UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea Triennale

Predizione della struttura secondaria dell'RNA con reti convoluzionali

Relatore

Prof. Samir Suweis

Correlatore

Dr. Francesco Mambretti

Laureando

Matteo Pedrazzi

1161774

Anno Accademico 2021/2022

Indice

Introduzione

In biologia e medicina, così come in diversi altri settori, può essere di fondamentale importanza conoscere le strutture secondarie di acidi nucleici come DNA e RNA. La struttura secondaria di questi polimeri consiste nella lista degli appaiamenti tra le basi azotate associate ai nucleotidi che li costituiscono. Tradizionalmente, per conoscere queste strutture sono stati adottati due approcci complementari: uno sperimentale, studiando le sequenze di interesse in laboratorio [1]; uno computazionale, sviluppando simulazioni numeriche in grado di prevederle [2]. Inizialmente tali metodi sfruttavano algoritmi in grado di minimizzare l'energia libera al fine di trovare la struttura termodinamicamente favorita. In seguito, anche grazie allo sviluppo dell'intelligenza artificiale, ed in particolare di Machine Learning e Deep Learning, è stato possibile utilizzare tecniche sempre più sofisticate in grado di prevedere la struttura finale senza la necessità di sfruttare modelli energetici, ma svolgendo una fase di apprendimento su strutture già determinate sperimentalmente. Nel recente articolo di Zhao *et al.* [3] si focalizza l'attenzione su questi metodi che, anche grazie alla crescente disponibilità di dataset contenenti strutture secondarie di RNA, hanno consentito di migliorare notevolmente la qualità delle previsioni sulle strutture ottenibili fino a pochi anni fa. In particolare sarà preso in esame il lavoro del 2022 di Booy, Ilin ed Orponen [4], che tramite le Convolutional Neural Networks riesce a sviluppare un modello innovativo per la previsione della struttura secondaria della molecola di RNA, partendo dalla rappresentazione delle possibili coppie delle basi di una sequenza di RNA di lunghezza L tramite un tensore, ossia una matrice L x L x c, con $c = 8$ canali di colore associati alle diverse possibili L x L coppie. L'obiettivo della CNN, che riceve in input questo tensore, è associare ad ogni coppia una probabilità di legame producendo così una matrice che verrà infine convertita in strutture secondarie a tutti gli effetti, tramite algoritmi appositi.
Gli ottimi risultati, ottenuti variando i dataset utilizzati per la fase di test del modello, assumono ancora più rilevanza se si considerano la relativa semplicità dell'architettura della rete rispetto a quelle presentate in metodi analoghi, oltre che la capacità di prevedere strutture complicate contenenti accoppiamenti non canonici tra le basi della catena dell'RNA.

Nel primo capitolo "Convolutional Neural Networks" saranno innanzitutto fornite le informazioni necessarie alla comprensione delle nozioni di Intelligenza Artificiale e Machine Learning: in particolare saranno trattate le Convolutional Neural Networks, che sfruttano Deep Learning al loro interno. In seguito nel secondo capitolo si introdurranno le generalità riguardanti la molecola dell'RNA, soffermandosi sull'evoluzione degli Studi che sono stati effettuati al fine di determinarne la struttura secondaria. Infine verrà illustrato il metodo proposto CNNFold, illustrando le Rappresentazioni matriciali scelte per la sequenza di RNA, seguite dalla struttura della rete utilizzata e dai Risultati ottenuti del modello sui dataset scelti.

Capitolo 1

Convolutional Neural Networks

1.1 Intelligenza Artificiale

Per la scrittura del seguente paragrafo sono stati consultati gli articoli [5] e [6].

Il termine Intelligenza Artificiale (AI, dall'acronimo inglese), è spesso trattato in relazione al ragionamento umano, basti pensare ai lavori iniziali di Alan Turing [7], o alla definizione data dall'Università di Stanford come «scienza e insieme di tecniche computazionali che vengono ispirate dal modo in cui gli esseri umani utilizzano il proprio sistema nervoso [...]» [8]. Quindi genericamente ogni modello, algoritmo o programma in grado di compiere azioni al fine di adempiere ad un compito specifico che necessita di un'intelligenza umana, possono essere considerati **intelligenza artificiale**. In particolare l'intelligenza artificiale si è rivelata fondamentale nello sviluppo ed applicazione di tecniche per risolvere problemi di vario genere nei campi più disparati, tra cui l'analisi predittiva di dati, gli assistenti virtuali, alcuni sistemi di suggerimento personalizzati sui clienti oltre che al riconoscimento di immagini e suoni [9–11]. In Figura 1.1 è mostrata l'inclusione nell'insieme più esteso AI, dei sottoinsiemi Machine Learning, Neural Networks e Deep Learning che verranno trattati nelle sezioni successive.

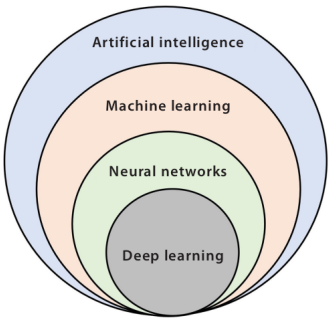
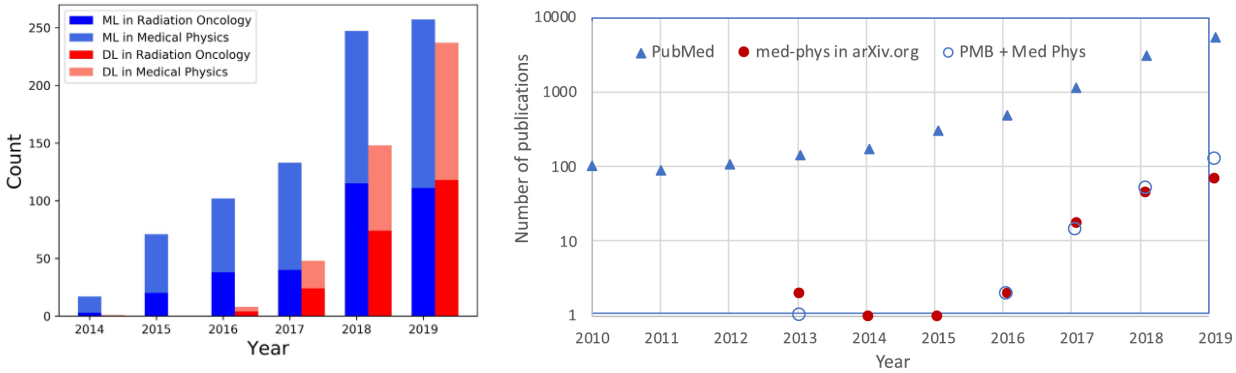


Figura 1.1: Relazione tra AI, ML, NNs e DL. Figura tratta da [12]

1.2 Machine Learning

Il Machine learning (ML) [13] è un insieme di metodi, in riferimento a modelli ed algoritmi numerici, che fornisce ai computer la capacità di apprendere (**learning**) ed in seguito effettuare previsioni riguardo a nuovi set di dati, senza essere programmati esplicitamente con tale finalità. Lo scopo del Machine Learning è quindi scoprire, tramite un processo di *training*, le relazioni o i pattern nascosti tra grandi quantità di dati, apprendendo le relazioni successivamente sfruttate per effettuare previsioni.



(a) Studi pubblicati su PubMed

(b) Pubblicazioni in scala logaritmica[1] riguardanti DL

Figura 1.2: Crescita del numero di pubblicazioni riguardanti ML e DL nel campo della fisica medica. Figure tratte rispettivamente da [14] e [6]

È possibile classificare i diversi algoritmi basati su ML utilizzando come criterio il tipo di input ed output del modello, oltre che al tipo di problema per cui sono stati ideati. Sfruttando tali criteri si possono distinguere tre categorie che rappresentano i tre diversi paradigmi legati all'apprendimento delle reti:

- ***Supervised learning***: il modello ha la possibilità di apprendere effettuando il processo di *training* su un set di dati (training set) in cui ad ogni input è associato l'output corrispondente, in dataset pre-etichettati (*pre-labeled*). Lo scopo è quello di approssimare la relazione tra variabili di input e output, con due diverse possibilità di applicazioni: predizione su input non etichettati oppure riduzione degli errori di classificazione del modello. Si tratta della strategia più diffusa di apprendimento, date le proprietà di immediatezza ed efficacia che la contraddistinguono;
- ***Unsupervised learning***: il training set non possiede labels, quindi lo scopo è quello di scoprire pattern ricorrenti nei dati ricevuti e ottimizzare una certa funzione costo associata;
- ***Reinforcement learning***: il sistema interagendo con l'ambiente apprende in base ai feedback positivi ("ricompense") ottenuti in caso di successo, puntando a massimizzare gli incentivi cumulati nel corso dell'esplorazione del problema.

1.3 Neural Networks

Uno dei possibili metodi implementabili per la soluzione di problemi di Machine Learning sono le reti neurali artificiali (*Artificial Neural Networks*), ovvero modelli computazionali ispirati alle reti neurali biologiche, i cui principali elementi strutturali saranno presentati a seguire, partendo dai nodi.

1.3.1 Nodi

Ogni rete è composta da nodi collegati tra loro tramite connessioni (dette *edges*), anche noti come **neuroni artificiali** dall'analogia con il sistema nervoso. Essi compiono in modo sequenziale 4 operazioni, espresse nella formula 1.1: per prima cosa ricevono un set di n segnali di input sotto forma di vettore $\vec{x} = (x_1, ..., x_n) \in \mathbb{R}^n$, a cui viene applicata, tramite moltiplicazione[2], un'operazione lineare definita dal vettore dei pesi (*weight vector*) $\vec{\omega} = (\omega_1, ..., \omega_n)$ ed al cui risultato viene sommato anche un bias ω_0. L'output $\hat{y}$ del nodo si ottiene alla fine applicando un'operazione non lineare (*activation function*, h) al risultato precedentemente ottenuto.

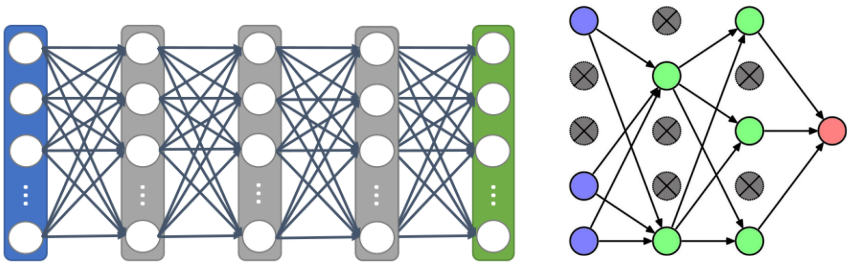
$$\hat{y} = \hat{f}(\vec{x}) = h(\vec{\omega}^T \cdot \vec{x} + \omega_0) \tag{1.1}$$

Il vettore contenente i parametri del modello è convenzionalmente definito $\vec{\theta} = (\omega_0, \omega_1, ..., \omega_n)$.

[1] Le pubblicazioni relative al 2019 sono state estese a tutto l'anno avendo raccolto dati per i soli primi due mesi.
[2] Nell'equazione 1.1 il termine $\vec{\omega}^T$ indica il vettore trasposto del vettore dei pesi $\vec{\omega}$.

1.3.2 *Layers*

All'interno di una rete, il *layer* è una struttura costituita da un gruppo di nodi, che ha il compito di trasmettere le informazioni dal layer che lo precede a quello successivo. Esistono vari tipi di layer con diversi tipi di connessioni tra neuroni e di conseguenza diversi possibili ruoli. Un criterio che si può assumere per differenziare i layers è ad esempio il numero di connessioni che consentono ai propri nodi.



(a) Fully-connected deep network (b) Alcuni dropout layer

Figura 1.3: Figure tratte rispettivamente da [6] e [14]

Si ottiene che da un lato esistono layers definiti ***fully-connected*** in cui ogni nodo è connesso a tutti i nodi del layer precedente e successivo (Figura 1.3(a)), all'opposto invece esistono i ***dropout* layers** che selezionano randomicamente una percentuale di nodi da ignorare (*dropout rate*, solitamente 20-50%) lasciando invariati i contributi degli altri neuroni (Figura 1.3(b)).

1.3.3 Artificial Neural Networks

Come già precedentemente accennato, le Artificial Neural Networks (ANNs), anche semplicemente Neural Networks (NNs), sono sistemi di elaborazione computazionale, ispirate al modo di lavorare e all'architettura del sistema nervoso umano ed animale. Analizzando la Figura 1.4 si può osservare un esempio di struttura tipica delle ANNs, che consiste innanzitutto di un layer di input composto da $n = 4$ nodi e rappresentabile come vettore multidimensionale $\vec{x} = (x_1, x_2, x_3, x_4)$. In casi in cui l'input è un'immagine si arriva ad avere un nodo per ogni pixel dell'immagine. Genericamente, all'input seguono un certo numero di *hidden layers* (in figura solamente uno), ossia livelli intermedi responsabili del processo di learning ed il cui numero all'interno della rete determina la complessità del modello, ed infine il layer relativo all'output.

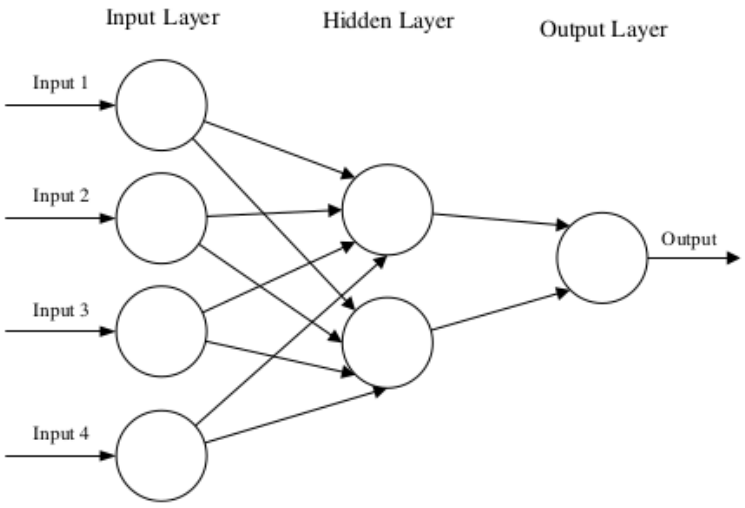


Figura 1.4: Esempio standard di ANN a tre layers. Figura tratta da [15]

1.4 Deep Learning

Fonte principale per il seguente paragrafo è stata [16].

Con il termine Deep Learning (DL) [17–19], si indica la sottocategoria del Machine Lerning che, operando in modo trasversale rispetto alle tre categorie di learning sopra citate (supervised, unsupervised e reinforcement), sfrutta le reti neurali artificiali (Artificial Neural Networks) a più strati (ossia dotate di *multiple layers*) per estrarre automaticamente dalla complessità dei dati una loro rappresentazione sintetica e astratta. Questo insieme di metodi ha permesso di ottenere successi in vari campi della scienza e per capire meglio il funzionamento di reti che implementano un sistema basato su DL, saranno presentati elementi di teoria e principali concetti dietro a tale sistema di apprendimento, partendo dalle fasi di *training* e *validation* del modello.

1.4.1 Training e validation

Durante la fase di **training** viene processato il training-dataset al fine di estrarre features significative per l'implementazione di un algoritmo specifico. Prendendo un input $\vec{x} = (x_1, ..., x_n)$, lo scopo del sistema è avvicinarsi il più possibile all'output y fornito dal dataset etichettato (*labeled*), di cui si ottiene una stima tramite la funzione $\hat{y} = \hat{f}(\vec{x})$. Concretamente nella fase di training vengono quindi determinati i parametri che rientrano nella stima di $\hat{y}$, ossia il vettore $\vec{\theta}$ precedentemente definito.
Il training è seguito da **validation**, nella quale non vengono aggiornati i valori di parametri in $\vec{\theta}$, bensì viene valutato, tramite il validation-dataset, il modello addestrato sul training set e vengono impostati i cosiddetti iperparametri del sistema, come ad esempio il numero e la tipologia di hidden layers da utilizzare. Infine esiste pure un test-dataset, non sempre distinto da quello di validation, grazie al quale è possibile avere un riscontro sulla bontà e validità del training effettuato.
Alla domanda riguardante il numero di dati necessari per un training soddisfacente non esiste una risposta universale: il problema va studiato caso per caso, anche se empiricamente all'aumentare della complessità del modello si è osservato che aumenta anche la richiesta di dati. Nelle applicazioni concrete è consuetudine cercare di bilanciare il notevole costo computazionale del training con la necessità di avere una predizione accurata e un modello sufficientemente generale: si tratta del *bias-variance dilemma*, ovvero la ricerca di minimizzare simultaneamente le due fonti di errore *bias* e *variance* di Figura 1.5. Un bias elevato può portare a difficoltà nell'individuare la relazione tra features appresa dal modello rispetto a quella reale.

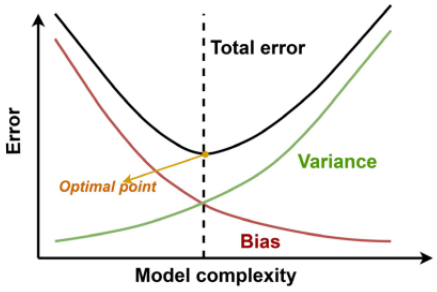


Figura 1.5: Bias-variance dilemma. Figura tratta da [14]

Inoltre è bene fare attenzione a non incorrere in *overfitting*, problema riscontrabile sia come effetto di alta varianza che quando il numero di parametri del modello è eccessivo se confrontato con la dimensione del campione. Ciò può portare a perdere la capacità di generalizzabilità del modello, ossia porta ad identificare e sfruttare relazioni tra features del training set in generale non realmente verificate.

1.4.2 Funzione costo

La fase di training del modello può essere riformulata matematicamente sotto forma di un problema di ottimizzazione, in cui lo scopo è trovare il set di parametri che minimizzi (o massimizzi, a seconda

della strada scelta) una certa funzione costo o obiettivo (da ***loss function***).
Alcune scelte comuni di loss function nell'ambito del DL possono essere l'errore quadratico medio (1.2) o la funzione di verosimiglianza (1.3), per cui la stima dei parametri ottimali si ottiene minimizzando (*least square estimator* o *maximum likelihood estimator*) la funzione costo:

$$\mathcal{L}(\hat{y}, y)_{LSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1.2}$$

$$\mathcal{L}(\vec{x}, \vec{\theta})_{MLE} = \prod_{i=1}^{n}\hat{f}(x_i, \theta_i) \tag{1.3}$$

1.4.3 Funzioni di attivazione

Altro elemento di fondamentale importanza per il Deep Learning sono le **funzioni di attivazione**, che permettono di trasformare in modo non lineare l'output di un nodo della rete, dato l'input o un insieme di essi, evidenziando le caratteristiche desiderate del segnale. Alcuni esempi mostrati in Figura 1.6, permettono di notare le caratteristiche di monotonia, limitatezza e continuità, comuni a funzioni di attivazione sia classiche (segno $sign(x)$, sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ e tangente iperbolica $tanh(x)$) che applicabili nel training di reti profonde, con queste ultime che verranno approfondite a seguire.

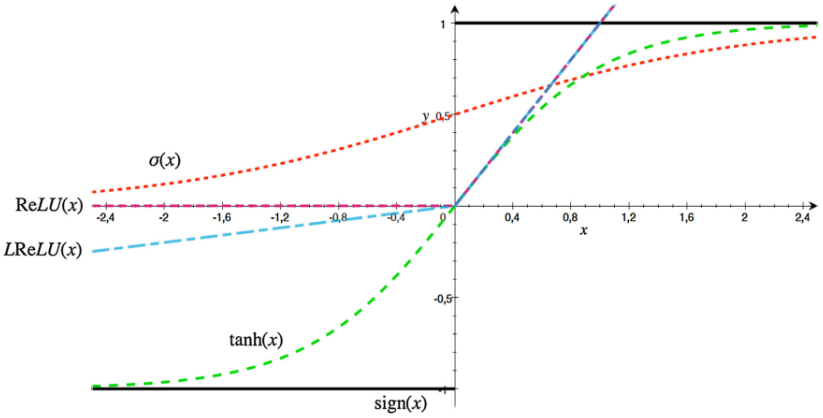


Figura 1.6: Esempi di funzioni di attivazione. Figura tratta da [16]

Una funzione di attivazione tipica delle reti neurali profonde è la Rectified Linear Unit, meglio nota come **ReLU** e definita in 1.4.

$$ReLU(x) = max\{0, x\} = \begin{cases} 0, & se\ x < 0 \\ x, & se\ x \geq 0. \end{cases} \tag{1.4}$$

L'utilizzo della ReLU porta diversi vantaggi, tra cui su tutti una migliore e più veloce fase di training per una rete ad elevato numero di layers, ma ha anche alcune lacune, come la non differenziabilità in $x = 0$ ed il problema della *dying ReLU*, per cui un neurone può arrivare in uno stato di inattività per qualsiasi input, fino ad un sostanziale "spegnimento". Questo è causato dal valore impostato a zero associato a tutti i neuroni con valori negativi e può portare a problemi delle performance del modello o addirittura all'arresto del processo di learning. Per ovviare a tale problematica è stata definita la variante **LeakyReLU** che è caratterizzata da una retta con coefficiente angolare molto piccolo per valori negativi di x:

$$LReLU(x) = max\{\alpha x, x\} = \begin{cases} \alpha x, & se\ x < 0 \\ x, & se\ x \geq 0 \end{cases} \quad \text{con } \alpha = 0.01^{3}. \tag{1.5}$$

[3]Valore tratto da [20], invece in librerie per deep learning e reti neurali come *TensorFlow* e *Keras* è automaticamente impostato rispettivamente a 0.2 e 0.3.

1.5 Deep Neural Networks

I modelli che più comunemente implementano DL sono le Deep Neural Networks (DNNs), essenzialmente una tipologia di ANNs che contengono un numero elevato di hidden layers.
La rappresentazione in termini matematici di una rete fully-connected, per un caso generico con n hidden layers, è esprimibile scrivendo per i singoli layers:

$$\text{primo hidden layer: } h_1 = \hat{f}_0(\vec{x}|\vec{\theta_0}) \tag{1.6}$$

$$i\text{-esimo hidden layer: } h_i = \hat{f}_{i-1}(h_{i-1}|\vec{\theta_i}) \tag{1.7}$$

$$\text{output layer: } \hat{y} = \hat{f}_n(h_n|\vec{\theta_n}) \tag{1.8}$$

che in alternativa si può scrivere in modo compatto:

$$\hat{y} = \text{DNN}(\vec{x}|\vec{\theta}) = \hat{f}_n(\hat{f}_{n-1}(\hat{f}_{n-2}(...\hat{f}_1(\hat{f}_0(\vec{x}|\vec{\theta_0})|\vec{\theta_1}))...|\vec{\theta_{n-1}})|\vec{\theta_n}).$$

1.5.1 Convolutional Neural Networks

Le Convolutional Neural Networks (CNNs) [15, 21] sono una tipologia particolare di DNNs proposte per la prima volta nel 1989 da LeCun *et. al.* [22], che risultano applicabili con successo nel campo del riconoscimento di pattern all'interno di immagini, con la possibilità quindi di sviluppare strumenti più adatti a questo tipo di compiti. L'utilizzo di *convolutional layers* ha come conseguenza una drastica riduzione del numero di parametri necessari rispetto alle DNNs tradizionali, che rendono le CNNs particolarmente adatte nel gestire immagini.

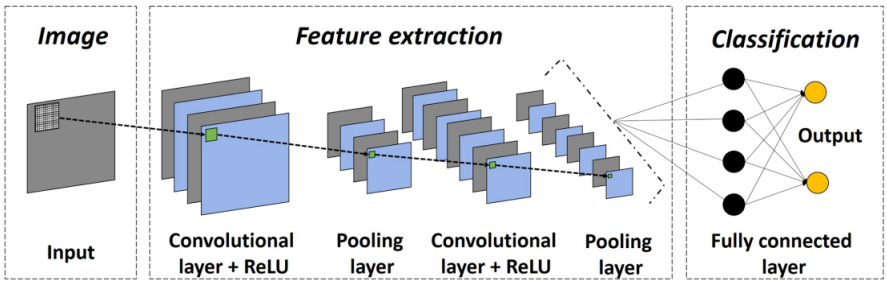


Figura 1.7: Una struttura comune per Convolution Neural Networks. Figura tratta da [12]

Essendo pensate per utilizzare come input immagini sotto forma di tensori, le CNNs possiedono una struttura caratterizzata da layers dotati di 3 "dimensioni": altezza (*height*), larghezza (*width*) e profondità (*depth*), dove width indica il numero di neuroni in un layer, mentre depth indica il numero totale dei layers di una rete, che coincidono con il numero di canali relativi al colore dell'immagine in input. Di seguito saranno presentate le idee chiave alla base delle CNNs, a partire da *Convolutional* layer e *Pooling* layer che, assieme ai fully-connected layers precedentemente introdotti, costituiscono una parte fondamentale delle CNN (Figura 1.7).

Convolutional layer

Anche noto come filtro, ha il compito di processare l'immagine ricevuta come input sotto forma di tensore, trasformandola in una mappa di attivazione da trasmettere al layer successivo. In particolare ogni nodo di una rete che sfrutta questa tipologia di layers riceve come input solo una regione ristretta del layer precedente, che prende il nome di ***receptive field***. Considerando come input un tensore che contenga come quarta dimensione la *depth* (ovvero il numero di colori dei pixel), la convoluzione[4] opera tramite un kernel (di dimensione 3x3 in Figura 1.8) utilizzando una funzione di attivazione ReLU: in

[4]Date le funzioni f e g reali, la loro convoluzione è data dall'integrazione del prodotto di una per l'altra traslata, ovvero nel caso discreto $(f * g)[n] := \sum_{m=-\infty}^{\infty} f(n-m) * g(m)$.

questo modo è in grado di ridurre dimensionalmente la depth in output e risulta computazionalmente molto più economica rispetto alla moltiplicazione tra matrici.

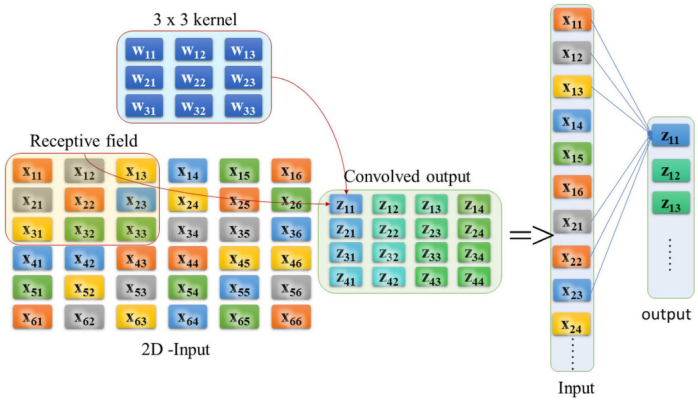


Figura 1.8: Convoluzione in 2 dimensioni con kernel 3x3. Figura tratta da [14]

La depth del layer, quindi, è modificabile al fine di impostare il numero di neuroni per ogni layer ed è una dei tre iperparametri usati per ottimizzare l'output della rete convoluzionale. Gli altri due sono il passo (*stride* S), responsabile della posizione del receptive field R (e conseguentemente della dimensione dell'output) ed il *zero-padding*, una tecnica che permette di conservare nell'output la dimensione dell'input, e consiste nell'aggiunta di pixel nulli ai margini dell'input. Il numero di zero-padding set è indicato con Z. In particolare l'alterazione spaziale della dimensione dell'output dei convolutional layers rispetto al volume in input V, è descritta dalla seguente formula:

$$\frac{(V - R) + 2Z}{S + 1} \in \mathbb{N}, \tag{1.9}$$

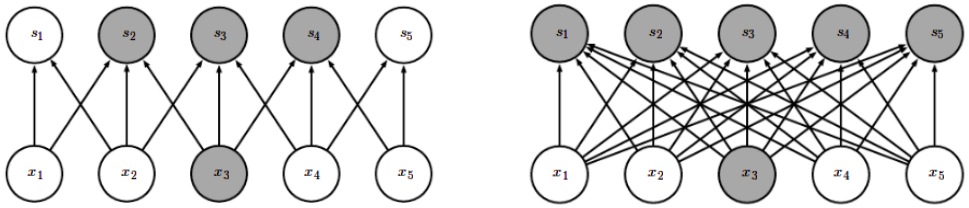
Pooling layer

Serve per ridurre gradualmente il numero di parametri del modello ed agisce sostituendo l'output di alcuni layer in una certa posizione, utilizzando le informazioni degli output adiacenti. Ad esempio l'operazione "max-pooling" riporta il valore massimo di output entro un intorno di forma rettangolare (*rectangular neighborhood*), a differenza di altre operazioni che invece utilizzano una media semplice o pesata in relazione alla distanza dal pixel centrale. Il processo di pooling permette di rendere la rappresentazione invariante rispetto a piccole traslazioni dell'input, proprietà utile in particolare se si è interessati alla presenza o meno più che alla esatta collocazione di una certa feature. Inoltre consentono anche alla rete di analizzare immagini a diverse risoluzioni, fattore di notevole importanza se si è interessati a dataset contenenti immagini di dimensioni non fissate.

1.5.2 Tecniche e strumenti nelle CNNs

Sparse interactions

Tecnica anche nota come *sparse connectivity*, sfrutta la dimensione del kernel inferiore rispetto alla matrice di input per implementare il numero minore possibile di interazioni nella rete, al fine migliorare l'efficienza statistica e di ridurre la richiesta di memoria, oltre che ridurre il tempo di esecuzione. In Figura 1.9 si nota visivamente la differenza nel numero di unità di output s coinvolte selezionando come input solo l'unità x_3: per convoluzione con kernel avente width pari a 3, sono individuati solo tre output (1.9(a)), mentre per la moltiplicazione tra matrici tutti gli output s_i $(i = 1, ..., 5)$ sono collegati ad x_3 (1.9(b)).



(a) (b)

Figura 1.9: Confronto tra convoluzione (a) e moltiplicazione di matrici (b). Figure tratte da [17]

Parameter sharing

Quando più funzioni nello stesso modello fanno uso degli stessi parametri si parla di condivisione di parametri, in particolare è comune in un modello avere la possibilità di riutilizzare più volte gli stessi *weights*, da cui il nome alternativo *tied weights*. Essi possono essere utilizzati in comune da neuroni in diverse regioni spaziali, permettendo una riduzione del numero di parametri nel convolutional layer rispetto alle reti neurali standard o ai layers fully-connected, in cui ogni *weight* di ogni neurone è usato esattamente una volta e poi dimenticato. Essendo ogni elemento del kernel applicato ad ogni elemento della matrice di input, questa tecnica non riduce il runtime bensì la richiesta di memoria da parte del modello.

Residual Blocks e *Batch normalization*

Uno dei problemi che ha afflitto per diversi anni studi nel campo del DL è il *vanishing/exploding gradient problem*[5], trattato nel dettaglio in [23]. Strategie introdotte per prevenire tale problema sono ad esempio l'utilizzo di *Residual Blocks* e *Batch normalization*.
I primi sfruttano le **skip** (o **residual**) **connections** tipiche delle Residual Networks (**ResNet**), utili perchè consentono di evitare il passaggio da uno o più nodi della rete tramite delle "scorciatoie" che collegano nodi di layers tra loro non adiacenti, permettendo ad informazioni importanti nei layers iniziali di essere trasmesse direttamente a quelli desiderati. Esse sono rappresentabili matematicamente considerando la connessione tra neuroni i e j in layers non adiacenti per $i > j$:

$$h_i = \hat{f}_{i-1}(\hat{f}_{i-2}(...h_j)) + h_j,$$

dove il termine additivo finale indica la skip connection.

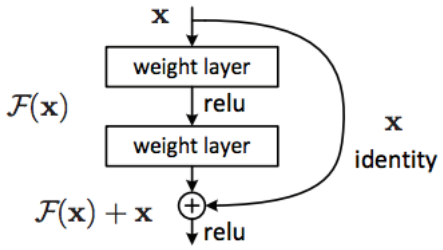


Figura 1.10: Residual Block. Figura tratta da [23]

La batch normalization, anche conosciuta come *batch norm*, invece fu proposta da Ioffe e Szegedy nel 2015 [24], e viene usata nelle NNs per normalizzare valori di input dei layers attraverso un processo di ricentramento e riscalamento, al fine di evitare valori fuori scala favorendo così maggiori velocità e stabilità nella rete.

[5] Fenomeno che comporta difficoltà nell'apprendimento per reti profonde. La causa sono le funzioni di attivazione non lineari che hanno derivate parziali a valori in (0,1), ed essendo usate in moltiplicazioni della rete, tale prodotto può decrescere o crescere in modo esponenziale.

Capitolo 2

RNA

Per la stesura di tale capitolo è stato consultato principalmente il documento [3].

2.1 Struttura

L'acido ribonucleico (in inglese *RiboNucleic Acid*, RNA) è una molecola polimerica lineare costituita da una catena di nucleotidi, ovvero unità ripetitive (monomeri) composte da una base azotata, uno zucchero pentoso ed un gruppo fosfato. Le basi azotate per l'RNA sono quattro: adenina (**A**), guanina (**G**), citosina (**C**) e uracile (**U**), con la differenza di quest'ultima sostituita alla timina rispetto al DNA. L'accoppiamento canonico di Watson-Crick permette, attraverso legami ad idrogeno, connessioni tra basi (A-U) e (G-C), a cui va aggiunta la coppia tra basi *wobble* (G-U). L'RNA è una molecola altamente versatile, responsabile di alcuni ruoli chiave nei processi cellulari, come la regolazione ed espressione dei geni, il trasporto di segnali cellulari e una funzione di catalizzatore in varie reazioni. Come visibile dalla Figura 2.1 a sinistra, la caratteristica distintiva dell'RNA rispetto al DNA, però, è la struttura a singolo filamento (*single stranded*) ripiegato su sè stesso piuttosto che i filamenti che formano la doppia elica tipica del DNA. In particolare la struttura piegata (*folded*) definisce la funzione biologica delle diverse famiglie[1] di RNA, e quindi il suo studio ha grande importanza in vari settori della biologia, con applicazioni in campo medico e nel design di nuove molecole.
La struttura è studiata su diversi livelli, a partire dalla semplice sequenza di basi da cui si forma rapidamente, attraverso un grande consumo di energia, la struttura secondaria la quale poi impiega più tempo per assumere una struttura tridimensionale:

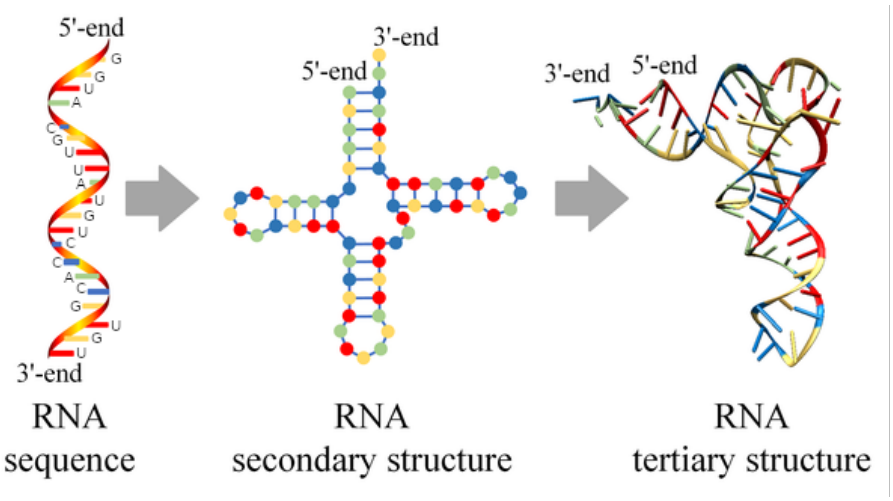
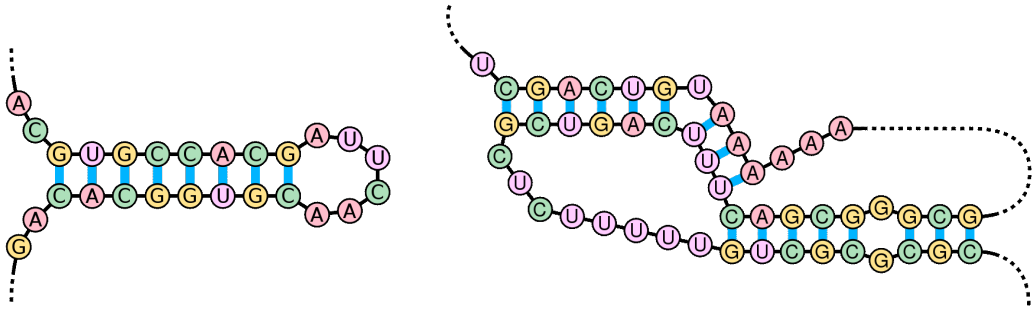


Figura 2.1: Figura tratta da [3]

[1] Nelle cellule esistono diversi tipi di RNA, detti appunto famiglie, distiguibili per funzione e struttura. Alcuni esempi sono RNA messaggero (mRNA), ribosomiale (rRNA) o transfer (tRNA).

- **Struttura primaria**: sequenza ordinata dei nucleotidi che costituiscono la catena;
- **Struttura secondaria**: determinata da collegamenti tra i nucleotidi della catena, ossia gli accoppiamenti tra le basi azotate. Strutture ricorrenti a questo livello sono filamenti accoppiati o steli (***stem segments***), alle cui estremità le basi non accoppiate possono costruire degli ***stem loops***, rappresentabili schematicamente come degli anelli (Figura 2.2(a)). Gli ***pseudoknots*** [25] invece sono strutture più complicate che si formano quando sono presenti almeno due strutture a loop concatenate, che comportano l'intersezione tra gli steli ad essi collegati (Figura 2.2(b)). Questo tipo di appaiamento risulta essere raro e difficile da studiare, tanto che cercando le soluzioni attraverso algoritmi computazionali si affronta un problema NP-completo[2], ma allo stesso tempo importante al fine di determinare la funzione della molecola. Infine esiste un altro tipo di accoppiamento non canonico tra basi, anch'esso ampiamente presente nella strutture secondarie dell'RNA, che porta alla formazione di tripletti di basi [26].



(a) Stem segment e stem loop (b) Pseudoknot

Figura 2.2: Figure tratte da *Wikipedia*

- **Struttura terziaria**: determina la forma tridimensionale del polimero ed è costituita da accoppiamenti tra le strutture di tipo secondario, che favoriscono la stabilità della molecola. Risulta essere estremamente complicata da studiare.

2.2 Studi

A livello della struttura secondaria avviene il ripiegamento della catena strettamente collegato alla funzione della molecola, pertanto gli studi sulla struttura dell'RNA si concentrano sulle coppie di basi presenti in tale struttura. Per determinare la struttura secondaria esistono principalmente due tipologie di approcci: uno sperimentale, perseguito tramite esperimenti di laboratorio ed un altro computazionale, che sfrutta algoritmi e metodi appositi per prevedere la struttura finale. Sperimentalmente, la **cristallografia ai raggi X** [1] e la **risonanza magnetica nucleare** (NMR) [27] sono i metodi più accurati per determinare la struttura secondaria della sequenza di RNA, con una risoluzione che permette l'osservazione fino alle singole basi, la quale li rende tuttora indispensabili per conoscere strutture con alta definizione. I problemi per questo approccio sono il costo elevato e la limitata possibilità di applicazione su larga scala. Per questo è importante considerare l'alternativa fornita dai metodi computazionali, partendo da quelli tradizionali per arrivare fino a quelli che sfruttano il machine learning nella predizione.

Metodi comparativi [28]

Sono i migliori per accuratezza tra i metodi di tipo computazionale, però possiedono vari difetti, tra cui su tutti l'estrema lentezza: considerando una sequenza di lunghezza n, il runtime è dell'ordine $\mathcal{O}(n^3)$

[2]Problema più complesso della classe NP, ovvero tra i problemi non deterministici che impiegano tempo limitato superiormente da un polinomio di grado pari alla lunghezza della sequenza. Nota la soluzione per un problema NP-completo si avrebbe soluzione immediata a tutti i problemi della classe NP.

(poi versioni ottimizzate arrivano a $\mathcal{O}(n^2 log(n))$). Tale metodo è fondato sull'idea che le strutture secondarie siano conservate nel tempo in misura maggiore rispetto a quelle primarie, attraverso delle "mutazioni compensatorie". Tra le limitazioni di questo approccio vanno annoverate quindi anche la richiesta di un set sequenze omologhe[3] necessarie al confronto per le migliaia di famiglie dell'RNA attualmente conosciute [29], oltre che alle scarse performances ottenute su strutture contenenti pseudoknots o altri accoppiamenti non canonici.

Metodi *score-based*

Sono i più usati negli ultimi decenni, e sfruttano un punteggio (*score*) associato ad ogni possibile struttura secondaria. In questo modo il problema viene trasformato in un'ottimizzazione al fine di ottenere lo score massimo attraverso algoritmi di programmazione dinamica (DP): nel primo e più semplice tra questi algoritmi [30], gli autori assegnano in particolare come punteggio l'energia libera per ogni base appaiata. Questi algoritmi lavorano per cercare le sottostrutture che, assemblate, diano una struttura ottimale, ovvero quella complessivamente avente energia libera minima (Minimum Free Energy, MFE). Il primo modello di questo tipo per la ricerca della MFE è stato il *nearest neighbor* (NN) model di Zuker e Stiegler [25], il quale però soffre di limitazioni come l'estrema lentezza computazionale (runtime ($\mathcal{O}(n^3)$)) e l'inefficienza per coppie non canoniche. Si può anche euristicamente apportare miglioramenti alle prestazioni del modello, pur non arrivando a previsioni completamente soddisfacenti per effetto della limitatezza dei metodi score-based che sono arrivati infatti ad un plateau per quanto riguarda le performances.

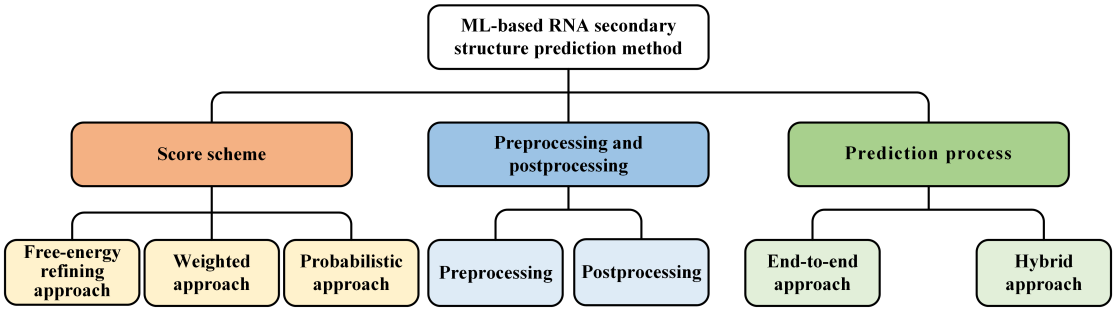


Figura 2.3: Schema dei metodi che sfruttano il Machine Learning. Figura tratta da [3]

Metodi *ML-based*

I vantaggi di un approccio basato sul Machine Learning sono molteplici, a partire dal funzionamento delle reti di tipo *data-driven* che permette di evitare la formazione di una dipendenza vincolante dalla conoscenza del meccanismo biologico del folding[4]. Altro aspetto importante è la rimozione di un modello energetico necessario per assegnare il punteggio alle possibili strutture, che permette di non introdurre immediatamente questa *prior*[5] che può essere fonte di errori. Infine il runtime per questi modelli è indipendente dalla lunghezza della catena di RNA, favorendo una buona flessibilità del modello. All'interno dei metodi *ML-based* esistono tre sottocategorie:

- *Score scheme*: il più accurato è CONTRAfold [31], modello che stima i parametri tramite un approccio probabilistico che sfrutta la conoscenza di strutture secondarie già determinate per indurre una distribuzione di probabilità sulle possibili strutture dell'RNA per una certa sequenza. In ogni caso, la previsione finale è invece risolta ancora come un problema di ottimizzazione, tramite la ricerca della MFE. La limitatezza di questo approccio sta nell'incapacità di prevedere coppie di basi non standard;

[3] Sequenze che hanno la stessa struttura secondaria al variare della sequenza di basi della catena.

[4] Si è osservato infatti che, in presenza di grandi quantità di dati, i modelli che "ignorano" il meccanismo biologico hanno risultati migliori di quelli che invece sono fondati su tali conoscenze, a testimonianza della possibile incompletezza o inaccuratezza delle attuali conoscenze nel campo.

[5] Relativamente ad una quantità ignota, in statistica la *prior* è la distribuzione di probabilità associata a tale quantità senza che dati o evidenze siano state prese in considerazione.

- *Pre/post-processing*: è possibile eseguire un pretrattamento per selezionare un gruppo di parametri o un metodo di predizione a seconda delle diverse famiglie di RNA in esame [32], con scopo di identificare prima le regole secondo cui avviene il folding e poi applicarle per ottenere la struttura finale;
- *Prediction*: tra gli approcci di tipo *end-to-end* si è distinto in particolare quello proposto da Takefuji *et al.* [33], il primo ad introdurre il ML nella previsione della struttura secondaria di RNA, ottenendo però in principio scarsi risultati soprattutto per effetto della scarsità dei dataset disponibili. Il punto di svolta è stato l'applicazione delle tecniche del DL alla previsione delle strutture: il primo metodo ad introdurle è stato SPOT-RNA (Singh *et al.* [34]), che impostando il problema attraverso una matrice di contatto[6], implementa l'utilizzo delle ResNet per le predizioni, riuscendo ad ottenere risultati ottimi, che poi sono stati addirittura superati da metodi più recenti come UFold [35], che può vantare un miglioramento di 10-30% rispetto ai modelli termodinamici tradizionali e del 14% rispetto agli altri metodi *ML-based*. Approcci di tipo ibrido provarono a combinare altri metodi con quelli basati sul ML ma senza ottenere performance di livello, probabilmente per effetto un bias legato alla composizione dei vari metodi.

Si entrerà ora nel dettaglio del metodo predittivo proposto nell'articolo esaminato, analizzando i particolari del modello utilizzato ed in seguito confrontando i risultati prodotti con quelli dei principali metodi di predizione della struttura secondaria dell'RNA.

[6] Matrice quadrata in cui l'elemento della i-esima riga e j-esima colonna indica l'interazione tra i-esimo e j-esimo nucleotide della catena di RNA.

Capitolo 3

CNNFold

Nel 2021 tramite l'articolo [4], i ricercatori[1] Booy, Ilin e Orponen, hanno proposto un modello innovativo che sfrutta il ML, ed in particolare le CNNs, per ottenere previsioni sulle strutture secondarie dell'RNA. Data l'imprecisione osservata per metodi basati sulla minimizzazione dell'energia, il modello proposto è fondato sul DL ed effettua il training solo su dataset contenenti strutture già caratterizzate sperimentalmente ottenendo, a differenza dei metodi *score-based*, ottimi risultati anche per strutture non canoniche, come quelle contenenti *pseudoknots*. Il punto focale dell'approccio presentato in questo articolo riguarda la generazione di una matrice di possibili contatti tra i nucleotidi del filamento di RNA, la quale viene poi convertita in un'immagine da sottoporre come input ad una CNN. In questo modo, sarà per prima cosa analizzata la strategia utilizzata per costruire la matrice (*input tensor*) a partire dalla sequenza iniziale, in seguito saranno presentati la struttura della rete convoluzionale impiegata ed i risultati da essa prodotti, che dopo una fase di *post-processing* verranno confrontati con quelli di alcuni altri modelli predittivi, sia di tipo score-based che ML-based.

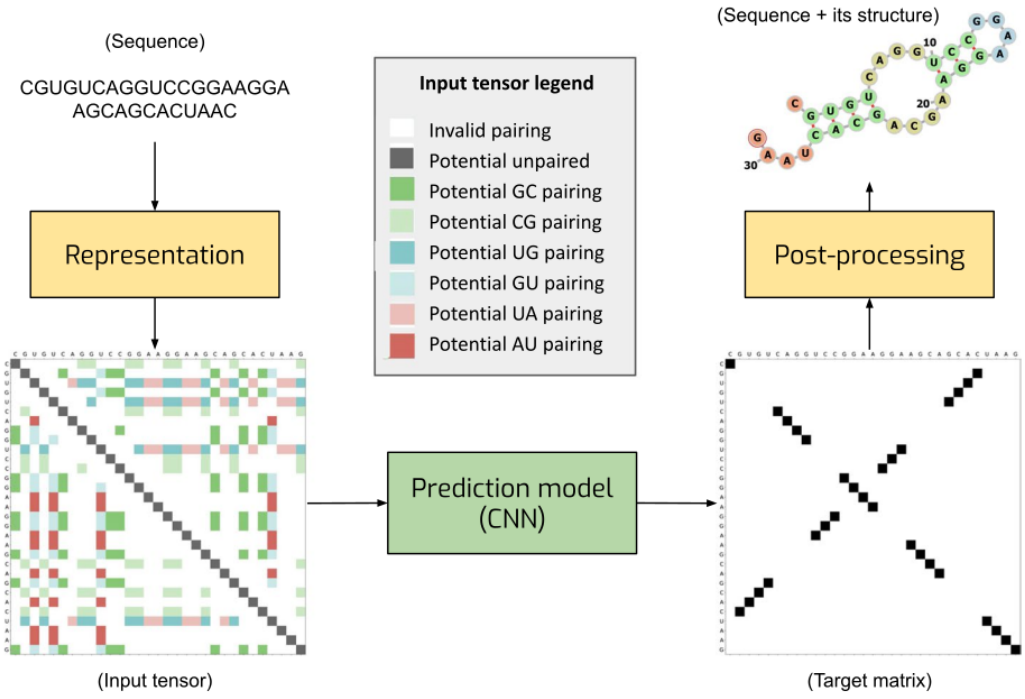


Figura 3.1: Schema riassuntivo del modello. Si rappresenta la sequenza come tensore, ossia una mappa 2D con 8 canali di colore, usata come input per la CNN che produce una *score matrix* per tutte le coppie possibili, infine tramite *post-processing* si converte la matrice nella struttura secondaria. Figura tratta da [4]

[1]Presso il dipartimento di Computer Science dell'Università finlandese di Aalto.

3.1 Rappresentazioni matriciali

3.1.1 Visualizzazione delle possibili coppie: *input tensor*

La struttura primaria del singolo filamento di RNA può essere rappresentata come una sequenza ordinata di L basi $q = (q_1, q_2, ..., q_L)$, dove $q_i = \{A, U, G, C\}$ sono le basi azotate. Tra tutte le sequenze su cui il modello è stato testato, viene presa come esempio pratico la sequenza di basi

$$\text{CGUGUCAGGUCCGGAAGGAAGCAGCACUAAC} \tag{3.1}$$

che ha la funzione di indice sia orizzontale che verticale nella matrice dei contatti, denotata X, con scopo di indicare le diverse possibili coppie $\{(q_i, q_j)\}$ tra basi della catena. Tale matrice L x L, dotata di 8 canali associati al colore è quindi rappresentabile come un **tensore** tridimensionale L **x** L **x** 8, in cui il vettore in 8 dimensioni associato al pixel (i, j) di X contiene le informazioni su possibili relazioni e vincoli esistenti tra le basi q_i e q_j, trasmesse alla Figura 3.2 attraverso l'uso dei colori presentati nella legenda:

- 6 possibili accoppiamenti simmetrici a coppie tra basi, di cui:
  - 4 sono garantiti da **Watson-Crick** (verde, rosso) in cui le due versioni simmetriche hanno una colorazione più chiara;
  - 2 sono la coppia ***wobble*** (azzurro) con due canali per le coppie simmetriche;
- 1 riguarda la **diagonale** $i = j$ che facilita il rilevamento di basi non appaiate e a cui è associato il canale grigio,
- 1 infine è il canale bianco che indica quando le coppie tra basi in i e j **non sono possibili**, per motivi legati all'incompatibilità tra basi (base spaiata) o ad altri vincoli come la distanza ($|i - j| \geq 3$).

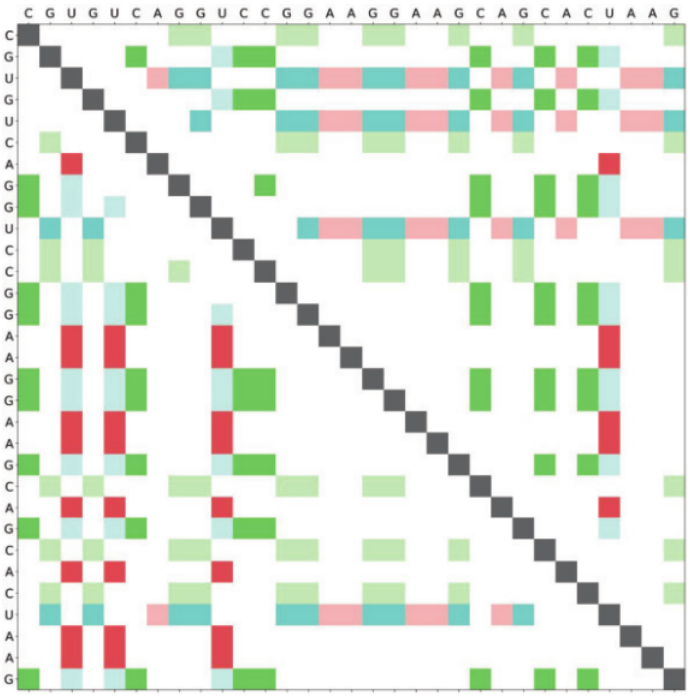


Figura 3.2: La matrice di contatto X. Figura tratta da [4]

3.1.2 Espressione dei risultati: *score* e *target matrix*

La matrice che funge da obiettivo (***target***) per l'output finale della CNN è una matrice binaria, ossia costituita di pixel alternativamente bianchi o neri, di dimensione L x L e denotata con T, in cui l'elemento ij-esimo ha valore:

$$t_{ij} = \begin{cases} 1, & \text{se } i \text{ e } j \text{ sono accoppiate (in rosso in Fig. 3.3)} \\ 0, & \text{altrimenti} \end{cases} \tag{3.2}$$

$$t_{ii} = 1, \quad i \text{ disaccoppiata (in azzurro in Fig. 3.3)} \tag{3.3}$$

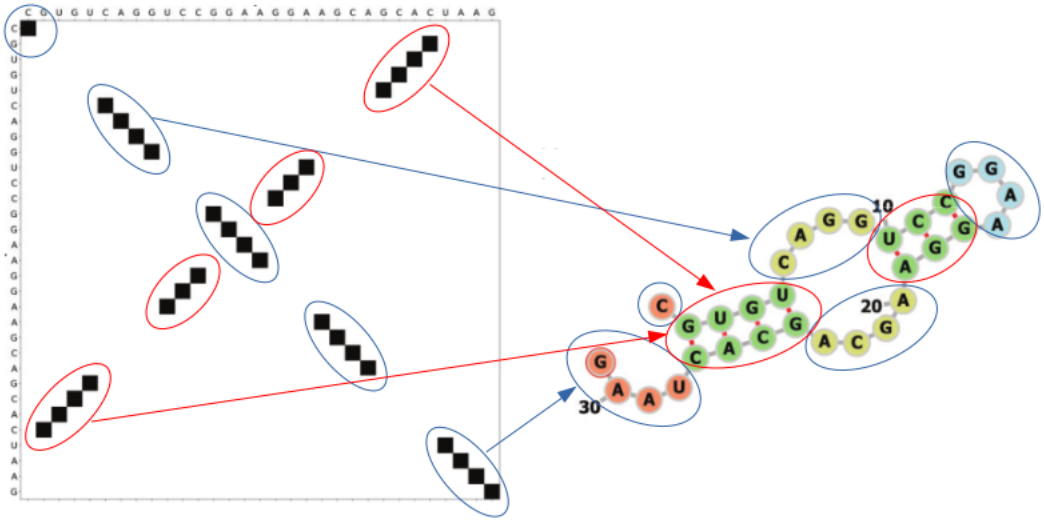


Figura 3.3: Relazione tra la *target matrix* T e la struttura secondaria prodotta. Le frecce collegano le caselle della matrice con le rispettive basi appaiate/singole nella struttura. Rielaborazione di figure tratte da [4]

Il vantaggio di questa rappresentazione è che rende semplice la previsione sia per coppie locali che a lunga distanza, ossia rispettivamente vicine e lontano dalle diagonale del tensore di input.
La ***score matrix*** Y prodotta invece come risultato dalla rete possiede gli stessi indici e dimensione di T, ma non è a valori binari, bensì presenta nei pixel (i, j) il punteggio (*score*) che il modello ha associato alla probabilità che la coppia tra basi q_i e q_j sia effettivamente realizzata nella struttura prevista.

3.2 Fasi operative

3.2.1 Architettura della CNN

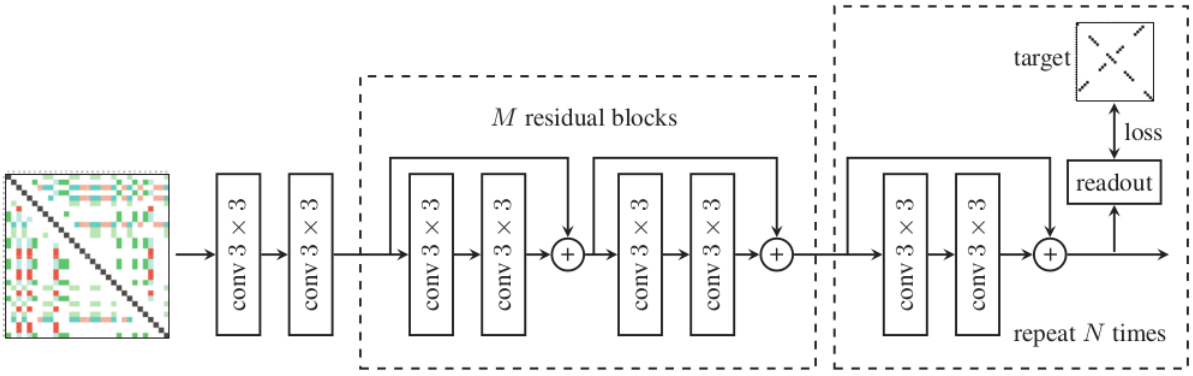


Figura 3.4: Architettura della CNN utilizzata. Figura tratta da [4]

La CNN utilizzata presenta innanzitutto due ***convolutional blocks***, in seguito nel primo riquadro tratteggiato di Figura 3.4 sono introdotti M ***residual blocks*** (che usano weights differenti) ed infine nel secondo riquadro un solo ***shared residual blocks*** ripetuto per N volte con parametri condivisi (*weight sharing*), che permette alla rete di processare sequenze di lunghezza variabile in modo equivariante, ovvero alla traslazione della sequenza in input varia anche l'output in modo analogo. I convolutional blocks sono composti da un convolutional layer di dimensione 3x3 (denotato in figura come "conv 3x3"), con 32 canali di output, seguiti da *batch-normalization* e funzione di attivazione LeakyReLU. Per lasciare invariata la dimensione dell'input dopo ogni convolutional block viene inoltre applicato un padding crescente all'aumentare della profondità della rete. L'ultimo layer incontrato prima dell'output finale è il ***readout layer***, che consiste in un convolutional block seguito da un convolutional layer, entrambi con kernel di dimensione 1.
Per arrivare alla soluzione ottimale nel minor tempo possibile la **loss** viene calcolata dopo ogni residual block, ed in particolare per questo modello è stato scelto, in base ai risultati osservati, di utilizzare come funzione costo l'errore quadratico medio, calcolato tra gli elementi della matrice Y_n ottenuta come output della rete dopo n residual blocks, rispetto alla target matrix T:

$$\mathcal{L}_n = \frac{1}{|V|} \sum_{ij \in V} (y_{ij}^{(n)} - t_{ij})^2 \tag{3.4}$$

dove $y_{ij}^{(n)}$ è l'elemento ij-esimo della matrice Y_n e V indica l'insieme di tutte le coppie possibili escluse quelle date da combinazioni di basi non valide o troppo distanti tra loro. La stima finale della loss si calcola facendo la media sugli N valori intermedi ottenuti:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n \tag{3.5}$$

3.2.2 *Post-processing*

La matrice Y prodotta come output dal modello deve ora essere convertita in una struttura secondaria a tutti gli effetti, in cui si possa osservare la struttura del filamento di RNA ripiegato. Gli approcci alternativi proposti in [4] per la fase di post-processing sono due: *Blossom post-processing* e *Argmax post-processing*.

Blossom post-processing

Lo scopo è estrarre una struttura secondaria in cui ogni base è appaiata ad un'altra base singola oppure spaiata. Si decide quindi di interpretare la matrice $Y{=}Y_n$ come una *weighted adjacency matrix*, ovvero una matrice delle adiacenze pesata[2] in cui i nodi della rete corrispondono alle basi e i pesi delle connessioni (*edges*) alle probabilità che esista la coppia tra tali basi secondo il modello. Per ridurre il costo computazionale sono state conservate in realtà solamente le tre edges di peso massimo per ogni nodo, con lo scopo di trovare un set di edges senza nodi in comune che massimizzi la somma dei pesi. Per risolvere questo problema di ottimizzazione viene applicato il *Blossom algorithm* [36], il quale però non funziona per strutture contenenti *self-loops*, per cui sono state create due copie del grafico originale pesato (senza self-loops) introducendo connessioni tra ogni coppia di nodi che rappresenta lo stesso nodo nel grafico originale. I pesi delle connessioni aggiunte, in questo modo sono associati ai pesi dei self-loops moltiplicati per due. L'algoritmo Blossom garantisce in questo modo che le basi siano alternativamente accoppiate con un'altra singola oppure spaiate, seppur dal punto di vista computazionale risulti essere molto costoso, soprattutto per lunghe sequenze.

Argmax post-processing

Il secondo approccio, denominato *Argmax post-processing*, sfrutta la struttura matriciale di Y scegliendo il valore massimo della casella y_{ij} nella riga i-esima di Y per determinare la connessione favorita

[2] Si tratta di un metodo di rappresentazione di strutture discrete finite, come può essere la struttura secondaria dell'RNA. In particolare il valore situato nella casella (i, j) della matrice rappresenta il peso associato alla connessione tra le basi in i e j.

tra i e j. Il tempo computazionale impiegato è conveniente rispetto al Blossom (andamento del tipo $\mathcal{O}(L^2)$), anche se la limitazione di questo metodo resta la produzione di strutture spesso non valide, in quanto non garantisce la simmetria tra coppie.

Le prestazioni legate a questi approcci differenti saranno osservabili in Tabella 3.3, anche se, visti i risultati analoghi a quelli del Blossom, è stato utilizzato Argmax per impostare gli iperparametri del sistema. In particolare in Figura 3.5 è possibile osservare l'estrema rapidità del metodo Argmax rispetto a Blossom, evidente soprattutto con l'utilizzo della CPU (Intel Xeon Gold 6230, 2.10Ghz) rispetto alla GPU[3] (NVIDIA Tesla V100 SXM2 with 32GB RAM).

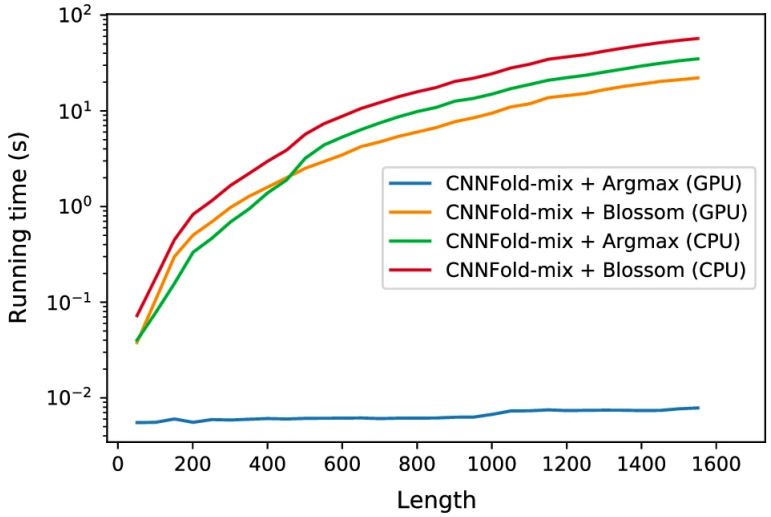


Figura 3.5: Tempi di esecuzione degli algoritmi al variare lunghezze della sequenza. Figura tratta da [4]

3.3 Risultati

3.3.1 Dasaset utilizzati

Per la analisi di tipo predittivo sulle strutture dell'RNA sono usati solitamente tre principali dataset, mostrati in Tabella 3.1 con il numero di strutture e le famiglie di RNA contenute, l'intervallo di lunghezza delle sequenze (in nucleotidi, nt) e l'utilizzo che ne viene fatto nel caso del modello studiato.

dataset	strutture	famiglie	lunghezza (nt)	utilizzo
RNAStrAlign [37]	37149[4]	8	[30,1851]	train, test
ArchiveII [38]	2975	10	[28,2968]	test
BpRNA [34]	1305		< 500	test

Tabella 3.1: Proprietà dei tre dataset

Grazie alla disponibilità di 3 diversi dataset, è stato possibile applicare al problema varie tipologie incrociate di test. Per primo, è possibile eseguire training e testing sullo stesso dataset (anche se ciò può portare a sovrastimare le prestazioni per overfitting), oppure in alternativa (tramite *cross-validation*, come suggerito in [39]) lo si può dividere in parti da dedicare separatamente a training, validation o test, come fatto su RNAStrAlign con percentuali rispettivamente dell'80%, 10% e 10%. Infine, per verificare la capacità di generalizzazione della rete così allenata, sono stati eseguiti training e test

[3]Siccome per il Blossom non è stata trovata un'implementazione adatta all'utilizzo della GPU, "CNNFold-mix + Blossom (GPU)" è stato eseguito su CPU.

rispettivamente su RNAStrAlign e ArchiveII.

3.3.2 Test utilizzato

Per valutare i modelli dopo il training sono stati usati il valore medio della *precision*, il *recall* ed il test *F1-score*, che saranno definiti in relazione al loro utilizzo.

Precision* e *Recall La precisione p è definita come il numero dei veri positivi (coppie previste in modo esatto) rispetto alla somma tra veri positivi e falsi positivi (coppie previste in realtà non esistenti) ed in particolare nel caso del modello da valutare, indica il grado di esattezza delle coppie previste. Il recupero r, invece, è definito come il numero di veri positivi rispetto alla somma tra veri positivi e falsi negativi (coppie esistenti ma non previste) e nel caso specifico studiato mostra quante delle coppie target è in grado di prevedere il modello.

F1-score È una classificazione binaria (ovvero assume valori in [0,1]) ampiamente usata nel campo del ML con scopo di misurare l'accuratezza di un test. Il calcolo è eseguito attraverso la media armonica tra precision e recall:

$$F1 = 2\frac{p \cdot r}{p + r} \tag{3.6}$$

3.3.3 Varianti del modello

Il training è stato effettuato per le varianti del modello di Figura 3.4 presentate in Tabella 3.2, dove M indica il numero di residual blocks ed N quello dei shared residual blocks, mentre con *epochs* si indica numero di cicli di training della NN.

	M	N	epochs	dataset
CNNFold	2	2	30	tutto
CNNFold-600	2	2	400	< 600 nt
CNNFold-600-big	10	2	45	< 600 nt[5]

Tabella 3.2: Differenti modelli testati

CNNFold-mix In base ai risultati associati alle tre varianti appena presentate in Tabella 3.2, si è deciso di introdurre il metodo CNNFold-mix, che sfrutta le prestazioni ottimali delle varianti CNNFold e CNNFold-600-big su porzioni particolari di RNAStrAlign, selezionate in base alla lunghezza delle sequenze. CNNFold-mix utilizza infatti i risultati di CNNFold per sequenze con lunghezza superiore a 600 nt (F1-score 0.916) e CNNFold-600-big per le sequenze più brevi di 600 nt (F1-score 0.970), consentendo di migliorare ulteriormente i risultati previsti dal modello.

È stato effettuato il training sui tre diversi modelli usando un algoritmo per l'ottimizzazione del tasso di apprendimento (*learning rate*), denominato *Adam*[6], con un *learning rate* pari a 0.005, in aggiunta all'utilizzo di *mini-batches*[7] rispettivamente con una sequenza o 16 sequenze nel caso di sequenze più lunghe di 1000 nt oppure sequenze brevi, per evitare problemi di memoria. Viene evidenziato nell'articolo inoltre il numero nettamente inferiore di parametri complessivamente presenti nel modello

[4] Visto che il dataset non è ridondante per sequenze ma per strutture secondarie da esse prodotte, il numero di strutture contenute in RNAStrAlign scende a 30451 se si considera in modo random solo una struttura secondaria nel caso ad un sequenza ne siano associate molteplici.

[5] Per limiti dovuti alla memoria, CNNFold-600-big non è applicabile per lunghe sequenze.

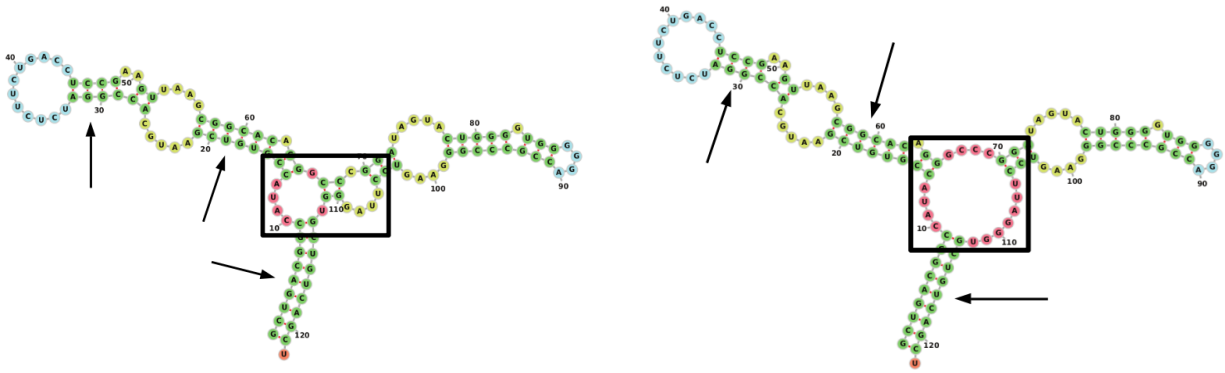
[6] Esegue un'ottimizzazione basata su gradiente del primo ordine di funzioni obiettivo stocastiche [40].

[7] Se con *epoch* si indica un processo sull'intero training set, esiste anche la possibilità di usare tutti i dati in una singola iterazione, in tal caso si parla di *batch*, oppure solo un sottoinsieme dei dati, nel caso di una *mini-batch*.

se confrontato con metodi "rivali" come E2EFold [39] e SPOT-RNA [34], rispetto ai quali il modello è stato messo alla prova nella sezione successiva.

3.3.4 Presentazione dei risultati

In Figura 3.6 sono visualizzate la struttura attesa rispetto a quella ottenuta tramite CNNFold-mix (con precisione del 93.4%) per E00001 della famiglia di RNA 5S. I diagrammi sono stati generati utilizzando Forna [41] ed i riquadri evidenziano le differenze strutturali negli *stem loops*, mentre le frecce indicano differenze di inclinazione e curvatura negli *stem segments* tra le due strutture.



(a) Target (b) CNNFold-mix

Figura 3.6: Confronto tra la struttura attesa (a) e quella prevista dal modello (b). Figure tratte da [4]

I risultati ottenuti da CNNFold-mix apportano miglioramenti significativi nella predizione delle strutture rispetto ad alcuni metodi attuali come E2EFold e MXFold2, rispettivamente sui dataset RNAStrAlign e ArchiveII, come osservabile dalla Tabella 3.3, mentre su BpRNA riesce ad eguagliare le prestazioni di SPOT-RNA pur non avendo ricevuto un training ottimizzato per tale dataset.

	RNAStrAlign			ArchiveII		
	precision	recall	F1	precision	recall	F1
Mfold [42]	0.450	0.398	0.420	0.428	0.383	0.401
CDPfold [43]	0.516	0.568	0.540	0.557	0.535	0.545
RNAstructure [44]	0.537	0.568	0.550	0.563	0.615	0.585
RNAfold [45]	0.620	0.606	0.609	0.565	0.627	0.592
LinearFold [46]	0.633	0.597	0.614	0.641	0.617	0.621
CONTRAfold [31]	0.608	0.663	0.633	0.607	0.679	0.638
E2Efold [39]	0.866	0.788	0.821	0.734	0.660	0.686
MXFold2 [47]				0.790	0.815	0.800
CNNFold+Argmax	0.955	0.861	0.900			
CNNFold-mix+Argmax	0.956	0.912	0.932			
CNNFold-mix+Blossom	**0.975**	**0.907**	**0.936**	**0.928**	**0.879**	**0.897**

Tabella 3.3: Risultati di vari modelli sui dataset RNAStrAlign e ArchiveII. Tabelle tratta da [4]

Altro elemento a favore di CNNFold, oltre agli ottimi risultati di precisione, recall e F1 ottenuti, è la validità biologivca delle sequenze prodotte grazie ai metodi di post-processing. In generale si osserva una grande dipendenza dei risultati dalle famiglie di RNA analizzate, con difficoltà riscontrate da CNNFold soprattutto su Telomerase e SRP, visualizzabili in Figura 3.7, in cui ogni punto rappresenta una sequenza ed i diversi colori indicano sequenze delle 8 diverse famiglie di RNA presenti in RNAStrAlign.

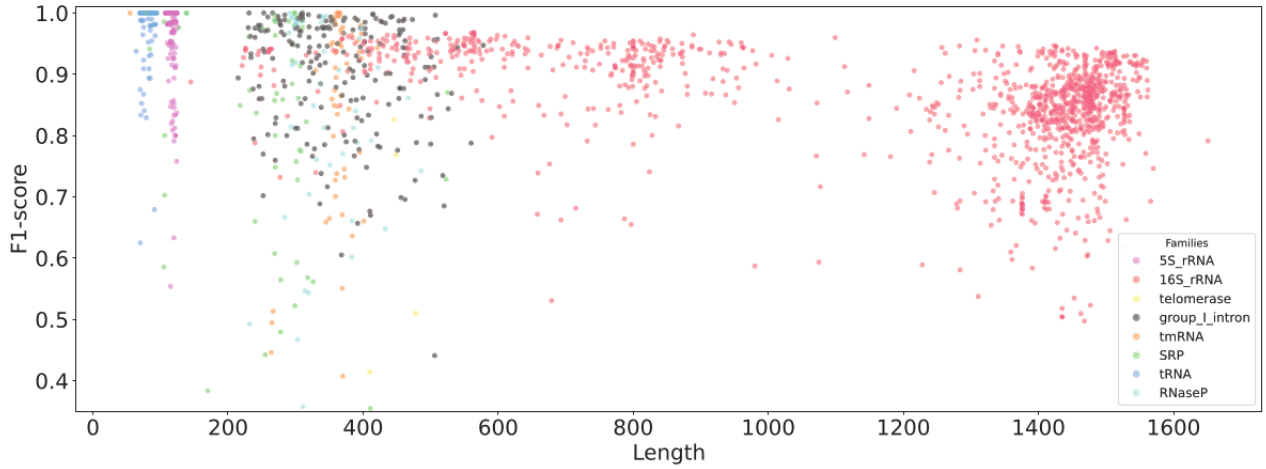


Figura 3.7: Grafico a dispersione di F1-score prodotto da CNNFold-mix rispetto alla lunghezza delle sequenze. Figura tratta da [4]

Per quanto riguarda le strutture non canoniche trattate, sulle 1413 **strutture *pseudoknotted***, notoriamente di difficile predizione, presenti in RNAStrAlign, sono state ottenute prestazioni analoghe (F1-score pari a 0.857) a quelle ottenute su strutture non contenenti *pseudoknots*, ad indicare l'ottimo funzionamento di CNNFold-mix su qualsiasi tipo di sequenze. Inoltre se si confrontano i risultati con quelli di altri metodi su questa porzione di dataset, si osserva che sia F1-score che il numero totale di strutture correttamente individuate come *pseudoknotted* (1412 su 1413) sono ampiamente superiori.

Capitolo 4

Conclusioni

Per la previsione delle strutture secondarie associate alle sequenze di RNA, l'approccio basato sulla minimizzazzione dell'energia ha raggiunto un limite dal punto di vista delle prestazioni, motivo per cui ha preso sempre più piede un'altro tipo di approccio, quello basato sul machine learning. Vari modelli che sfruttano ML, hanno ottenuto prestazioni superiori a quelle precedentemente osservabili, apportando miglioramenti sensibili sotto vari punti di vista: le strutture prevedibili possono essere più complicate e contenere coppie non canoniche, non si ha una dipendenza dal modello energetico adottato ed il processo è computazionalmente più veloce.
In particolare tra i modelli *ML-based*, CNNFold si è distinto per l'utilizzo di una matrice di contatto per rappresentare le possibili coppie tra basi della sequenza, presa come input della CNN. L'output della rete è a sua volta una matrice, che indica quante e quali basi sono appaiate tra loro e quali invece rimangono spaiate. Da questa struttura, tramite alcuni algoritmi di *post-processing*, sono infine ottenibili le strutture secondarie dell'RNA.
Si è potuto osservare nella sezione di esposizione dei risultati come la variante CNNFold-mix ottenga risultati analoghi, se non in gran parte dei casi migliori, rispetto agli altri modelli per prevedere strutture secondarie dell'RNA, nonostante la rete adoperata sia strutturalmente meno complessa di altre. È stato usato come test il *F1-score*, lavorando su 3 dataset diversi o dividendo un dataset in base alla lunghezza delle sequenze contenute, al fine di testare CNNFold su dati i quali non fossero precedentemente stati utilizzati per il training.
In un campo in cui non è possibile effettuare *data argumentation*, ossia aggiungere ai dati nuovi elementi "sintetici" per aumentare la dimensione del campione, sono stati utilizzati i dati a disposizione, pur essendo questo limitante per via del numero ancora ridotto di strutture caratterizzate sperimentalmente in laboratorio.
Una critica che si può avanzare all'esposizione dei dati da parte dell'articolo è la scelta dei modelli rispetto ai quale confrontare i risultati in Tabella 3.3. Buona parte di essi hanno un approccio *score-based* e sfruttano algoritmi di minimizzazione dell'energia, producendo risultati nettamente inferiori. Non è nemmeno menzionato, ad esempio, UFold [35], che produce strutture con valore di F1-score attorno a 0.9, mentre è solamente citato ma non inserito in tabella SPOT-RNA [34].
Se si osservano i risultati di *precision* e *recall* prodotti dai test su CNNFold, si osserva che la precisione è costantemente la grandezza maggiore delle due. Questo indica che i falsi negativi (nucleotidi considerati liberi che in realtà sono appaiate) sono più dei falsi positivi (basi singole che vengono erroneamente considerate appaiate). Gli autori affermano il modello sia ulteriormente migliorabile, con lo scopo in futuro di risolvere le problematiche che rendono al giorno d'oggi ancora difficile prevedere elementi strutturali di importanza biologica, dato che le molecole di RNA con funzione dipendente dalla struttura possiedono una conformazione *folded* in modo unico ed univoco [48]. Alcune ottimizzazioni proposte dagli autori sono l'utilizzo nella fase di training di informazioni legate alla lunghezza e alla famiglia di appartenenza della sequenza in input. Per l'obiettivo di arrivare a progettare sequenze di RNA con le proprietà richieste potrebbe essere utile inoltre estendere il modello alla previsione di strutture multiple alternative a partire dalla stessa sequenza, ottenibili attraverso differenti algoritmi di *post-processing*.

Bibliografia

[1] Eric Westhof. Twenty years of RNA crystallography. *RNA*, 21(4):486–487, March 2015.

[2] Ian Kings Oluoch, Abdullah Akalin, Yilmaz Vural, and Yavuz Canbay. A review on rna secondary structure prediction algorithms. In *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pages 18–23, 2018.

[3] Qi Zhao, Zheng Zhao, Xiaoya Fan, Zhengwei Yuan, Quian Mao, and Yudong Yao. Review of machine learning methods for rna secondary structure prediction. *PLoS Computational Biology*, 17(8), 2021.

[4] Mehdi Saman Booy, Alexander Ilin, and Pekka Orponen. RNA secondary structure prediction with convolutional neural networks. *BMC Bioinformatics*, 23(58), February 2022.

[5] Dimiter Dobrev. A definition of artificial intelligence. *arXiv preprint arXiv:1210.1568*, 2012.

[6] Chenyang Shen, Dan Nguyen, Zhiguo Zhou, Steve B. Jiang, Bin Dong, and Xun Jia. An introduction to deep learning in medical physics: advantages, potential, and challenges. *The international journal of biomedical physics and engineering*, 65(5):05TR01, March 2020.

[7] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, October 1950.

[8] Peter Stone, Rodney Brooks, Erik Brynjolfsson, and et al. Artificial intelligence and life in 2030: the one hundred year study on artificial intelligence. In *One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel*, Stanford, CA, September 2016. Stanford University.

[9] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Rev. Mod. Phys.*, 91:045002, Dec 2019.

[10] Juan M. Górriz, Javier Ramírez, Andrés Ortíz, Francisco J. Martínez-Murcia, Fermin Segovia, John Suckling, Matthew Leming, Yu-Dong Zhang, Jose Ramón Álvarez Sánchez, Guido Bologna, Paula Bonomini, Fernando E. Casado, David Charte, Francisco Charte, Ricardo Contreras, Alfredo Cuesta-Infante, Richard J. Duro, Antonio Fernández-Caballero, Eduardo Fernández-Jover, Pedro Gómez-Vilda, Manuel Graña, Francisco Herrera, Roberto Iglesias, Anna Lekova, Javier de Lope, Ezequiel López-Rubio, Rafael Martínez-Tomás, Miguel A. Molina-Cabello, Antonio S. Montemayor, Paulo Novais, Daniel Palacios-Alonso, Juan J. Pantrigo, Bryson R. Payne, Félix de la Paz López, María Angélica Pinninghoff, Mariano Rincón, José Santos, Karl Thurnhofer-Hemsi, Athanasios Tsanas, Ramiro Varela, and Jose M. Ferrández. Artificial intelligence within the interplay between natural and artificial computation: Advances in data science, trends and applications. *Neurocomputing*, 410:237–270, 2020.

[11] Yang Lu. Artificial intelligence: a survey on evolution, models, applications and future trends. *Journal of Management Analytics*, 6(1):1–29, 2019.

[12] Burak Koçak, Emine Şebnem Durmaz, Ece Ateş, and Özgür Kılıçkesmez. Radiomics with artificial intelligence: a practical guide for beginners. *Diagnostic and Interventional Radiology*, 25(6):485–495, 2019.

[13] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[14] Sunan Cui, Huan-Hsin Tseng, Julia Pakela, Randall K. Ten Haken, and Issam El Naqa. Introduction to machine and deep learning for medical physicists. *Medical Physics*, 47(5):e127–e147, 2020.

[15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[16] Andreas Maier, Christopher Syben, Tobias Lasser, and Christian Riess. A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik*, 29(2):86–101, 2019. Special Issue: Deep Learning in Medical Physics.

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

[18] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, May 2019.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[20] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3. Citeseer, 2013.

[21] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology*, 5(23):495, 2017.

[22] Yann LeCun, Bernhard Boser Boser, John S. Denker, Don Henderson, Richard E. Howard, W. E. Hubbard, and Larry D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[25] Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, January 1981.

[26] Amal S. Abu Almakarem, Anton I. Petrov, Jesse Stombaugh, Craig L. Zirbel, and Neocles B. Leontis. Comprehensive survey and geometric classification of base triples in RNA structures. *Nucleic Acids Research*, 40(4):1407–1423, Nov 2011.

[27] Boris Fürtig, Christian Richter, Jens Wöhnert, and Harald Schwalbe. NMR spectroscopy of RNA. *ChemBioChem*, 4(10):936–962, September 2003.

[28] Robin R. Gutell, Jung C. Lee, and Jamie J. Cannone. The accuracy of ribosomal RNA comparative structure models. *Current Opinion in Structural Biology*, 12(3):301–310, June 2002.

[29] Sarah W. Burge, Jennifer Daub, Ruth Eberhardt, John Tate, Lars Barquist, Eric P. Nawrocki, Sean R. Eddy, Paul P. Gardner, and Alex Bateman. Rfam 11.0: 10 years of RNA families. *Nucleic Acids Research*, 41(D1):D226–D232, November 2012.

[30] Ruth Nussinov and A.B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

[31] Chuong B. Do, Daniel A. Woods, and Serafim Batzoglou. Contrafold: Rna secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, July 2006.

[32] Yu Zhu, ZhaoYang Xie, YiZhou Li, Min Zhu, and Yi-Ping Phoebe Chen. Research on folding diversity in statistical learning methods for rna secondary structure prediction. *International journal of biological sciences*, 14(8):872–882, 2018.

[33] Yoshiyasu Takefuji, Li-Lin Chen, Kuo-Chun Lee, and John Huffman. Parallel algorithms for finding a near-maximum independent set of a circle graph. *IEEE Transactions on Neural Networks*, 1(3):263–267, 1990.

[34] Jaswinder Singh, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. Rna secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning. *Nature Communications*, 10(1):5407, Nov 27 2019.

[35] Laiyi Fu, Yingxin Cao, Jie Wu, Qinke Peng, Qing Nie, and Xiaohui Xie. Ufold: Fast and accurate rna secondary structure prediction with deep learning. *bioRxiv*, 2021.

[36] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[37] Zhen Tan, Yinghan Fu, Gaurav Sharma, and David H. Mathews. TurboFold II: RNA structural alignment and secondary structure prediction informed by multiple homologs. *Nucleic Acids Research*, 45(20):11570–11581, Nov 2017.

[38] Michael F. Sloma and David H. Mathews. Exact calculation of loop formation probability identifies folding motifs in RNA secondary structures. *RNA*, 22(12):1808–1818, October 2016.

[39] Xinshi Chen, Yu Li, Ramzan Umarov, Xin Gao, and Le Song. Rna secondary structure prediction by learning unrolled algorithms. *arXiv preprint arXiv:2002.05810*, 2020.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Peter Kerpedjiev, Stefan Hammer, and Ivo L. Hofacker. Forna (force-directed rna): Simple and effective online rna secondary structure diagrams. *Bioinformatics*, 31(20):3377–3379, Jun 2015.

[42] Nicholas R. Markham and Michael Zuker. *UNAFold*, pages 3–31. Humana Press, Totowa, NJ, 2008.

[43] Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for molecular biology*, 6(1):1–14, 2011.

[44] Stanislav Bellaousov, Jessica S. Reuter, Matthew G. Seetin, and David H. Mathews. Rnastructure: web servers for rna secondary structure prediction and analysis. *Nucleic Acids Research*, 41(W1):W471–W474, 2013.

[45] Dezhong Deng, Kai Zhao, David Hendrix, David H. Mathews, and Liang Huang. Linearfold: Linear-time prediction of rna secondary structures. *bioRxiv*, 2018.

[46] Hao Zhang, Chunhe Zhang, Zhi Li, Cong Li, Xu Wei, Borui Zhang, and Yuanning Liu. A new method of rna secondary structure prediction based on convolutional neural network and dynamic programming. *Frontiers in genetics*, 10:467, 2019.

[47] Kengo Sato, Manato Akiyama, and Yasubumi Sakakibara. Rna secondary structure prediction using deep learning with thermodynamic integration. *Nature communications*, 12(1):1–9, 2021.

[48] Shu-Yun Le, Kaizhong Zhang, and Jacob V. Maizel Jr. Rna molecules with structure dependent functions are uniquely folded. *Nucleic acids research*, 30(16):3574–3582, 2002.