ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

CLASSIFICAZIONE DI RADIOGRAFIE TORACICHE PER LA DIAGNOSI DEL COVID-19 CON RETI CONVOLUZIONALI E VISION TRANSFORMER

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Davide Domini

Seconda Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

COVID-19

Transfer Learning

Convolutional Neural Networks

Radiografie del torace

Vision Transformer

A chiunque mi abbia supportato
in questo lungo percorso.

Introduzione

Negli ultimi due anni – per via della pandemia generata dal virus Covid19 – la vita in ogni angolo del nostro pianeta è drasticamente cambiata. Ad oggi – nel mondo – sono oltre duecentoventi milioni le persone che hanno contratto questo virus e sono quasi cinque milioni le persone decedute. In alcuni periodi si è arrivati ad avere anche un milione di nuovi contagiati al giorno e mediamente – negli ultimi sei mesi – questo dato è stato di più di mezzo milione al giorno.

Gli ospedali – soprattutto nei paesi meno sviluppati – hanno subito un grande stress e molte volte hanno avuto una carenza di risorse per fronteggiare questa grave pandemia. Per questo motivo ogni ricerca in questo campo diventa estremamente importante, soprattutto quelle che – con l'ausilio dell'intelligenza artificiale – riescono a dare supporto ai medici. Queste tecnologie una volta sviluppate e approvate possono essere diffuse a costi molto bassi e accessibili a tutti.

In questo elaborato verranno sperimentati e valutati due diversi approcci alla diagnosi del Covid-19 a partire dalle radiografie toraciche dei pazienti: il primo metodo si basa sul transfer learning di una rete convoluzionale inizialmente pensata per la classificazione di immagini. Il secondo approccio utilizza i Vision Transformer (ViT), un'architettura ampiamente diffusa nel campo del Natural Language Processing adattata ai task di Visione Artificiale.

I capitoli sono divisi come segue:

- *Capitolo 1*, vengono presentate la nomenclatura e le tecniche di base della moderna Intelligenza Artificiale.

- *Capitolo 2*, vengono presentate reti neurali più complesse e le loro caratteristiche.

- *Capitolo 3*, viene presentata nel dettaglio l'architettura dei Vision Transformer.

- *Capitolo 4*, viene effettuato un riassunto delle varie tecnologie utilizzate per sviluppare questo progetto.

- *Capitolo 5*, viene presentato lo sviluppo del progetto con una valutazione dei modelli ottenuti.

Indice

Elenco delle figure

Capitolo 1

Panoramica delle tecniche di Intelligenza Artificiale

In questa sezione si vogliono illustrare i principali aspetti di questa disciplina, essendo una branca dell'informatica molto ampia dopo una breve panoramica ad alto livello ci addentreremo nella parte più legata agli argomenti di questa tesi.

1.1 Introduzione

Questa materia si compone di un vastissimo numero di tecniche, le quali, sono a loro volta usate in molti contesti diversi. Questo porta a confondere alcuni termini fra loro, per chiarezza definiamo come verranno usati i vari nomi nel corso dei prossimi capitoli.
L'intelligenza artificiale è l'insieme dei programmi che hanno abilità di imparare e ragionare in modo simile agli umani, un suo sottoinsieme è il *machine learning*, ovvero tutti gli algoritmi che riescono ad imparare senza avere delle regole hand-coded ben definite su cui basarsi. A sua volta il machine learning è in parte composto dal *deep learning*, cioè tecniche basate su un elevato numero di strati di neuroni artificiali che riescono ad imparare partendo da un insieme di dati molto ampio. Questa situazione è rappresentabile molto bene da un diagramma di Eulero Venn come quello in figura 1.1.

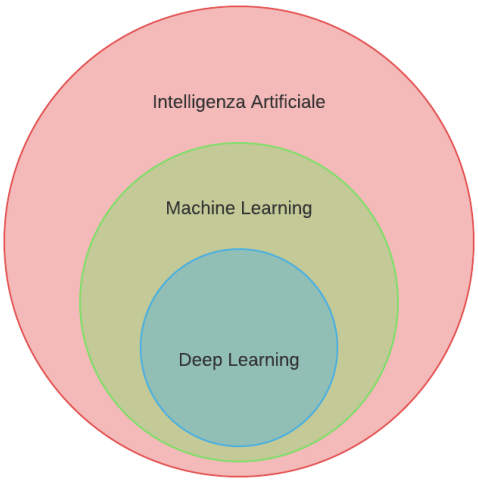


Figura 1.1: Diagramma di Eulero Venn per l'intelligenza artificiale

La principale differenza fra Machine Learning e Deep Learning sta nel fatto che nei modelli di ML le feature dei dati ritenute importanti vengono selezionate da un esperto del dominio applicativo mentre nei modelli di DL queste vengono selezionate in automatico dal modello stesso (figura 1.2).

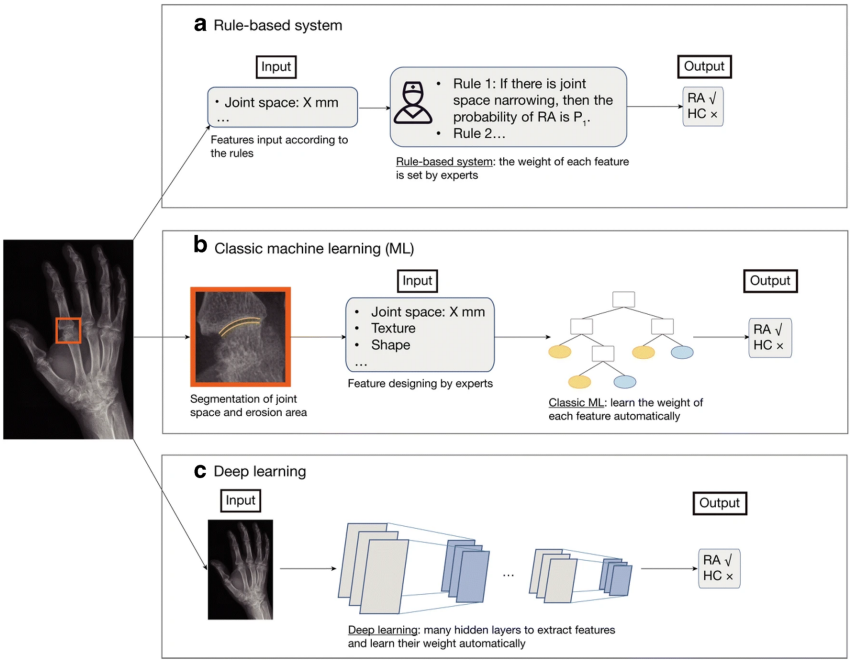


Figura 1.2: Differenze fra ML e DL.
Fonte: [1]

Addentrandoci nella parte riguardante il machine learning possiamo schematizzare un'altra divisione (come in figura 1.3) :

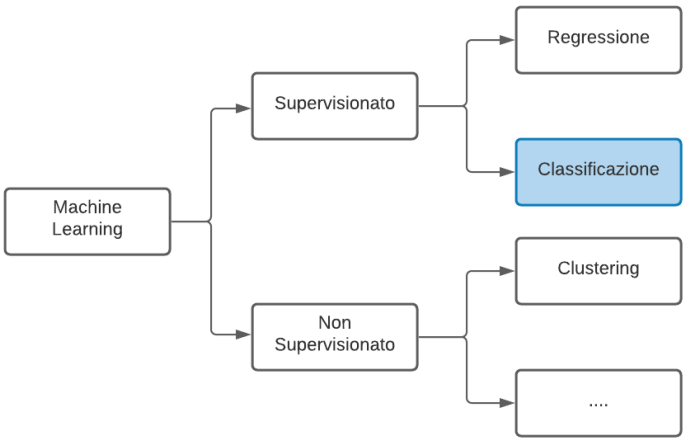


Figura 1.3: Schematizzazione del machine learning

In questa tesi si discuterà principalmente della parte riguardante la classificazione, ovvero la predizione di una variabile discreta (ad esempio una variabile binaria 0/1). Questi algoritmi rientrano nella categoria "Supervisionato" in quanto in fase di training conosciamo il valore reale della variabile da predire e possiamo sfruttarlo per calcolare un errore da minimizzare.

1.2 Classificazione

La classificazione ha come scopo la predizione di una variabile discreta. I valori che può assumere questa variabile rappresentano le classi a cui può appartenere una singola istanza del dataset. Nel caso le classi siano solo due si parla di classificazione semplice. Invece nel caso siano più di due si parla di classificazione multiclasse.
La più semplice forma di classificazione prevede l'individuazione di un iperpiano che separi le istanze delle due classi nel miglior modo possibile, questo è visualizzabile nello spazio tridimensionale come in figura 1.4.

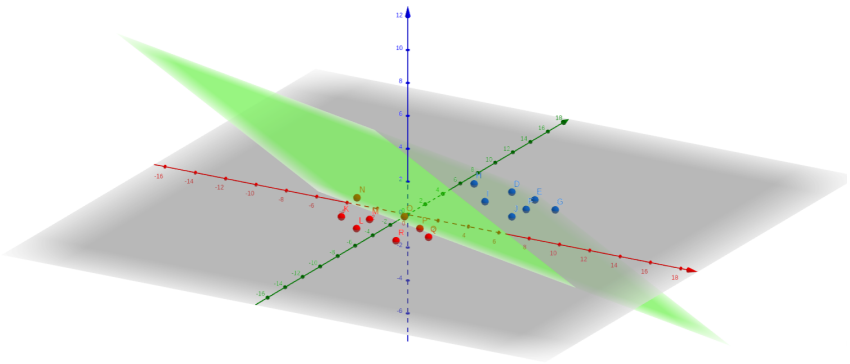


Figura 1.4: Iperpiano nello spazio R^3

Un iperpiano in un generico spazio R^n è definito dall'equazione:

$$h = b + w_1 \cdot x_1 + ... + w_n \cdot x_n = b + \mathbf{w^T} \cdot \mathbf{x}$$

Dunque per trovare l'iperpiano ottimale rispetto ai nostri dati dobbiamo calcolare i parametri b e $\mathbf{w}$, questo si può fare applicando l'algoritmo di discesa sul gradiente (descritto nella sezione 1.3) e minimizzando la seguente funzione di errore detta *Logistic Loss*:

$$\sum_{i=1}^{m} log(1 + e^{-y_i(b+\mathbf{w^T} \cdot \mathbf{x_i})}) + \lambda ||\mathbf{w}||_2^2$$

Una volta ottenuta l'equazione dell'iperpiano si usa la seguente funzione – detta *regressione logistica* o *sigmoide* – per calcolare la classe di appartenenza C di una data istanza x_j:

$$C = \frac{1}{1 + e^{-(b+\mathbf{w^T} \cdot \mathbf{x_j})}}$$

In particolare la sigmoide approssima una funzione a gradino che assume valori nell'intervallo $[0; 1]$ (come in figura 1.5), quindi è interpretabile come la probabilità $P(Y = 1|X = x_j)$ che una data istanza appartenga alla classe positiva.

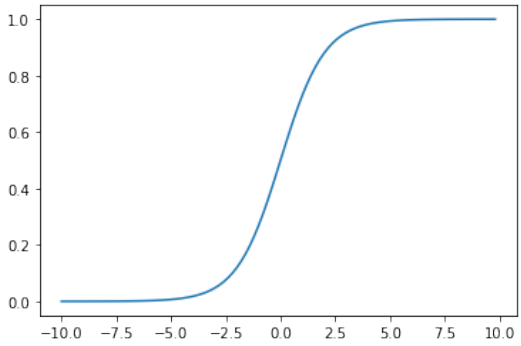


Figura 1.5: Grafico della funzione sigmoide

L'aspetto di questo grafico è dato dal fatto che se l'istanza presa in considerazione fosse collocata esattamente sull'iperpiano allora non sapremmo con certezza la classe di appartenenza, infatti in questo caso l'equazione dell'iperpiano assumerebbe valore 0 e la sigmoide valore $\frac{1}{1+e^0} = \frac{1}{2} = 50\%$. Se invece ci allontanassimo dall'iperpiano (posizionandoci nel semispazio superiore) la sigmoide assumerebbe valore tendente ad 1 (100% di probabilità). Nel caso si volesse invece calcolare la probabilità di appartenenza alla classe negativa basterebbe utilizzare la formula inversa:

$$P(Y = 0|X = x_j) = 1 - P(Y = 1|X = x_j) = 1 - \frac{1}{1 + e^{-(b+\mathbf{w^T}\cdot\mathbf{x_j})}}$$

1.3 Discesa sul gradiente

La discesa sul gradiente è un metodo matematico iterativo utile per trovare un minimo locale di una certa funzione, se questa è la funzione di errore di un metodo di machine learning possiamo usarlo per trovare i parametri ottimali.

Il gradiente di una generica funzione $f : \Re^n \to \Re^n$ è il vettore che ha per componenti le sue derivate parziali: $\nabla f(x_1, ..., x_n) = (\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, ..., \frac{\delta f}{\delta x_n})$. Geometricamente questo vettore indica la direzione di massima crescita della funzione in un dato punto, dunque lo stesso vettore cambiato di segno indica la direzione di massima decrescita. Questo si può osservare nella figura 1.6, in cui sono rappresentate in scala di grigi due differenti funzioni (il colore nero indica i valori più piccoli mentre il bianco i valori più grandi).

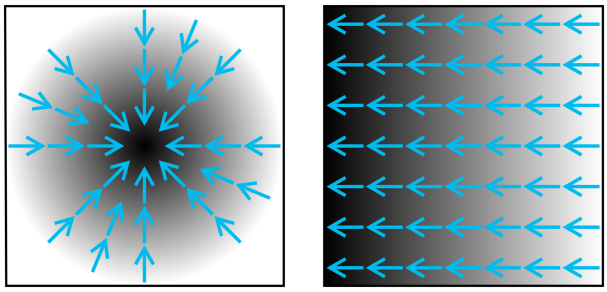


Figura 1.6: Gradiente inverso di due funzioni

Il funzionamento di base di questo algoritmo è il seguente:

1. Trovare una funzione di errore per il problema corrente (ad esempio logistic loss per classificazione o MSE per regressione).

2. Iniziare con dei parametri casuali e calcolare l'errore.

3. Calcolare il gradiente in quel punto.

4. Sottrarre ai parametri attuali il gradiente calcolato in modo da spostarsi verso un punto in cui l'errore è minore di quello attuale.

5. Tornare al punto 3 finché non si è raggiunta una qualche condizione di stop.

Matematicamente, dunque, data una generica funzione di errore $E(w)$ si calcola il gradiente $\nabla_w(E)$ e si esegue iterativamente la seguente operazione:

$$w_{i+1} = w_i - \eta \cdot \nabla_{w_i}(E)$$

In modo da ottenere $E(w_{i+1}) < E(w_i)$ finché non si raggiunge la condizione di stop. Lo scalare η che viene moltiplicato al gradiente è un iper-parametro detto *passo di discesa* che controlla l'intensità della discesa ed influenza l'accuratezza del risultato.

Questa discesa si può osservare graficamente in figura 1.7, inoltre si può notare anche che limitandoci solo ad un minimo locale se si cambiasse punto di inizio allora si potrebbe trovare un minimo locale diverso e di conseguenza dei parametri ottimali diversi.

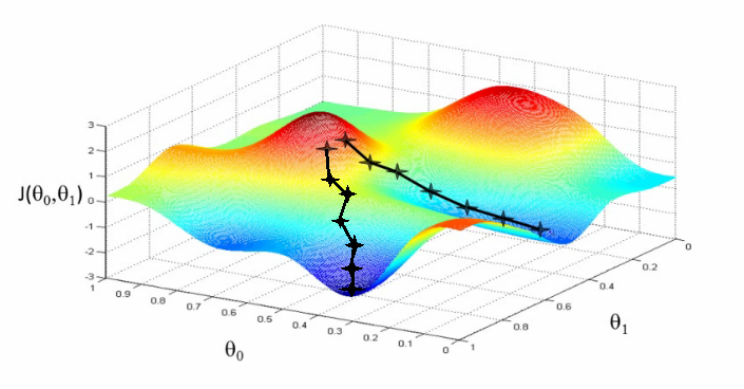


Figura 1.7: Discesa sul gradiente.
Fonte: https://forum.huawei.com/enterprise/en/machine-learning-training-method-gradient-descent-method/thread/708303-893

Nei problemi di classificazione – dove si utilizza la logistic loss – otteniamo come gradiente:

$$\nabla_{\tilde{w}}(\sum_{i=1}^{m} log(1 + e^{-y_i h_{\tilde{w}}(\tilde{x_i})}) + \frac{1}{2}\lambda||\tilde{w}||_2^2) = \sum_{i=1}^{m} -\frac{y_i \cdot \tilde{x_i}}{1 + e^{y_i \cdot h_{\tilde{w}}(\tilde{x_i})}} + \lambda\tilde{w}$$

Dove $\tilde{x} = (1, \mathbf{x})$, $\tilde{w} = (b, \mathbf{w})$ e $h_{\tilde{w}}(\tilde{x_i}) = \tilde{w} \cdot \tilde{x_i}$

1.4 Reti neurali

Le reti neurali sono modelli matematici complessi – ispirati alla struttura del cervello umano – in grado di rappresentare relazioni non lineari fra i dati.

Il neurone biologico

Il cervello umano è composto da miliardi di neuroni interconnessi fra loro, questi scambiandosi dati (sotto forma di impulsi elettro-chimici) riescono a compiere tutte le funzioni a cui siamo abituati, come: imparare, pensare, avere una memoria a lungo termine e molte altre.
La medicina odierna è riuscita a scoprire come sono strutturati i singoli neuroni (figura 1.8) e in che modo comunicano fra loro:

- I *dendriti* sono le fibre minori che ramificandosi a partire dal soma raccolgono gli input per il neurone
- *L'assone* è una fibra che, tramite i vari *terminali assonici*, espone l'output del neurone

I dendriti e i terminali assonici dei vari neuroni sono collegati attraverso le *sinapsi*.

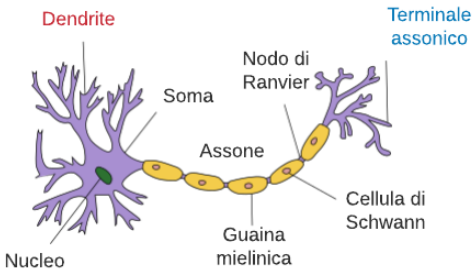


Figura 1.8: Struttura di un neurone biologico.
Fonte: https://it.wikipedia.org/wiki/Neurone

Il neurone artificiale

Il primo tentativo di simulare il comportamento di un neurone biologico fu condotto da McCulloch e Pitts nel 1943 [2]. Successivamente si è costruito un modello matematico (figura 1.9) somigliante in grado di approssimare e rappresentare i dati.

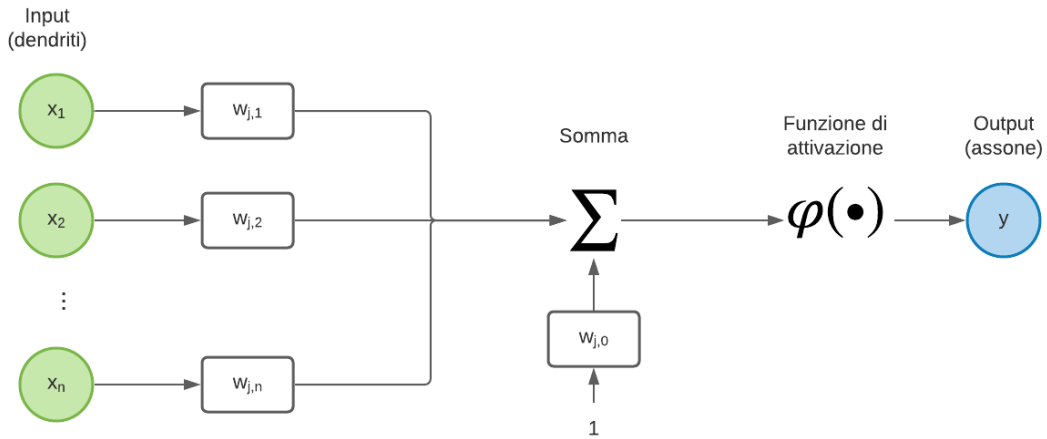


Figura 1.9: Struttura di un neurone artificiale

In particolare alle variabili di input viene associato un peso e successivamente vengono sommate fra loro, con l'aggiunta di un bias utile a controllare il punto

di lavoro ottimale del neurone. Il risultato di queste operazioni viene usato come input per una funzione di attivazione (solitamente la funzione sigmoide) da cui si ottiene l'output del neurone, questo ultimo passaggio è utile per rappresentare relazioni non-lineari fra i dati di input. Il risultato di un singolo neurone è quindi calcolabile con la seguente formula:

$$out_j = \varphi(w_{j,0} + \sum_{i=1}^{m} x_{j,i} \cdot w_{j,i})$$

I pesi w sono i parametri da ottimizzare in fase di training, inoltre se, come detto in precedenza, la funzione di attivazione è la funzione sigmoide otteniamo che ogni neurone effettua semplicemente l'ottimizzazione di un singolo iperpiano. Come funzione di attivazione, oltre alla sigmoide, vengono solitamente utilizzate anche: tangente iperbolica, ReLu (Rectified Linear Unit) e una funzione a gradino. (Grafici in figura 1.10)

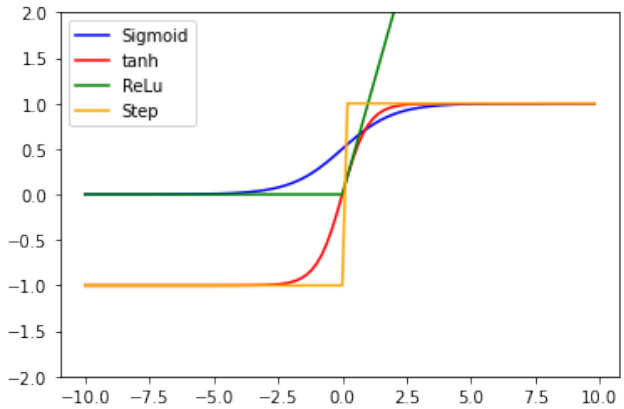


Figura 1.10: Grafici delle più comuni funzioni di attivazione

Reti di neuroni

Nelle reti neurali i singoli neuroni artificiali descritti in precedenza vengono disposti in strati e poi interconnessi fra di loro, il tipo di connessione comunemente usata è una connessione densa, ovvero l'output di un neurone nello strato i-esimo è propagato in input a tutti i neuroni dello strato $i + 1$. (come in figura 1.11)

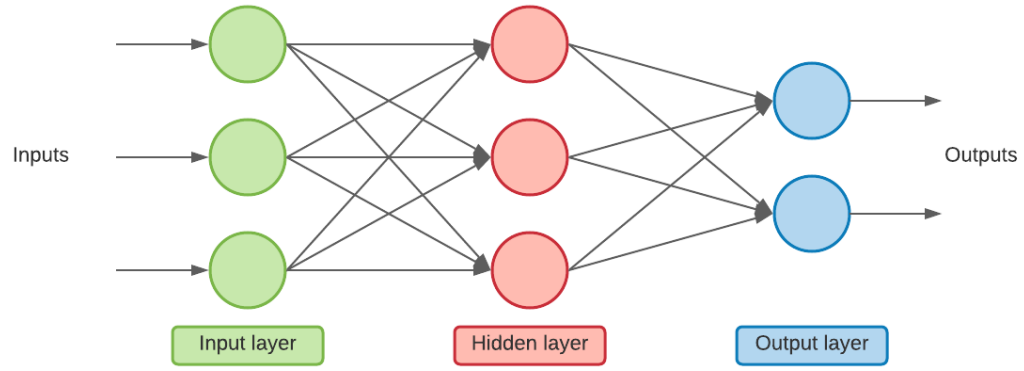


Figura 1.11: Struttura di una rete neurale densamente connessa

Geometricamente queste reti neurali trasformano lo spazio di partenza dei dati per renderli linearmente separabili, come nei grafici 1.12 e 1.13, dove viene automaticamente aggiunta la feature X^2 per poter separare le istanze con una semplice retta.

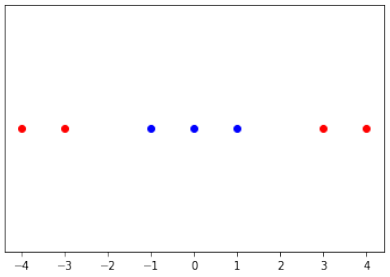


Figura 1.12: Spazio originale

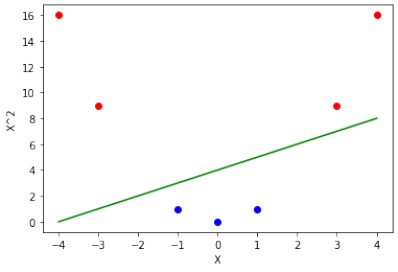


Figura 1.13: Spazio trasformato

Ad esempio se avessimo come input i dati presenti in figura 1.14, certamente non avremmo chances di ottenere buoni risultati con un classificatore lineare. Usando invece una rete neurale lo spazio di partenza subirebbe molteplici trasformazioni facendo diventare i dati linearmente separabili. Andando a riportare il risultato nello spazio dei dati originario (figura 1.15) quello che si ottiene è una separazione non-lineare con un'accuratezza molto elevata.

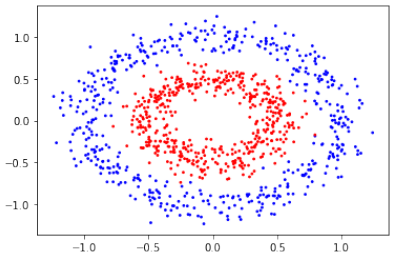


Figura 1.14: Dati non lineari

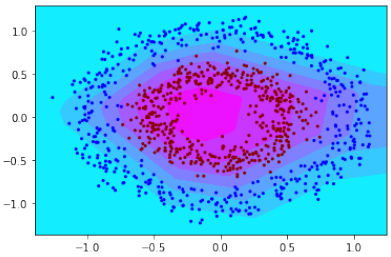


Figura 1.15: Risultato classificazione

Le reti neurali possono avere diverse architetture, possiamo distinguerle in due grandi gruppi:

- Reti *feedforward*, sono reti che consentono collegamenti fra neuroni solo in avanti, ovvero l'output di un neurone del livello i-esimo può essere collegato solo all'input di un neurone del livello $i + 1$.
- Reti *ricorrenti*, sono consentiti anche collegamenti con neuroni dello stesso livello o dei livelli precedenti. Questo complica la fase di training ma permette di avere un effetto di memoria a breve termine che in alcuni casi è utile ad incrementare l'accuratezza.

Per effettuare il training di queste reti viene usato un algoritmo di backpropagation introdotto da Rumelhart et al. 1985 [3], nella pratica è una discesa del gradiente che ottimizza e automatizza la fase di calcolo del gradiente stesso. L'algoritmo funziona essenzialmente nel seguente modo: in un primo momento vengono passate le istanze di training alla rete (fase di forward), una volta arrivati al layer di output viene calcolata una funzione di loss che indica quanto l'output ottenuto si discosta dall'output reale. A questo punto viene effettuata una fase di backward in cui viene stimato quanto, ogni collegamento presente nella rete abbia influito sull'errore, infine basandosi sui risultati ottenuti dalla fase precedente si passa alla discesa sul gradiente che aggiorna i vari parametri in modo da ridurre l'errore.

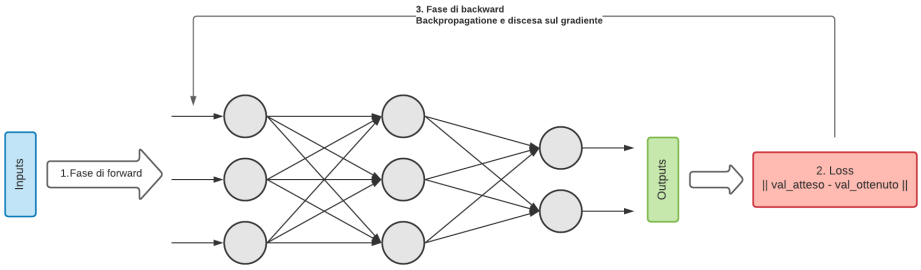


Figura 1.16: Schematizzazione dell'algoritmo di backpropagation

Quando queste reti vengono utilizzate come classificatori si deve usare una speciale rappresentazione delle possibili classi di output detta: One-Hot-Encoding. In questa rappresentazione si utilizzano tante variabili binarie quante sono le possibili classi, così facendo avranno tutte valore 0 tranne quella corrispondente alla classe dell'istanza corrente che avrà valore 1 (Come in figura 1.17). Nella rete neurale si avrà un neurone di output per ogni variabile binaria, ognuno di questi neuroni poi, grazie alla funzione di attivazione softmax, esprimerà una probabilità di appartenenza dell'istanza alla relativa classe, dunque la classe con probabilità massima sarà la prescelta.

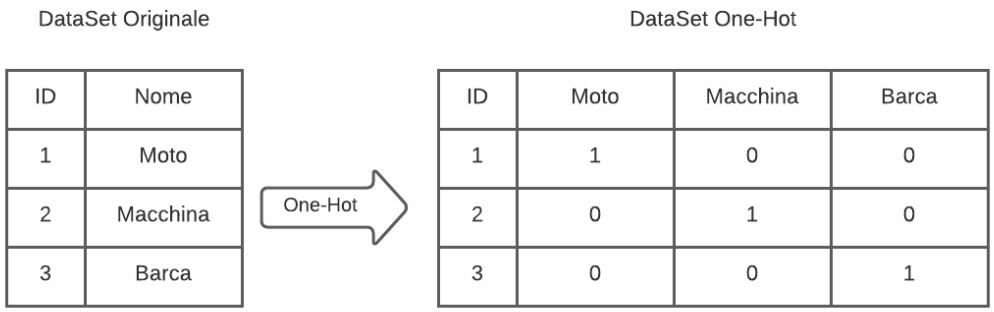


Figura 1.17: One-Hot-Encoding

1.5 Valutazione di un modello di classificazione

Una volta che si è costruito un qualsiasi modello di classificazione bisogna anche misurarne le prestazioni con delle metriche che ci indichino la bontà di questo modello in modo da poterlo confrontare con altri modelli e riuscire a capire se è utilizzabile o meno.

Matrice di confusione e metriche correlate

Un primo strumento molto utile per valutare un generico modello è la matrice di confusione, questa riassume i risultati che ottiene il modello stesso su un set di dati di test. Questa matrice ha la seguente forma (prendiamo, per semplicità, il caso di una classificazione con due classi):

		Valori Reali		
		0	**1**	**Totale**
Valori Predetti	**0**	Veri Negativi	Falsi Negativi	$TN + FN$
	1	Falsi Positivi	Veri Positivi	$FP + TP$
	Totale	$TN + FP$	$FN + TP$	

Da questa matrice si possono ricavare le seguenti metriche:

- *Accuratezza*: $\frac{TP+TN}{TP+TN+FP+FN}$
- *Precisione*, indica il rapporto di istanze che effettivamente appartengono ad una delle due classi rispetto al totale delle istanze predette come tali dal modello.
  Si calcola come: $Precision(0) = \frac{TN}{TN+FN}$
- *Recall*, indica il rapporto di istanze che sono state predette come appartenenti ad una determinata classe rispetto a tutte le istanze che realmente appartengono a tale classe.
  Si calcola come: $Precision(0) = \frac{TN}{TN+FP}$
- *F1-Measure*, indica un riassunto di precisione e recall.
  Si calcola come: $F1 - Measure(0) = \frac{2 \cdot precision(0) \cdot recall(0)}{precision(0)+recall(0)}$

Tutte queste metriche possono essere similmente calcolate per la classe positiva 1 e, inoltre, di tutte si può calcolare la media per avere una metrica unica riassuntiva.

Intervallo di confidenza dell'accuratezza

Una volta costruita la matrice di confusione si può calcolare, con la formula vista in precedenza, l'accuratezza ottenuta dal modello sul test set di riferimento. Questa accuratezza è rappresentata da un singolo numero, ad esempio 84%, che però potrebbe variare leggermente utilizzando dati diversi, quindi si vuole calcolare un intervallo nel quale siamo certi, con una certa confidenza, che ricada l'accuratezza reale. Per ottenere questo risultato si può modellare un processo di classificazione (indipendentemente dal modello usato) come un

processo Bernoulliano di N (dimensione del test set) eventi indipendenti, ovvero ad ogni istanza del test set viene associata una variabile aleatoria di Bernoulli (successo o insuccesso). In questo modo otteniamo una successione di variabili aleatorie che ci descrivono se una data istanza è stata classificata correttamente (successo) o in maniera errata (insuccesso), come nel seguente schema (figura 1.18):

X_1	X_2	X_3	X_4	...	X_N
1	1	0	1	...	0

Figura 1.18: Successione di variabili aleatorie nel processo di Bernoulli

A questo punto possiamo ricordare il Teorema del limite centrale (TLC):

Teorema 1. *La somma (o la media) di un grande numero di variabili aleatorie indipendenti e dotate tutte della stessa distribuzione è approssimativamente normale, indipendentemente dalla distribuzione soggiacente.*

Dunque, se abbiamo un test set di N istanze (con N $>$ 30 per poter applicare il TLC) e otteniamo S successi, facendo la media di tutte le variabili aleatorie ottenute in precedenza troviamo la nostra accuratezza f:

$$f = \frac{1}{N} \sum_i X_i = \frac{S}{N}$$

Applicando ad f il TLC possiamo approssimarla con una distribuzione z e scrivere la seguente equazione:

$$PR[-z \leq (f - p) \leq z] = confidenza$$

Dove:

- f è l'accuratezza precedentemente calcolata
- p è la reale accuratezza da trovare, ed unica incognita
- La confidenza viene fissata da noi, in genere si usa il 95% o il 99%. Spesso viene indicata nella forma $1 - \alpha$, α è detto p-value ed indica la probabilità che un certo risultato sia frutto del caso.
- $-z$ e z vengono determinate nella tabella della z distribution a partire dalla confidenza.

Risolvendo la precedente equazione in p si trova dunque il reale intervallo in cui ricade p:

$$p = \frac{f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

Dalla formula possiamo notare due cose particolarmente interessanti:

1. Quando N aumenta l'intervallo si restringe
2. Quando la confidenza aumenta allora aumenta anche il valore di z e di conseguenza l'intervallo si allarga

Come esempio possiamo prendere il caso in cui abbiamo un test set di $N = 100$ istanze e una confidenza fissata al 95% (p-value 0.05), se risolvendo la precedente equazione otteniamo $p \in [65.7; 76.4]$ allora possiamo dire che siamo sicuri al 95% che la reale accuratezza del nostro modello ricada in tale intervallo.

Capitolo 2

Il Deep Learning

2.1 Reti neurali profonde

Le *Reti Neurali Profonde* (o Deep Neural Network) sono reti neurali formate da almeno due – ma spesso molti di più – hidden-layer.

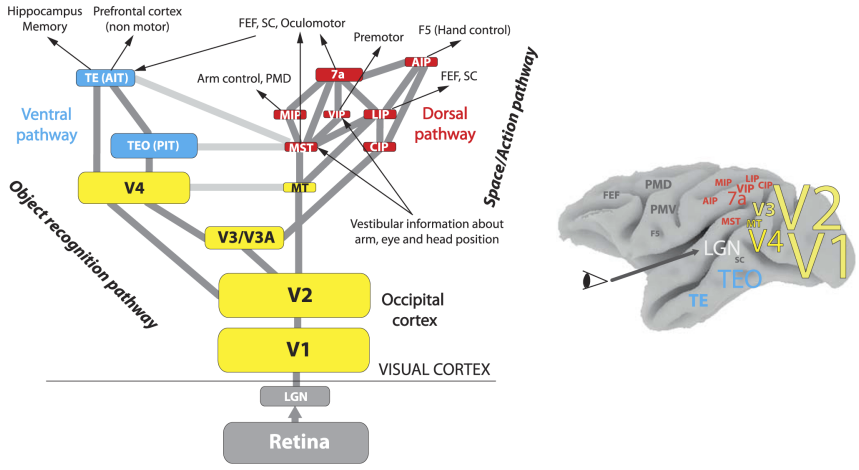


Figura 2.1: Schema di funzionamento del sistema visivo umano.
Fonte: [4]

Questa architettura prende spunto dal sistema visivo umano che – come mostrato in diversi studi tra cui Kruger et al. 2013 [4] – si articola su circa dieci livelli. Gli input visivi entrano dalla retina, per poi passare dalla corteccia occipitale e proseguire in tutti i livelli più profondi (Come in figura 2.1).
Le DNN più diffuse al giorno d'oggi utilizzano circa fra i 5 e 50 livelli, questo range riesce a garantire una buona efficienza in fase di training. Ogni livello di queste reti deep serve per trovare feature sempre più complesse, il livello di input utilizza i dati grezzi per poi passare il suo output ai livelli successivi

che utilizzano quest'ultimo per ricavare automaticamente le nuove feature. Per esempio potremmo avere una DNN costruita per classificare delle recensioni (come quella in figura 2.2). In questo caso i dati grezzi su cui lavora il layer di input sono i singoli caratteri della frase, il secondo livello ricostruisce delle singole parole ritenute importanti, il terzo livello riassume la positività e la negatività delle parole chiave estratte in precedenza ed infine l'ultimo livello, di output, assegna una classe alla recensione (le classi potrebbero essere ad esempio: pessima, discreta ed ottima).

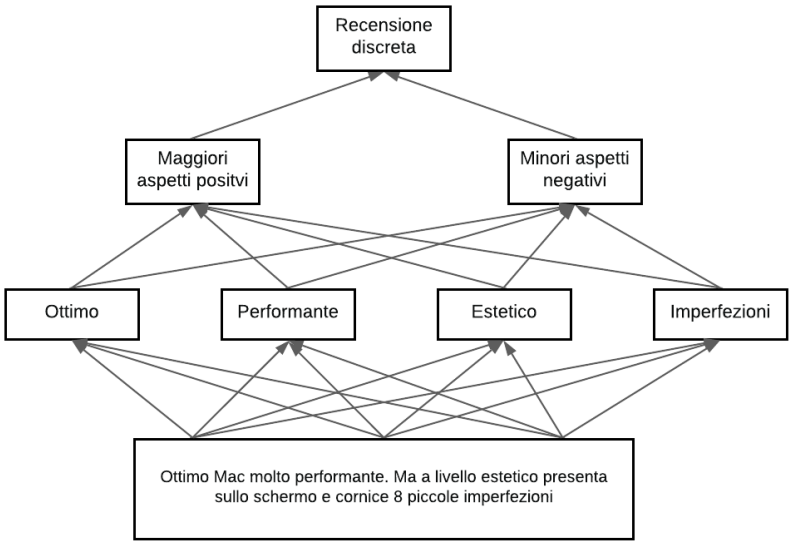


Figura 2.2: DNN per classificare recensioni, recensione estratta da *Amazon.it*

Le reti neurali profonde hanno diversi tipi di architetture, le principali usate in campo di classificazione sono:

- Convolutional Neural Network
- Fully Connected DNN
- Hierarchical Temporal Memory

L'utilizzo di tutti questi vari tipi di DNN ha permesso di superare diverse accuratezze che erano state raggiunte nello stato dell'arte con altre tecniche, soprattutto in campo di visione artificiale. Queste reti hanno però bisogno di:

- Una grande mole di dati di training, che, nella pratica potrebbero non essere sempre disponibili o potrebbe essere molto dispendioso metterli insieme.

- Un tempo di training spesso elevato.
- Hardware costoso, come le GPU, per incrementare le prestazioni e diminuire i tempi di training.

2.2 Convolutional Neural Network

Le CNN sono un'architettura di Deep Neural Network – introdotta da Yann LeCun nel 1989 [5] – usata in particolar modo nel campo di visione artificiale e anche in qualche applicazione di Natural Language Processing, come in Chen et al. [6] dove una rete convoluzionale viene impiegata per l'estrazione di eventi.

Queste reti trovano la loro origine nel sistema visivo umano, due studi di fine anni cinquanta condotti da David Hubel e Torsten Wiesel – poi vincitori del premio Nobel per la medicina – hanno scoperto che la corteccia visiva è composta da neuroni aventi un campo ricettivo limitato e diverso. Questo vuol dire che un dato neurone reagisce e processa l'input che arriva solo da una piccola porzione dell'intera scena visibile dall'occhio umano, andando ad unire queste porzioni – che possono anche essere sovrapposte – si ottiene l'intera immagine (figura 2.3). Dallo studio si evince anche che i neuroni oltre ad avere un campo visivo limitato sono anche in grado di processare solo linee aventi certi orientamenti (alcuni verticali, altri orizzontali e così via). Man mano che si va in profondità nella corteccia visiva l'immagine viene ricostruita, è importante notare che un dato neurone è connesso solo ad un piccolo sottoinsieme di tutti gli altri.

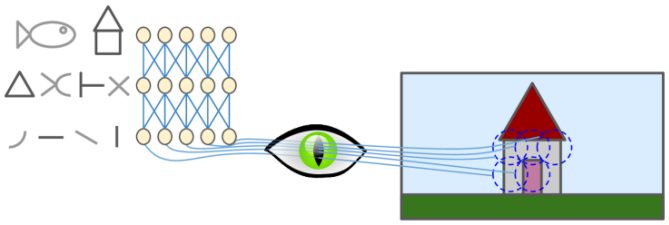


Figura 2.3: Schema di funzionamento della corteccia visuale.
Fonte: https://www.researchgate.net/figure/A-convolutional-neural-networks-CNN_fig6_321286547

Per implementare questa architettura sono stati introdotti principalmente due nuovi layer: Convolutional Layer e Pooling Layer. Solitamente, questi, vengono disposti nei primi livelli della DNN per poi essere seguiti da alcuni livelli "classici" fino ad arrivare al livello di output con una softmax (come in figura 2.4).

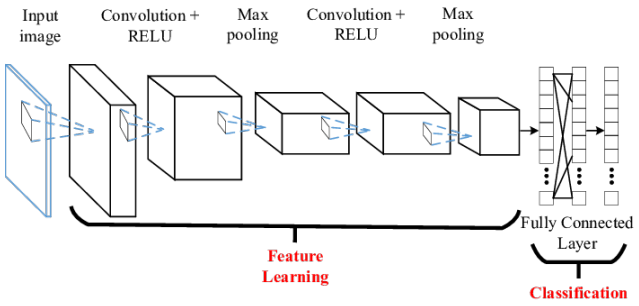


Figura 2.4: Classica architettura CNN.
Fonte: https://www.researchgate.net/figure/A-convolutional-neural-networks-CNN_fig6_321286547

Convolutional Layers

I Convolutional Layers riproducono il comportamento dei neuroni descritto nella sezione precedente, per cui, ogni neurone di questi layer non è collegato ad ogni pixel dell'immagine di input ma solo ad una sua sotto-porzione rettangolare (come in figura 2.5).

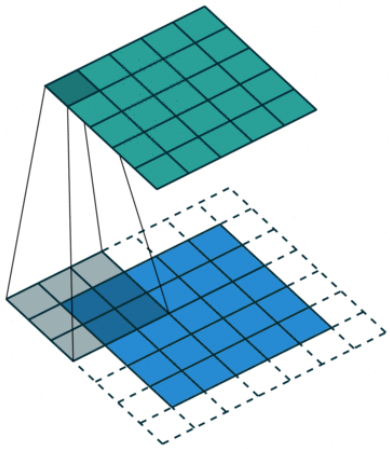


Figura 2.5: Campo ricettivo nei convolutional layers.
Fonte: https://medium.com/syncedreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-42f33d4378e0

Una seconda novità introdotta è l'utilizzo dei filtri per riprodurre il comportamento secondo cui un neurone riconosce solo linee che hanno un certo orientamento. Questi filtri – o kernel – si possono rappresentare come una piccola immagine solitamente di dimensioni 3x3 pixel o 5x5 pixel. Nell'esempio

in figura 2.6 si può vedere sia la rappresentazione che il risultato dell'applicazione di un filtro diagonale. Nelle CNN, ovviamente, i filtri non devono essere pre-impostati manualmente ma vengono scelti dalla rete stessa in fase di training a seconda dello specifico task su cui si lavora, inoltre uno stesso layer potrebbe applicare più filtri in successione.

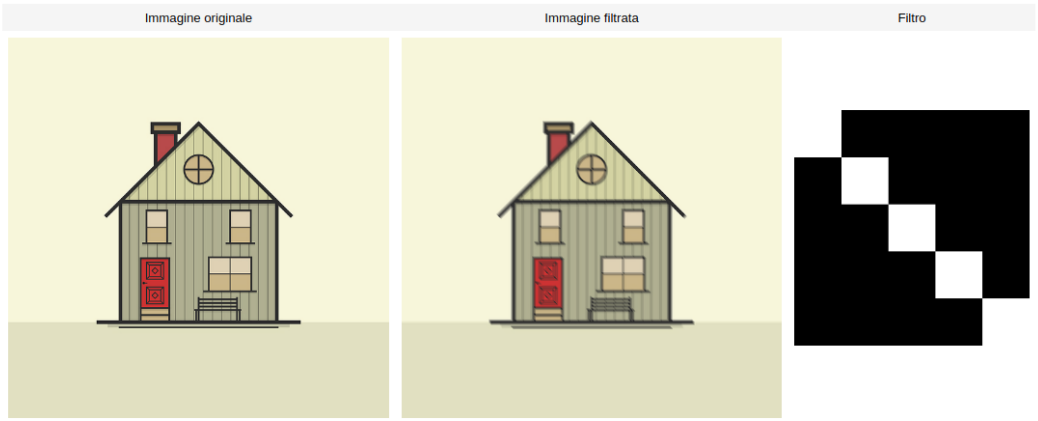


Figura 2.6: Esempio di applicazione di un filtro 5x5

L'applicazione di un filtro F di dimensioni $m \times m$ ad una determinata immagine I per ottenere un'immagine I' avviene mediante una tecnica detta *convoluzione* la cui formula è:

$$I'[y,x] = \sum_{i=-d}^{d} \sum_{j=-d}^{d} F[i,j] \cdot I[y-i, x-j] \; con \; d = \lfloor \frac{m}{2} \rfloor$$

Pooling layers

I pooling layers sono stati introdotti allo scopo di limitare l'utilizzo di memoria e il numero di pesi delle CNN. Per raggiungere questo scopo viene fatto un campionamento dell'output prodotto dai layer convoluzionali nel seguente modo:

1. Dopo il layer convoluzionale viene aggiunto un layer di pooling.
2. Ogni neurone di questo nuovo layer è collegato solo ad un piccolo numero di neuroni del livello precedente, possiamo chiamare questo insieme *finestra ricettiva*.
3. Un dato neurone del pooling layer applica ai valori presenti nella sua finestra ricettiva una funzione (ad esempio massimo o media) per ottenere un unico valore riassuntivo.

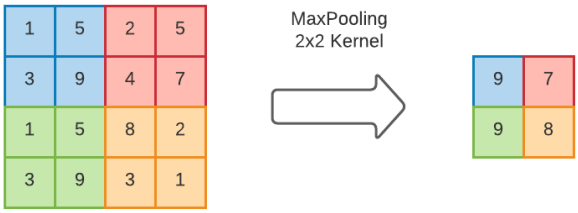


Figura 2.7: Esempio di max pooling

Oltre ai benefici elencati nel paragrafo precedente questo layer introduce anche un effetto di invarianza rispetto ai piccoli cambiamenti.

2.3 Transfer Learning

Il Transfer Learning è una tecnica di Machine Learning che ha come obiettivo l'incremento delle prestazioni di un modello utilizzando una base di conoscenza già appresa dal modello stesso per un task simile.

In termini più rigorosi il Transfer Learning può essere definito nel seguente modo:

Teorema 2. *Un generico dominio $\mathcal{D}$ è formato da: uno spazio degli input $\mathcal{X}$ e una distribuzione di probabilità marginale $P(X)$, dove $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Dato un dominio $\mathcal{D} = \{\mathcal{X}, P(X)\}$, un task è formato da: uno spazio degli output $\mathcal{Y}$ e da una funzione obiettivo – predittiva – $f : \mathcal{X} \to \mathcal{Y}$. La funzione f viene usata per predire la corretta classe $f(x)$ di una nuova istanza x. Questo task, denotato come $\mathcal{T} = \{\mathcal{Y}, f(x)\}$ viene appreso a partire dai dati di training $\{x_i, y_i\}$, dove $x_i \in \mathcal{X}$ e $y_i \in \mathcal{Y}$.*

Dunque dato un dominio di partenza $\mathcal{D}_S$ e un task $\mathcal{T}_S$ e dato un nuovo dominio target $\mathcal{D}_T$ e un task $\mathcal{T}_T$, dove $\mathcal{D}_S \neq \mathcal{D}_T$ e $\mathcal{T}_S \neq \mathcal{T}_T$, il transfer learning mira ad aumentare le capacità predittive della funzione $f_T(\cdot)$ usando la base di conoscenza in $\mathcal{D}_S$ e $\mathcal{T}_S$

Poniamo – ad esempio – di voler costruire un modello che classifichi correttamente delle immagini in base a se contengono una bicicletta o una motocicletta, possiamo procedere nel seguente modo (schematizzato in figura 2.8):

1. Si trova nello stato dell'arte una soluzione che effettua una classificazione simile ma su classi diverse.

2. Si prende questa rete scartando il livello di output, a volte anche qualche hidden layer immediatamente precedente all'output, e lo si sostituisce con un nuovo output layer pulito.

3. Vengono bloccati i pesi dei livelli che sono stati mantenuti identici dalla soluzione base.

4. Si effettua il training dei nuovi livelli inseriti.

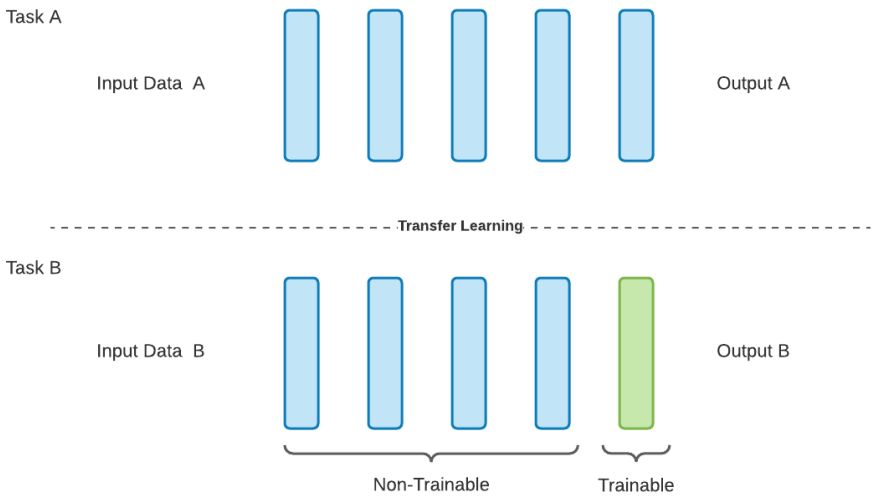


Figura 2.8: Esempio di transfer learning

Questa tecnica è molto usata in deep learning perchè, nella maggior parte dei casi, permette di aumentare di molto le prestazioni (grafico 2.9) e in più permette anche di avere un dataset piccolo quando, invece, per costruire un modello da zero servirebbe una notevole quantità di dati.

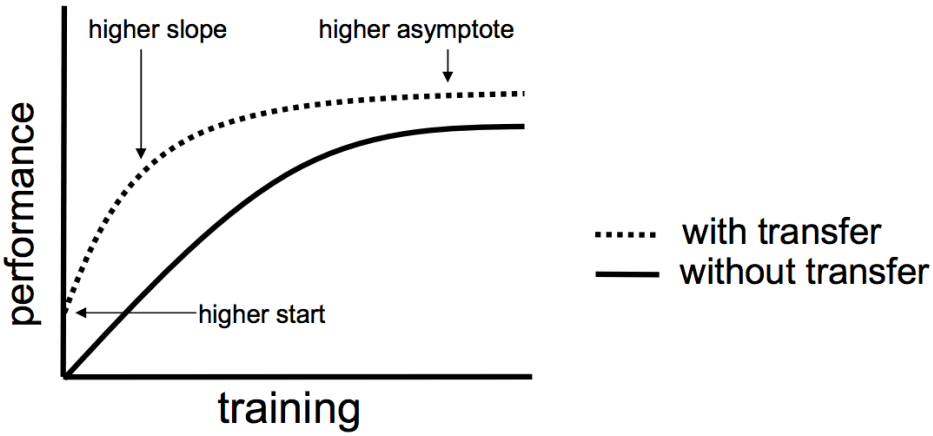


Figura 2.9: Benefici – in termini di performance – del transfer learning.

2.4 Interpretabilità

L'interpretabilità indica la facilità con cui si riesce a spiegare il motivo per cui un modello di machine learning ha preso una certa decisione. Non tutti i modelli sono interpretabili allo stesso modo, questo è dato principalmente dalla complessità con cui il modello stesso rappresenta i dati. Al tempo stesso per ottenere dei buoni risultati di accuratezza – molto spesso – serve aumentare questa complessità, dunque esiste una proporzionalità inversa fra interpretabilità e accuratezza (come si può vedere nel grafico in figura 2.10).

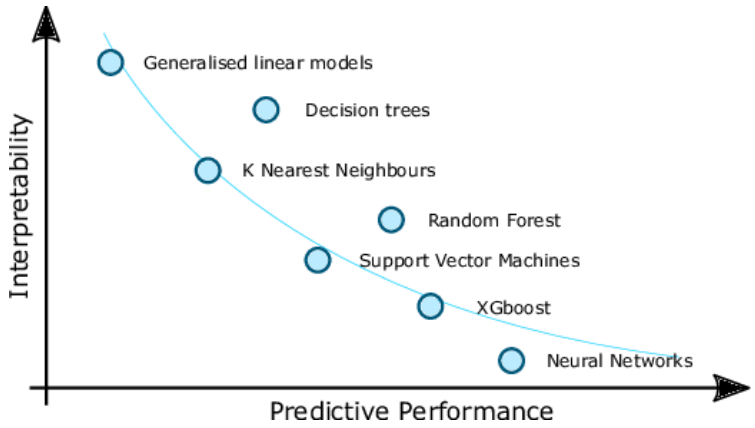


Figura 2.10: Trade-off accuratezza-interpretabilità.
Fonte: [7]

Riuscire ad interpretare il risultato di un generico modello risulta essere molto importante per svariati motivi, tra cui:

- Etica, in alcuni campi di utilizzo del ML, per esempio in medicina o nelle macchie a guida autonoma, è importante sapere i motivi dietro alle varie decisioni per accertarsi di non poter procurare danni a terzi.

- Controllare la casualità presente nel risultato ed effettuare un debug.

- Equità, per assicurarsi che la decisione presa dal modello non vada a nuocere qualcuno (ad esempio nel caso in cui il modello sia utilizzato per classificare le performance dei dipendenti di un'azienda).

- Rispettare i regolamenti, il GDPR comprende un articolo che sancisce il diritto dei singoli individui a ricevere spiegazioni su una certa scelta algoritmica (ad esempio perchè una transazione bancaria lecita sia stata erroneamente classificata come illecita e rigettata).

Il processo di interpretazione utilizzato consiste nell'usare il modello come una sorta di Black-Box per poi valutarne i risultati ottenuti (come in figura 2.11).

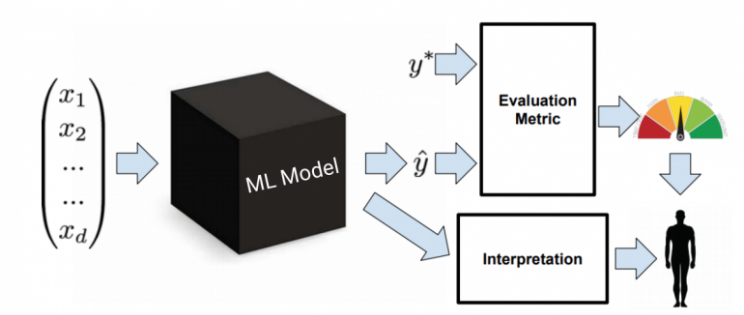


Figura 2.11: Processo di interpretazione di un modello di ML.
Fonte: https://www.analyticsvidhya.com/blog/2019/08/decoding-black-box-step-by-step-guide-interpretable-machine-learning-models-python/

Come si può notare dal grafico 2.10 alcuni dei modelli più semplici da interpretare sono:

- Modelli lineari, si può avere molto facilmente un'interpretazione valutando i vari coefficienti. Prendiamo in esempio un caso di studio in cui si vuole predire il prezzo di vendita delle macchine a partire da alcune caratteristiche come: cilindrata, chilometraggio e numero di posti. Una volta che tutte le feature di partenza sono state normalizzate e i coefficienti ottimali trovati otteniamo una funzione del tipo:

$$prezzo = w_0 + w_1 \cdot cilindrata + w_2 \cdot km + w_3 \cdot numeroposti$$

  A questo punto possiamo confrontare i vari coefficienti per capire quali feature abbiano maggiore importanza e quali meno.

- Alberi decisionali, in questo caso si ottiene un'interpretazione grafica molto veloce e comprensibile anche ai non esperti della materia, infatti basterà seguire i vari rami dell'albero per capire quali scelte hanno portato a quali risultati. Per esempio se si volesse predire un punteggio di gradimento dei videogiochi da parte delle persone in base alla loro età e al loro sesso si potrebbe ottenere un albero decisionale come quello in figura 2.12.

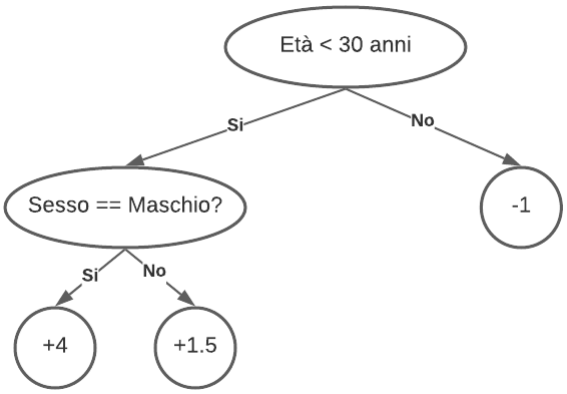


Figura 2.12: Albero decisionale.

La ricerca scientifica in questo settore è ancora in pieno sviluppo ed esistono ancora modelli la cui l'interpretazione risulta molto difficile o addirittura impossibile da effettuare.

2.5 Il Deep Learning in medicina

La medicina è un area che si presta particolarmente bene all'utilizzo delle tecniche di deep learning. Molte sotto-aree della medicina come la genomica, la chimica-computazionale e la dermatologia sono ricche di dati che possono essere sfruttati per addestrare le reti, si stima che un ospedale di dimensioni medie produca dai 15 ai 20 terabyte di dati nuovi ogni anno (Yue et al. [8]). Questo genera diverse opportunità e sfide, sono innumerevoli i lavori presenti nello stato dell'arte che utilizzano tecniche di deep learning per risolvere problemi medici, un esempio molto interessante si può trovare in DeFauw et al. [9], uno studio portato avanti dall'azienda DeepMind in cui viene usata una DNN per fare una diagnosi sulle malattie dell'occhio umano. Un sunto delle accuratezze raggiunte da tutte queste tecniche si può trovare in articoli come [10] e [11].
Analizzando i vari lavori si può effettuare una divisione in categorie, dal punto di vista medico, come la seguente:

1. ***Immagini cliniche***, riguardo a: oncologia, neurologia, gastroenterologia, pneumologia, ...

2. ***Bio-segnali***, riguardo a: elettrocardiogramma, elettroencefalogramma, elettromiogramma, ...

3. ***Bio-medicina***, riguardo a: genomica, metabolismo, ...

4. ***EHR (Electronic Health Records)***, riguardo a: analisi dei rischi, predizione di patologie, ...

Nonostante le premesse per l'applicazione del deep learning in questo campo siano eccellenti i ricercatori hanno dovuto fronteggiare e risolvere anche alcune problematiche, le principali sono le seguenti:

- I dati sono di molti tipi diversi (testo, dati numerici strutturati, immagini, ...), questo porta a difficoltà quando vanno combinati insieme per ottenere una diagnosi.
- I dati che sono disponibili sono spesso affetti da molto rumore e da una elevata variabilità.

Viste le caratteristiche dei dati appena illustrate è necessario l'impiego di un gran numero di architetture di DL diverse, una schematizzazione efficace si può trovare in figura 2.13.

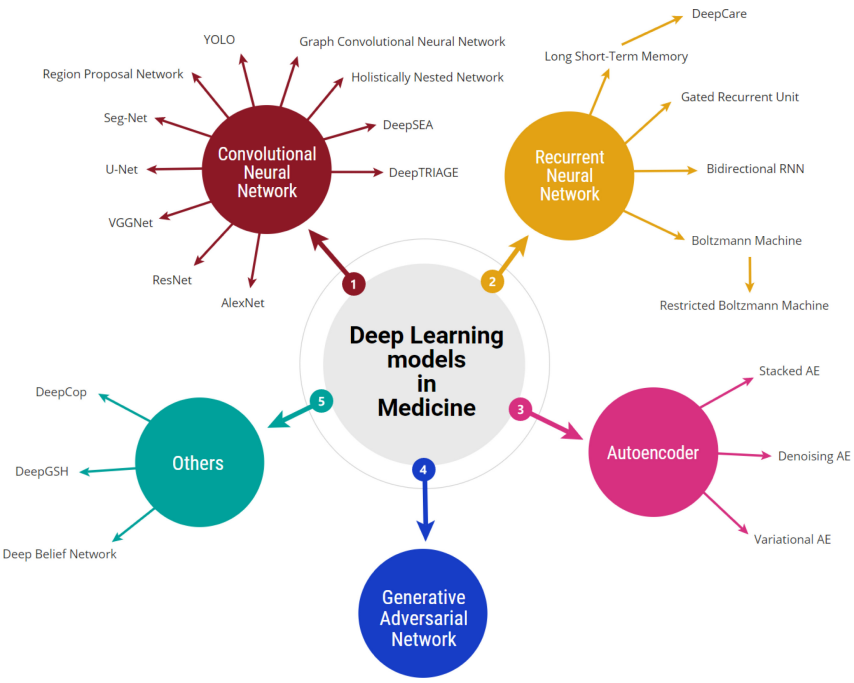


Figura 2.13: Riassunto delle tecnologie di DL usate in medicina.
Fonte: [10]

Infine possiamo addentrarci nelle varie aree – descritte nella precedente suddivisione – per capire quali sono le architetture che hanno ottenuto i migliori risultati. (Dati tratti da [10] e [11])

Area	Compito	Architettura
Oncologia	Classificazione	CNN
	Detection	CNN
	Segmentazione	CNN, GAN
	Stima del dosaggio di una sostanza	CNN
Neurologia	Classificazione	AlexNet
	Segmentazione	GAN
	Ricostruzione	CNN, DAE
Gastroenterologia	Classificazione	CNN
	Detection	ResNet
	Segmentazione	RNN
ECG	Aritmia	RNN
	Apnea	VNN
	Glucosio	CNN
Genomica	Struttura del DNA	GCNN, AE
	Predizione di malattie	DAE, RNN
EHR	Predizione di malattie	LSTM
	Analisi dei rischi	GCNN

Capitolo 3

Vision Transformer Architecture (ViT)

3.1 Origini e funzionamento dei Transformer

I Transformer sono una particolare architettura di rete neurale introdotta da Vaswani et al. nell'articolo *Attention is all you need* [12]. Originariamente – nel 2017 – sono stati sviluppati per migliorare le prestazioni ottenute dalle Recursive Neural Network nei task di Natual Language Processing.

All'epoca per risolvere un task di NLP – come ad esempio la traduzione di una frase da una lingua ad un'altra – venivano sviluppate grandi reti neurali ricorrenti. Queste reti hanno una memoria sequenziale in grado di rappresentare le interconnessioni fra le parole vicine in una frase, il funzionamento di base è il seguente:

1. La frase viene decomposta nelle singole parole (anche dette Token).
2. Le parole vengono rappresentate in un formato vettoriale comprensibile alla rete.
3. Il primo vettore viene passato alla rete che produrrà un hidden state corrispondente.
4. A questo punto il secondo vettore verrà passato alla rete, questa volta però viene aggiunto anche l'hidden state della fase precedente, in questo modo il secondo hidden state conterrà sia le informazioni della parola corrispondente sia le informazioni delle parole precedenti (con un certo limite temporale).

Continuando in questo modo per tutte le parole della frase otterremo una sequenza di hidden state che potranno poi essere usati per decodificare la frase e ottenere la traduzione (come nello schema in figura 3.1).

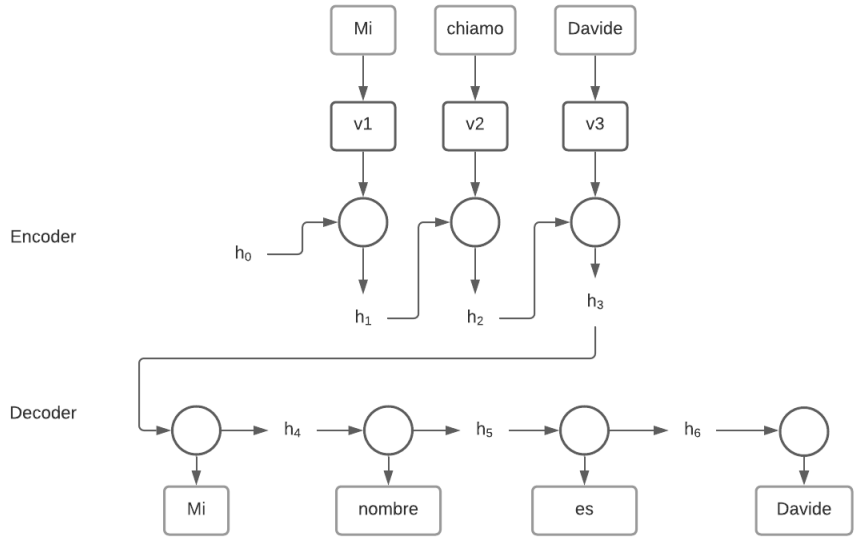


Figura 3.1: Schema di una RNN per la traduzione

L'utilizzo di queste reti permetteva di raggiungere buoni risultati ma era limitato da un tempo di addestramento molto alto, per questo motivo i Transformer sono stati sviluppati senza l'utilizzo di alcun layer ricorrente (o convoluzionale) e basano il loro funzionamento interamente sul meccanismo di attention (spiegato nella sezione 3.2), permettendo di ottenere risultati migliori e con tempi di training notevolmente ridotti.
Come si può vedere nello schema in figura 3.2 i Transformer sono composti da due componenti principali: l'encoder (a sinistra) e il decoder (a destra). La parte di encoding serve per estrarre dagli input le informazioni di conoscenza necessarie per il training della rete, questo viene fatto prima trasformando le parole in vettori numerici (embedding) e aggiungendo loro delle informazioni che rappresentano la loro posizione all'interno del testo (positional encoding). Una volta ottenuti – questi vettori – vengono passati attraverso un livello di attention che serve per rappresentare i collegamenti fra le varie parole, infine questi valori vengono passati ad un livello densamente connesso. La prima parte del decoding è molto simile all'encoding ma viene fatta sulle frasi output di target. Una volta finita questa prima parte tutte le informazioni vengono passate al terzo e ultimo layer di attention, le informazioni provenienti dall'encoder fungono da Key e Value mentre quelle provenienti dal decoder fungono da Query. Come fase finale sono presenti dei layer densamente connessi utili per arrivare alla vera e propria predizione.

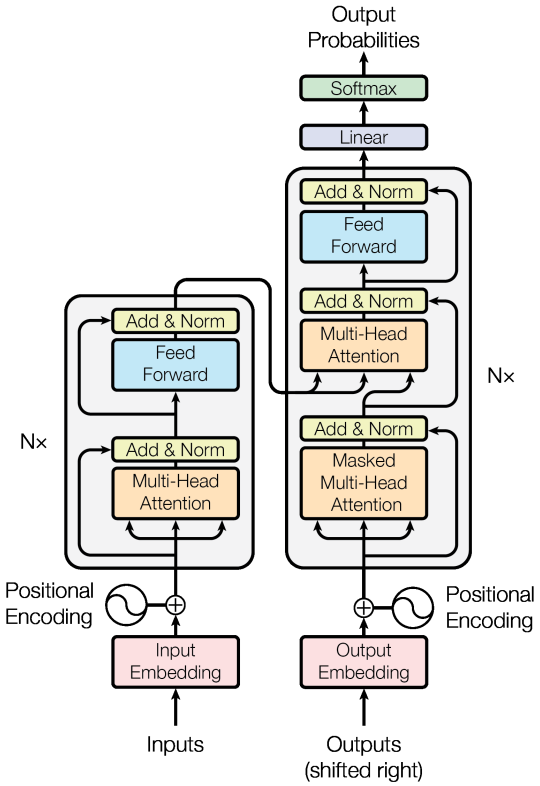


Figura 3.2: Architettura di un Transformer.
Fonte: [12]

3.2 Attention

L'Attention – come si evince anche dal titolo del paper *Attention is all you need* – è il componente chiave dei Transformer. Questo meccanismo ha come obiettivo rappresentare in modo efficacie ed efficiente le relazioni fra tre tipi di valori detti V (Value), K (Keys) e Q (Query). L'idea di base è capire quali chiavi K rispondono meglio alla query Q per poi trovare i valori V associati, questo viene espresso matematicamente con la seguente formula:

$$Attention(V, K, Q) = soft - max(\frac{Q \cdot K}{\sqrt{d}}) \cdot V$$

Il primo passaggio di questa formula calcola il prodotto vettoriale fra i vari vettori K e la query Q, questo prodotto sarà tanto maggiore quanto più piccolo è l'angolo fra i due vettori – quindi quanto più sono simili i vettori, similarità coseno – secondariamente il tutto viene diviso per la radice della dimensionalità d come normalizzazione. Infine viene prima applicata la funzione soft-max a

questi valori per limitarli tutti nell'intervallo $[0; 1]$ e come ultimo passaggio si esegue il prodotto vettoriale con il vettore V, questo prodotto va in automatico a selezionare (i.e. porre *attenzione*) la componente V_i ritenuta più importante, graficamente si può schematizzare come in figura 3.4 dove il vertice della curva a campana indica il valore V selezionato.

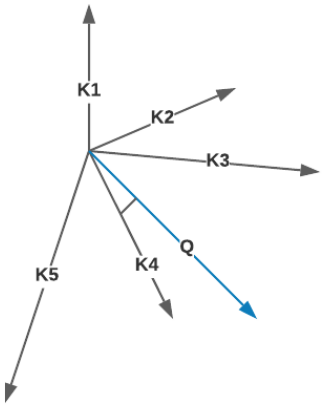


Figura 3.3: Prodotto $Q \cdot K$

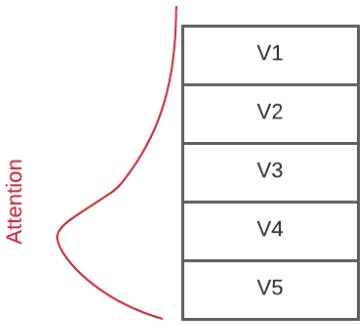


Figura 3.4: Attention

Una variante di questo meccanismo è la *Multi-Head Attention* (figura 3.5), in questo caso i vettori di input vengono divisi in h sotto-insiemi e vengono calcolate h attention separate (multi-head) per poi concatenare il risultato finale, in questo modo ogni attention "impara" aspetti leggermente diversi.

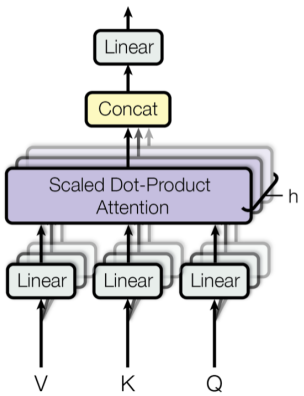


Figura 3.5: Multi-Head Attention.
Fonte: [12]

3.3 Transformer in visione artificiale

I Transformer sono, al giorno d'oggi, uno standard de-facto delle applicazioni legate al mondo del Natural Language Processing mentre sono molto meno usati nel campo della Visione Artificiale dove si preferisce ancora utilizzare reti convoluzionali. Invece in Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [13] viene proposta un'architettura – chiamata ViT – per la classificazione delle immagini basata interamente sui Transformer.

L'architettura ViT (figura 3.6) è molto simile a quella dei Transformer descritti nelle sezioni precedenti: l'immagine di input viene scomposta in molteplici patch che a loro volta, attraverso una proiezione lineare, vengono trsformate in vettori numerici a cui viene aggiunto il position embedding, oltre ai vettori corrispondenti alle sotto-sezioni dell'immagine viene aggiunto anche un vettore che rappresenta la classe di appartenenza, questo viene inizializzato in modo randomico per poi essere "imparato" dalla rete in fase di training. Questi vettori vengono passati ad un Transformer Encoder standard per poi finire nell'ultimo livello, una sotto-rete densamente connessa, che effettua la classificazione vera e propria partendo dalla patch di classificazione.

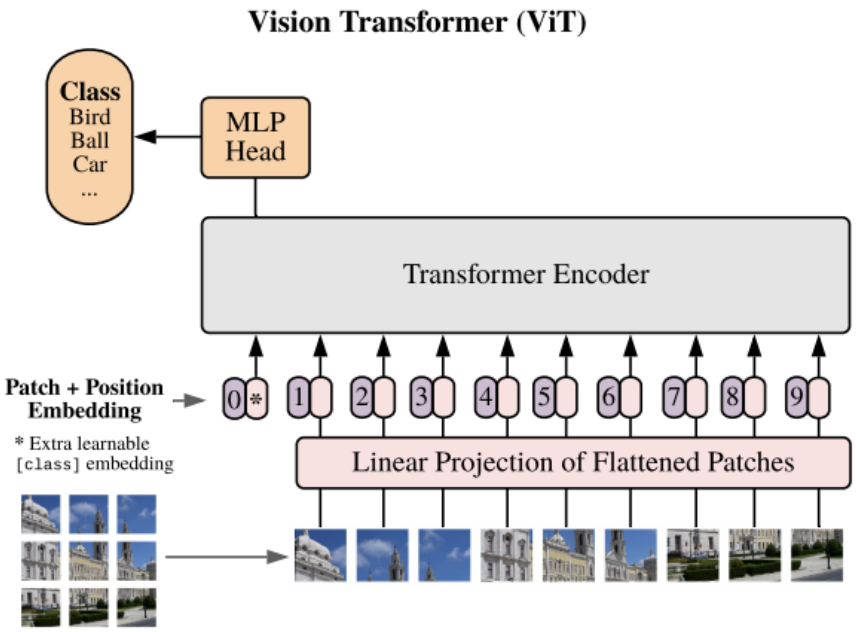


Figura 3.6: Architettura ViT.
Fonte: [13]

Questa soluzione apporta sia vantaggi che svantaggi, i primi sono: un miglioramento dei tempi di training ed inferenza dato dall'esclusione dei layer convoluzionali per l'estrazione delle feature e una migliore capacità di trovare collegamenti fra le patch, questo è dato dal fatto che l'encoder calcola l'attention fra ogni coppia di patch mentre sia i layer convoluzionali che quelli ricorrenti hanno informazioni solo sulle patch più vicine. Come svantaggio invece è presente la necessità di avere un training set più ampio per poter arrivare allo stesso livello di accuratezza delle reti convoluzionali.

Capitolo 4

Tecnologie utilizzate

4.1 Python

Per la programmazione è stato usato Python: un linguaggio ad alto livello, orientato agli oggetti e particolarmente utilizzato nelle applicazioni di Data Science e IA viste le molteplici librerie esterne che mette a disposizione.

4.2 Google Colaboratory

Google Colaboratory è un cloud basato su notebook jupyter che permette di scrivere ed eseguire codice Python in remoto su macchine virtuali messe a disposizione da Google. In questo modo si hanno degli environment diversificati per ogni progetto e si ha accesso all'hardware di Google – GPU e TPU oltre che a CPU – di qualità notevolmente superiore all'hardware di cui di solito dispongono dei normali computer portatili. Inoltre essendo basato su jupyter si ha la possibilità di inserire celle di testo, immagini, formule tramite LaTeX e molto altro in modo da rendere altamente leggibile e presentabile tutto il codice.

4.3 Keras

Keras è un'interfaccia che offre semplici e ottimizzate API per il machine learning e il deep learning. È stata sviluppata per la prima volta da François Chollet – un dipendente di Google – e nel 2017 è stata riconosciuta ufficialmente dal team di sviluppo di Tensor Flow. L'obiettivo primario di questa libreria è minimizzare il numero di linee di codice necessarie per sviluppare un proprio modello di learning. Keras ha molteplici pregi come:

1. Scalabilità, grazie al back-end basato su Tensor Flow è in grado di scalare anche su architetture molto complesse come cluster di GPU e TPU.

2. Deployment, è possibile utilizzare modelli costruiti con Keras sia in ambienti web tramite JavaScript che in ambienti mobile (Android e iOS) o embedded grazie a TensorFlow Lite.

Inoltre è utilizzata in moltissime soluzioni presenti nello stato dell'arte oltre che da importanti organizzazioni come: NASA, NIH (National Institute of Health) e CERN (anche per l'acceleratore di particelle). Un esempio di modello creato e allenato con Keras potrebbe essere il seguente, si noti quante poche righe di codice bastino:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential([
        Dense(128, activation = "relu", input_shape=(16,)),
        Dropout(0.5),
        Dense(128, activation = "relu"),
        Dropout(0.5),
        Dense(output_shape=2, activation = "softmax")
])

model.compile(loss="categorical_crossentropy",
                optimizer="adam", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=15, batch_size=15,
                validation_data=(X_test, y_test))
```

In questo progetto sono state utilizzate le seguenti classi di Keras:

- *to_ categorical*, per trasformare le etichette delle varie istanze in formato one-hot-encoding.
- *Sequential*, per creare una pila di layer neuronali.
- *Dense*, per creare un layer di neuroni densamente connessi.
- *Conv1D*, per creare un layer convoluzionale.

- *Flatten*, per appiattire l'output del layer convoluzionale in modo da poterlo usare come input di un layer Dense.
- *Dropout*, per prevenire l'overfitting. Imposta casualmente alcuni input a 0 con una frequenza che gli viene passata in input.
- *ImageDataGenerator*, per gestire il dataset.
- *vit_keras*, per utilizzare i Vision Transformer.

4.4 Scikit Learn

Scikit-Learn è una libreria open source per Python utile per il machine learning. Contiene diversi algoritmi, notevolmente ottimizzati, per: regressione, classificazione e clustering. In paricolare per questo progetto sono state utilizzate le seguenti classi, utili per ottenere le metriche di valutazione del modello:

- *confusion_matrix*
- *classification_report*

Capitolo 5

Sviluppo del progetto

5.1 Dataset e stato dell'arte

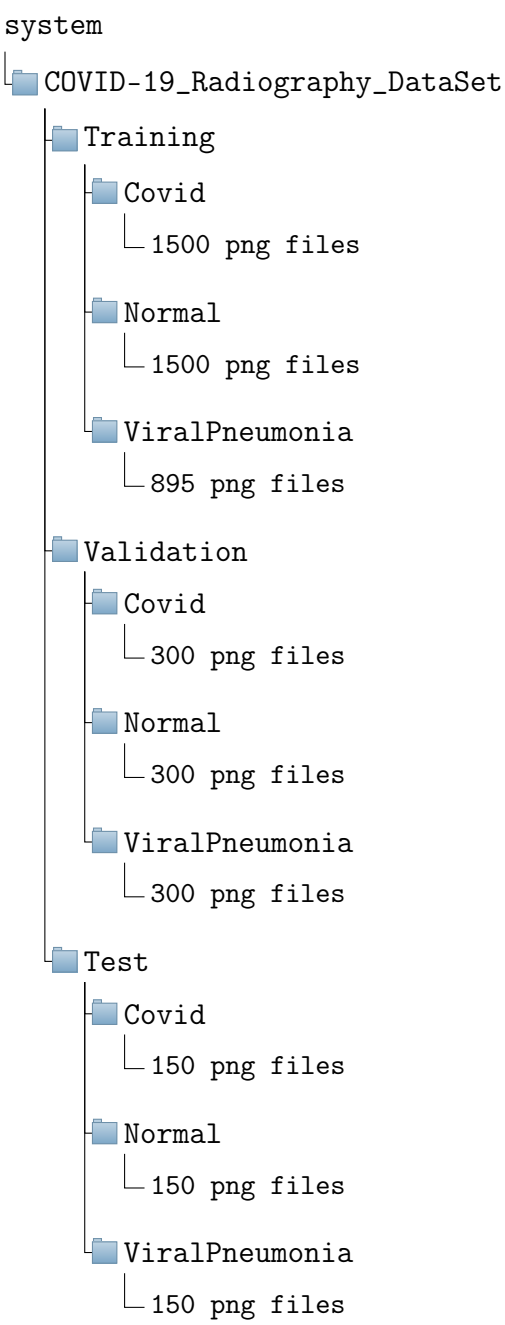
I dati usati in questo progetto sono stati reperiti online su Kaggle.com, un famoso sito di machine learning. Questo dataset è stato creato da alcuni ricercatori delle università di Doha (Qatar) e Dhaka (Bangladesh), in collaborazione con alcuni medici della Malesia e del Pakistan. Le radiografie presenti sono state tratte da altri dataset di dimensioni minori allo scopo di creare un unico insieme ed essere in grado di aumentare l'efficienza del training dei vari modelli.

Tutti i dettagli delle soluzioni proposte da questi team di ricerca possono essere trovati in Chowdhury et al. [14] e in Rahman et al. [15], di seguito è riportato un riassunto delle migliori accuratezze raggiunte:

Autore	Soluzione	Accuratezza	F1-Score
Chowdhury et al.	CheXNet (TL*)	97.74	96.61
Chowdhury et al.	DenseNet201 (DA** + TL*)	97.94	97.94
Rahman et al	Deep CNN	96.29	96.28

Tabella 5.1: * TL = Transfer Learning, ** DA = Data Augmentation

Allo scopo di velocizzarne l'uso – per questo progetto – il dataset è stato riorganizzato come segue:


system
COVID-19_Radiography_DataSet
Training
Covid
1500 png files
Normal
1500 png files
ViralPneumonia
895 png files
Validation
Covid
300 png files
Normal
300 png files
ViralPneumonia
300 png files
Test
Covid
150 png files
Normal
150 png files
ViralPneumonia
150 png files

5.2 Soluzioni proposte e codice

Pre-processing del dataset

Inizialmente sono state importate tutte le librerie e componenti necessarie.

```
pip install tensorflow_addons
import pandas as pd
import seaborn as sns
import numpy as np
import tensorflow_addons as tfa
import glob, warnings
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation,
        Dense, Flatten, BatchNormalization, Conv2D,
        MaxPool2D, AveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import itertools
import os
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
%matplotlib inline
```

È stato importato il dataset e sono state definite le path dei dati di traning, validation e test.

```
from google.colab import drive
drive.mount('/gdrive')
!cp '/gdrive/My Drive/Tesi/COVID-19_Radiography_DataSet.zip' ./
```

```
!unzip 'COVID-19_Radiography_DataSet.zip'
train_path = 'COVID-19_Radiography_DataSet/Training/'
validation_path = 'COVID-19_Radiography_DataSet/Validation/'
test_path = 'COVID-19_Radiography_DataSet/Test/'
```

È stato impostato keras per l'utilizzo della GPU.

```
physical_devices = tf.config.experimental
                    .list_physical_devices('GPU')
tf.config.experimental
    .set_memory_growth(physical_devices[0], True)
```

VGG16

Una prima soluzione è stato il fine tuning di VGG16: una rete convoluzionale addestrata per la classificazioni sul dataset ImageNet, un insieme di immagini appartenenti a 1000 classi differenti. L'architettura di questa rete è rappresentata in figura 5.1.

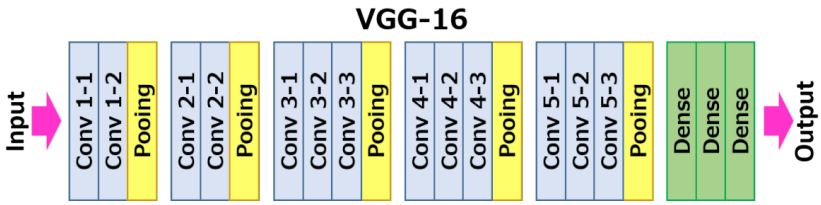


Figura 5.1: Architettura VGG16

Inizialmente sono stati definiti i batch per i vari set di dati, specificando come funzione di pre-processing quella standard di VGG16, questa funzione sottrae ad ogni pixel il valore medio di tutti i pixel del set di dati (per ogni canale RGB).

```
train_batches = ImageDataGenerator(preprocessing_function=
    tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=train_path,
    target_size=(224,224),
```

```
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=10)

validation_batches = ImageDataGenerator(preprocessing_function=
    tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=validation_path,
    target_size=(224,224),
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=10)

test_batches = ImageDataGenerator(preprocessing_function=
    tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=test_path,
    target_size=(224,224),
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=10,
    shuffle=False)
```

È stato poi importato il modello VGG16 pre-addestrato e se ne è creato un altro identico eliminado solo il layer di output, inoltre tutti i livelli copiati sono stati impostati come "non-trainable" per far sì che in fase di training i pesi non vengano cambiati.

```
vgg16_model = tf.keras.applications.vgg16
                .VGG16(weights='imagenet')
model = Sequential()
for layer in vgg16_model.layers[:-1]:
    model.add(layer)

for layer in model.layers:
    layer.trainable = False
```

È stata aggiunta una nuova sotto-rete densamente connessa che verrà addestrata sul dataset di questo progetto.

```
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=3, activation='softmax'))
```

Si è poi proceduto al training effettivo della nuova rete.

```
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(x=train_batches,
validation_data=validation_batches, epochs=10, verbose=2)
```

Infine si è usato il modello per classificare le istanze del test-set in modo da poter procedere ad una valutazione.

```
predictions = model.predict(x=test_batches, verbose=0)
```

Vision Transformer

La seconda proposta testata è stata l'applicazione dell'architettura Vision Transformer (ViT), anche in questo caso mediante fine tuning.

Come in precedenza – per prima cosa – sono stati creati i batch per i vari set di dati.

```
train_batches = ImageDataGenerator(preprocessing_function=None,
    rescale = 1./255,
    samplewise_center = True,
    samplewise_std_normalization = True) \
    .flow_from_directory(directory=train_path,
    target_size=(224,224),
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=16)
```

```
validation_batches = ImageDataGenerator(preprocessing_function=None,
    rescale = 1./255,
    samplewise_center = True,
    samplewise_std_normalization = True) \
    .flow_from_directory(directory=validation_path,
    target_size=(224,224),
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=16)

test_batches = ImageDataGenerator(preprocessing_function=None,
    rescale = 1./255,
    samplewise_center = True,
    samplewise_std_normalization = True) \
    .flow_from_directory(directory=test_path,
    target_size=(224,224),
    classes=['Covid','Normal','ViralPneumonia'],
    batch_size=16,
    shuffle=False)
```

È stato poi importato il modello pre-addestrato, nella libreria sono presenti tre tipi di modelli ViT la cui differenza sta nella grandezza del training set originario, in questo caso si è scelto il modello più robusto. Avendo specificato – al momento di importare il modello – il parametro *"include_top = False"* verranno importati i layer dell'architettura ViT solo fino al Transformer Encoder, per questo motivo è stata aggiunta manualmente la parte finale densamente connessa. Nel penultimo layer Dense è stata utilizzata la funzione di attivazione Gelu – *Gaussian Error Linear Units* – al posto della più comune funzione Relu. Questa particolare funzione è stata introdotta da Hendrycks et al. [16] e presenta caratteristiche più performanti nei task di Visione Artificiale.

```
!pip install --quiet vit-keras
from vit_keras import vit
vit_model = vit.vit_b32(
        image_size = 224,
        activation = 'softmax',
```

```
        pretrained = True,
        include_top = False,
        pretrained_top = False,
        classes = 3)
model = Sequential()
model.add(vit_model)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(11, activation = tfa.activations.gelu))
model.add(BatchNormalization())
model.add(Dense(3, 'softmax'))
```

Si è proceduto al training del modello, in questo caso sono state utilizzate anche due callback utili per apportare alcune modifiche quando una determinata metrica smette di migliorare per un dato lasso di tempo, nello specifico:

1. ReduceLROnPlateau, riduce dinamicamente il learning rate quando la metrica *val_ accuracy* non migliora per due epoche consecutive.
2. EarlyStopping, ferma il training nel caso in cui la metrica *val_ accuracy* non migliora per cinque epoche consecutive.

```
opt = tfa.optimizers
                .RectifiedAdam(learning_rate = 1e-4)

model.compile(optimizer = opt,
              loss = tf.keras.losses
                    .CategoricalCrossentropy(label_smoothing = 0.2),
              metrics = ['accuracy'])

STEP_SIZE_TRAIN = train_batches.n // train_batches.batch_size
STEP_SIZE_VALID = validation_batches.n // validation_batches.batch_size

reduce_lr = tf.keras.callbacks
            .ReduceLROnPlateau(monitor = 'val_accuracy',
                               factor = 0.2,
```

```
                                patience = 2,
                                verbose = 1,
                                min_delta = 1e-4,
                                min_lr = 1e-6,
                                mode = 'max')

earlystopping = tf.keras.callbacks
                .EarlyStopping(monitor = 'val_accuracy',
                                        min_delta = 1e-4,
                                        patience = 5,
                                        mode = 'max',
                                        restore_best_weights = True,
                                        verbose = 1)

callbacks = [earlystopping, reduce_lr]

history = model.fit(x = train_batches,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = validation_batches,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = callbacks)
```

Infine si è usato il modello per classificare le istanze del test-set in modo da poter procedere ad una valutazione.

```
predictions = model.predict(test_batches,
        steps = test_batches.n // test_batches.batch_size + 1)
predicted_classes = np.argmax(predictions, axis = 1)
true_classes = test_batches.classes
```

5.3 Valutazione dei modelli sviluppati

Risultati ottenuti Il primo modello sviluppato ha ottenuto un'accuratezza pari a 0.83, un risultato accettabile ma peggiore del modello ViT che ha incrementato notevolmente le performance ottenute portando l'accuratezza a 0.92. Di seguito sono riportate le matrici di confusione e le metriche di valutazione dei due modelli sviluppati, si precisa che l'etichetta "0" equivale alla calsse *"Covid"*, l'etichetta "1" equivale alla calsse *"Normal"* ed, infine, l'etichetta "2" equivale alla calsse *"Viral Pneumonia"*.

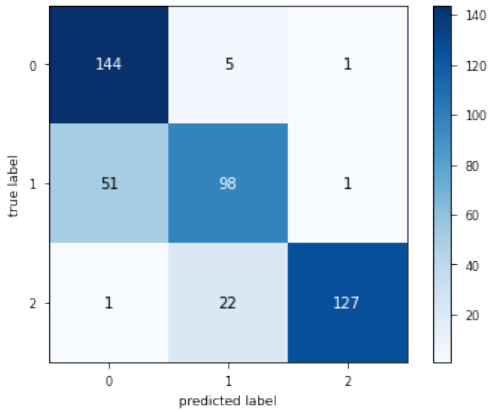


Figura 5.2: VGG16 CM

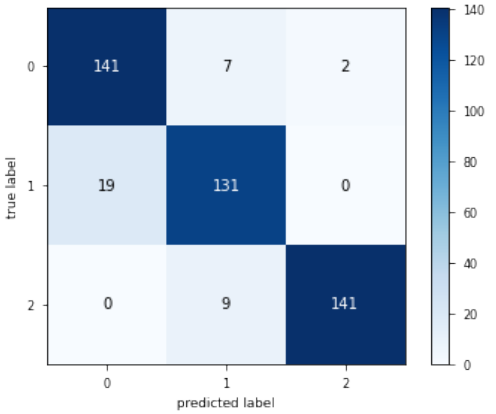


Figura 5.3: ViT CM

Soluzione	Classe	Precision	Recall	F1	Support	Accuracy
VGG16	0	0.73	0.96	0.83	150	0.83
	1	0.78	0.65	0.71	150	
	2	0.98	0.85	0.91	150	
ViT	0	0.88	0.94	0.91	150	0.92
	1	0.89	0.87	0.88	150	
	2	0.99	0.94	0.96	150	

Tabella 5.2: Metriche di valutazione della classificazione

Confronto delle due soluzioni proposte Un aspetto da considerare è che la differenza fra le due soluzioni proposte in questa tesi non sia solo frutto del caso ma sia effettivamente statisticamente significativa. Questo può essere fatto – se il test set è abbastanza grande – approssimando gli errori commessi dai due modelli e_{vgg16} ed e_{vit} tramite il Teorema del Limite Centrale con due

distribuzioni normali: $e_{vgg16} \sim N(\mu_1, \sigma_1)$ ed $e_{vit} \sim N(\mu_2, \sigma_2)$. A questo punto una volta calcolata la differenza $d = |e_{vgg16} - e_{vit}|$ possiamo calcolare – fissata una certa confidenza $Z_{\frac{\alpha}{2}}$ – il reale intervallo di differenze:

$$d_r = d \pm Z_{\frac{\alpha}{2}} \cdot \sigma_t \;\; con\; \sigma_t^2 = \sigma_1^2 + \sigma_2^2$$

Svolgendo i calcoli con gli errori dei due modelli precedenti si ottiene come intervallo $[0.05; 0.13]$, non contenendo il valore 0 possiamo concludere che *le differenze sono statisticamente significative* e che il modello con architettura ViT è effettivamente migliore del primo.

Confronto con lo stato dell'arte Dalla tabella 5.2 si può notare come l'architettura ViT riesca a migliorare tutti i risultati ottenuti dal fine-tuning della rete VGG16 riuscendo ad avere un buon risultato anche rispetto ad altre soluzioni presenti nello stato dell'arte:

- In Chowdhury et al. [14] effettuando il fine-tuning della rete DenseNet201 è stata ottenuta un'accuratezza di 0.97. Il dataset utilizzato è lo stesso di questa tesi.
- In Tabik et al. [17] è stato effettuato il fine-tuning di una rete convoluzionale per diagnosticare il virus Covid-19 discriminando in base alla gravità dei casi (lieve, moderata o severa). È stata ottenuta un'accuratezza di 0.86 per la classe lieve, 0.61 per la classe moderata e 0.97 per la classe severa. Il dataset utilizzato è COVIDGR-1.0 [18], un'insieme di 825 radiografie.
- In Afshar et al. [19], sono state proposte due soluzioni. La prima usando una rete non pre-addrestrata ha ottenuto un'accuratezza di 0.95, nella seconda effettuando il fine-tuning di una rete pre-addestrata è stata raggiunta un'accuratezza di 0.98. Il dataset utilizzato è CovidX, un insieme di circa 14000 radiografie introdotto da Wang et al. in [20].

Conclusioni e sviluppi futuri

Al giorno d'oggi qualsivoglia applicazione utile a monitorare ed effettuare predizioni sul virus Covid-19 può diventare estremamente importante come aiuto per uscire da questo periodo pandemico, ogni piccolo miglioramento può portare i vari ospedali e gli stati a risparmiare una grande quantità di tempo e anche denaro.

Questa tesi aveva come obiettivo testare una nuova architettura per la visione artificiale – i vision transformer – in modo da velocizzare ed incrementare le performance delle soluzioni già presenti nello stato dell'arte. I risultati ottenuti sono molto incoraggianti soprattutto vista la piccola quantità di dati di training che sono stati necessari.

In futuro questo progetto potrà essere ulteriormente migliorato sia incrementando il set di dati con nuove radiografie, sia unendo tecniche di data augmentation. Un altro aspetto da considerare potrebbe essere l'utilizzo di alcune varianti di Transformer per performare meglio nei task di visione artificiale e anche per ridurre la complessità computazionale da quadratica a lineare.

Ringraziamenti

Il primo ringraziamento vorrei esprimerlo al relatore di questo lavoro, il Prof. Gianluca Moro, che ha acceso il mio interesse nei confronti di questa splendida disciplina.

Grazie anche alla mia famiglia e ai miei amici per essermi stati vicini in questi anni e avermi permesso di rendere al meglio nei miei studi.

Una menzione speciale va anche a tutti i professori del corso di studio i quali hanno contribuito a migliorare le mie conoscenze e il modo di affrontare i problemi.

Infine – ma non per importanza – vorrei ringraziare anche Veronika per essermi stata sempre accanto e avermi aiutato a migliorare le mie capacità di cooperare.

Bibliografia

[1] Mengdi Jiang, Yueting Li, Chendan Jiang, Lidan Zhao, Xuan Zhang, and Peter E. Lipsky. Machine learning in rheumatic diseases. *Clinical Reviews in Allergy & Immunology*, 60(1):96–110, Feb 2021.

[2] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[3] Rumelhart, David E. Hinton, Geoffrey E. Williams, and Ronald J. Learning internal representations by error propagation. 1985.

[4] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, and L. Wiskott. Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 35(08):1847–1871, aug 2013.

[5] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[6] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 167–176, Beijing, China, July 2015. Association for Computational Linguistics.

[7] Giovanni Ciatto, Andrea Omicini, Michael Schumacher, and Davide Calvaresi. Agent-based explanations in ai: Towards an abstract framework. 05 2020.

[8] Lin Yue, Dongyuan Tian, Weitong Chen, Xuming Han, and Minghao Yin. Deep learning for heterogeneous medical data analysis. *World Wide Web*, 23(5):2715–2737, Sep 2020.

[9] Jeffrey De Fauw, Joseph R. Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, George van den Driessche, Balaji Lakshminarayanan, Clemens Meyer, Faith Mackinder, Simon Bouton, Kareem Ayoub, Reena Chopra, Dominic King, Alan Karthikesalingam, Cían O. Hughes, Rosalind Raine, Julian Hughes, Dawn A. Sim, Catherine Egan, Adnan Tufail, Hugh Montgomery, Demis Hassabis, Geraint Rees, Trevor Back, Peng T. Khaw, Mustafa Suleyman, Julien Cornebise, Pearse A. Keane, and Olaf Ronneberger. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature Medicine*, 24(9):1342–1350, Sep 2018.

[10] Francesco Piccialli, Vittorio Di Somma, Fabio Giampaolo, Salvatore Cuomo, and Giancarlo Fortino. A survey on deep learning in medicine: Why, how and when? *Information Fusion*, 66:111–137, 2021.

[11] Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M. Hoffman, Wei Xie, Gail L. Rosen, Benjamin J. Lengerich, Johnny Israeli, Jack Lanchantin, Stephen Woloszynek, Anne E. Carpenter, Avanti Shrikumar, Jinbo Xu, Evan M. Cofer, Christopher A. Lavender, Srinivas C. Turaga, Amr M. Alexandari, Zhiyong Lu, David J. Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K. Wiley, Marwin H. S. Segler, Simina M. Boca, S. Joshua Swamidass, Austin Huang, Anthony Gitter, and Casey S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[14] Muhammad E. H. Chowdhury, Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandakar Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al Emadi, Mamun Bin Ibne Reaz, and Mohammad Tariqul Islam. Can ai help in screening viral and covid-19 pneumonia? *IEEE Access*, 8:132665–132676, 2020.

[15] Tawsifur Rahman, Amith Khandakar, Yazan Qiblawey, Anas Tahir, Serkan Kiranyaz, Saad Bin Abul Kashem, Mohammad Tariqul Islam, Somaya Al Maadeed, Susu M. Zughaier, Muhammad Salman Khan, and Muhammad E.H. Chowdhury. Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images. *Computers in Biology and Medicine*, 132:104319, 2021.

[16] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

[17] S. Tabik, A. Gómez-Ríos, J. L. Martín-Rodríguez, I. Sevillano-García, M. Rey-Area, D. Charte, E. Guirado, J. L. Suárez, J. Luengo, M. A. Valero-González, P. García-Villanova, E. Olmedo-Sánchez, and F. Herrera. Covidgr dataset and covid-sdnet methodology for predicting covid-19 based on chest x-ray images. *IEEE Journal of Biomedical and Health Informatics*, 24(12):3595–3605, 2020.

[18] S. Tabik, A. Gómez-Ríos, J. L. Martín-Rodríguez, I. Sevillano-García, M. Rey-Area, D. Charte, E. Guirado, J. L. Suárez, J. Luengo, M. A. Valero-González, P. García-Villanova, E. Olmedo-Sánchez, and F. Herrera. Covidgr dataset and covid-sdnet methodology for predicting covid-19 based on chest x-ray images, 2020.

[19] Parnian Afshar, Shahin Heidarian, Farnoosh Naderkhani, Anastasia Oikonomou, Konstantinos N. Plataniotis, and Arash Mohammadi. Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*, 138:638–643, 2020.

[20] Linda Wang, Zhong Qiu Lin, and Alexander Wong. Covid-net: a tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Scientific Reports*, 10(1):19549, Nov 2020.