*Bioinformatica Progetto*

***Predizione di Regioni Regolatorie Attive nella cell line A549 mediante l’utilizzo di metodi di Deep Learning***

*Alessandro Frangiamone*

*Anno Accademico 2020/2021*

*Università degli Studi di Milano*

*1 – Introduzione*

Nel **DNA**, la regolazione dell’espressione del gene avviene a livello di trascrizione del **RNA**, ed è accompagnato attraverso la sequenza specifica di legatura delle proteine (Transcription Factors) che attivano o inibiscono la trascrizione.

I **Transcription Factors** agiscono come attivatori o repressori.

I **repressori** prevengono l’RNA polymerase dal formare produttività complesse con la regione di inizializzazione per la trascrizione (promoters), gli **attivatori** invece la facilitano.

Identificare e determinare se determinate regioni hanno uno stato attivo o meno è un task essenziale per comprendere la modulazione dell’espressione dei geni e capire l’impatto genetico di varianti nelle malattie umane.

L’effetto di varianti genetiche in regioni non-coding è legato strettamente alla predizione di regioni regolatorie attive.

L’utilizzo di metodi di Machine Learning può portare ottime opportunità nell’identificare la posizione e lo stato di attivazione di queste regioni.

In questo progetto in particolare ci occuperemo di identificare se due regioni regolatorie (**Enhancers** e **Promoters**) sono attive.

*2 – Modelli*

Per il progetto sono stati utilizzati tre modelli di Deep Learning:

* **Feed-Forward Neural Network,** rete neurale dove le connessioni tra i layer vanno in un sola direzione (forward, da input ad output), quindi non formando cicli. È una delle reti neurali più classiche in cui l’output è determinato semplicemente dall’input. La rete Feed-Forward più semplice è il percettrone a singolo layer, costituito da un layer di ingresso seguito direttamente dall’uscita.

Le Feed-Forward più usate sono le cosiddette **Multi Layer Perceptron** (MLP) o Deep Feed-Forward Network, in cui tra il layer di input e quello di output vi sono più layer, definiti ‘hidden layer’. Ogni layer ha connessioni entranti dal precedente layer e uscenti da quello successivo.

* **Convolutionale Neural Network,** rete neurale che consiste in un input layer, diversi hidden layer e un output layer; usa la convoluzione al posto della moltiplicazione tra matrici.

Più in profondità la stuttura della network è composta da differenti hidden layer, generalmente il primo layer è un **convolutional layer**, che trasforma l’input (generalmente un tensore con una forma: numero di input x altezza x larghezza x canali) che viene astratto in una feature map.

Vi sono anche **pooling layer**, che hanno lo scopo di ridurre la dimensionalità combinando output di cluster di neuroni in un singolo neurone nel layer successivo. Generalmente vengono usati filtri di dimensione 2x2 e ci sono due metriche di pooling: Max, prendere il valore massimo del cluster selezionato. Average, prende il valore medio.

Infine, vi sono i **fully connected layers** che connettono ogni neurone di un layer con ogni neurone del layer successivo, come in una rete tradizionale MLP. Questi layer sono presenti alla fine nel CNN, dopo la parte di convoluzione e si occupano di classificare l’immagine

Rispetto ad altri metodi di machine learning CNN usa dati poco pre-processati, la rete ottimizza i filtri/kernel, in modo automatico con il learning.

* **Multi Modal Neural Network,** consiste nella combinazione di due Deep Neural Network, un hidden layer iniziale rappresenta il punto di giuntura tra le due network.

Spesso le informazioni nel mondo reale provengono da fonti rappresentate in modalità diverse, ad esempio immagini possono essere associate con spiegazioni di testo; due fonti di informazioni così diverse hanno anche proprietà statistiche differenti. La rete multimodal ha l’obiettivo di unire i benefici di entrambe le modalità combinando due NN.

Nel progetto le due modalità sono i dati epigenomici, gestiti dalla rete feed-forward, e i dati genomici, gestiti dalla rete convoluzionale.

Le performance dei tre modelli verrà poi trattata nell’ultimo paragrafo.

*3 – Setup Sperimentale*

Questo paragrafo consiste nell’analisi di ogni step sperimentale, dalla definizione dei task, fino alle metriche utilizzate per comparare i modelli.

*3.1 – Tasks*

I tasks analizzati nel progetto sono due: “**Active Enhancers vs Inactive Enhancers**” e “**Active Promoters vs Inactive Promoters**”.

Identificare e determinare se determinate regioni hanno uno stato attivo o meno è un task essenziale per comprendere la modulazione dell’espressione dei geni e capire l’impatto genetico di varianti nelle malattie umane.

*3.2 – Dataset*

I modelli di Feed Forward Neural Network e Convolutional Neural Network sono stati trainati e testati rispettivamente con dati epigenomici e genomici.

Per quanto riguarda i dati **epigenomici** per comodità è stato utilizzata il package *epigenomic\_dataset 1* che comprende al suo interno i dati già pre-processati per ogni regione enhancers e promoters di sette cell lines, tra cui la cell line utilizzata per il progetto ovvero la **A549**. Il package scarica e utilizza i dati di **FANTOM 5 2** un dataset che da accesso a circa 250 cell lines labels relative a regioni regolatorie.

Per le reti neurali convoluzionali invece sono richiesti dati sequenziali, con più alta località rispetto ai dati epigenomici, dato che sfrutta maggiormente la posizionalità. I dati **genomici** vengono presi dall’***USCS Genomes browser 3*** un repository contenente alcuni dataset genomici umani, in particolare è stato utilizzato il genome ‘**HG38**’. Per automatizzare il processo di coding è stato utilizzato il package *ucsc\_genomes\_downloader 4*.

*3.3 – Data Pre-Processing*

Prima di passare i dati ai modelli è necessario processarli in modo da massimizzare la qualità dell’informazione e ridurre il rumore.

Per prima cosa è stato eseguito un check dei **NaN value**, ovvero tutti i valori nulli all’interno delle celle dei vari sample.

La procedura per trattare questi dati è la **Data Imputation**, che consiste nel sostituire il contenuto delle celle contenenti i missing value con valori sostitutivi ottenuti mediante varie metodologie. La data imputation è necessaria dato che i NaN value potrebbero causare problemi per l’analisi dei dati.

La metodologia di data imputation utilizzata nel progetto è la **Neighbours Imputation**, che sfrutta l’algoritmo **k-Nearest Neighbours** per riempire i valori mancanti. Di default come metrica è utilizzata la distanza euclidea dal valore mancante ai suoi vicini, viene poi fatta la media dei valori vicini oppure viene pesato il valore in base alla distanza di ogni vicino.

*3.4 – Data Correlations*

Un filtro che è possibile applicare alle features è la correlazione con le label d’output e correlazione tra features. Idealmente vogliamo bassa correlazione tra le feature, altrimenti implica che contengono informazioni simili. E alta correlazione con l’output.

Nel progetto è stata eseguita la **correlazione di Pearson**, che misura la relazione lineare tra due dataset. Brevemente si ottiene un risultato di valore compreso tra +1 e -1, dove +1 corrisponde a correlazione positiva, 0 a non correlazione e -1 a correlazione negativa.

In seguito è stata eseguita la **correlazione di Spearman**, misura non parametrica (non implica che sia un dataset con distribuzione normale) di monotonicità di due dataset, ovvero permette di stabilire quanto bene una relazione tra due variabili può essere descritta usando una funzione monotona. Spesso Spearman identifica correlazioni nelle feature che Pearson non identifica.

*3.5 – Feature Selection*

La **Feature Selection** è una forma di riduzione della dimensionalità di un determinato dataset. Lo scopo della feature selection è quello di avere un modello funzionale.

L’idea generale è scegliere un subset rilevante di feature dal pool originale, eliminando le feature che non mi interessano in modo da ridurre l’overfitting causato dai dati noisy, aumentare l’accuracy e ridurre il tempo di training.

Nel progetto per eseguire la feature selection utilizziamo l’algoritmo **Boruta** un wrapper (passaggi iterativi per scoprire le feature di valore) costruito mediante l’algoritmo di classificazione **Random Forest**.

Random Forest è generalmente un algoritmo veloce, che viene eseguito senza la regolazione dei parametri e fornisce una stima numerica dell’importanza della feature. La classificazione avviene tramite voto di un insieme di decision tree, sviluppati indipendentemente su diversi campioni del training set.

Boruta è basato sulla stessa idea, aggiungendo casualità al sistema e raccogliendo risultati dall’insieme di campioni randomizzati si può ridurre l’impatto delle fluttuazioni casuali.

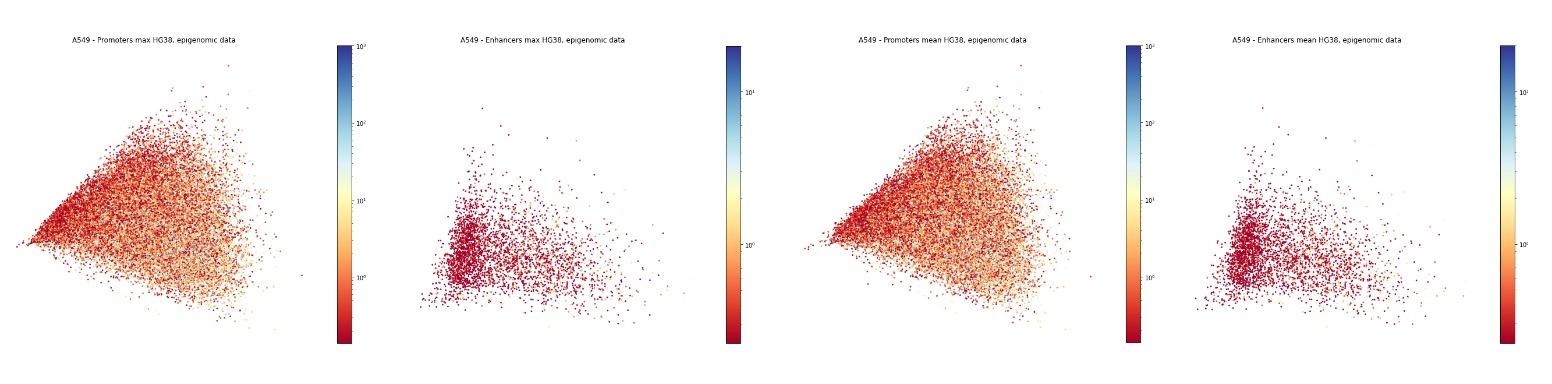
Nel progetto l’algoritmo è stato eseguito nel ciclo di training del modello.

*3.6 – Data Visualization*

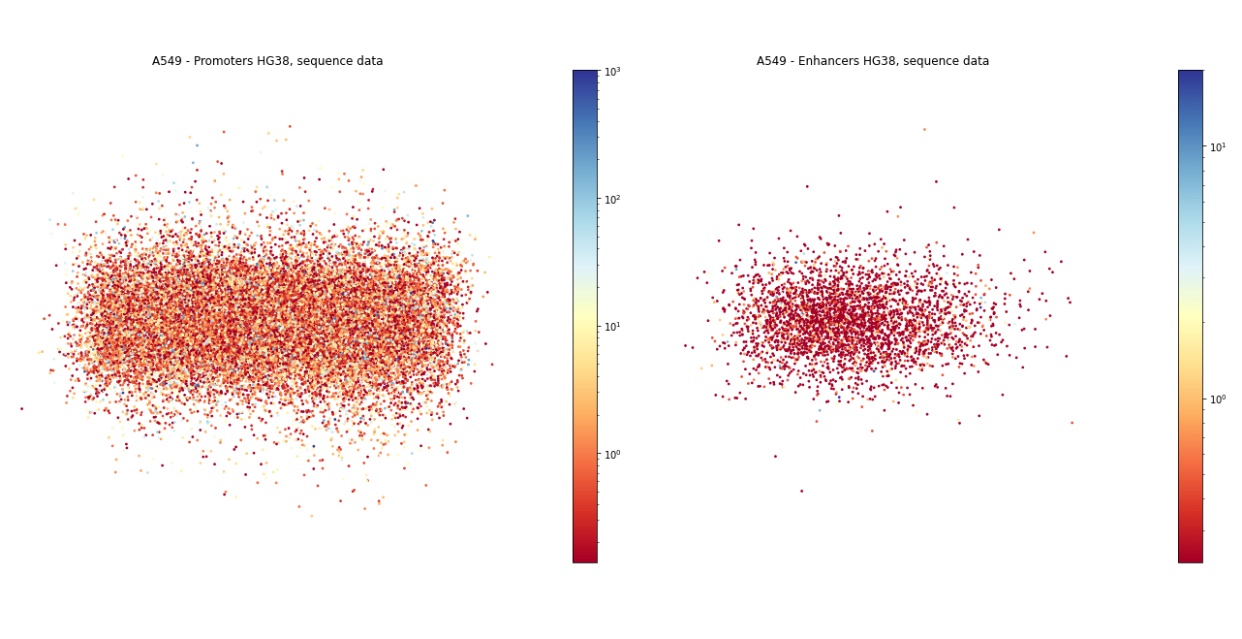
Nel progetto viene utilizzata una tecnica per la semplificazione (riduzione di dimensionalità) e visualizzazione dei dati: **PCA (Principal Component Analysis)**.

L’Idea di PCA è: dati N datapoints in uno spazio D-dimensionale, proiettarli in uno spazio a dimensionalità inferiore come un grafico 2-D preservando comunque più informazioni possibili. In particolare, ricavare una proiezione che minimizza l’errore quadratico nella ricostruzione dei dati originali.

Di seguito alcuni scatterplot ricavati dai dati epigenomici e genomici del progetto.



*Fig.1: Data Visualization [Epigenomic Data] Enhancers e Promoters (max e mean) cell line A549*

**

*Fig.2: Data Visualization [Genomic Data] Enhancers e Promoters cell line A549*

*3.7 – Holdouts*

Prima di trainare i dati viene fatto un semplice holdouts con 10 split e i dati vengono divisi: 80% training e 20% testing.

**Holdouts** è un approccio che consiste nel dividere la porzione di dati con cui fare il training, dalla porzione di dati con cui fare il testing; questo per fare in modo di avere una verifica dell’efficacia del modello trainato su dati mai visti dal modello, i dati di test.

Per avere un approccio più consistente è possibile usare il **repeated holdout,** che consiste nel dividere randomicamente in k sottoparti il dataset da cui ricavo il test set. Questo processo lo ripeto più volte, così da avere l’errore reale stimato come la media delle stime separate.

*3.8 – Struttura dei Risultati*

In seguito al training e testing del modello sono state scelte delle metriche di valutazione, utili poi a comparare i dati, le metriche usate sono: **Accuracy**, **Precision** e **AUROC**.

* **Accuracy**, metrica molto semplicistica, è definita come il numero di previsioni corrette diviso per il numero totale di previsioni, moltiplicato per 100.
* **Precision**, metrica definita come il numero di elementi etichettati correttamente diviso il numero totale di elementi etichettati come appartenenti alla classe (sia corretti che errati), in formule:

Precision = True Positive / ( True Positive + False Negative )

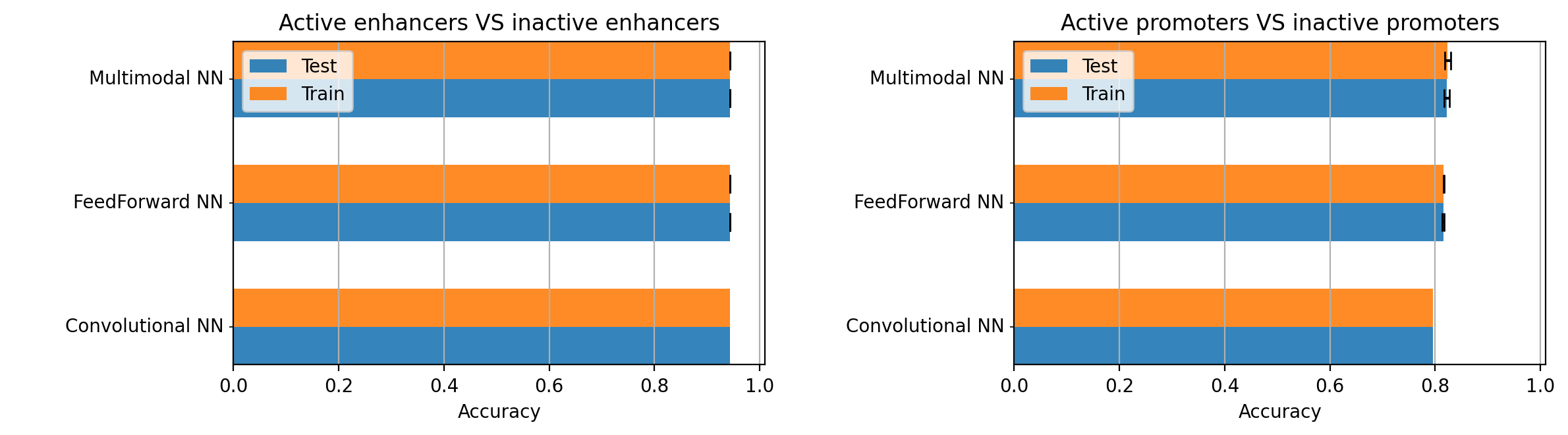
* **AUROC (Area Under Receiver Operating Characteristic curve)**, AUROC è una misura aggregata delle prestazioni di un classificatore binario su tutti i possibili valori di threshold. AUROC calcola l’area sotto la curva ROC (curva che mostra il ratio dei veri positivi / il ratio dei falsi positivi per vari valori di threshold) e quindi compresa tra 0 e 1.

Come parametro standard viene considerato il rapporto 0, ovvero quello che rappresenta casualità, tanti veri positivi quanti falsi positivi. Idealmente quello che si vuole raggiungere è un AUROC il più alto possibile (il più vicino a 1 possibile).

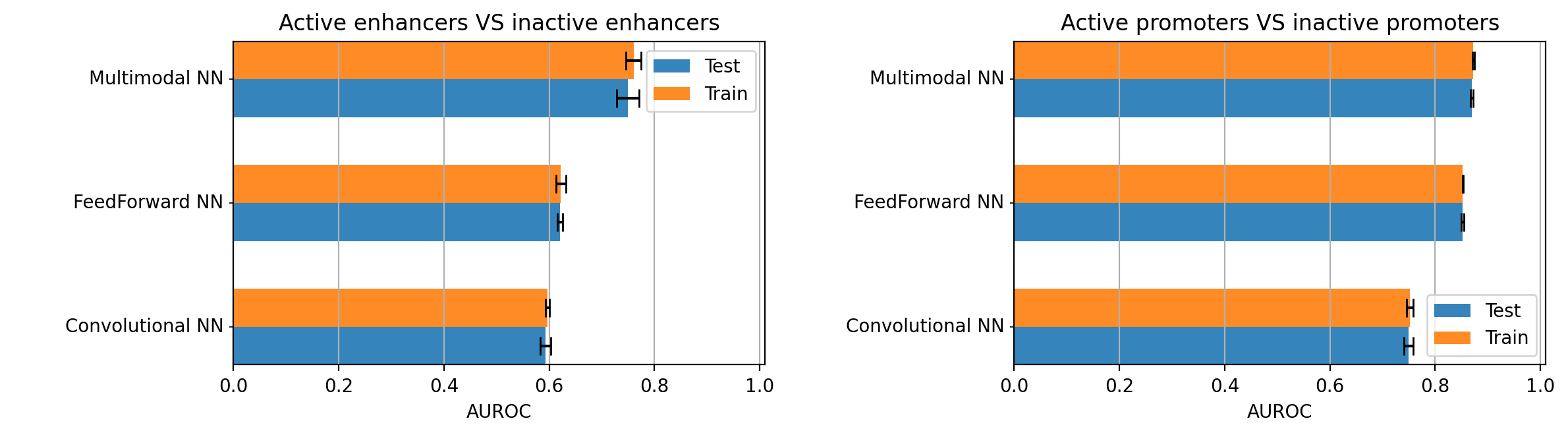
*4 – Risultati*

I risultati ottenuti sono stati rappresentati su un grafico a barra orizzontale, utilizzando il package python *barplots 5*.

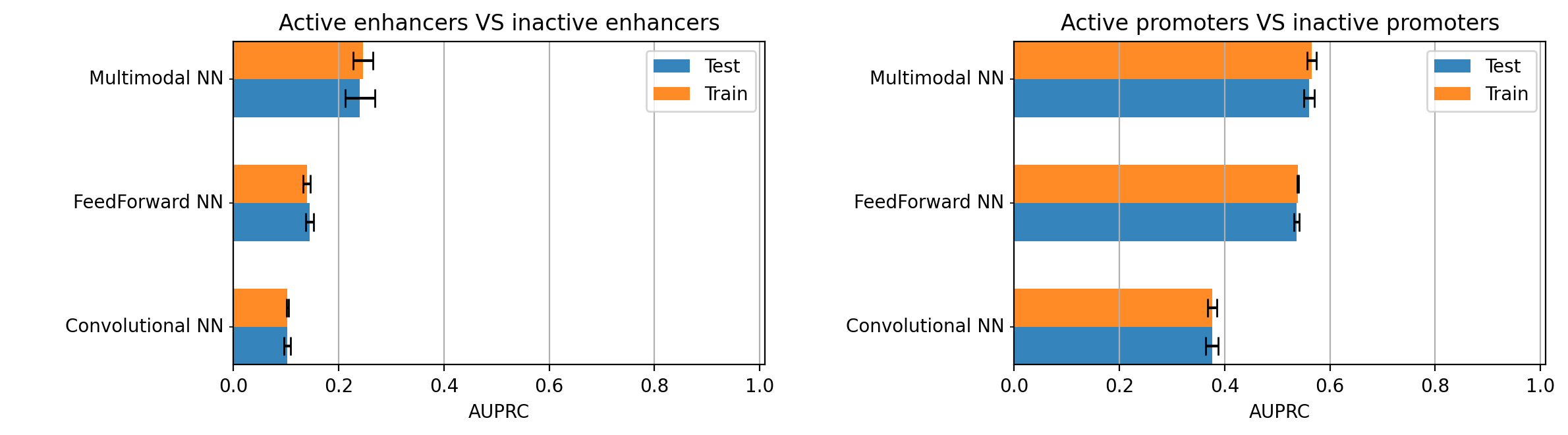
Oltre alle metriche scelte e discusse nel paragrafo 3.8 sono presenti i bar plot di altre metriche inserite per completezza.



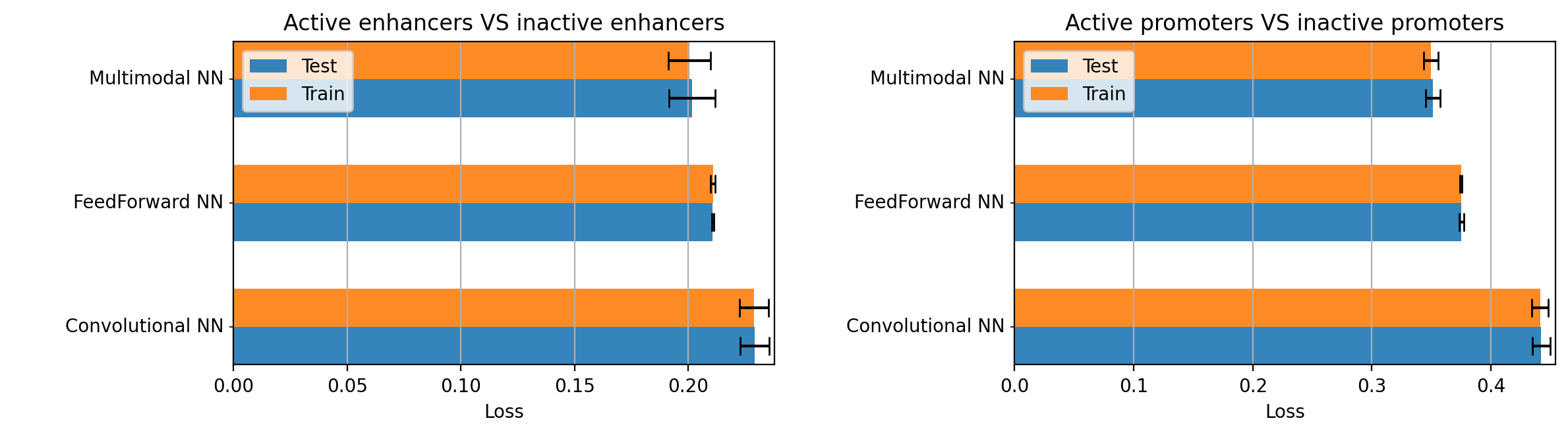
*Fig.3: Accuracy dei tre modelli in entrambi i task*



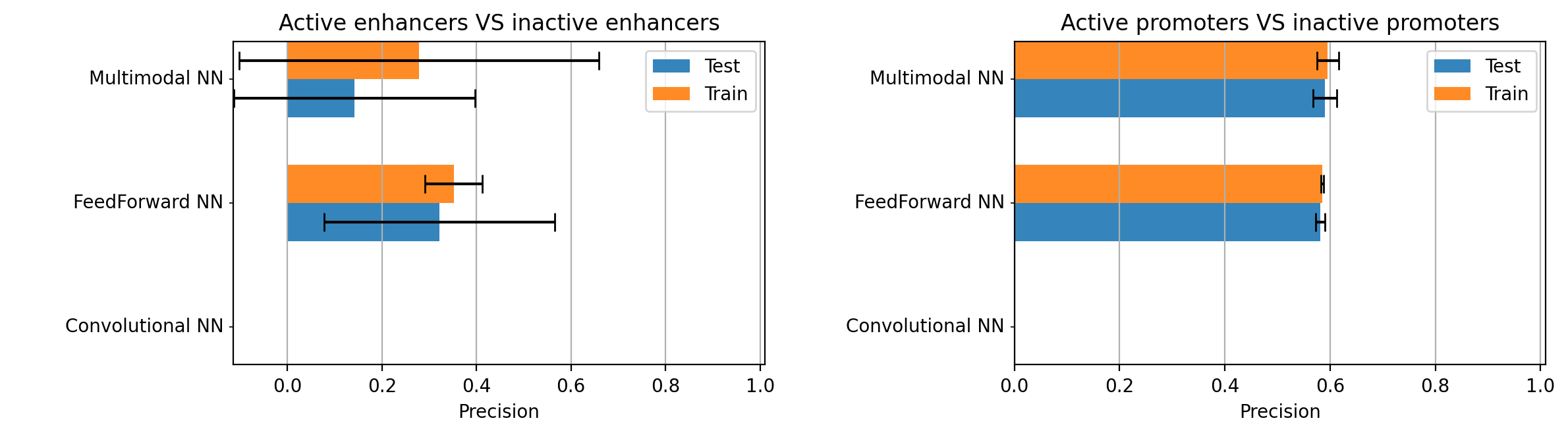
*Fig.4: AUROC dei tre modelli in entrambi i task*



*Fig.5: AUPRC dei tre modelli in entrambi i task*



*Fig.6: Loss dei tre modelli in entrambi i task*



*Fig.7: Precision dei tre modelli in entrambi i task*

Dai dati e barplot ottenuti possiamo concludere che:

Le performance migliori sono state ottenute dalla **Multimodal Neural Network**, sfruttando la combinazione di FFNN per dati epigenomici e CNN per dati genomici, in molti casi reali le MMNN hanno performance migliori rispetto a singole network dato che sfruttano la potenzialità di entrambe. Soprattutto data la differente varietà dei dati.

Il modello che ha performato peggio invece è la **Convolutional Neural Network**, probabilmente dato che i task eseguiti sono binari. Questa tipologia di rete neurale non performa in modo ottimale su task binari, infatti generalmente le CNN vengono utilizzate ad esempio per multiclass classification e in particolare le operazioni di convoluzione sono ottimali quando si ha a che fare con immagini (es. Image Classification, Object Detection).

In generale i risultati ottenuti non sono ottimali, è possibile migliorare i risultati facendo tuning più accurati di tutti e tre i modelli.

*Referenze:*

* [1] *Epigenomic\_dataset*: <https://github.com/AnacletoLAB/epigenomic_dataset>
* [2] *Fantom5*: <https://fantom.gsc.riken.jp/5/>
* [3] *Ucsc genomes browser*: <https://hgdownload.soe.ucsc.edu/downloads.html>
* [4] *Ucsc\_genomes\_downloader*: <https://github.com/LucaCappelletti94/ucsc_genomes_downloader>
* [5] *Barplots*: <https://github.com/LucaCappelletti94/barplots>