Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Biomedica

Analisi delle prestazioni di DeepVirFinder nell'identificazione di sequenze virali

Relatore:
Prof. Matteo Comin

Laureando:
Luca Pellegrini

Anno Accademico 2023/2024
Data di laurea **23 settembre 2024**

Sommario

Uno dei principali campi di applicazione della moderna bioinformatica è la metagenomica, che si occupa di studiare le comunità microbiche direttamente nel loro ambiente naturale, senza la necessità di isolare e coltivare in laboratorio le singole specie. Le tecnologie di sequenziamento di nuova generazione permettono di sequenziare interi genomi in poche ore, generando quotidianamente enormi quantità di dati, che necessitano di strumenti efficaci per essere analizzati.

Negli ultimi anni, sono stati sviluppati diversi software che sfruttano moderne tecniche di deep learning per studiare i dati metagenomici. In questo elaborato ci concentreremo in particolare su uno di questi software, DeepVirFinder, che sfrutta una rete neurale convoluzionale per identificare sequenze virali all'interno di dati metagenomici. Descriveremo nel dettaglio l'architettura del modello di DeepVirFinder, e ne analizzeremo le prestazioni attraverso alcuni esperimenti.

Abstract

Metagenomics is one of the main fields of application of bioinformatics. Metagenomics deals with the study of communities of microbial organisms directly in their natural environments, bypassing the need for isolation and lab cultivation of individual species. Next-generation sequencing technologies enable us to sequence whole genomes in just a few hours, thus producing huge quantities of data on a daily basis. We therefore need efficient tools to analyze this data.

In recent years, many different pieces of software have been developed, which levarage modern deep learning techniques to study this metagenomic data. In this work, we'll focus on one specific such software, DeepVirFinder, which uses a convolutional neural network to identify viral sequences in metagenomic data. We'll describe the architecture of DeepVirFinder in great detail, and we'll analyze its performace through some experiments.

Indice

Capitolo 1

Introduzione

La *metagenomica* è l'applicazione delle moderne tecniche di sequenziamento genomico allo studio delle comunità microbiche direttamente nel loro ambiente naturale, superando la necessità di isolare e coltivare in laboratorio le singole specie [1]. La metagenomica è uno dei principali campi di applicazione della *bioinformatica*, una "scienza ibrida" che sfrutta gli strumenti dell'informatica - come le tecniche di archiviazione, trasmissione ed elaborazione delle grandi moli di dati (i *Big Data*) ed i recenti sviluppi nel campo dell'IA/*machine learning* - per analizzare i dati biologici e supportare la ricerca scientifica in vari campi [2].

In particolare, la metagenomica si occupa di studiare vari ambienti microbici, in cui un gran numero di specie di microorganismi (batteri, virus, archeobatteri, funghi e protisti) convivono [3]. Questi habitat nel loro complesso, inclusi i microorganismi, i loro genomi e l'ambiente circostante, vengono indicati col termine di *microbiomi*; l'insieme dei microorganismi costituisce il *microbiota*, mentre l'insieme del loro materiale genetico (genomi e geni) viene indicato con il termine *metagenoma* [4].

A differenza dell'approccio classico, basato sull'isolamento e la coltivazione in laboratorio di alcune specie, la metagenomica permette di cogliere l'effettiva diversità del microbiota, studiando anche quelle specie difficilmente coltivabili *in vitro*.

Campioni di microbiomi possono essere raccolti da varie fonti, tra cui ad esempio i microbiomi ambientali (come quelli presenti nel suolo e nell'acqua di mari, laghi e fiumi) e le comunità microbiche residenti all'interno del corpo di altri organismi ospiti, come ad esempio il microbioma intestinale di noi esseri umani. In particolare, il microbioma intestinale svolge un ruolo fondamentale nel regolare varie funzioni del nostro organismo, dalle quali dipende il nostro stato di salute [3]. Di conseguenza assume grande importanza lo sviluppo di strumenti che permettano di studiare il metagenoma di tali microbiomi ed indagare i collegamenti tra il microbioma e la salute umana. Il software *DeepVirFinder* [5],

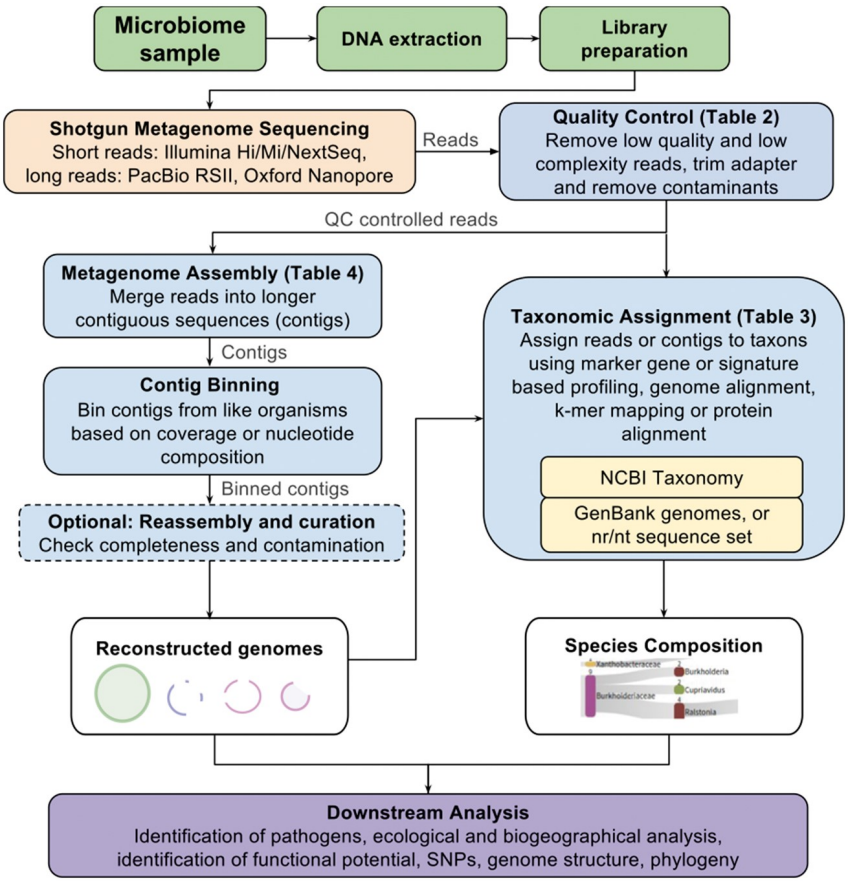

Figura 1.1: "Tipica procedura di analisi di dati metagenomici". Fonte [6] Fig. 1

che descriveremo dettagliatamente nel seguito, analizzandone le prestazioni, si inserisce proprio in questo contesto.

Il miglioramento delle tecnologie di sequenziamento di nuova generazione (*next-generation sequencing*, NGS) ha permesso un rapido sviluppo nel campo della metagenomica: è diventato possibile sequenziare, in poche ore e ad un costo contenuto, interi genomi. Nel corso del tempo, si sono affermati due principali approcci, che garantiscono un sequenziamento ad elevato *throughput*: la metagenomica basata su geni marcatori (*marker genes*) e quella basata su sequenziamento *whole-genome shotgun* (WGS) [3].

L'approccio basato su geni marcatori, che cronologicamente è stato il primo, consiste nel sequenziamento di specifiche regioni di diversi geni marcatori (geni 16S rRNA per i procarioti e geni 18S rRNA per gli eucarioti); le sequenze ricavate vengono poi assegnate a specifici gruppi tassonomici. Ciò permette di caratterizzare la composizione del campione di microbiota [3]. Questo approccio tuttavia non permette di studiare la componente virale dei microbiomi, in quanto non esistono geni marcatori universali per i virus [3, 4].

Nel sequenziamento WGS, invece, si procede al sequenziamento e all'analisi del materiale genetico di tutti gli organismi presenti nel campione. È così possibile cogliere l'effettiva

biodiversità del microbioma, inclusi gli archeobatteri, i batteri, i virus e gli eucarioti. I dati ottenuti da sequenziamento WGS sono più complessi, e permettono un'elevata risoluzione nella classificazione tassonomica [3].

Le piattaforme comunemente usate per il sequenziamento si distinguono per alcuni fattori, tra cui la lunghezza e la precisione delle letture. Alcune piattaforme, come Illumina, producono principalmente letture corte (150-300bp), mentre altre, come PacBio e Nanopore, producono letture più lunghe (in media con una lunghezza tra le 10k e le 30kbp). Tuttavia, il sequenziamento con Nanopore presenta tendenzialmente un tasso d'errore più elevato. Per questo motivo, la maggior parte dei dati metagenomici generati ad oggi derivano da letture corte ricavate da sequenziamento WGS [3].

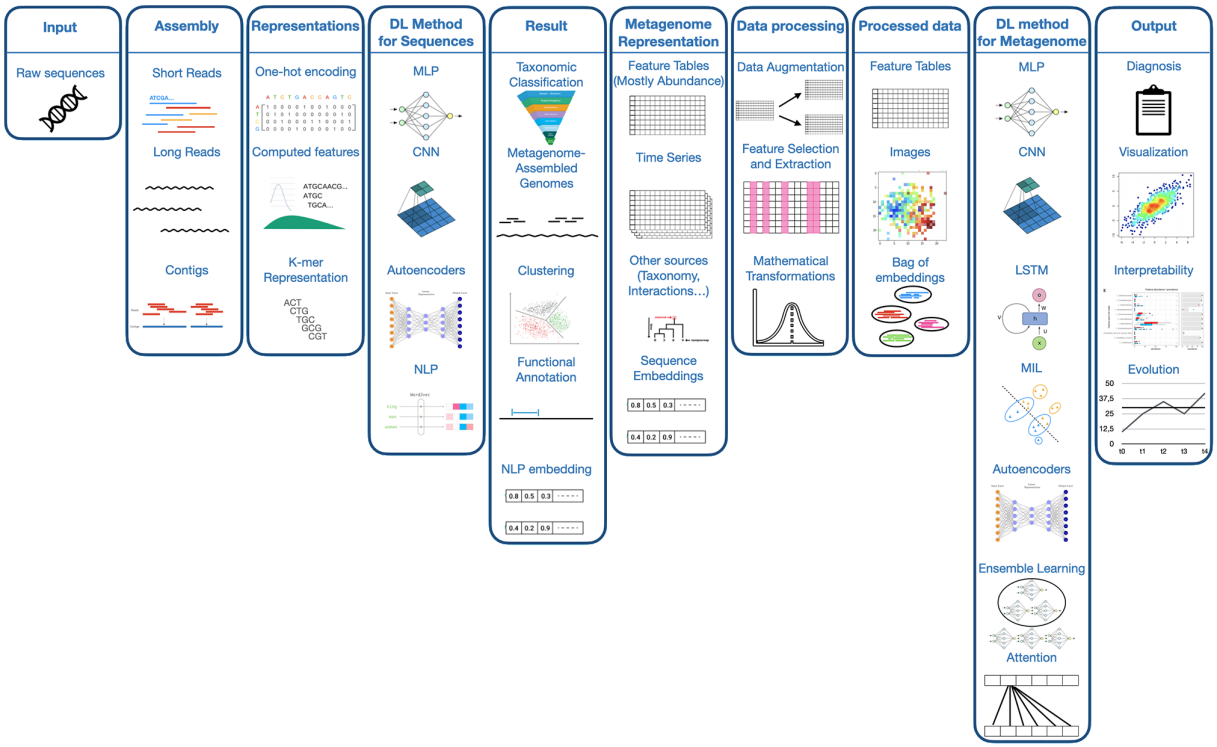

Figura 1.2: Panoramica di diversi metodi e approcci usati nell'analisi di dati metagenomici, dal sequenziamento alla diagnosi di malattie. Fonte [3] Fig. 7

Al giorno d'oggi, un singolo laboratorio di metagenomica produce quotidianamente enormi quantità di dati metagenomici (nell'ordine delle centinaia di Terabyte al mese); risulta quindi evidente la necessità di sviluppare soluzioni informatiche in grado di gestire dataset di tali dimensioni in modo efficiente. Il *deep learning* è uno dei principali strumenti a disposizione dei bioinformatici per affrontare queste sfide, e il gran numero delle sue applicazioni nel campo della metagenomica ne testimonia il potenziale [3, 7, 8].

L'obiettivo di questo elaborato è analizzare in particolare una di queste applicazioni, il software DeepVirFinder [5]: studiare l'architettura della sua rete neurale, capire il

funzionamento (almeno ad alto livello) degli script Python di cui si compone, usarlo per addestrare alcuni modelli su un nostro dataset, e analizzare le prestazioni di questi modelli.

Capitolo 2

Materiali e metodi

2.1 Deep learning

Il *deep learning* è una branca del *machine learning* che sfrutta reti neurali profonde (da cui l'attributo "deep"), composte da diversi strati (*layer*), per individuare automaticamente *feature* rilevanti nei dati in input, a differenza degli approcci tradizionali al machine learning, in cui spesso si richiede di progettare manualmente le feature da fornire in input agli algoritmi [9]. Ciò permette agli algoritmi di deep learning di individuare efficacemente pattern anche molto complessi all'interno dei dati grezzi.

Con il termine *feature* si intende una singola proprietà o caratteristica misurabile del fenomeno che si sta osservando. Le reti neurali profonde usate nel deep learning sono in grado, a partire dalle feature elementari (ad esempio i valori RGB dei singoli pixel di un immagine di un tessuto), di ricavare automaticamente feature più complesse (ad esempio la morfologia e l'organizzazione spaziale delle cellule del tessuto), ottenendo così un significativo miglioramento nell'accuratezza dei risultati prodotti dagli algoritmi [7].

Nel machine learning (e anche nel deep learning), si distinguono principalmente due approcci al *training* delle reti neurali: *supervised learning* (apprendimento supervisionato) e *unsupervised learning* (non supervisionato).

Nel supervised learning, ad ogni insieme di valori di input, detto *esempio* (nel caso di DeepVirFinder, una sequenza genomica), è associata una *label* (etichetta), che indica il valore corretto dell'output (nel caso di un compito di classificazione, ad esempio, indica la classe a cui appartiene l'esempio) [10]. Si consideri un dataset di esempi (X_i, Y_i), con X_i l'i-esimo input e Y_i l'i-esima label di output. Ogni *data point* X_i viene immesso nella rete neurale, e si valuta l'output rispetto alla label corretta Y_i, per calcolare una perdita (*loss*) $L(Y_i^{'}, Y_i)$. La funzione *loss* è la somma, su tutti gli esempi, dell'errore fatto nel

prevedere il valore di Y_i. Minore è il valore della *loss*, migliore è il modello. Esempi di funzioni *loss* sono l'errore quadratico e la cross-entropia, usate a seconda che l'output sia un valore continuo o una categoria [8].

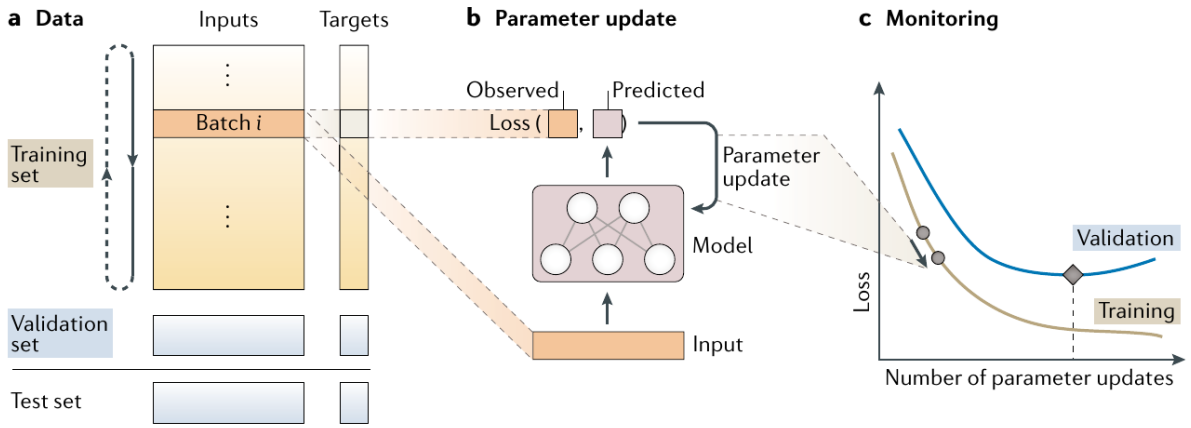

Figura 2.1: Training tramite apprendimento supervisionato. Fonte [7]

L'apprendimento supervisionato infatti è spesso usato per compiti di classificazione (in cui l'output appartiene a un insieme finito di valori possibili) e regressione (l'output può assumere valori in un intervallo continuo).

Nell'unsupervised learning, si chiede al modello di apprendere pattern dai dati in input senza alcun feedback. L'apprendimento non supervisionato è usato tipicamente per compiti di *clustering* [10].

La maggior parte delle reti neurali (*neural networks*, NN, in inglese) usate nel supervised learning appartengono a quattro classi: completamente connessa (*fully connected*), convoluzionale (*convolutional*), ricorrente (*recurrent*), convoluzionale a grafo (*graph convolutional*).

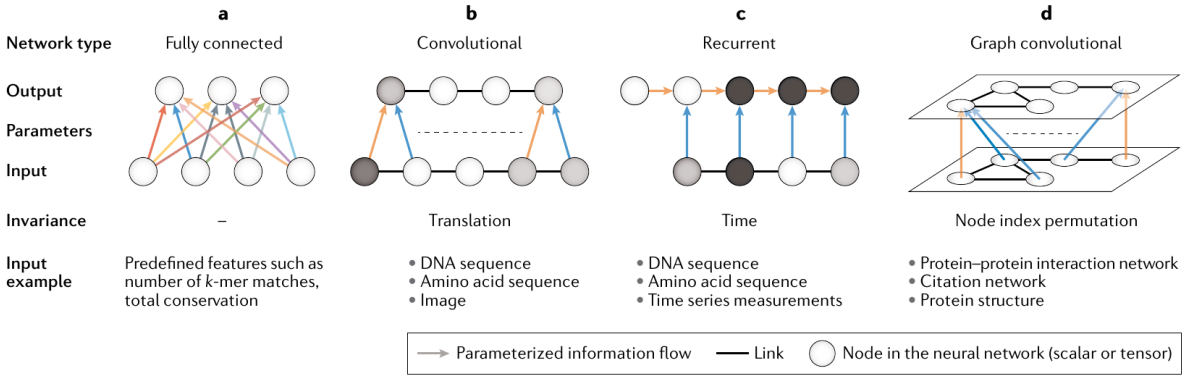

Figura 2.2: Quattro tipi di reti neurali. Fonte [7]

Le reti completamente (o densamente) connesse (dette anche *feed-forward*) sono l'architettura più semplice, in cui ogni neurone dello strato i è connesso solo a neuroni dello strato $i + 1$, e ogni neurone dello strato $i + 1$ riceve un input da tutti i neuroni dello strato i. Nell'architettura feed-forward si assume che non vi sono particolari relazioni o un ordine tra le feature in input, e quindi tutti i rami di connessione possono avere pesi (parametri) diversi [8].

Le reti convoluzionali (CNN) sono progettate per elaborare dati in cui ci si aspetta di trovare pattern invarianti rispetto alla posizione. Ogni neurone dello strato convoluzionale viene scansionato lungo la matrice di input, e in ogni posizione di questa matrice viene calcolata una somma pesata locale e prodotto un valore di output [8].

Le reti ricorrenti (RNN) sono progettate per elaborare dati appartenenti a serie temporali (organizzati in modo sequenziale). In ogni punto della sequenza, viene applicata una rete neurale (di tipo feed-forward o convoluzionale) per generare un valore (un segnale interno), che viene fornito come input al successivo step della rete neurale ricorrente. Il segnale interno può essere visto come una forma di memoria, che tiene conto delle informazioni ricavate dalla porzione di sequenza già elaborata dalla RNN [8].

Le reti convoluzionali a grafo (GCN) sono progettate per elaborare dati organizzati in strutture a grafo, ad esempio le reti di interazioni tra proteine, o le reti delle funzioni regolatrici dei geni [7].

2.2 DeepVirFinder

DeepVirFinder [5] è un software che sfrutta tecniche di *deep learning* per costruire modelli in grado di individuare sequenze virali. Data una sequenza campione tratta da dati metagenomici, il modello genera in output un valore tra 0 e 1, ove un valore più vicino a 1 indica maggiore probabilità che la sequenza in esame sia virale (è un esempio di classificatore binario, per cui $Y = 0$: sequenza ospite, $Y = 1$: sequenza virale).

DeepVirFinder sfrutta l'apprendimento supervisionato nel training dei propri modelli.

Prima di DeepVirFinder, altri metodi (e.g. VirFinder [11]) hanno usato le frequenze dei k-mer come feature per addestrare modelli di machine learning nell'identificazione di sequenze virali da dati metagenomici. La validità di metodi come VirFinder conferma l'assunto che vi sono differenze significative nella distribuzione dei k-mer tra virus e i loro ospiti procarioti.

Gli autori di DeepVirFinder hanno generalizzato i k-mer con il concetto di *motivi*. Ogni motivo viene rappresentato con una matrice, detta *position weight matrix* (PWM), di

dimensioni 4 x k, dove in ogni colonna sono riportate le probabilità in avere un nucleotide A/C/G/T in una data posizione. L'idea è che l'uso dei motivi permetta al modello una maggiore flessibilità e capacità di identificare pattern nei dati, migliorando l'accuratezza delle previsioni [5].

La rete neurale di DeepVirFinder è composta da un layer convoluzionale, un layer di *global max pooling*, un layer completamente connesso (detto anche layer "denso") e alcuni layer di *dropout*.

I filtri del layer convoluzionale sono rappresentati come matrici (*weight matrices*) di dimensione 4 x k, dove k è la lunghezza del filtro: rappresentazione analoga alle PWM. Ogni filtro corrisponde ad uno dei possibili motivi.

Una sequenza di DNA di lunghezza L, $X = X_1...X_l...X_L$, $X_l \in A, C, G, T$ viene codificata secondo la codifica detta *one-hot*, ottenendo una matrice 4 x L, $\boldsymbol{Z}^{(1)} = Z_1^{(1)}...Z_l^{(1)}...Z_L^{(1)}$, ove:

$$Z_l^{(1)} = \begin{cases} [1,0,0,0]^T, & \text{se } X_l = A \\ [0,1,0,0]^T, & \text{se } X_l = C \\ [0,0,1,0]^T, & \text{se } X_l = G \\ [0,0,0,1]^T, & \text{se } X_l = T \end{cases}$$

In caso di nucleotide ambiguo, "**N**", $Z_l^{(1)}$ viene codificato come $\left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]^T$.

La sequenza di DNA così codificata viene fornita come input al layer convoluzionale (con funzione di attivazione ReLU), il quale contiene M motivi di lunghezza K. Il motivo m-esimo può essere rappresentato con una matrice 4 x K, U_m, di coefficienti $U_{m,i,k}$, $i = 1, ..., 4$, $k = 1, ..., K$. Ogni motivo scansiona la sequenza $\boldsymbol{Z}^{(1)}$ dall'inizio alla fine, ottenendo una serie di valori dell'intensità del motivo, in funzione della posizione lungo la sequenza. L'intensità del motivo è calcolata come la cross-correlazione tra le sottosequenze di $\boldsymbol{Z}^{(1)}$ di lunghezza K e ciascun motivo. I risultanti valori di intensità per tutti gli M motivi possono essere raccolti in una matrice $\boldsymbol{Z}^{(2)}$ di dimensioni M x $(L - K + 1)$, ove:

$$Z_{m,l}^{(2)} = \sum_{i=1}^{4} \sum_{k=1}^{K} Z_{i,l+k-1}^{(2)} U_{m,i,k}$$

A ogni motivo viene applicata la funzione ReLU (rectified linear unit), ottenendo la matrice $\boldsymbol{Z}^{(3)}$ delle stesse dimensioni di $\boldsymbol{Z}^{(2)}$, ove $Z_{m,l}^{(3)} = \max(0, Z_{m,l}^{(2)})$.

La funzione ReLU, definita come $\text{ReLU}(x) = \max(0, x)$, è una delle funzioni comunemente usate come funzioni di attivazione nelle reti neurali, in quanto permette alle reti neurali

di apprendere relazioni non lineari tra feature e label [12].

La matrice $\boldsymbol{Z}^{(3)}$ viene passata al layer di *max pooling*, il quale per ogni motivo conserva solo il valore di intensità massima, riducendo le dimensioni della matrice a M x 1: $Z_m^{(4)} = \max(Z_{m,1}^{(3)}, ..., Z_{m,L-K+1}^{(3)})$.

La matrice $\boldsymbol{Z}^{(4)}$ viene data in input al layer denso contenente N neuroni completamente connessi, ove l'n-esimo neurone ha vettore dei pesi W_n e bias b_n. La matrice restituita in output, $\boldsymbol{Z}^{(5)}$, ha dimensioni N x 1, ove:

$$Z_n^{(5)} = b_n + \sum_{m=1}^{M} W_{n,m} Z_m^{(4)}$$

A ogni neurone viene quindi applicata la funzione di attivazione ReLU, ottenendo la matrice $\boldsymbol{Z}^{(6)}$, $Z_n^{(6)} = \max(0, Z_n^{(5)})$, che viene infine sintetizzata attraverso un layer denso con funzione di attivazione sigmoide per generare un valore tra 0 e 1 (0 = sequenza procariote, 1 = sequenza virale): $Z^{(7)} = \sigma(z)$ con $z = q + \sum_{n=1}^{N} V_n Z_n^{(6)}$.

La funzione sigmoide, $\sigma(z) = \frac{1}{1+exp\{-z\}}$, è un'altra funzione di attivazione comunemente utilizzata, in quanto permette di mappare un qualsiasi numero reale a un valore compreso nell'intervallo $[0, 1]$ [12].

Riassumendo, fornendo una sequenza $\mathbf{X}$ in input alla rete neurale, si ottiene un valore Y dato da: $Y(\mathbf{X}) = \sigma(\text{Dense}(\text{Dense}(\text{MaxPool}(\text{Conv}(\text{Encode}(\mathbf{X}))))))$.

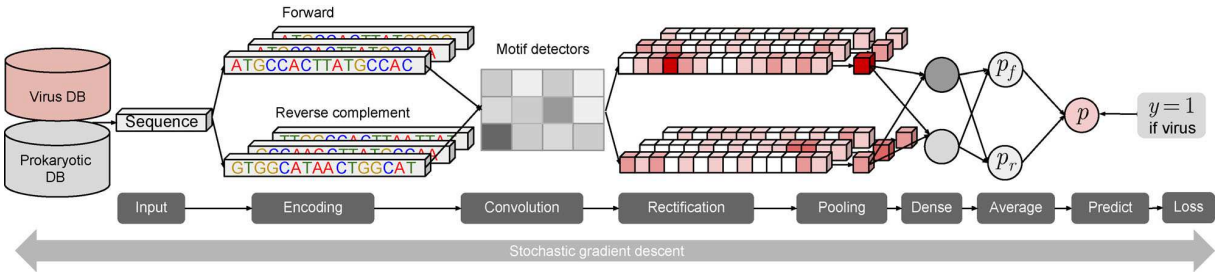

Figura 2.3: Architettura di DeepVirFinder. Fonte [5]

Tuttavia, è necessario tenere in considerazione che i frammenti di DNA hanno struttura a doppia elica, e che il materiale genetico sequenziato può derivare da entrambe le eliche; perciò il valore della previsione ottenuto per una sequenza ($\mathbf{X}_F$) dovrebbe essere uguale a quello per il suo complemento inverso ($\mathbf{X}_R$). Il modello prevede quindi di applicare la stessa rete neurale anche al complemento inverso di ciascuna stringa, e il valore finale della previsione è dato dalla media delle due previsioni:

$$Y_{\text{final}} = \frac{Y(\mathbf{X}_F) + Y(\mathbf{X}_R)}{2}$$

La funzione obiettivo è minimizzare la perdita (*loss*) di cross-entropia binaria tra il valore previsto Y_{final} e le label corrette.

Durante il training, i parametri dei layer della rete neurale vengono aggiornati attraverso *back-propagation* secondo l'algoritmo di ottimizzazione di Adam per la discesa stocastica del gradiente (*stochastic gradient descent*), con tasso di apprendimento (*learning rate*) pari a 0.001. Inoltre viene applicata una regolarizzazione di tipo *dropout*, con tasso 0.1, dopo il layer di max pooling e il layer densamente connesso, al fine di ridurre l'overfitting della rete neurale.

Il termine *overfitting* indica la condizione in cui i parametri di una rete neurale si adattano troppo bene ai dati di training, peggiorando la capacità di generalizzazione della rete stessa. Tale condizione può presentarsi quando una rete neurale viene allenata per troppe iterazioni (epoche) sullo stesso dataset. Per monitorare il processo di training e individuare una possibile condizione di overfitting, a ogni iterazione si calcola una metrica per valutare l'errore di generalizzazione, usando un dataset separato (detto di *validazione*); tipicamente, dopo un certo numero di iterazioni, la funzione obiettivo (la *loss function*) assume un andamento asintotico, mentre l'errore di generalizzazione può iniziare a peggiorare; il tal caso, il training viene interrotto (si parla di *early stopping*) [12].

Per ridurre la tendenza di una rete neurale ad andare in overfitting, è possibile applicare una strategia di *regolarizzazione*, ovvero apportare delle modifiche all'algoritmo di training del modello (modificando la *loss function* oppure la procedura di ottimizzazione dei parametri) al fine di ridurre l'errore di generalizzazione. Un esempio classico è la regolarizzazione di tipo L^2, che prevede di modificare la loss function sommandovi i quadrati dei valori dei parametri del modello: ciò introduce una preferenza nell'algoritmo di training, che sarà indirizzato a scegliere parametri con valore assoluto più piccolo [9]. Un altro esempio di regolarizzazione è il metodo Dropout [13] (usato anche da DeepVirFinder), che prevede di rimuovere casualmente alcuni neuroni dalla rete neurale durante la fase di training. Ciò permette di ridurre sensibilmente la tendenza all'overfitting delle reti neurali anche molto grandi.

La rete neurale di DeepVirFinder ha quattro iperparametri principali: la lunghezza dei filtri (motivi), il numero dei filtri nello strato convoluzionale, il numero di neuroni densamente connessi, e il numero di epoche nel training. I primi tre determinano la complessità del modello, mentre l'ultimo controlla il passaggio da underfitting a overfitting.

Il codice sorgente di DeepVirFinder è disponibile all'indirizzo:
`http://github.com/jessieren/DeepVirFinder`

Il software si compone principalmente di tre script Python: `encode.py`, `training.py` e `dvf.py`. Gli script `encode.py` e `training.py` servono per eseguire il training di modelli,

anche con propri dataset. Lo script `dvf.py`, sfruttando i modelli prodotti durante la fase di training, produce per ogni sequenza in input una previsione $Y \in [0, 1]$.

2.2.1 Script `encode.py`

Lo script `encode.py` riceve in input un file in formato FASTA (parametro `-i FILENAME`), contenente le sequenze genomiche da elaborare. Frammenta le sequenze in sotto-sequenze di lunghezza fissata (parametro `-l CONTIGLENGTH`); per ciascuna sequenza così ottenuta calcola il complemento inverso, e codifica entrambe secondo la codifica one-hot descritta sopra. Un ulteriore parametro (`-p CONTIGTYPE`) specifica se l'insieme di sequenze elaborate appartengono a host procarioti o a virus. Questa informazione sarà necessaria successivamente nella fase di training.

Le sequenza codificate sono inserire in due matrici (di tipo array di NumPy), una contenente le sequenze originali (*forward sequences*) e una i complementi inversi (*backward sequences*), e salvate in corrispondenti file sul filesystem, per essere utilizzate successivamente dallo script `training.py`.

2.2.2 Script `training.py`

Lo script `training.py`, sfruttando le sequenze codificate precedentemente, esegue il training di un modello secondo i parametri e gli iperparametri specificati. Parametri di `training.py`:

- `-l CONTIGLENGTH` lunghezza delle sequenze (*contig*)
- `-i INDIRTR` input directory per il dataset di training
- `-j INDIRVAL` input directory per il dataset di validazione
- `-o OUTDIR` output directory per i modelli
- `-f FILTER_LEN1` lunghezza dei filtri
- `-n NB_FILTER1` numero di filtri nel layer convoluzionale
- `-d NB_DENSE` numero di neuroni nel layer denso
- `-e EPOCHS` numero di epoche

Lo script riceve in input i dataset codificati di training e validazione, e salva i modelli generati nella cartella specificata. È necessario specificare la lunghezza delle sequenze tramite il parametro `-l CONTIGLENGTH`, che deve essere la stessa per i dati di training e

di validazione. Poiché nei casi di studio reali le sequenze genomiche ottenute dal sequenziamento sono di lunghezza variabile, è opportuno eseguire più volte il training con valori di lunghezza diversi, ad esempio 150, 300, 500, 1000 bp. Nel seguito si confronteranno le prestazioni di modelli addestrati con sequenze di lunghezza diversa.

Gli iperparametri principali del modello vengono specificati tramite le rispettive opzioni: la lunghezza (`-f FILTER_LEN1`) e il numero (`-n NB_FILTER1`) dei filtri dello strato convoluzionale, il numero di neuroni dello strato denso (`-d NB_DENSE`), il numero di epoche (`-e EPOCHS`) per cui eseguire il training.

La costruzione del modello è implementata nelle seguenti righe di codice:

```
print("...building model...")
## if model exists
if os.path.isfile(modName):
    model = load_model(modName)
    print("...model exists...")
else :
    ## siamese
    forward_input = Input(shape=(None, channel_num))
    reverse_input = Input(shape=(None, channel_num))
    hidden_layers = [
        Conv1D(filters = nb_filter1, kernel_size = filter_len1,
           activation='relu'),
        GlobalMaxPooling1D(),
        Dropout(dropout_pool),
        Dense(nb_dense, activation='relu'),
        Dropout(dropout_dense),
        Dense(1, activation='sigmoid')
    ]
    forward_output = get_output(forward_input, hidden_layers)
    reverse_output = get_output(reverse_input, hidden_layers)
    output = Average()([forward_output, reverse_output])
    model = Model(inputs=[forward_input, reverse_input], outputs=output)
    model.compile(Adam(lr=learningrate), 'binary_crossentropy', metrics
       =['accuracy'])
```

2.2.3 Script `dvf.py`

Lo script `dvf.py`, ricevendo in input un file (in formato FASTA) contenente sequenze genomiche, genera un file di output (`.txt`), in cui ogni riga riporta la descrizione di una sequenza e la previsione corrispondente (un valore tra 0 e 1, come descritto sopra).

A differenza di `training.py`, lo script `dvf.py` non richiede che il file FASTA venga precedentemente codificato tramite `encoding.py`, ma esegue al suo interno la codifica delle sequenze lette. La codifica segue il metodo one-hot descritto sopra.

Parametri di `dvf.py`:

- `-i INPUT_FA` input file (formato FASTA)
- `-m MODDIR` directory contenente i modelli (default `./models`)
- `-o OUTPUT_DIR` output directory
- `-l CUTOFF_LEN` esegue previsioni solo per sequenze di lunghezza >= L bp (default 1)
- `-c CORE_NUM` numero di core per esecuzione in parallelo (default 1)

Lo script sfrutta i modelli presenti nella directory specificata per eseguire le previsioni. Nella versione del software attualmente scaricabile da GitHub, lo script è configurato per usare modelli diversi (allenati su sequenze di lunghezza diversa: 150, 300, 500, 1000bp) in base alla lunghezza della sequenza di cui deve calcolare la previsione:

```
def pred(ID) :
    codefw = code[ID]
    codebw = codeR[ID]
    head = seqname[ID]
    seqL = len(codefw)

    if seqL < 300 :
        model = modDict['0.15']  # 0.15 corrisponde a 150bp
        null = nullDict['0.15']
    elif seqL < 500 and seqL >= 300 :
        model = modDict['0.3']  # 0.3 corrisponde a 300bp
        null = nullDict['0.3']
    elif seqL < 1000 and seqL >= 500 :
        model = modDict['0.5']  # 0.5 corrisponde a 500bp
        null = nullDict['0.5']
    else :
        model = modDict['1']  # 1 corrisponde a 1000bp
        null = nullDict['1']
    ...
```

Da questo frammento di codice, si può vedere che lo script `dvf.py` usa:

- per le sequenze minori di 300bp, il modello allenato su sequenze di 150bp
- per le sequenze minori di 500bp e $\geqslant$ 300bp, il modello allenato su sequenze di 300bp
- per le sequenze minori di 1000bp e $\geqslant$ 500bp, il modello allenato su sequenze di 500bp
- per le sequenze $\geqslant$ 1000bp, il modello allenato su sequenze di 1000bp

Capitolo 3

Esperimenti e risultati

Gli esperimenti, che saranno qui descritti, sono stati eseguiti tramite il cluster di calcolo BLADE del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova.

Il software di DeepVirFinder è stato installato secondo le istruzioni presenti alla rispettiva pagina GitHub.

Nei primi esperimenti, in analogia a quanto fatto dagli autori di DVF, si è cercato di determinare i valori ottimali per gli iperparametri del modello (lunghezza e numero dei filtri, numero dei neuroni densamente connessi).

3.1 Dataset

I dataset utilizzati nei nostri esperimenti sono un sottocampionamento del dataset usato dagli autori di DeepVirFinder [5]. Ren et al. hanno scaricato dal database del *National Center for Biotechnology Information* (NCBI) 2'314 genomi di riferimento (detti *RefSeq*) di virus che infettano procarioti (batteri e archeobatteri). Questi genomi sono stati suddivisi in sequenze (non sovrapposte) di varia lunghezza (150, 300, 500, 1000, 3000bp) e raggruppati in tre distinti dataset: dataset di training (genomi scoperti prima di gennaio 2014), dataset di validazione (tra gennaio 2014 e maggio 2015), dataset di test (dopo maggio 2015).

Alle sequenze virali sono state affiancate sequenze di procarioti, frammentate a partire da 38'234 genomi RefSeq, suddivisi in tre dataset secondo lo stesso schema (training, validazione e test).

I dataset usati da Ren et al. comprendono diversi file FASTA di enorme dimensione. Al fine di ottenere un campione più piccolo per i nostri esperimenti, per ciascuno dei file FASTA originali sono state mantenute solo il primo milione di righe (in alcuni casi, i

file FASTA originali contenevano meno di un milione di righe, e quindi non sono stati troncati).

3.2 Misure di prestazione

Prima di descrivere nel dettaglio gli esperimenti fatti, occorre dare una definizione delle misure di prestazione usate per valutare i modelli.

Nel caso di un classificatore binario, come per DeepVirFinder, si vuole mappare l'output Y della rete neurale a una tra due possibili classi, tipicamente indicate con i termini *positiva* ($Y = 1$, la classe che si sta cercando, nel nostro caso le sequenze virali) e *negativa* ($Y = 0$, l'altra classe, nel nostro caso le sequenze procarioti).

Nella maggior parte dei casi, l'output Y della rete neurale è un valore continuo compreso tra 0 e 1 (interpretabile come la probabilità che un dato esempio appartenga alla classe positiva); perciò è necessario stabilire una *soglia* (T, *threshold*), tale che tutti gli esempi per cui $Y_i \geqslant T$ vengono classificati nella classe positiva, mentre tutti quelli per cui $Y_i < T$ nella classe negativa [14].

La scelta del valore di soglia ha importanti ripercussioni sul calcolo delle metriche di prestazione. Fissata una soglia, è possibile costruire una matrice 2x2, detta *matrice di confusione*, in cui raccogliere tutti gli output del classificatore binario. Lungo le colonne si ha la verità effettiva (*ground truth*), lungo le righe la previsione del classificatore:

	Actual positive	Actual negative
Predicted positive	**True positive (TP)**	**False positive (FP)**
Predicted negative	**False negative (FN)**	**True negative (TN)**

Se il totale dei positivi effettivi è simile al totale dei negativi effettivi, si dice che il dataset è *bilanciato*.

Differenti valori di soglia determinano differenti proporzioni di true positives, true negatives, false positives e false negatives.

I valori di TP, FP, TN, FN sono usati per calcolare diverse metriche (accuratezza, *recall*, precisione, *false positive rate*), al fine di valutare le prestazioni di un classificatore (fissato un certo valore di soglia). A seconda del modello, del compito e del dataset specifico, alcune metriche possono essere più significative di altre.

3.2.1 Accuratezza (*accuracy*)

L'accuratezza è la proporzione di tutti gli esempi correttamente classificati, sia positivi che negativi [15]:

$$\text{Accuratezza} = \frac{\text{classificazioni corrette}}{\text{classificazioni totali}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Nel caso di DeepVirFinder, l'accuratezza indica la frazione di tutte le sequenze correttamente classificate (virus o procarioti).

L'accuratezza è spesso usata come un indicatore grossolano della qualità di un modello, ad esempio per seguire il processo di training del modello stesso. Tuttavia, è fortemente consigliato unire all'accuratezza anche un'altra (almeno) delle metriche di seguito descritte, per ottenere una valutazione più completa.

3.2.2 Precisione (*precision*)

La precisione è la proporzione di tutti gli esempi classificati come positivi che erano effettivamente positivi [15]:

$$\text{Precisione} = \frac{\text{effettivi positivi classificati correttamente}}{\text{tutti gli esempi classificati positivi}} = \frac{TP}{TP + FP}$$

3.2.3 *Recall* o *true positive rate*

Il *true positive rate* (TPR), noto anche come *recall* o *sensitivity*, è la proporzione di tutti gli effettivi positivi correttamente classificati come positivi [15]:

$$\text{Recall (o TPR)} = \frac{\text{effettivi positivi classificati correttamente}}{\text{tutti gli effettivi positivi}} = \frac{TP}{TP + FN}$$

Il true positive rate è a volte indicato come probabilità di rilevamento (*probability of detection*).

3.2.4 *False positive rate*

Il *false positive rate* (FPR) è la proporzione di tutti gli effetti negativi classificati erroneamente come positivi [15]:

$$\text{FPR} = \frac{\text{effettivi negativi classificati erroneamente}}{\text{tutti gli effettivi negativi}} = \frac{FP}{FP + TN}$$

Il false positive rate è a volte chiamato probabilità di un falso allarme.

3.2.5 Specificità (*specificity*)

La specificità (*specificity*) è la proporzione di tutti gli effettivi negativi correttamente classificati come negativi:

$$\text{Specificity} = \frac{\text{effettivi negativi classificati correttamente}}{\text{tutti gli effettivi negativi}} = \frac{TN}{TN + FP}$$

3.2.6 F1 score

La *F1 score* è definita come la media armonica tra la precisione e il recall (TPR) [15]:

$$\text{F1} = \frac{2 \times \text{precisione} \times \text{recall}}{\text{precisione} \times \text{recall}} = \frac{2\text{TP}}{2\text{TP} + FP + FN}$$

La F1 score viene a volte usata per descrivere le prestazioni di un classificatore su un dataset sbilanciato, in quanto combina le metriche di precisione e recall. Quando entrambe hanno un valore perfetto di 1.0, anche F1 tende a 1.0. In generale, quando precisione e recall assumono valori vicini tra loro, la F1 score tende ai loro valori; quando invece le due metriche hanno valori molto distanti tra loro, F1 tende al valore peggiore tra i due.

3.2.7 AUROC (*Area under the ROC curve*)

Come già evidenziato, le metriche sopra descritte (accuratezza, TPR, FPR, precisione, etc.) dipendo dal valore di soglia scelto. Spesso però si ha la necessità di confrontare le prestazioni di due modelli rispetto a tutti i possibili valori di soglia.

Le curve ROC (*receiver-operating characteristic curve*) forniscono una rappresentazione grafica delle prestazioni di un modello rispetto a tutte le possibili soglie.

In pratica, si suddivide il range dei possibili valori di soglia in intervalli piccoli a piacere; per ogni valore di soglia si calcolano il true positive rate (TPR) e il false positive rate (FPR), quindi si plottano le coppie di valori (FPR, TPR), con FPR sull'asse delle ascisse e TPR sull'asse delle ordinate [16].

L'area sottesa da una curva ROC, nota come AUROC (*Area under the ROC curve*) o semplicemente AUC, rappresenta la probabilità che il corrispondente modello, ricevuti due esempi, uno positivo e uno negativo, scelti casualmente, assegni all'esempio positivo un valore Y maggiore rispetto che all'esempio negativo [16].

I valori di AUROC sono quindi compresi tra 0 (classificatore pessimo, che sbaglia tutte le previsioni) e 1 (classificatore perfetto). Maggiore è l'AUROC, migliore è il corrispondente modello. Si noti che il caso con AUROC = 0.5 corrisponde ad un classificatore che assegna casualmente un esempio ad una delle due classi (come tirare una moneta).

Nel seguito, adotteremo l'AUROC come metrica di riferimento per valutare le prestazioni dei modelli di DeepVirFinder da noi addestrati.

3.3 Esperimento: determinazione del valore ottimale della lunghezza dei filtri

Nel primo esperimento l'obiettivo era cercare il valore ottimale per il parametro `-f FILTER_LEN1` (lunghezza dei filtri/motivi, di seguito indicato con l'abbreviazione fl), che insieme al numero dei filtri nello strato convoluzionale e al numero di neuroni densi determinano la complessità del modello.

Si sono addestrati vari modelli: per ognuna delle cinque *contig length* (150, 300, 500, 1000, 3000bp) si è variata la lunghezza dei filtri da 4 a 15 (con numero di filtri e numero di neuroni densi fissati a 1000), e si sono valutate le prestazioni dei modelli ottenuti calcolando l'AUROC sui dataset di training e validazione. I risultati ottenuti sono riassunti nella figura [3.1].

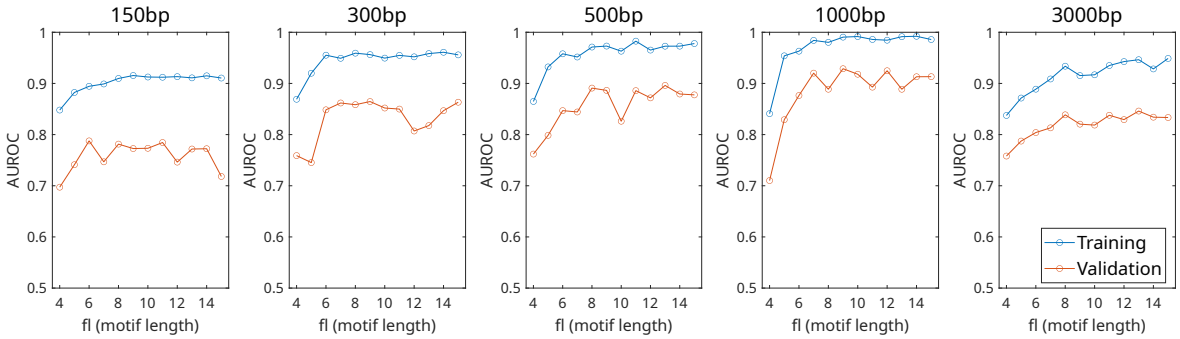

Figura 3.1: Risultati

In generale, si può notare un iniziale rapido miglioramento dell'AUROC di validazione all'aumentare della lunghezza dei filtri da 4 a 8. L'AUROC di validazione raggiunge un

massimo per valori di fl tra 8 e 10, per poi mantenersi a valori più o meno costanti al successivo aumentare di fl.

I valori di AUROC da noi ottenuti sul dataset di validazione, tuttavia, risultavano avere un andamento altalenante: ad esempio, nei casi di sequenze di 500 e 1000bp si vede come l'AUROC di validazione, dopo un primo massimo locale in corrispondenza di $fl = 8$ (500bp, AUROC $= 0.8908$) e $fl = 7$ (1000bp, AUROC $= 0.9200$), presenta numerosi altri picchi e flessioni nel suo andamento al crescere di fl.

I risultati ottenuti dagli autori di DeepVirFinder, invece, mostravano un andamento dell'AUROC di validazione qualitativamente diverso, come è possibile vedere nella figura [3.2].

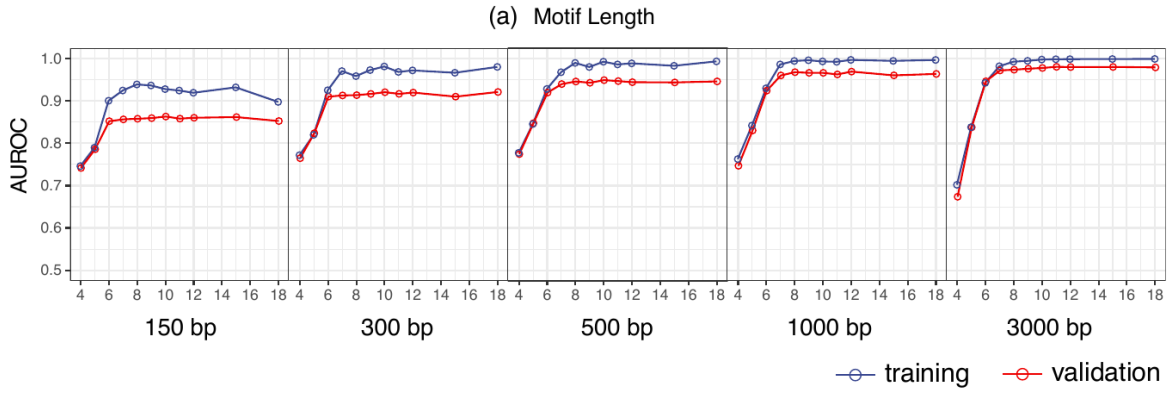

Figura 3.2: "Effetto della lunghezza dei motivi sull'AUROC di validazione". Fonte [5] Supplementary Fig. S1A

Si è quindi deciso di ripetere l'esperimento, considerando i casi con $fl = 8, 9, 10, 11, 12$, ottenendo i risultati in figura [3.3], che evidenziano un andamento più simile a quelli della figura [3.2].

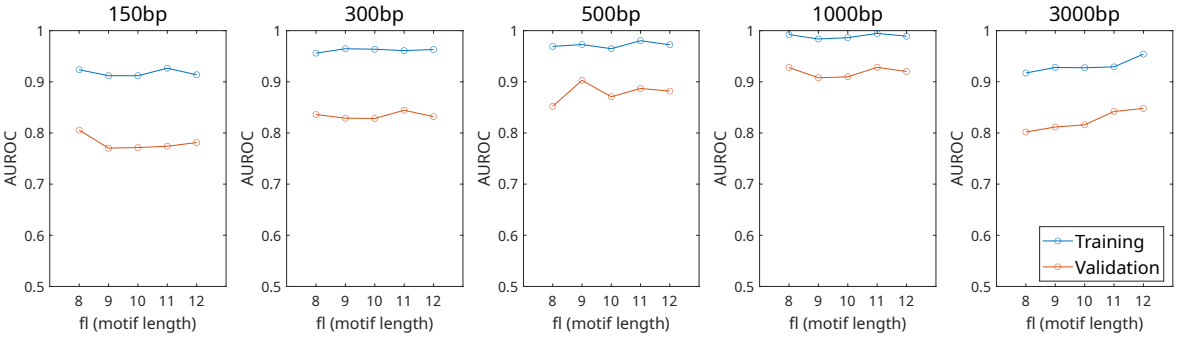

Figura 3.3: Risultati

Per ciascuna delle cinque *contig length* (150, 300, 500, 1000, 3000bp), si vede che il massimo valore di AUROC di validazione è associato a valori di fl diversi:

- 150bp: max AUROC = 0.8056 per $fl = 8$
- 300bp: max AUROC = 0.8443 per $fl = 11$
- 500bp: max AUROC = 0.9028 per $fl = 9$
- 1000bp: max AUROC = 0.9281 per $fl = 11$
- 3000bp: max AUROC = 0.8479 per $fl = 12$

Considerando nel complesso gli andamenti per le cinque *contig length*, si è deciso di fissare $fl = 11$, in quando anche per le sequenze di 150, 500 e 3000bp l'AUROC di validazione, calcolata per $fl = 11$, assume un valore vicino al rispettivo massimo. In particolare, si sono ottenuti i valori di AUROC di validazione: 0.774097, 0.844374, 0.886926, 0.928169 e 0.842045 per le sequenze di lunghezza 150, 300, 500, 1000, 3000bp rispettivamente.

3.4 Esperimento: determinazione del valore ottimale del numero di filtri e di neuroni densi

Nel secondo esperimento, l'obiettivo era determinare il valore ottimale per altri due iperparametri del modello, il numero di filtri (nb) e il numero di neuroni densi. In analogia con quanto fatto da Ren et al. [5], si è deciso di testare configurazioni con numero di filtri uguale al numero di neuroni densi, variando nb da 100 a 1500. Per comprendere meglio la dipendenza dell'AUROC di validazione dal valore dei tre iperparametri, si è deciso di ripetere il training dei modelli per $fl = 9, 10, 11$ (e per ciascuna delle cinque *contig length*). I risultati ottenuti sono visibili in figura [3.4].

I risultati mostrano un generale miglioramento dell'AUROC all'aumentare di nb, anche se in alcuni casi l'AUROC per $nb = 1500$ risulta minore dell'AUROC per $nb = 1000$. Concentrandoci sui risultati ottenuti con i modelli per cui $fl = 11$ (valore che si era scelto dopo il primo esperimento), possiamo notare che:

- per le sequenze di 150 e 300bp, l'AUROC di validazione ottenuta per $nb = 1000$ migliora nettamente rispetto al caso con $nb = 500$, mentre risulta praticamente identica (150bp) o migliore (300bp) dell'AUROC ottenuta per $nb = 1500$;
- per le sequenze di 500bp, l'AUROC di validazione si mantiene pressoché allo stesso valore per tutti i valori di nb;
- per le sequenze di 1000 e 3000bp, si è ottenuto il massimo valore di AUROC in corrispondenza di $nb = 500$, mentre l'AUROC per $nb = 1000$ risulta leggermente minore.

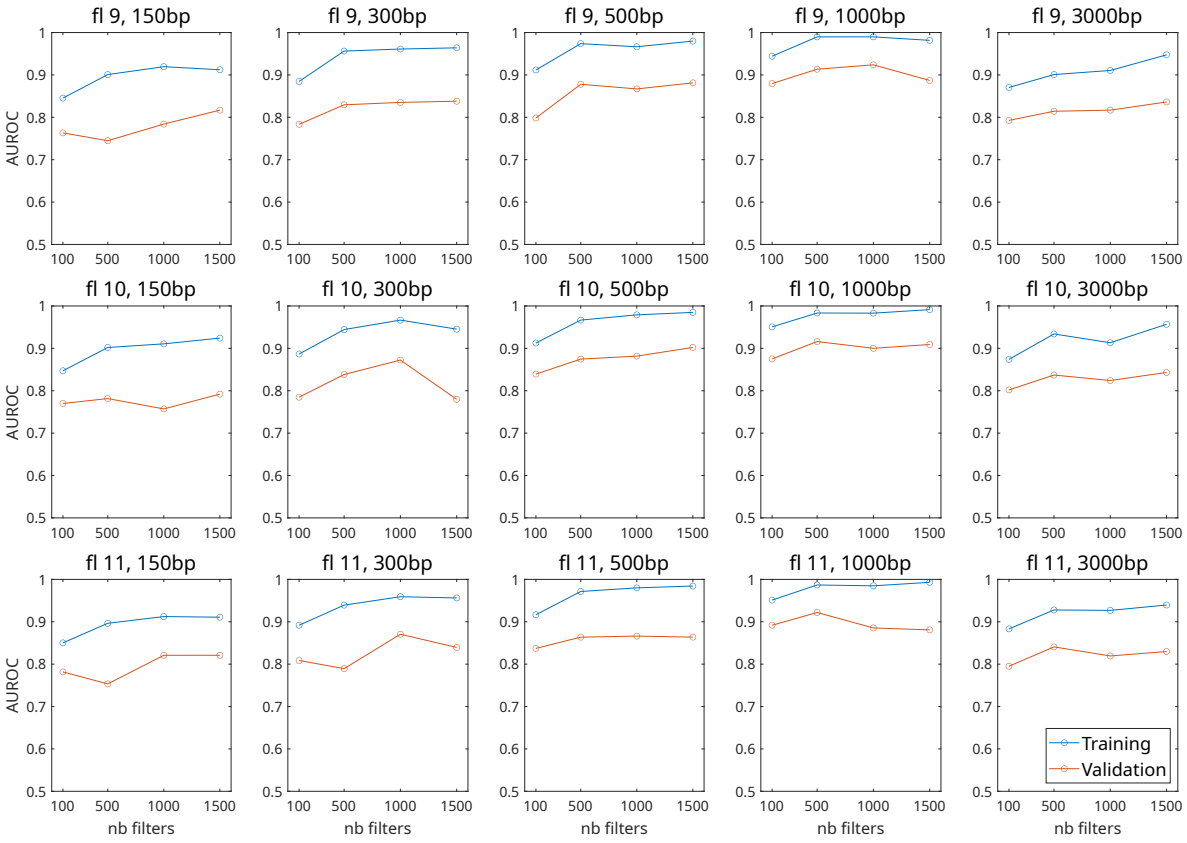

Figura 3.4: Risultati

Da una prima lettura dei risultati emergono quindi due possibili valori (500 e 1000) da poter assegnare al parametro nb per ottimizzare il modello. In considerazione del fatto che l'AUROC di validazione ottenuta per sequenze di 150 e 300bp con $nb = 500$ risultava minore di 0.8 (valore abbastanza basso), si è scelto di fissare $nb = 1000$, al fine di non penalizzare eccessivamente i modelli addestrati sulle sequenze più corte.

3.5 Esperimento: valutazione delle prestazioni per sequenze di lunghezza diversa

Mentre nei primi due esperimenti, l'AUROC di validazione era stata calcolata su un insieme di sequenze della stessa lunghezza di quelle usate per addestrare il corrispondente modello (ad esempio, il modello addestrato su sequenze di 150bp era stato valutato solo su sequenze di 150bp), nel terzo esperimento si è voluto misurare la capacità di un modello di classificare sequenze di lunghezza diversa da quella delle sequenze su cui è stato addestrato. Nei casi reali, i campioni metagenomici possono contenere sequenze di lunghezze molto diverse tra loro, e risulta quindi importante stabilire se esiste una correlazione tra le

lunghezze delle sequenze usate per il training, la lunghezza della sequenza in esame e la qualità della previsione generata dal modello.

Come primo passo, si sono addestrati cinque modelli, ciascuno con sequenze di una lunghezza fissata (150, 300, 500, 1000, 3000bp). Le sequenze usate per addestrare i modelli sono state ricavate dall'unione dei dataset di training e di validazione (RefSeq scoperti entro maggio 2015, come descritto nella sezione 3.1). Il dataset di test (RefSeq successivi a maggio 2015) è stato usato per calcolare i punteggi di AUROC dei vari modelli. Ogni modello è stato testato su sequenze di lunghezza diversa (150, 300, 500, 1000, 3000bp), ottenendo in totale 25 valori di AUROC. I risultati sono riportati in tabella e riassunti in figura [3.5].

	Test contig length						
	150bp	300bp	500bp	1000bp	3000bp		
AUROC	0.8243	0.8648	0.8713	0.8711	0.8823	150bp	Train contig length
	0.8210	0.9075	0.9343	0.9456	0.9341	300bp	
	0.7902	0.8907	0.9336	0.9562	0.9593	500bp	
	0.7453	0.8451	0.9051	0.9517	0.9696	1000bp	
	0.8147	0.8845	0.9129	0.9264	0.9154	3000bp	

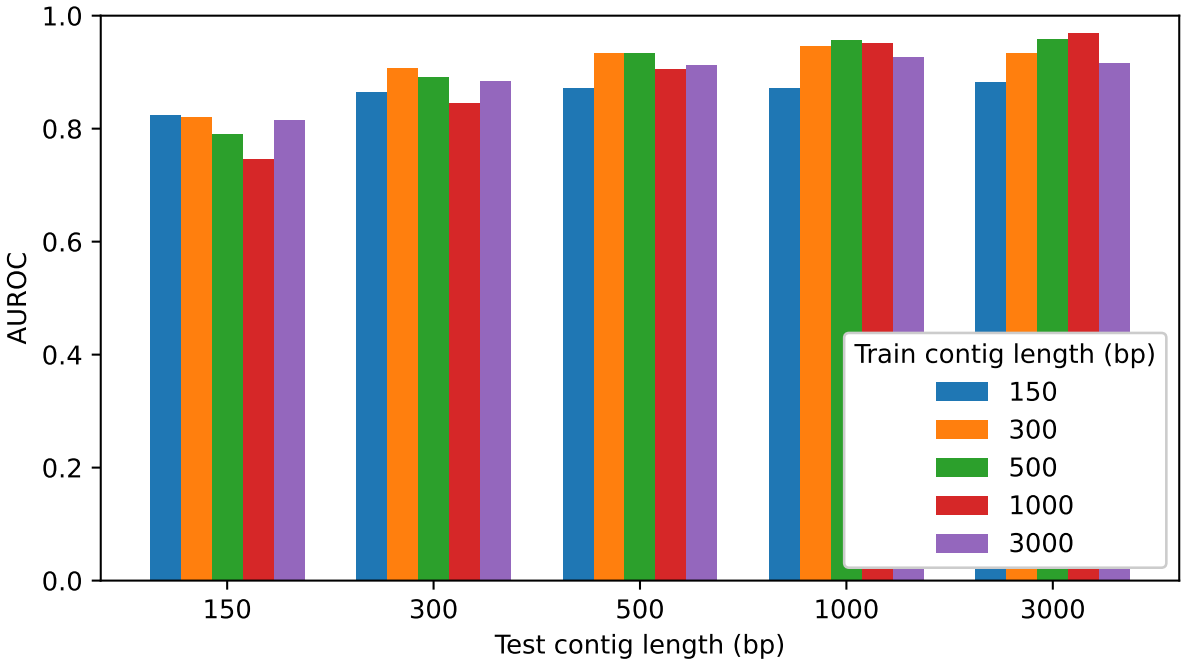

Figura 3.5: Risultati

Nel testare i modelli su sequenze di 150bp, il modello con l'AUROC più elevata (0.8243) è risultato essere quello addestrato su sequenze di 150bp, anche se di poco migliore rispetto

al modello addestrato su quelle di 300bp (AUROC 0.8210). Per le sequenze di 300bp, le prestazioni migliori si sono ottenute con il modello addestrato su sequenze di 300bp (AUROC 0.9075). Per le sequenze di 500bp, i modelli addestrati su sequenze di 300 e 500bp si sono dimostrati praticamente equivalenti (AUROC 0.9343 e 0.9336 rispettivamente). Interessante notare che in questi tre casi, i modelli addestrati su sequenze di 3000bp si sono posizionati in terza posizione, mentre nei risultati ottenuti da Ren et al. [5] (si veda la figura [3.6]) questi modelli avevano ottenuto i valori di AUROC più bassi.

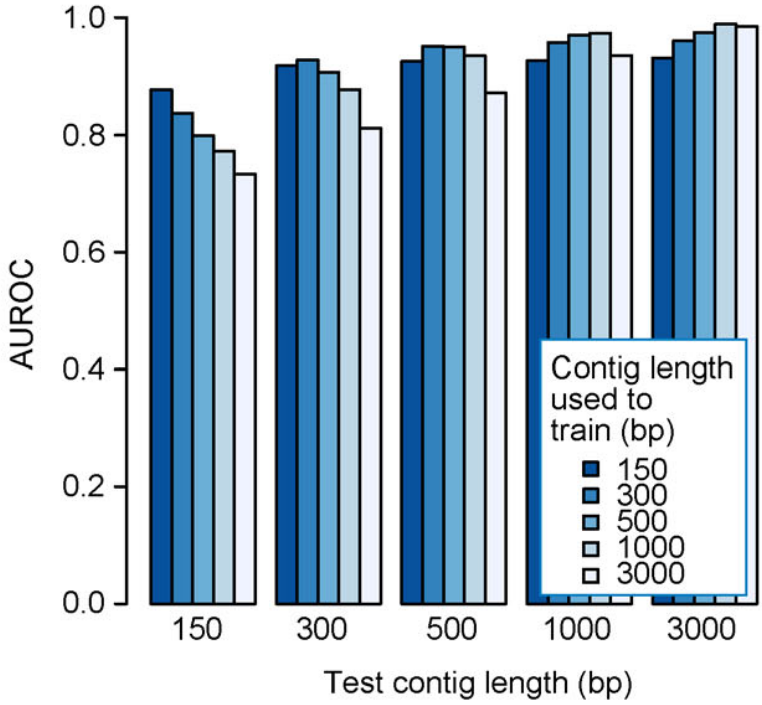

Figura 3.6: "Effetto della lunghezza delle sequenze sulle prestazioni di DeepVirFinder". Fonte [5] Fig. 2B

Nel caso delle sequenze di 1000bp, il modello migliore (AUROC 0.9562) è stato quello addestrato su sequenze di 500bp; i modelli addestrati su sequenze di 300 e 1000bp hanno ottenuto comunque AUROC molto vicine (0.9456 e 0.9517 rispettivamente). Infine, per le sequenze di 3000bp, il modello addestrato su sequenze di 1000bp ha mostrato prestazioni migliori (AUROC 0.9696).

Da queste osservazioni, possiamo dedurre che, data una sequenza da classificare, il modello che può fornire la previsione migliore sarà quello addestrato su sequenze di lunghezza simile a quella in esame. In particolare, considerando i cinque modelli da noi addestrati, useremo il modello addestrato su sequenze di 150bp per classificare sequenze di lunghezza < 300bp, il modello addestrato su sequenze di 300bp per classificare sequenze di 300-500bp, il modello addestrato su sequenze di 500bp per le sequenze di 500-1000bp,

e il modello addestrato su sequenze di 1000bp per le sequenze di lunghezza > 1000bp, analogamente a quanto ottenuto da Ren et al.

Interessante notare che in questo esperimento si sono ottenuti valori di AUROC nettamente migliori rispetto a quelli dei precedenti esperimenti, a parità di lunghezza delle sequenze testate. Tale miglioramento è principalmente dovuto all'ampliamento del dataset usato per il training, che in questo esperimento ha incluso i genomi RefSeq fino a maggio 2015.

Nello svolgimento di quest'ultimo esperimento, abbiamo riscontrato tuttavia dei tempi di elaborazione eccezionalmente lunghi. Inizialmente, si erano sottoposti al cluster di calcolo diversi job contemporaneamente, ciascuno dedicato ad eseguire uno dei cinque modelli addestrati su uno dei file FASTA contenenti le sequenze di test, di varie lunghezze. Mentre alcuni job terminarono l'elaborazione dei risultati nell'arco di alcune ore, come ci aspettavamo, la maggior parte degli altri job continuavano a risultare in esecuzione anche dopo 48 ore dall'avvio. Inoltre molti di questi job, ancora in esecuzione, non producevano più alcuna previsione nei file di output. Terminati i job che risultavo in stallo, si è proceduto ad avviarli nuovamente, imbattendosi però nello stesso problema.

Consultando alcuni articoli che si erano occupati di confrontare le prestazioni di DeepVirFinder con quelle di altri software equivalenti [17], si è scoperto che la causa di questi lunghi tempi di elaborazione risiede nella scelta del backend (Theano) usato da DeepVirFinder per generare le previsioni. Sebbene possano essere avviati più job in parallelo, la backend Theano è in grado di elaborare un solo dataset alla volta, in maniera seriale, e ciò spiega perché la maggior parte dei job in stallo non producesse output.

Capitolo 4

Conclusioni

Come si è detto nell'introduzione, i batteriofagi (i virus che infettano gli organismi procarioti, come batteri e archeobatteri) svolgono un ruolo importante nelle comunità microbiche, regolando il metabolismo dei propri ospiti e quindi le loro interazioni con l'ambiente. Di conseguenza, nel caso dei microbiomi presenti all'interno del nostro organismo, tali virus hanno un impatto rilevante sulla nostra salute. Comprendere a fondo la composizione e il ruolo di queste comunità virali è una delle principali sfide della metagenomica.

Le moderne tecniche di deep learning, adottate con successo in molti altri campi come la *computer vision* e il *natural language processing*, potrebbero apportare grandi miglioramenti nel campo della ricerca metagenomica, di cui peraltro sono già disponibili alcuni esempi.

Questo elaborato è stato per noi l'occasione di approcciarci allo studio di una materia (il *deep learning*, in particolare nella sua applicazione alla metagenomica) completamente nuova, coniugando lo studio teorico dei concetti fondamentali con alcune prove sperimentali.

Ad oggi, sono stati sviluppati numerosi software per identificare sequenze genomiche di origine virale all'interno di dati metagenomici. Questi software si distinguono principalmente in due categorie: alcuni cercano somiglianze tra le sequenze in esame e quelle presenti in un database di genomi di riferimento (sono detti *alignment-based* o *homology-based*), altri invece fanno uso di tecniche di machine learning/deep learning e analizzano *feature* insite nelle sequenze (come le frequenze dei k-mer o dei motivi) per generare una classificazione. In questo elaborato, ci siamo concentrati principalmente su uno strumento, DeepVirFinder, appartenente a questa seconda categoria.

Sia negli esperimenti condotti dagli stessi autori del software [5], sia in studi di ricercatori

indipendenti [17], DeepVirFinder ha dimostrato ottime prestazioni, collocandosi tra i migliori software per l'identificazione di sequenze virali, in particolare per le sequenze di lunghezza ridotta (< 500bp).

L'uso delle reti convoluzionali permette a DeepVirFinder di classificare sequenze sia codificanti che non codificanti. Data la sua capacità di individuare sequenze virali molto corte (150 o 300bp) con una buona precisione e sensitività, può essere applicato anche direttamente alle letture grezze ottenute dal sequenziamento, omettendo la fase di assemblaggio in *contig* più lunghi.

Nei nostri esperimenti abbiamo potuto osservare come la dimensione del dataset usato per il training influenzi in modo rilevante le prestazioni dei modelli. Avendo usato un sottoinsieme del dataset originale raccolto da Ren et al., i valori di AUROC da noi ottenuti, in particolare nei primi due esperimenti, erano nettamente minori di quelli ottenuti dagli autori. Nel terzo esperimento, in cui i modelli sono stati addestrati combinando i dataset di training e validazione, l'AUROC misurata sul dataset di test è stata maggiore, e più vicina ai valori ottenuti dagli autori di DeepVirFinder. I valori ottimali da noi determinati per gli iperparametri del modello ($fl = 11$ e $nb = 1000$) sono molto vicini a quelli fissati da Ren et al.

Tuttavia, nel terzo esperimento abbiamo individuato un difetto che penalizza DeepVirFinder rispetto ad altri software equivalenti. L'uso della backend Theano per lo svolgimento delle operazioni tensoriali limita la possibilità di scalare l'applicazione di DeepVirFinder, e usarlo per esaminare diversi set di sequenze in parallelo: poiché la backend Theano pone un lock globale, sarebbe necessario eseguire le varie istanze di DeepVirFinder ciascuna in un container separato, ciascuno dotato della propria copia della libreria Theano [17].

Studi come quello di Schackart et al. [17], in cui si sono confrontati 9 strumenti (tra cui DeepVirFinder) su 5 dataset di benchmark diversi, hanno evidenziato che esistono differenze anche molto nette nelle prestazioni dei diversi software per l'identificazione di sequenze virali. Queste differenze dipendono da vari fattori, quali il metodo usato nell'analisi delle sequenze (*alignment-based* oppure basato su reti neurali), il dataset di training (dimensione, distribuzione relativa delle specie virali e ospiti), e il tipo di applicazione. Si è visto che lo strumento che risulta migliore in un certo contesto, e rispetto a certe metriche, non necessariamente lo è anche in altri contesti.

Il campo della ricerca metagenomica è in costante evoluzione. Un approccio che potrebbe risultare vincente, nella sfida di classificare con elevata precisione e sensitività le sequenze virali, consisterebbe nello sviluppare strumenti basati su architetture ibride, che possano trarre il meglio dalle diverse tipologie di software.

Bibliografia

[1] Kevin Chen e Lior Pachter. «Bioinformatics for whole-genome shotgun sequencing of microbial communities». In: *PLoS computational biology* 1.2 (lug. 2005), pp. 106–112. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.0010024.

[2] Arthur M. Lesk. *Bioinformatics*. In: *Encyclopaedia Britannica*. URL: https://www.britannica.com/science/bioinformatics (visitato il 06/09/2024).

[3] Gaspar Roy et al. «Deep learning methods in metagenomics: a review». In: *Microbial Genomics* 10.4 (apr. 2024), p. 001231. ISSN: 2057-5858. DOI: 10.1099/mgen.0.001231.

[4] Julian R. Marchesi e Jacques Ravel. «The vocabulary of microbiome research: a proposal». In: *Microbiome* 3.1 (dic. 2015). Number: 1 Publisher: BioMed Central, pp. 1–3. ISSN: 2049-2618. DOI: 10.1186/s40168-015-0094-5. URL: https://microbiomejournal.biomedcentral.com/articles/10.1186/s40168-015-0094-5.

[5] Jie Ren et al. «Identifying viruses from metagenomic data using deep learning». In: *Quantitative Biology (Beijing, China)* 8.1 (mar. 2020), pp. 64–77. ISSN: 2095-4689. DOI: 10.1007/s40484-019-0187-4.

[6] Florian P. Breitwieser, Jennifer Lu e Steven L. Salzberg. «A review of methods and databases for metagenomic classification and assembly». In: *Briefings in Bioinformatics* 20.4 (19 lug. 2019), pp. 1125–1136. ISSN: 1477-4054. DOI: 10.1093/bib/bbx120.

[7] Gökcen Eraslan et al. «Deep learning: new computational modelling techniques for genomics». In: *Nature Reviews Genetics* 20.7 (lug. 2019). Publisher: Nature Publishing Group, pp. 389–403. ISSN: 1471-0064. DOI: 10.1038/s41576-019-0122-6. URL: https://www.nature.com/articles/s41576-019-0122-6.

[8] James Zou et al. «A primer on deep learning in genomics». In: *Nature Genetics* 51.1 (gen. 2019), pp. 12–18. ISSN: 1546-1718. DOI: 10.1038/s41588-018-0295-5.

[9] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. Adaptive computation and machine learning. http://www.deeplearningbook.org. Cambridge, Mass: The MIT press, 2016. ISBN: 978-0-262-03561-3. URL: http://www.deeplearningbook.org.

[10] Stuart J. Russell e Peter Norvig. *Intelligenza Artificiale: Un approccio moderno: Vol. 2*. collaborator Francesco Amigoni. 4a edizione. Milano Torino: Pearson Italia, 2021. xix+410. ISBN: 978-88-919-2748-4.

[11] Jie Ren et al. «VirFinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data». In: *Microbiome* 5.1 (6 lug. 2017), p. 69. ISSN: 2049-2618. DOI: 10.1186/s40168-017-0283-5. URL: https://doi.org/10.1186/s40168-017-0283-5.

[12] *Machine Learning Glossary*. Google for Developers. URL: https://developers.google.com/machine-learning/glossary (visitato il 25/08/2024).

[13] Nitish Srivastava et al. «Dropout: a simple way to prevent neural networks from overfitting». In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313. URL: https://www.semanticscholar.org/paper/Dropout%3A-a-simple-way-to-prevent-neural-networks-Srivastava-Hinton/34f25a8704614163c4095b3ee2fc969b60de4698.

[14] *Classification: Thresholds and the confusion matrix | Machine Learning*. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/classification/thresholding (visitato il 30/08/2024).

[15] *Classification: Accuracy, recall, precision, and related metrics | Machine Learning*. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall (visitato il 30/08/2024).

[16] *Classification: ROC and AUC | Machine Learning*. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (visitato il 30/08/2024).

[17] Kenneth E. Schackart et al. «Evaluation of computational phage detection tools for metagenomic datasets». In: *Frontiers in Microbiology* 14 (25 gen. 2023). Publisher: Frontiers. ISSN: 1664-302X. DOI: 10.3389/fmicb.2023.1078760. URL: https://www.frontiersin.org/journals/microbiology/articles/10.3389/fmicb.2023.1078760/full.