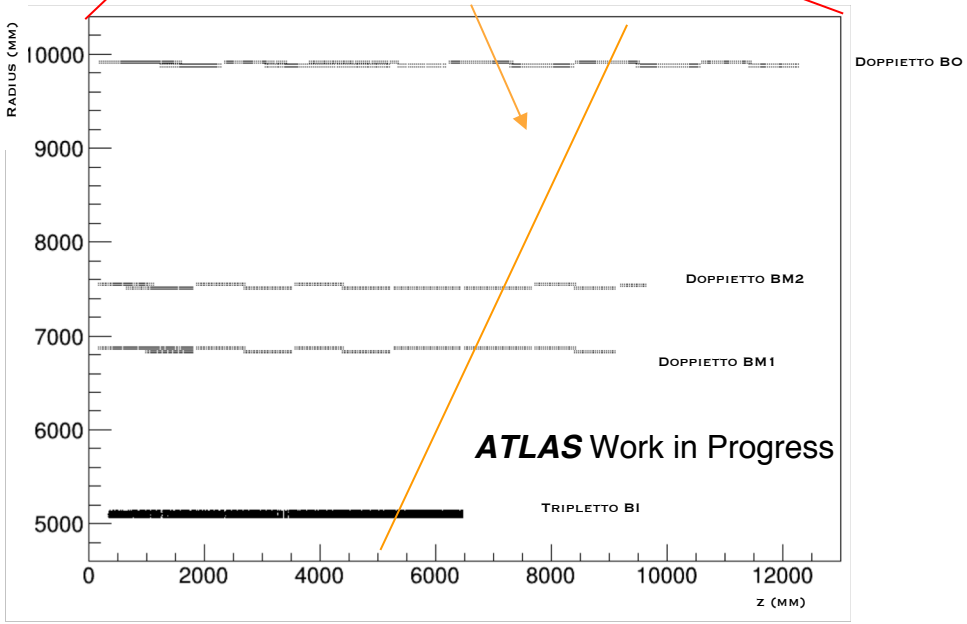
- Necessitá di essere molto flessibile e veloce

Reti Neurali → valido candidato

Informazioni disponibili per il trigger



Muone di impulso trasverso infinito

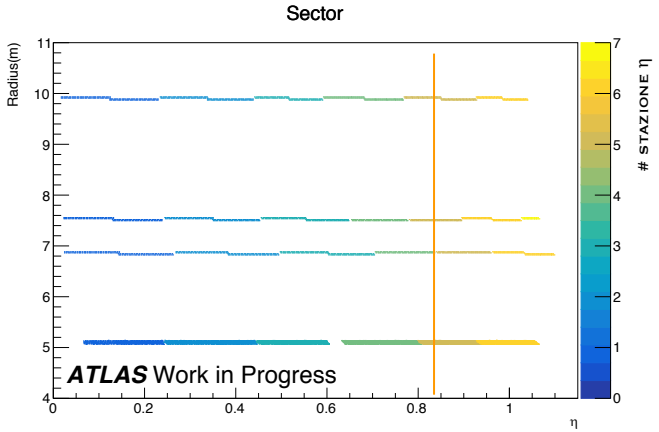


Resistive Plate Chamber(RPC):

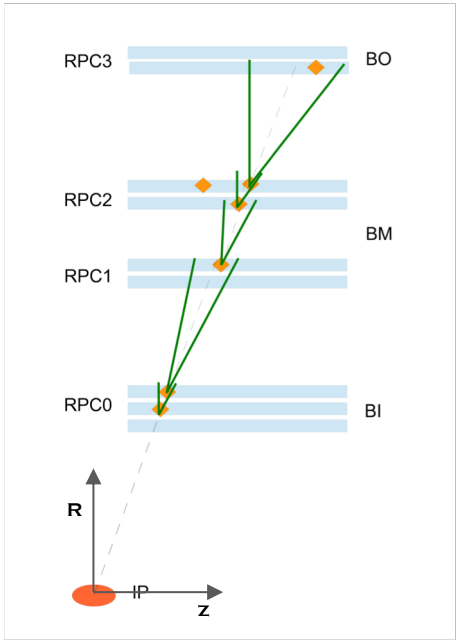
- Detector a gas veloci
- Sorgente di trigger in ATLAS

Settore:

- Si divide ATLAS in 2 parti ($z \gtrless 0$)
- Si divide la cordinata ϕ in 8 settori, ciascuno dei quali viene a sua volta diviso in 2 (Small e Large)
- Si prendono in considerazione tutte le strip degli RPC del settore ottenuto



Strategia standard di trigger



ATLAS-TDR-026

Algoritmo standard:

- Controlla la presenza di coincidenze all'interno di finestre aperte sequenzialmente in base alle presenza di segnale nei layers precedenti

👍 Stabile ed affidabile

👍 Buone performance

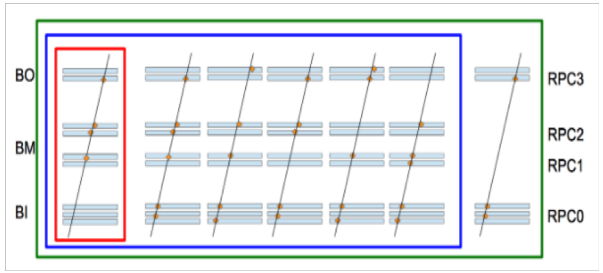
👎 Le finestre di coincidenza devono essere regolate "a mano"

- La dimensione delle finestre dipende dalla soglia di trigger in impulso trasverso (p_T)
- forte dipendenza dalle modalitá di trigger e dalla geometria locale del rivelatore

👎 Assume tracce che puntano al vertice primario di interazione

- Vertici di decadimento secondari lontani?

👎 Non fornisce nessun tipo di stima sul p_T



Trigger e Neural Nets (NN)

Output minimale trigger → fino a **tre candidati**

Una **NN** può essere sfruttata per:

- Individuare un candidato muone (=almeno un muone con p_T>soglia)
- Effettuare regressione su p_T ed η del muone con p_T più alto (leading)
- Effettuare regressione su p_T ed η del secondo muone con p_T più alto (subleading) → **BONUS!**
- Controllare (classificare) il numero di muoni → **BONUS!**

Regressione sul p_T a livello 0:

- L'algoritmo standard risponde semplicemente alla domanda "è sopra soglia?" (impostando diverse soglie si può individuare un range di impulso)

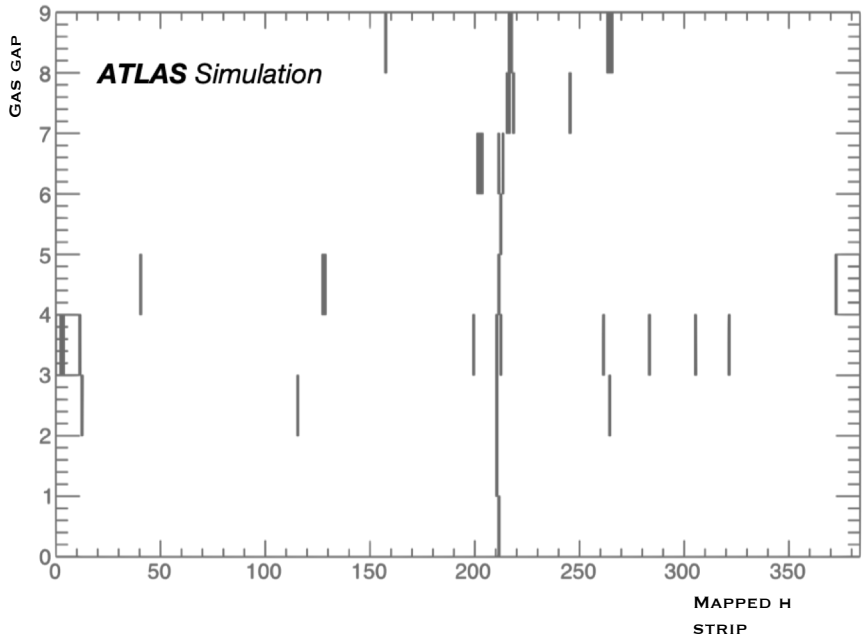
La **NN** viene addestrata per fornire un output **5D**

0 → nessun muone
1/2 → uno/due muoni
3 → più di 2 muoni

$$\left(p_T^{lead} \quad \eta^{lead} \quad p_T^{sublead} \quad \eta^{sublead} \quad N^{muons}\right)$$

Da mappa di strip a Immagini

https://twiki.cern.ch/twiki/bin/view/AtlasPublic/L1MuonTriggerPublicResults



Esempio di un muone con p_T=19 GeV + noise

Mappaggio → $i^{strip} = \frac{\eta_{hit} - \eta^{min}}{\eta^{max} - \eta^{min}} \times 384$

Si possono cosí costruire immagini da usare come input per una NN:

- Nel caso minimale 384 x 9 pixel
- Muone → retta

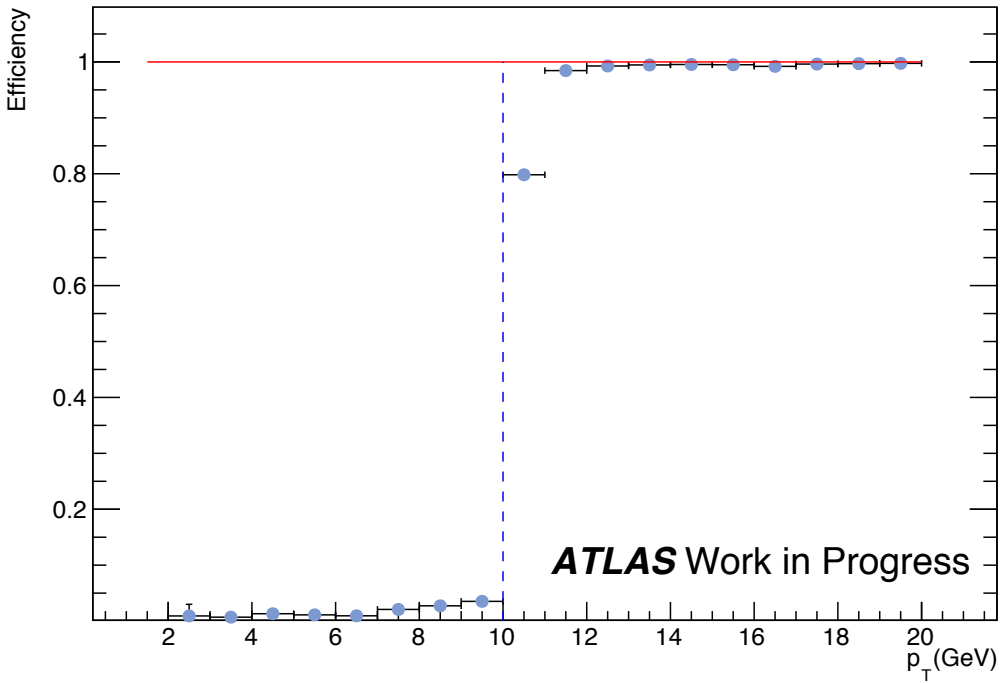
Vogliamo fare regressione sul p_T del muone

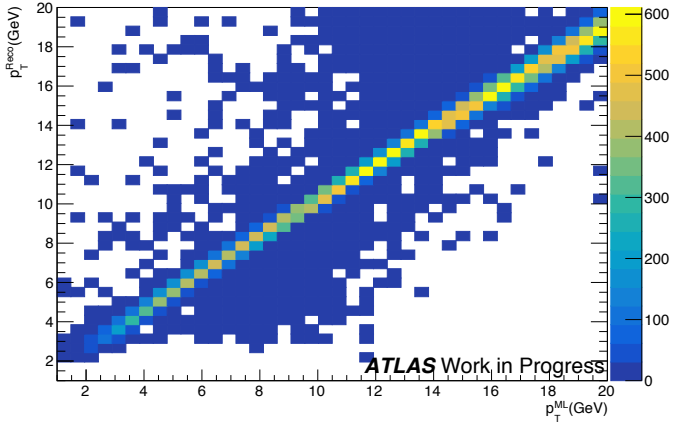
Problema:

- Le immagini sono altamente "sparsificate"

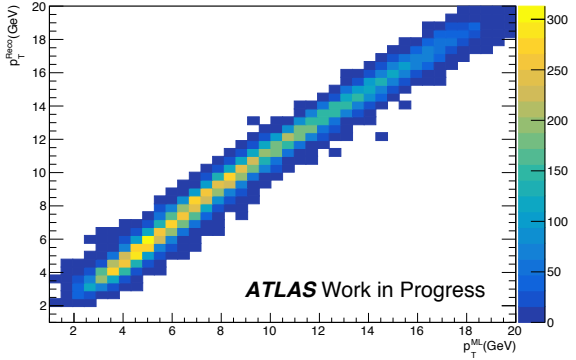
Rete Neurale Densa - Performance

Rete neurale completamente connessa (2 da layers da 2048 neuroni)
Input → immagini linearizzate





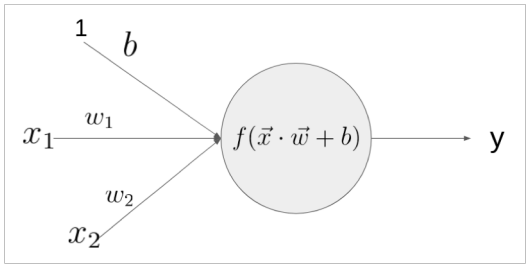
Muone leading (↑) e subleading (↓)

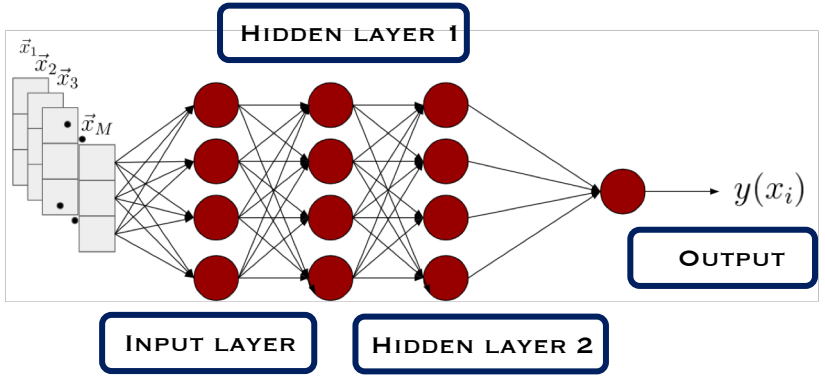


~11M di parametri → troppo grande da sintetizzare

Reti Convoluzionali

Neurone



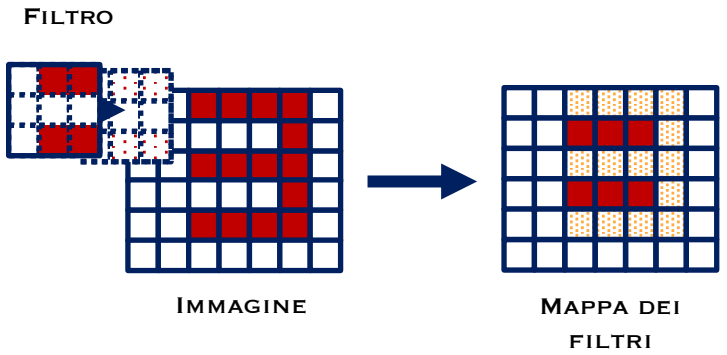


Rete Neurale Convoluzionale (CNN):
Reti neurali profonde ottimizzate per l'identificazione di immagini.

↓

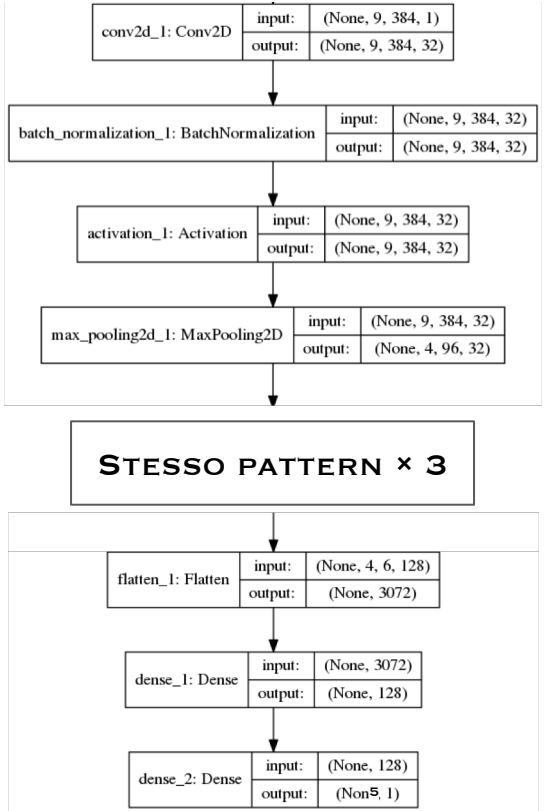
Efficaci per problemi che presentano simmetrie traslazionali e rotazionali

→



In questo modo il numero di parametri si riduce notevolmente

Struttura della CNN Floating Point



Filtri [3 × 3]

Pool. [2 × 4]

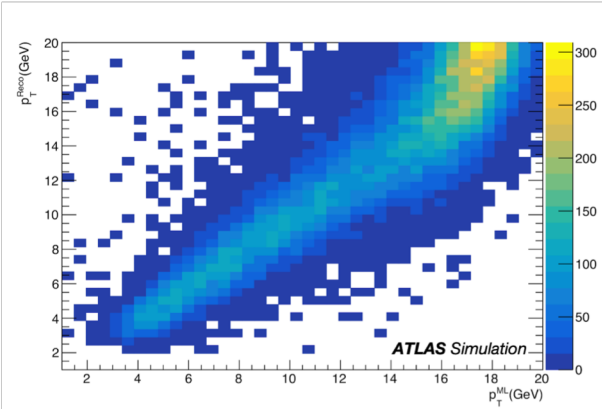
Architettura:

- (Conv2D + Batch Norm. + Max Pooling) x 3
- (Dense) x 2 layers per arrivare all'output (5D)

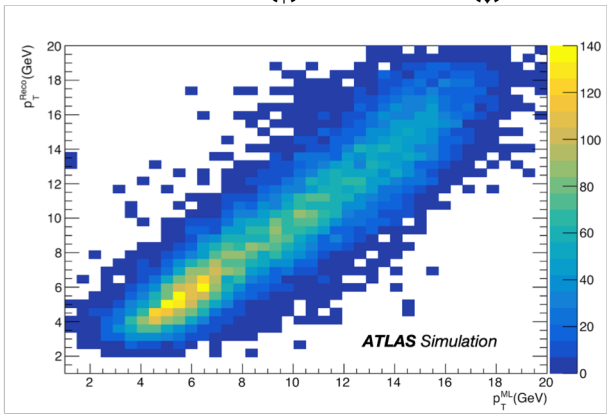
Numero totale di parametri: 500k

- Meno di una NN dense pura

Performance della CNN FP– Quantitá Fisiche



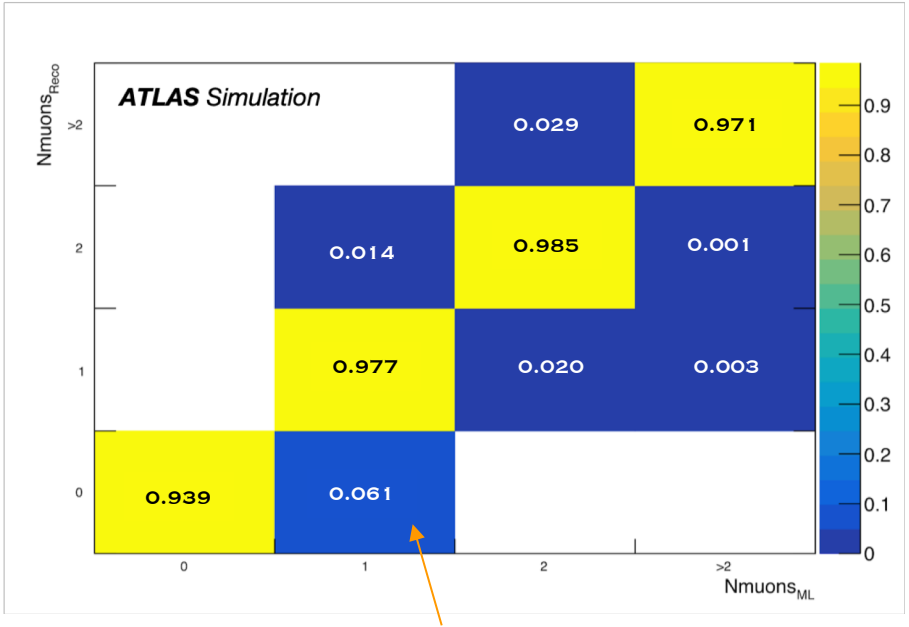
Muone leading (↑) e subleading (↓)



https://twiki.cern.ch/twiki/bin/view/AtlasPublic/L1MuonTriggerPublicResults

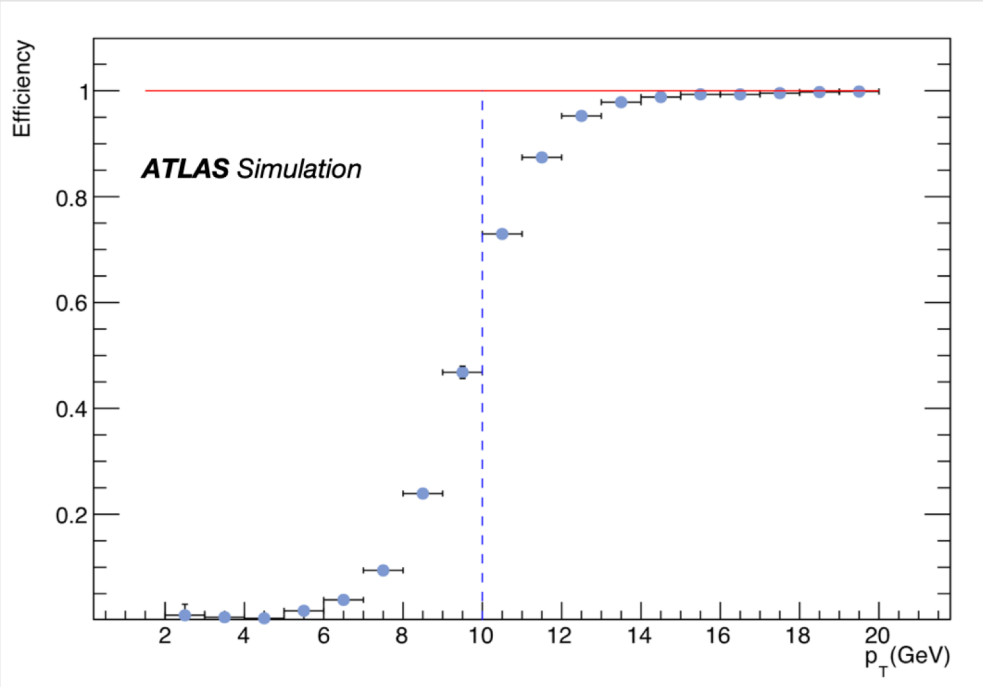
Le quantitá fisiche interessanti sono ben rappresentate

Gli eventi risultano ben classificati per quanto riguarda n^{muoni}



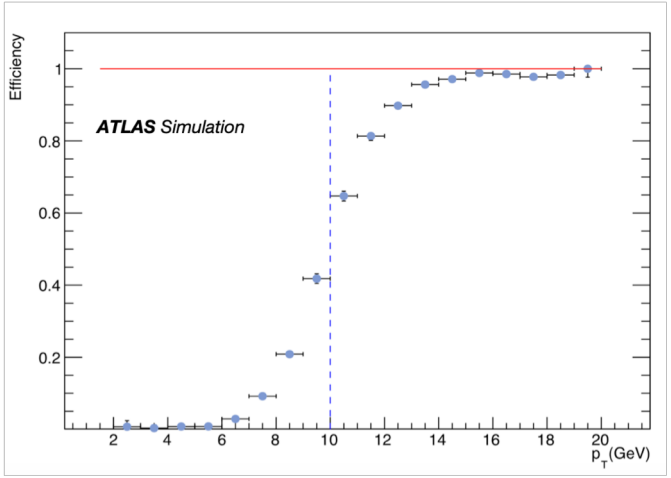
Quasi totalmente muoni con p_T fra 0-2 GeV: rimossi dal trigger

Performance della CNN FP- Trigger



Muone leading (↑) e subleading (→)

Le curve di efficienza di trigger sono soddisfacenti sia per il muone leading che per il sub-leading



https://twiki.cern.ch/twiki/bin/view/AtlasPublic/L1MuonTriggerPublicResults

Implementazione - CNN Ternaria

La Sector Logic del trigger di livello 0 opererá su FPGA

É necessario quindi sintetizzare questa CNN su di una FPGA

- NN Standard funzionano con pesi descritti da numeri con precision Floating Point a 32 bit
- Pesi FP non sono la scelta ottimale
  - Grosso impiego di risorse logiche

Si puó pensare di realizzare una CNN ternaria:

- Pesi =-1, 0, +1 (Descritti da soli due bit di memoria)

Una NN terniara determina una perdita di precisione

- Pochi punti percentuali rispetto ad una NN FP32 con la stessa struttura

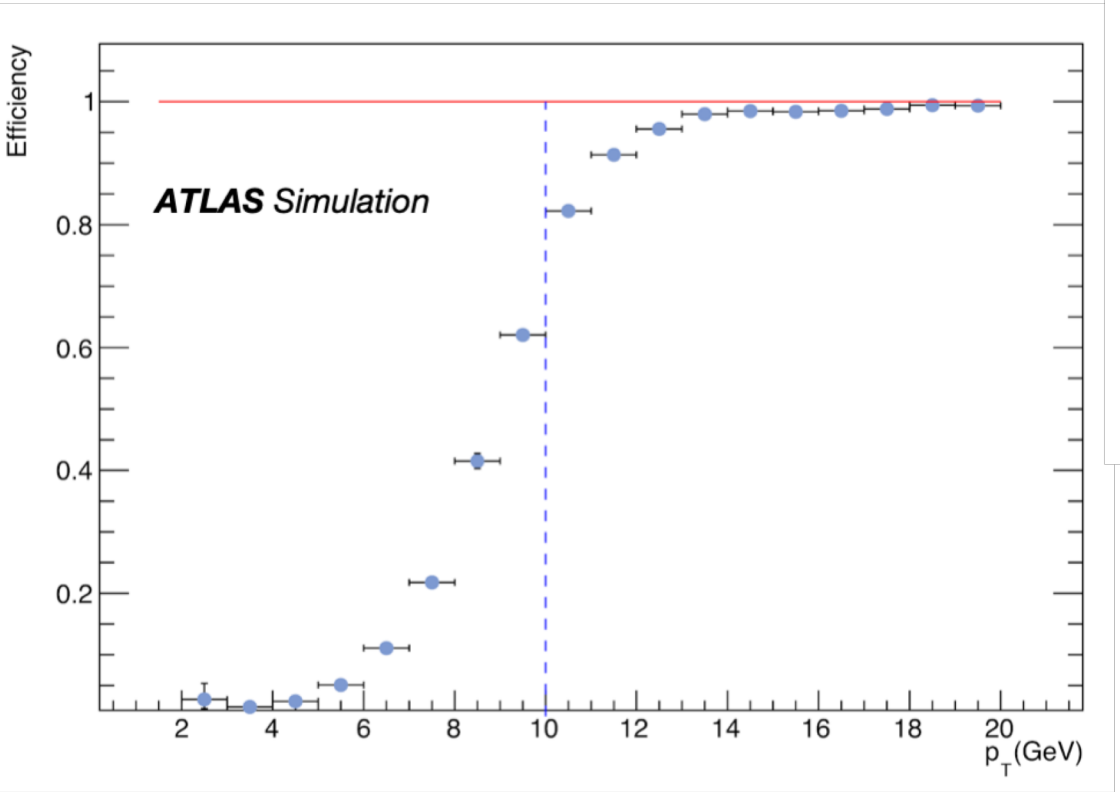
Ma dimensioni minori

- Inferiore consumo di risorse logiche
- Fino a 16 volte piú piccola rispetto ad una NN FP32

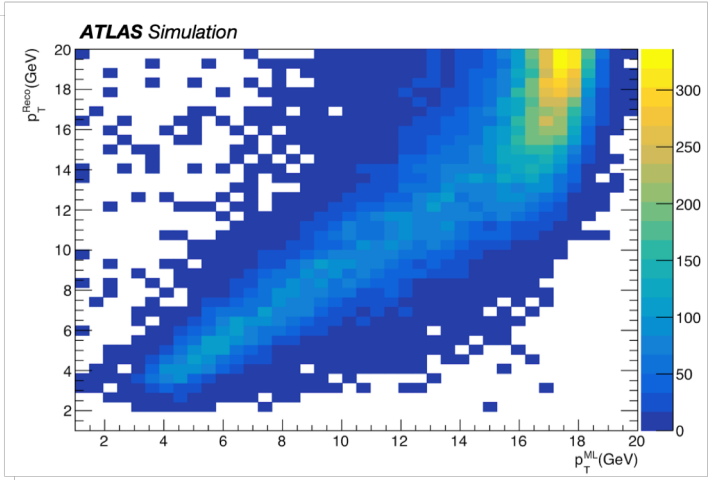
Puó in linea di principio essere resa piú profonda

- Piú layer recuperano la precisone persa

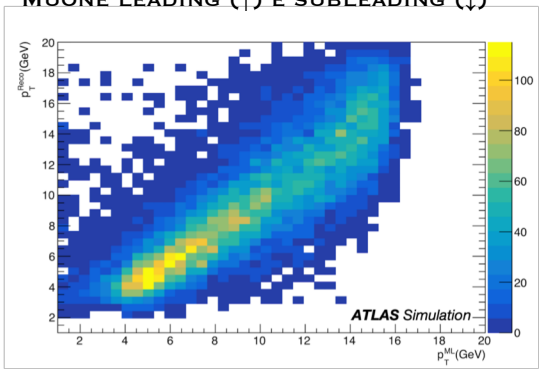
CNN Ternaria- Performance



https://twiki.cern.ch/twiki/bin/view/AtlasPublic/L1MuonTriggerPublicResults



Muone leading (↑) e subleading (↓)



Stato attuale dei lavori

Una CNN puó essere utilizzata per implementare l'algoritmo di trigger muonico di livello 0 per l'upgrade di fase II di ATLAS

- Buona risoluzione e risposta
- Regressione su p_T ed η di muone lead. e sublead.

Abbiamo una rete ternaria convoluzionale che migliora le prestazioni dell'algoritmo convenionale

- In termini di risorse logiche, puó essere sintetizzata Nella FPGA che si pianifica di impiegare per il trigger lo

Abbiamo sintetizzato in una FPGA con successo per il momento semplici architetture Dense e Conv2D

- Modelli firmware realizzati usando HLS4ML

Attualmente stiamo lavorando per riuscire a sintetizzare il nostro modello di rete ternaria convoluzionale

NN Dense F32 con 3 layer

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	13	3376	44226	131419	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	4450	-	-
Total	13	3376	48676	131461	0
Available	960	1824	433920	216960	64
Utilization (%)	1	185	11	60	0

NN Dense Ternaria con 3 layer

* Summary:

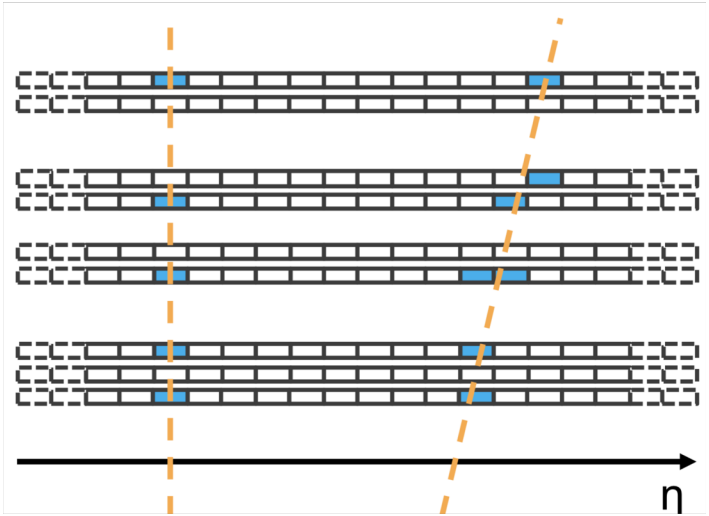
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	-	123	9626	59344	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	5906	-	-
Total	0	123	15532	59386	0
Available	960	1824	433920	216960	64
Utilization (%)	0	6	3	27	0

Grazie per l'attenzione

BACKUP

Mappaggio in strip-η vs layers

$$i^{strip} = \frac{\eta_{hit} - \eta^{min}}{\eta^{max} - \eta^{min}} \times 384$$



Il mappaggio previene alcune dipendenze geometriche, come:

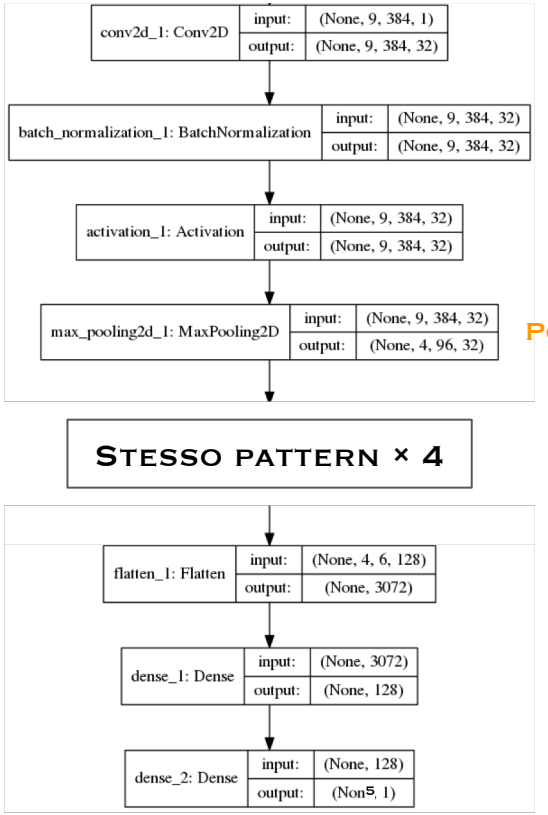
- Buchi
- Sovrapposizione di camere

Da notare che un muone di impulso trasverso infinito resta sempre una linea verticale, indipendentemente da η

Il mappaggio in strip fittizie permette di aumentare/diminuire la granularitá del detector a piacere:

- Nel caso minimale, ad ogni strip reale corrisponde una strip fittizia

Struttura della CNN Ternaria



Filtri [3 × 3]

Pool. [2 × 4]

Architettura:

- (Conv2D + Batch Norm. + Max Pooling) x4
  - Stessa struttura della rete FP32
- (Dense)x2 layers per arrivare all'output (5D)

Numero totale di parametri: 1M

- Maggiore rispetto alla CNN FP
- Ma qui ogni peso puó essere solamente +1/0/-1
  - Potenzialmente solo 2 bit

Per massimizzare le prestazioni della rete, il training (operato preliminarmente su GPU) viene effettuato in maniera non ternaria

NN su FPGA – Come implementarle

Architettura in python

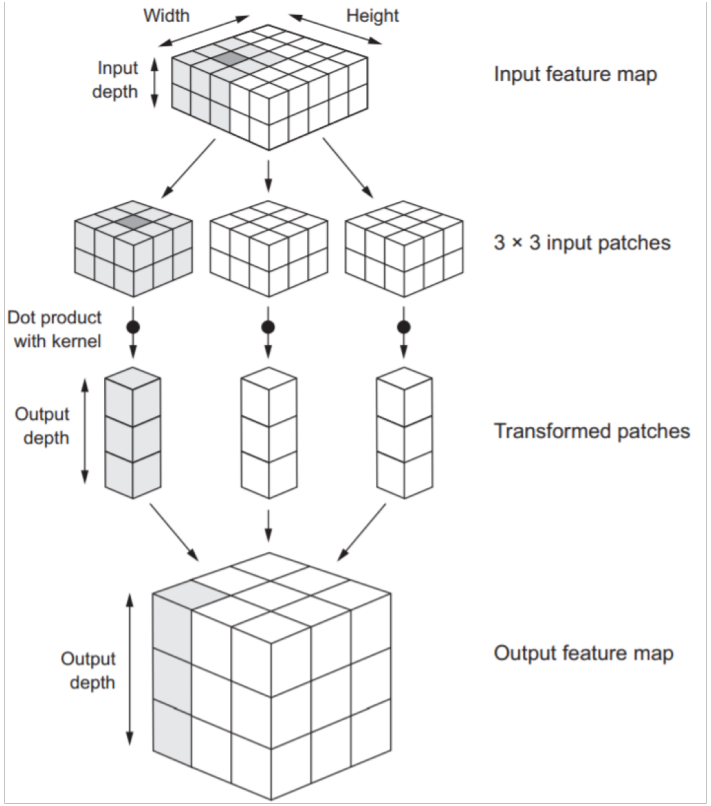
Per implementare una NN in una FPGA bisogna tradurre architetture tradizionali realizzate con pacchetti per la costruzione di reti neurali (keras) in linguaggio HLS che possa essere sintetizzato:

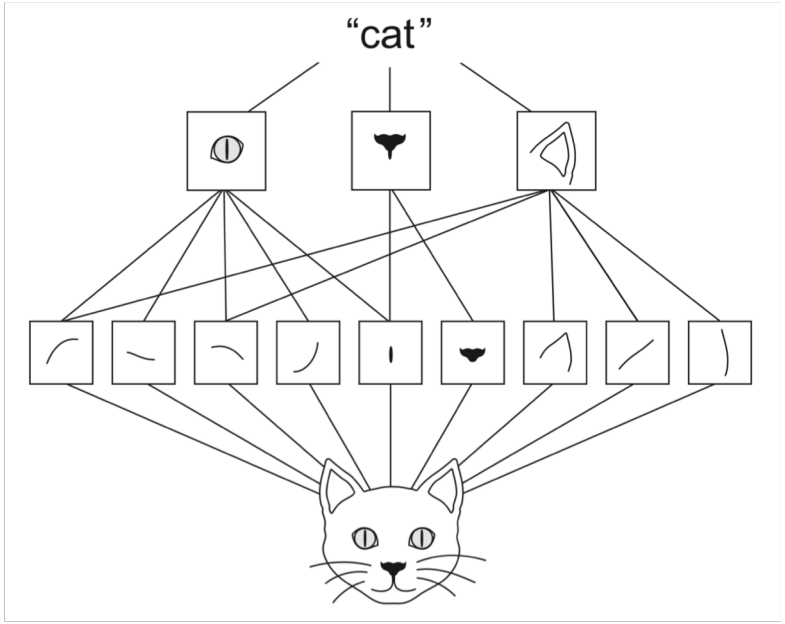
- Abbiamo usato HLS4ML per realizzare implementazioni firmware di algoritmi ML

C/C++ → Vivado HLS → VHDL/Verilog → Sintesi ed Implementazione

Lunghe operazioni di programmazione

CNN - Funzionamento





Architettura rete densa