UNIVERSITÀ DEGLI STUDI DEL SANNIO

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

Tesi di Laurea

Individuazione del morbo di Parkinson: tecniche di machine learning e image processing applicate a Spiral Test

Relatore:
Prof.ssa.
Lerina Aversano

Correlatore:
Dott.ssa.
Martina Iammarino

Candidato:
Marco Ruta
Matr. 863002147

Anno Accademico 2020-2021

Indice

Indice delle figure

Introduzione

In questo lavoro di tesi verrà analizzata l'applicazione di tecniche di machine learning e deep learning su Spiral Test, metodica utilizzata per diagnosticare la malattia di Parkinson. La scelta di tale argomento è stata adeguata al completamento di questo mio percorso di studi triennale in quanto mi ha fornito ottimi spunti di riflessione e mi ha messo di fronte ad una delle difficoltà che più apprezzo nell'ambito ingegneristico: dover eseguire scelte progettuali al fine di risolvere problemi e migliorare i risultati.

Il lavoro di tesi è composto in tre capitoli:

Nel primo capitolo verrà affrontata la malattia, nello specifico andando ad analizzare *epidemiologia, fisiopatologia, sintomatologia, fattori di rischio, diagnosi, scale di misura* e *trattamenti.*

Nel secondo capitolo verrà affrontato il topic *machine learning*, con un'ampia parentesi relativa a *deep learning* e *reti neurali convoluzionali*, qui vengono descritti tutti gli strumenti che verranno poi utilizzati nel caso di studio.

Nel terzo capitolo, relativo al caso di studio, verranno adottati due *dataset* molto diversi tra loro. L'intero capitolo verterà quindi su un lavoro svolto in parallelo, in una prima fase verranno analizzati i dataset e sarà eseguito *pre-processing* dei dati, in seguito verrà eseguita una *classificazione,* prima utilizzando le stesse tecniche di machine learning per entrambi i dataset, poi, andando ad applicare tecniche di deep learning per migliorare i risultati ottenuti.

La natura dissimile dei due dataset e il loro numero ridotto di istanze sono alcune tra le difficoltà che si sono presentate durante il lavoro. L'applicazione di tecniche di *data extraction* e *data augmenting* hanno permesso di superare tali difficoltà, ottenendo dei buoni risultati.

Capitolo 1: Il Morbo di Parkinson

1.1 Definizione ed epidemiologia

Il morbo di Parkinson è una malattia cronica e neurodegenerativa caratterizzata dalla progressiva perdita di neuroni dopaminergici all'interno della substantia nigra e dallo sviluppo di corpi di Lewi all'interno dei neuroni celebrali. La malattia fu individuata e descritta per la prima volta dal medico inglese James Parkinson, il quale la definì come una "paralisi agitante" nel suo saggio "An Essay of the Shaking Palsy". Successivamente la malattia fu etichettata con il suo attuale nome dal Neurologo James Martin Charcot [1, 2]. È il secondo disturbo neurodegenerativo per incidenza mondiale dopo la malattia di Alzheimer, con diffusione pari a 7-10 milioni di persone e tasso di incidenza annuo di circa 10-20 nuovi casi su 100 000 persone [3].L'incidenza della malattia è correlata ad alcuni fattori intrinsechi come età, sesso ed etnia.

L'età media di diagnosi va dai 55 ai 60 anni e l'incidenza aumenta dell'1-2% in persone con più di 60 anni, fino ad arrivare al 3.5% in individui di 85-89 anni. L'etnia con maggiore incidenza è quella caucasica, la quale presenta un tasso doppio rispetto alle altre. Pazienti di sesso maschile sono 1.5 volte più propensi a sviluppare il morbo rispetto a pazienti di sesso femminile. Questo è probabilmente dovuto all'effetto neuro-protettivo dell'estrogeno, il principale ormone sessuale femminile [3].

Anche i fattori ambientali hanno un ruolo importante, ad esempio nella zona dei grandi laghi (USA) è stata dimostrata la correlazione tra il morbo e l'esposizione prolungata a tossine presenti nella zona.[4]

1.2 Fisiopatologia

La progressiva degenerazione di neuroni dopaminergici nella substanza nigra porta alla progressiva diminuzione e limitazione delle funzioni motorie nel paziente.

La presenza di "sintomi non motori" suggerisce però l'alterazione, da parte della malattia, di altre zone del cervello, come la comparsa nei neuroni di corpi di Lewi [2].

La malattia viene definita come idiopatica in quanto le cause scatenanti non sono ancora del tutto note, in letteratura sono presenti due ipotesi, non necessariamente esclusive, riguardanti la patogenesi di tale malattia:

- Misfolding e aggregazione di proteine portano alla morte cellulare dei neuroni dopaminergici, tali proteine possono infatti essere neurotossiche e possono causare danni cellulari se l'organismo non è in grado di riconoscerle ed eliminarle in autonomia [6].
- Disfunzione mitocondriale e conseguente stress ossidativo portano ad una aumentata produzione di specie reattive dell'ossigeno (ROS) le quali interagendo con acidi nucleici, proteine e lipidi causano danni cellulari. Il metabolismo della dopamina rende i neuroni dopaminergici territorio fertile per la produzione di ROS [7].

I corpi di Lewi sono aggregati citoplasmatici di proteine di forma sferica con diametro che può variare tra 7 e 15 nm [5]; vengono rilevati all'interno dei neuroni dopaminergici e sono composti per la maggior parte da α-sinucleina insolubile, sottoforma di filamenti, in combinazione con una serie di proteine.

Non è del tutto chiaro il meccanismo che porta ad un eccessivo accumulo di α-sinucleina insolubile; tuttavia, la letteratura suggerisce che un errato funzionamento del sistema ubiquitina-proteasoma (UPS), il quale si occupa della proteolisi intercellulare, porti ad un'aggregazione anomala di proteine, tra le quali anche l'α-sinucleina [2, 8].

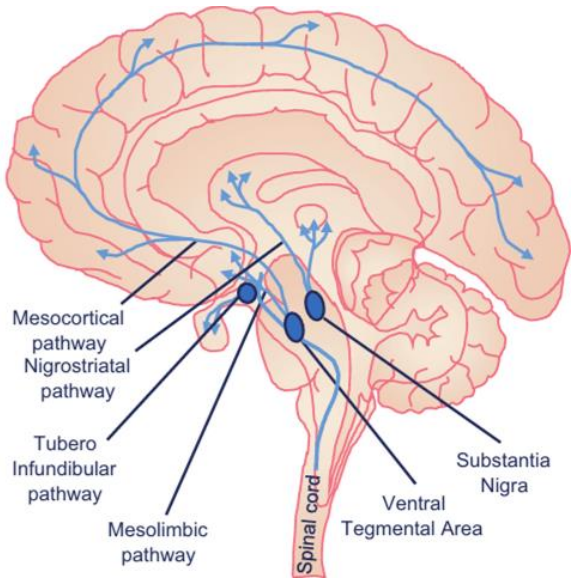

Figura 1 – Sistema Dopaminergico

1.3 Sintomatologia

Il morbo di parkinson viene classificato come disturbo del movimento. I sintomi principali sono difatti di prevalenza motoria e sono dovuti alla costante ed irreversibile diminuzione di dopamina all'interno dei neuroni; le caratteristiche motorie più frequenti vengono raggruppate sotto il termine "parkinsonismo" e sono *bradicinesia*, *tremore a riposo* e *rigidità posturale* [9].

Tuttavia, in molti casi sono presenti una serie di "sintomi non motori" molti dei quali spesso precedono la diagnosi e contribuiscono ad incrementare il livello di disabilità ed a ridurre la qualità di vita del paziente. Purtroppo, i sintomi non motori vengono spesso fraintesi e non correlati allo sviluppo della malattia [10, 11, 12].

1.3.1 Sintomi motori

- *Bradicinesia:* segno motorio tipico e più facilmente riconoscibile della malattia di Parkinson. Consiste nell'eccessiva lentezza nei movimenti del paziente inducendo grandi difficoltà nell'iniziare e portare a termine attività sequenziali che in generale richiedano un controllo motorio molto fine, come ad esempio la scrittura o l'utilizzo di piccoli utensili. Tale sintomo viene valutato chiedendo al paziente di compiere rapidi movimenti ripetuti andando ad analizzare eventuali perdite di velocità ed ampiezza [10].
- *Tremore a riposo:* movimento involontario e oscillatorio con frequenza compresa tra i 4 e i 6 Hz che solitamente si manifesta negli arti ma può essere anche presente in labbra, mento e mascella. Viene definito tremore "a riposo" in quanto si attenua temporaneamente quando il paziente intraprende movimenti o azioni. Questo sintomo è presente nel 75% dei pazienti e tende a diminuire negli stadi più avanzati della malattia [10, 11].
- *Rigidità:* aumento spropositato del tono muscolare con conseguente maggior resistenza a movimenti passivi. Può coinvolgere collo, spalle, anche e polsi causando nel paziente una sensazione di tensione permanente. È uno dei sintomi principali della malattia ed è spesso associato a dolore nelle zone interessate [11].
- *Instabilità posturale:* sintomo comune soprattutto negli stadi più avanzati della malattia ed è dovuto principalmente alla perdita di riflessi posturali. Questo porta ad una compromissione dell'equilibrio e alla comparsa di blocchi motori ("freeze of gait"), con conseguente difficoltà nel cammino e nell'esecuzione di attività quotidiane. La camminata del paziente affetto da Parkinson diventa lenta e strisciante, vengono persi i

movimenti oscillatori delle braccia e risulta più difficile cambiare direzione [10].

1.3.2 Sintomi non motori

- *Disturbi neuropsichiatrici:* il disturbo più diffuso è la *depressione*, di cui si stima siano affetti circa il 40% dei pazienti. Sintomi secondari ad essa correlati sono mancanza di energia, disturbi del sonno, poca concentrazione, ansia, irritabilità e ritardo psicomotorio. Altro disturbo meno frequente è la *psicosi*, la quale si presenta sottoforma di allucinazioni visive e paranoia. Altro sintomo rilevante che colpisce il 40% dei pazienti è l'*ansia* che si manifesta sotto forma di sudorazione eccessiva, dolore al petto, mancanza di respiro e vertigini [14].
- *Disturbi cognitivi:* circa il 30-40% dei pazienti affetti dalla malattia sviluppa forme di demenza. Tale disturbo comporta un rallentamento psicomotorio, deficit della memoria, alterazioni di umore e personalità, perdita di attenzione e apatia. Tali sintomi sono invalidanti e portano ad una rapida progressione della malattia oltre che al peggioramento della qualità di vita del paziente [13].
- *Disfunzione olfattiva:* è uno dei primi disturbi diagnosticanti la malattia, in quanto si manifesta nei primi stadi e colpisce più del 90% dei pazienti; la causa di tale disfunzione è la compromissione del SNC [15].
- *Costipazione:* rientra tra i primi sintomi della malattia e può presentarsi con largo anticipo rispetto all'insorgenza dei sintomi motori; si stima che colpisca più del 90% dei pazienti ed il disturbo tende ad aumentare con l'avanzare della terapia [13, 14].
- *Disturbo del sonno:* è il sintomo non motorio più frequente, con incidenza che varia tra il 70-98% dei pazienti. Tale disturbo viene enfatizzato dalla somministrazione di terapie dopaminergiche. Le varie manifestazioni sono eccessiva sonnolenza nelle ore diurne, sonnolenza improvvisa e disturbo della fase REM. Quest'ultimo è un sintomo molto invalidante caratterizzato dalla perdita dell'atonia muscolare fisiologica, che induce improvvisi sobbalzi e movimenti, anche violenti durante le ore notturne [13, 16].
- *Ipotensione ortostatica:* sintomo caratteristico delle fasi avanzate. Si stima sia presente in forma sintomatica nel 30% dei pazienti; consiste in un rapido calo della pressione arteriosa quando il paziente passa in posizione eretta, con conseguente debolezza, senso di vertigine, visione offuscata fino ad arrivare a sincope nei casi peggiori. I trattamenti dopaminergici incrementano lo sviluppo di tale sintomo [13, 14].

1.4 Fattori di rischio

Caratteristiche intrinseche come età, sesso ed etnia influenzano fortemente l'incidenza della malattia, tuttavia esistono una serie di fattori estrinseci che condizionano questo processo.

L'insorgere della malattia è correlato a fattori genetici e fattori ambientali, come l'utilizzo di alcuni pesticidi, erbicidi e metalli pesanti. Da alcuni studi è difatti emerso come l'esposizione prolungata di sostanze come *rotenone* (pesticida) e *paraquart* (erbicida) incrementi del 70% la probabilità di contrarre la malattia nell'arco di 10-20 anni [17].

Diversi studi dimostrano come la *familiarità*, ovvero la presenza di un parente di primo grado avente la malattia di Parkinson, incrementi la possibilità di sviluppare la patologia. Si stima che più del 5% dei pazienti sia affetto da una forma di "Parkinson giovanile" ereditario [18].

In letteratura sono stati individuati anche una serie di "fattori protettivi" che sembrano ridurre le possibilità di sviluppo della malattia. È stato dimostrato come il consumo di nicotina, il quale attiva un rilascio di dopamina e una serie di recettori neuro-protettivi, riduca fino al 60% le possibilità di contrarre il Parkinson. Anche l'assunzione di caffeina, antagonista del recettore adenosina A_{2A}, ne riduce del 30% le possibilità [30].

1.5 Diagnosi

La diagnosi della malattia di Parkinson non si basa su test diagnostici che ne certifichino la presenza, ma da un'indagine clinica durante la quale vengono analizzati i sintomi, la storia clinica del paziente e la risposta al trattamento. Per la diagnosi deve essere presente bradicinesia accompagnata da tremore e/o rigidità. La malattia in fase precoce si presenta spesso in forma monolaterale, è quindi necessario porre particolare attenzione all'asimmetria degli eventuali sintomi [15].

Specialmente nelle fasi precoci della malattia, può essere utile ricorrere a tecniche di *imaging* per aiutare il medico ad eseguire la diagnosi. La *risonanza magnetica* (RM) è utile per escludere malattie che presentano gli stessi sintomi del Parkinson; tuttavia, tale strumento non possiede la sensibilità necessaria per individuarlo.

Tecniche di *imaging nucleare* come PET (Positron Emission Tomography) e SPECT (Single Photon Emission Computed Tomography) sono invece adatte allo scopo.

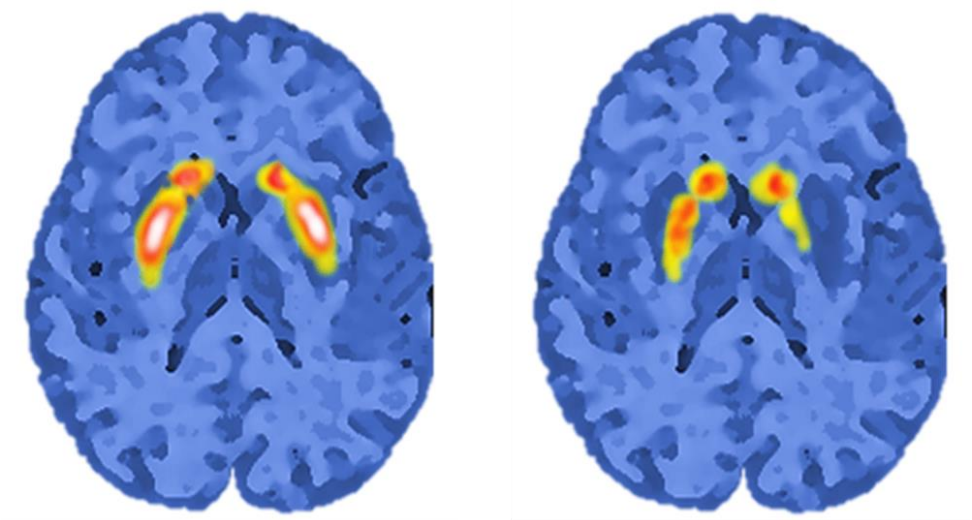

Figura 2 – DaTscan di un paziente sano (a sinistra)
e di un paziente con malattia di Parkinson (a destra)

DaTscan è una particolare tecnica di SPECT che consente di individuare la degenerazione dei trasportatori dopaminergici e di distinguere pazienti sani o malati. Il risultato di tale tecnica si presenta come un'immagine dove il tracciante è interamente localizzato nel corpo striato, il luogo in cui avviene la maggior parte delle trasmissioni dopaminergiche. In condizioni normali il corpo striato si presenta sotto forma di due "virgole" simmetriche, nei pazienti affetti dal morbo invece tale zona risulta asimmetrica e/o ridotta [19, 20].

1.6 Scale di misura

Una scala di misura clinica molto diffusa è la scala di Hoehn and Yahr (H&Y), definita per la prima volta nel 1967 da Margaret Hoehn e Melvin Yahr. Tale scala permette di categorizzare l'evoluzione della malattia in cinque livelli. I sintomi valutati sono compromissione del cammino e dell'equilibrio, ponendo particolare importanza alla bilateralità dei sintomi. L'attuale versione della scala H&Y presenta due livelli intermedi (1.5 e 2.5) in modo da poter meglio descrivere agli stadi intermedi della malattia [21].

Stage	Modified Hoehn and Yahr Scale
1	Unilateral involvement only
1.5	Unilateral and axial involvement
2	Bilateral involvement without impairment of balance
2.5	Mild bilateral disease with recovery on pull test
3	Mild to moderate bilateral disease; some postural instability; physically independent
4	Severe disability; still able to walk or stand unassisted
5	Wheelchair bound or bedridden unless aided

Figura 3 – Scala di Hoehn and Yahr modificata

La scala di misura clinica più utilizzata per la definizione della severità della malattia di Parkinson è la Unified Parkinson's Disease Rating Scale (UPDRS). Viene definita per la prima volta negli anni 80, nel 2001 subisce una revisione da parte della Movement Disorder Society.

La versione attuale è denominata MDS sponsored UPDRS revision (MDS-UPDRS) e si presenta come una serie di 65 domande suddivise in 4 sezioni ben distinte:

- sintomi non motori provati dal paziente durante le attività di vita quotidiana
- sintomi motori provati dal paziente durante la vita quotidiana
- valutazione clinica dei sintomi motori del paziente
- analisi delle complicazioni motorie

Ognuna delle 65 domande prevede come risposta un numero compreso tra 0 e 4, identificativo della frequenza in cui si presenta tale disturbo [22, 23].

1.7 Trattamenti

I trattamenti attualmente utilizzati si dividono in farmacologici e chirurgici. I trattamenti farmacologici mirano ad alleviare temporaneamente i sintomi, il farmaco più utilizzato è la Levodopa, per cui si stima un'effettiva riduzione dei sintomi nell'80% dei pazienti. Tale farmaco, nonostante la sua efficacia, presenta una serie di effetti collaterali associati al trattamento per medio-lungo termine. Per questo motivo viene somministrato in maniera combinata insieme alla Carbidopa, con l'obiettivo di ridurre gli effetti collaterali [24, 25, 26].

I trattamenti chirurgici vengono considerati come primo approccio alla cura, precedente ad eventuali trattamenti farmacologici. La stimolazione profonda del cervello (DBS) è la tecnica più frequentemente utilizzata ed efficace, con uno spettro molto ristretto di effetti collaterali [27].

Consiste nella stimolazione elettrica di aree specifiche del cervello attraverso alcuni elettrodi (localizzati nell'area interna del globus pallidus o nei nuclei subtalamici) alimentati da un generatore interno di impulsi. La stimolazione di tali aree comporta un netto miglioramento del tremore, della rigidità e della bradicinesia, con risultati che si stima rimangano stabili fino a 8 anni dall'intervento [25, 28].

Non tutti i pazienti possono essere sottoposti a tale tecnica, il risultato è garantito solo per pazienti con età inferiore a 70 anni e in assenza di disturbi cognitivi o psichiatrici [29].

Capitolo 2: Machine Learning

2.1 Introduzione

Il machine learning è una branca dell' artificial intelligence, sviluppatasi negli ultimi decenni del XX secolo, che studia i sistemi e gli algoritmi capaci di *imparare* dai dati, estraendo e sintetizzando conoscenza da essi.
Una definizione puntuale e formale viene fornita da T.M. Mitchell:

> *"Un programma apprende da una certa esperienza **E** se nel rispetto di una classe di compiti **T**, con una misura della prestazione **P**, la prestazione **P** misurata nello svolgere il compito **T** è migliorata dall'esperienza **E**"* [34].

Il machine learning è di grande aiuto nello svolgimento di task che sarebbero troppo complicati per essere affrontati utilizzando classici paradigmi di programmazione. In questo contesto il task viene descritto in termini di come il nostro sistema dovrebbe processare un campione d'esempio. Un campione viene solitamente indicato attraverso un vettore $x \in \mathrm{R}^n$, dove ogni elemento x_i rappresenta una *feature* dello stesso. Un esempio tipico riguarda le immagini: nella maggior parte dei casi verranno utilizzati i valori dei pixel dell'immagine come vettore delle *features*.
Le tipologie di task che possono essere affrontate attraverso algoritmi di machine learning sono molteplici, di cui le due principali sono:

- *Classificazione:* task in cui l'algoritmo deve riuscire ad associare ad ogni input una tra k classi. Per poter risolvere questa task l'algoritmo produrrà una funzione $f : \mathrm{R}^n \rightarrow \{1, \dots, k\}$ tale che $y = f(x)$, ovvero il modello classifica l'input descritto dal vettore *x* con la categoria identificata dal codice numerico *y*.

- *Regressione:* task in cui l'algoritmo deve riuscire a predire un valore numerico dato un input. Per poter risolvere questa task l'algoritmo produrrà una funzione $f : \mathrm{R}^n \rightarrow \mathrm{R}$, che si differenzia da quella di classificazione per il formato dell'output [35].

Per poter valutare le abilità predittive di un algoritmo di machine learning è necessario definire delle metriche capaci di quantificarne le performance. La metrica più utilizzata per i problemi di classificazione è l'*accuracy*, intesa come la proporzione tra il numero di esempi classificati correttamente e il totale delle predizioni eseguite dal modello. Può assumere valori compresi tra 0 e 1, rispettivamente se ogni predizione risulta errata e se tutte risultano corrette. Possiamo ottenere la stessa informazione andando a valutare la metrica complementare dell'accuracy: l'*error rate.*

Gli algoritmi di classificazione si dividono in *algoritmi di apprendimento supervisionato* e *algoritmi di apprendimento non supervisionato,* in base al tipo di esperienza sperimentata durante la fase di apprendimento.

- *Algoritmi di apprendimento non supervisionato:* sperimentano un dataset contenente molte *features* e apprendono autonomamente proprietà della struttura del dataset. In questo caso i dati non sono classificati e/o strutturati ed è compito del modello poter riconoscere schemi e strutture ricorrenti. In particolare, vengono osservati molti esempi random del vettore *x* e si tenta di ricavare, esplicitamente o implicitamente, la *distribuzione di probabilità P(x)*. Un esempio classico è il *clustering*, il cui obiettivo è dividere il dataset in diversi *cluster* contenenti elementi tra loro simili.

- *Algoritmi di apprendimento supervisionato:* sperimentano un dataset contenente molte *features* in cui però, ad ogni elemento del dataset, è associata una *label* o *target*. In questo caso il modello osserva molti esempi random del vettore *x*, associati ognuno alla propria *label y* e cerca di ricavare la *probabilità condizionata P*(x | *y*). Un esempio classico sono algoritmi di classificazione o regressione.

Esistono anche altre tipologie di apprendimento, meno frequentemente utilizzate, che si discostano dalle due principali:

- A*lgoritmi di apprendimento semi-supervisionato:* sperimentano un dataset in cui la label è presente solo per alcuni degli elementi, mentre per gli altri no.

- *Algoritmi di apprendimento multi-instance:* sperimentano un dataset in cui per una collezione finita di elementi esiste una label comune, ma per il singolo elemento dell'insieme la label è assente.

- *Algoritmi di apprendimento rinforzato:* interagiscono direttamente con un ambiente capace di fornire un riscontro diretto, sotto forma di *premi* e *penalità*. Si crea un *feedback-loop* tra il sistema e l'ambiente, l'ambiente reagisce fornendo una risposta e l'algoritmo la analizza in funzione dell'obiettivo da raggiungere [35].

  Tipicamente si è interessati ai risultati ottenuti dall'algoritmo su dati mai sperimentati, in modo da poterne simulare il funzionamento nel mondo reale. Per questo motivo le performance vengono valutate su un set specifico, definito *testing set*, distinto dal set su cui viene allenato l'algoritmo: il *training set.*

- *Training set*: dati che verranno utilizzati esclusivamente nella fase di addestramento, da questi dati il modello *apprenderà* le relazioni tra il vettore X (delle *features*) e Y (*target*). È molto importante che tali dati siano completi, ovvero ad ogni vettore X deve corrispondere un target Y. I campioni presenti in questo set devono essere scelti con cura: non devono essere troppo specializzati e differenti dai dati che verranno utilizzati per il testing, per non incorrere nel fenomeno dell'*overfitting* [35].
- *Testing set*: dati che verranno utilizzati per la valutazione del modello. A tale scopo questi dati non devono assolutamente essere presenti nel testing set in quanto il modello riuscirebbe ad eseguire predizioni senza problemi e ciò andrebbe a falsificare i risultati. Per questo motivo il testing set viene separato preventivamente dal training set [35].

2.2 Algoritmi di Machine Learning

In questo paragrafo verranno analizzati alcuni algoritmi di machine learning, in particolare quelli che verranno poi impiegati nel caso di studio (Capitolo 3).

2.2.1 Decision Tree

Classificatore basato su un albero decisionale costituito da *nodi*, *archi* e *foglie*. Ogni nodo dell'albero di decisione è una condizione che verrà testata per ogni istanza del testing set ed è costituito da un elemento di *split*: una feature ricavata dai dati di training tale da poter produrre una distribuzione omogenea dei dati in accordo con le *label* della classe. Ogni *arco* consente la connessione tra un nodo padre ed un nodo figlio e presenta uno specifico valore per l'attributo di *split*. Ogni *foglia*, invece, rappresenta una *label* di classe [36].

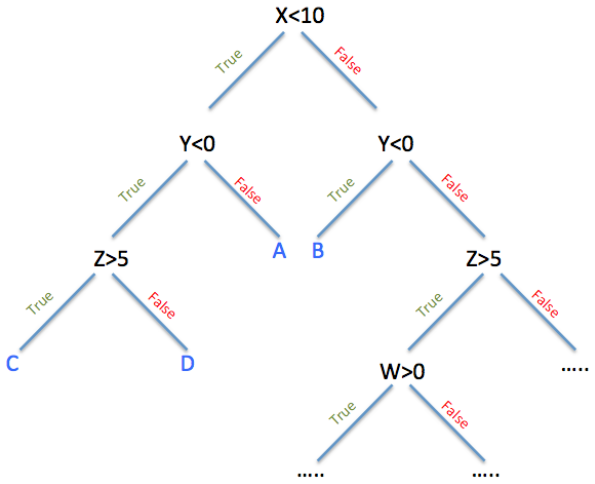

Figura 4 – Decision Tree

Una delle problematiche più note riguardanti gli alberi decisionali è il fenomeno dell'*overfitting*, che si presenta quando il modello è fin troppo specializzato sul training set. Questa eccessiva specializzazione porta alla creazione di alberi molto complessi, con conseguente numero di nodi molto elevato. Questo fenomeno può essere risolto utilizzando due tecniche di *potatura*, ovvero di riduzione del numero di nodi:

- *Pre-pruning:* tecnica che tenta di prevenire l'overfitting andando ad interrompere preventivamente la costruzione dell'albero decisionale quando viene superata una determinata soglia. Solitamente l'algoritmo di training termina quando è stato esaminato l'intero training set, utilizzando questa tecnica si va ad interrompere l'addestramento quando ad esempio l'accuracy supera la soglia.

- *Post-pruning:* in questo caso l'albero viene costruito analizzando ogni istanza del training set ed in seguito viene potato. La potatura viene eseguita andando a valutare l'eventuale miglioramento delle prestazioni in seguito alla rimozione di uno o più nodi.

Andando ad applicare queste tecniche senza criterio si può incorrere nel fenomeno opposto, *l'underfitting.* Questo succede quando il modello è stato generato in maniera troppo semplicistica oppure è stato eseguito *pruning* eccessivo, portando ad un alto numero di errori di classificazione.
Una caratteristica molto positiva di questa tipologia di classificatore è sicuramente l'interpretabilità del modello creato, sempre a patto di mantenere un numero di foglie limitato [36] .

2.2.2 Random Forest

Classificatore *esembled*, ovvero realizzato come combinazione ottimizzata di più *Decision Tree*, rappresentato come una foresta di alberi decisionali. La classificazione di un'istanza viene eseguita da ognuno degli alberi, i quali lavorano utilizzando ognuno un diverso sub-set di attributi. I risultato finale sarà quello più popolare generato dai vari alberi [36].

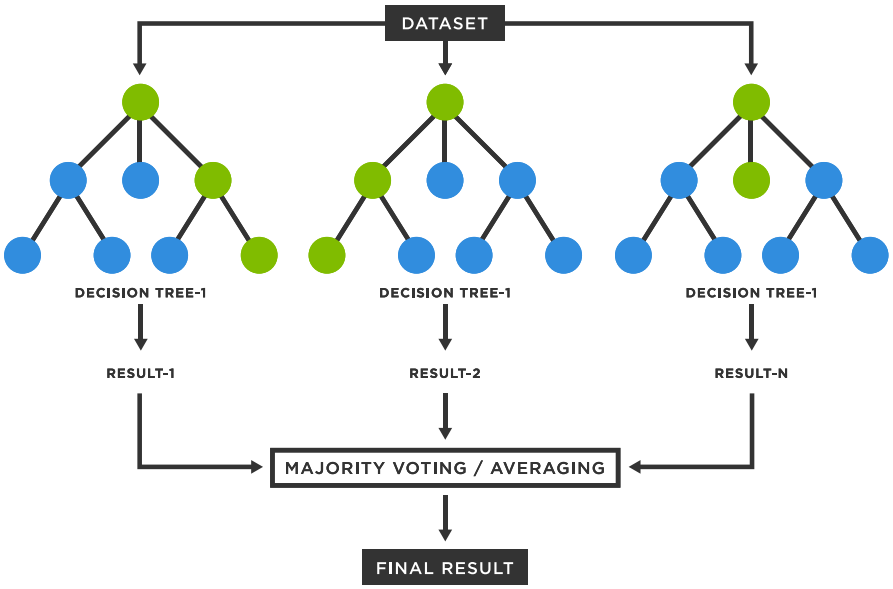

Figura 5 – Random Forest

2.2.3 Gradient Boosting

Altra tipologia di classificatore *esembled,* realizzato come combinazione ottimizzata di più classificatori semplici, tipicamente *Decision Tree.* Vengono addestrati sull'insieme di training *N* classificatori, l'addestramento dell' *i-esimo* classificatore produrrà dei pesi, da cui partirà l'addestramento dell'*i+1-esimo* classificatore. In questo modo si possono andare a smussare incrementalmente gli errori commessi dai precedenti classificatori. Il risultato finale, come nel caso precedente, sarà quello più popolare generato dai vari classificatori [36].

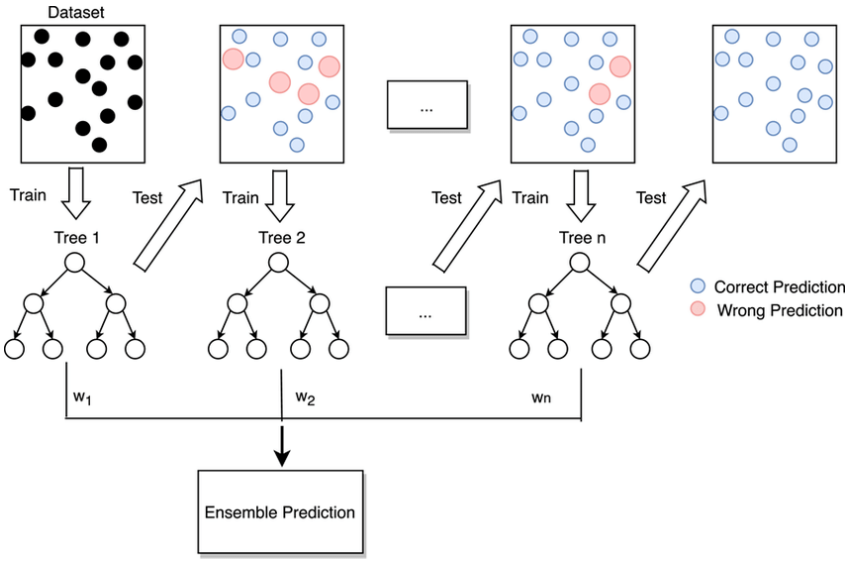

Figura 6 – Gradient Boosting

2.2.4 K-Nearest Neighbours

Tecnica di classificazione in cui non è prevista la creazione di un modello capace di etichettare le istanze di test, quello che viene fatto è conservare tutte le istanze di training in un piano. Ogni istanza è caratterizzata dalla sua posizione, dettata dal valore dei suoi attributi. Il grafico risultante verrà utilizzato per classificare le istanze del testing set, andando a valutare per ogni nuova istanza quali sono le classi dei K elementi più vicini. Il risultato della classificazione sarà dato dalla classe più frequente dei K vicini. La scelta del parametro K diventa essenziale: in caso di K troppo piccolo si va incontro a problemi di *rumore,* al contrario invece si rischia di includere nei vicini anche istanze appartenenti ad altre classi [36].

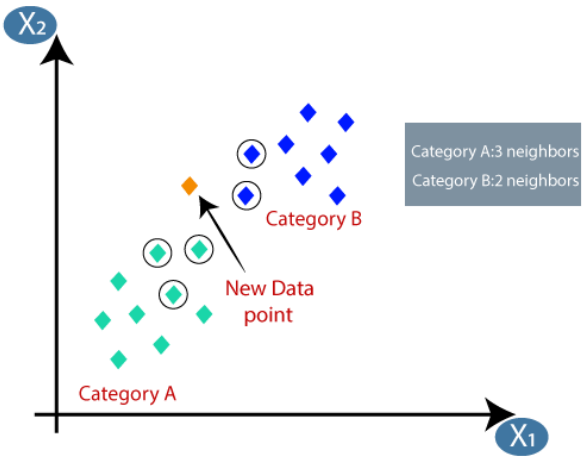

Figura 7 – K-Nearest Neighbours

2.2.5 Logistic Regression

Tecnica di regressione non lineare utilizzata per stabilire la probabilità con cui un sample possa generare uno o l'altro valore della variabile *target*. È una tecnica di regressione che opera solo su variabili dipendenti *dicotomiche*, ossia che possono assumere come valori solo 0 o 1; per questo motivo è una tecnica che può esser utilizzata per eseguire classificazione binaria [36].

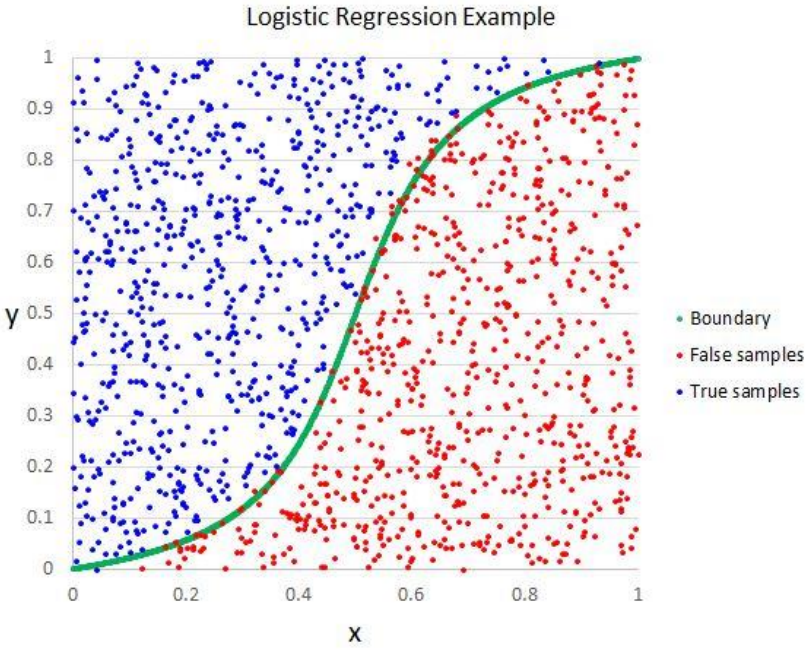

Figura 8 – Logistic Regression

2.4 Deep Learning

Il deep learning è una specializzazione del machine learning, entrambi condividono lo stesso scopo, ma il modo in cui lo si raggiunge è del tutto diverso. Nei modelli di machine learning il programmatore deve, esplicitamente o implicitamente, guidare l'algoritmo durante la fase di training. Nel deep learning si tenta invece di simulare il funzionamento del cervello umano, capace di correggere *autonomamente* le future previsioni a fronte di un errore. Il termine *deep* deriva dal fatto che le *reti neurali* utilizzate sono dotate di molteplici layer nascosti, rendendole, di fatto, profonde e di struttura simile a quella del cervello umano. La logica con cui vengono affrontati i problemi da un modello di deep learning è di decomposizione e gerarchizzazione; ogni rappresentazione di un dato si basa su un insieme di rappresentazioni più semplici e può essere sempre decomposta in esse [37].

2.4.1 Reti Neurali

Le reti neurali basano il loro funzionamento su quello di un neurone biologico. I principali componenti di quest'ultimo sono:

- Soma: parte principale del neurone.
- Assone: filamento che fuoriesce dal soma e trasporta impulsi elettrici, si ramifica nelle sue estremità di *sinapsi*, avente ognuna un proprio *bottone sinaptico*.
- Dendriti: filamenti che fuoriescono dal *soma.*

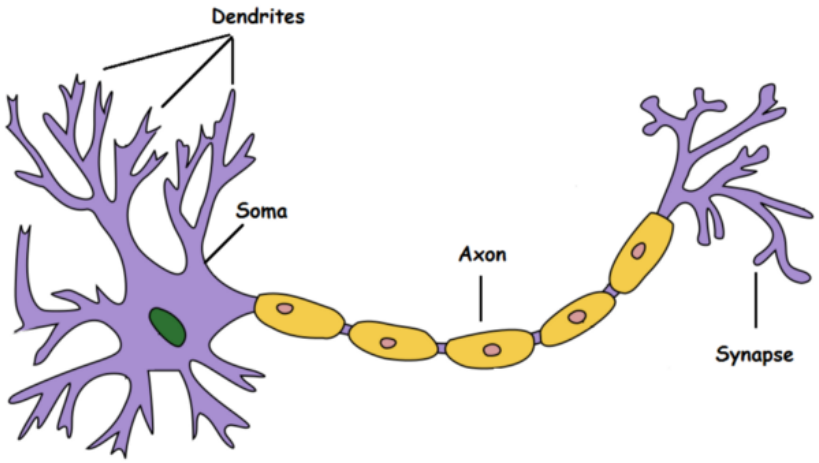

Figura 9 – Rappresentazione neurone biologico

Il neurone è capace di captare segnali in ingresso attraverso i *dendriti*, elaborarli nel *soma* e successivamente trasmetterli ad altri neuroni tramite l'*assone.* Quando un segnale si trova presso una *sinapsi*, questa rilascia particolari *neurotrasmettitori* che ne determinano la sua conduttività, ovvero quanto la sinapsi attenua o amplifica il segnale proveniente dall'*assone* [38].

Il primo modello di neurone artificiale è stato sviluppato nel 1943 da Warren McCulloch e Walter Pitts, in tale modello sono previsti n input *i*, ognuno pesato con un proprio peso *w* che vengono sommati. I pesi di ogni input vengono scelti in base all'importanza di quest'ultimo, in modo che possano contribuire di più o di meno al risultato finale. Se la somma dei vari input supera la funzione di soglia *T*, si produrrà un segnale in output [39].

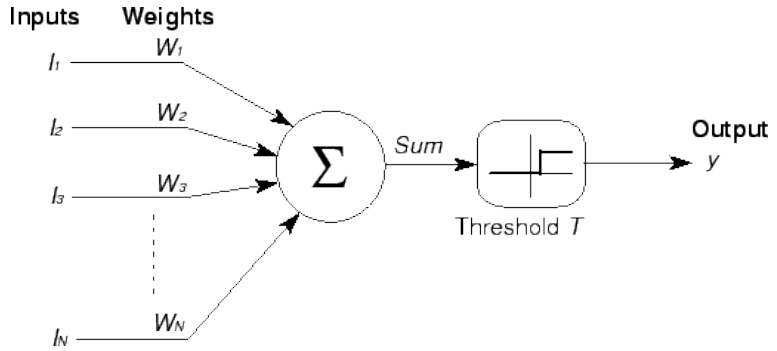

Figura 10 – Rappresentazione neurone di McCulloch e Pitts

I limiti di tale modello sono dettati dal fatto che i pesi e le connessioni devono essere calibrati manualmente per poter ottenere buoni risultati. Intorno alla fine degli anni Cinquanta, Frank Rosenblatt introdusse una rete composta di unità perfezionate del modello di McCulloch – Pitts, denominata *Perceptron.* Il modello aggiunge un ulteriore valore di input costante, denominato bias, e la sua equazione è la seguente:

$$output = \begin{cases} 0, & \sum_j w_j x_j \leq soglia \\ 1, & \sum_j w_j x_j > soglia \end{cases}$$

considerando x e w come vettori e bias come opposto del valore di soglia, possiamo riscrivere l'equazione come:

$$a = \begin{cases} wx + b \leq 0 \\ wx + b > 0 \end{cases}$$

a è la funzione di attivazione, mentre il bias può essere interpretato come una soglia che

consente di influenzare rapidamente l'output del singolo neurone.

La novità principale di questo modello risiede nella sua capacità di poter ricalibrare i propri pesi a fronte di un errore [39].

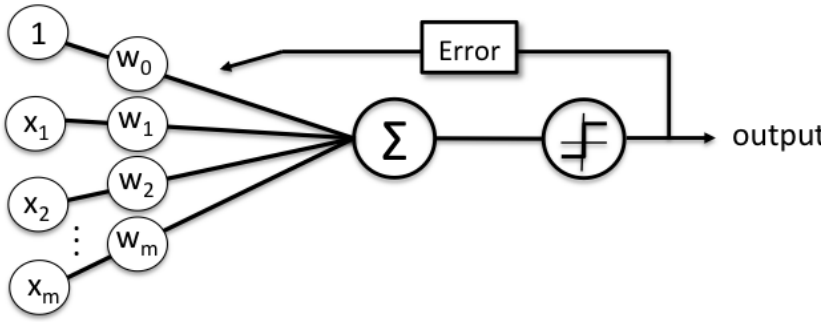

Figura 11 – Rappresentazione Perceptron

Il limite più grande del singolo perceptron è l'incapacità nell'affrontare problemi non lineari. Al fine di poter risolvere problemi più complessi si iniziarono a connettere gli input di neuroni artificiali con output di altri neuroni artificiali, creando delle reti layered chiamate *perceptron multistrato.*

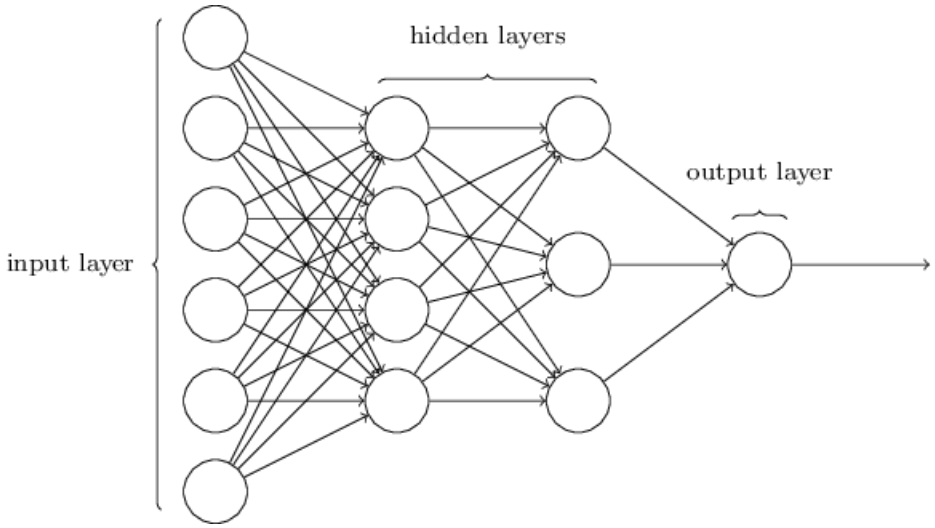

Figura 12 – Rappresentazione Multi Layer Perceptron

Ci troviamo di fronte ad una vera e propria rete neurale artificiale composta da più perceptron. Sono presenti un input layer che capta il segnale ed un output layer che esegue una classificazione, tra questi due layer sono presenti un numero arbitrario di layer *nascosti*. Ogni neurone di un livello è connesso a tutti i neuroni del livello precedente,

questo porta alla creazione di una rete *completamente connessa* [39].

2.4.2 Addestramento di una Rete Neurale

Il metodo più conosciuto ed utilizzato per addestrare una rete neurale è l'algoritmo di *retropropagazione dell'errore*, proposto nel 1986 da David. E. Rumelhart, G. Hinton e R.J. Williams [40]. L'algoritmo si basa sulla modifica sistematica dei pesi delle connessioni presenti tra i vari neuroni, affinché il risultato si avvicini a quello aspettato. Nella fase di addestramento possono essere distinte due fasi:

Forward propagation: fase in cui vengono calcolate tutte le attivazioni dei neuroni della rete, dal primo all'ultimo, per conservarne temporaneamente i risultati intermedi. I valori in output stimati sono il risultato di questa fase di training e possono essere confrontati con i valori attuali per ottenere l'*errore* commesso:

$$E(w) = \sum_{i=1}^{N} (y_i - F_w(x_i))^2$$

Backward propagation: fase in cui, attraverso il metodo di *discesa del gradiente*, si ricercano i pesi *w* che minimizzino l'errore quadratico definito in precedenza.

Siano x_d i valori di input, t_d le etichette, o_d i valori di output osservati e w_i l'i-esimo peso, si calcolano le derivate parziali dell'errore rispetto alle singole componenti:

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{N} (t_d - o_d)^2$$

Il gradiente $\nabla E(w)$ è l'insieme di tutte le derivate rispetto alle n componenti:

$$\nabla E(w) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n} \right]$$

E ci consente di calcolare la variazione da applicare al vettore *w* al fine di minimizzare l'errore:

$$\nabla w = -\mu \nabla E(w) = -\mu \frac{\partial E}{\partial w_0}$$

Con μ definito *learning rate*, un valore positivo molto piccolo, tale da poter correggere ma anche da poter mantenere le classificazioni corrette. Il vettore a questo punto *w* viene aggiornato con la seguente formula, nota come *discesa del gradiente:*

$$w = w - \mu \nabla E(w)$$

Questi passi vengono eseguiti per ogni neurone della rete, in modo da poterla bilanciare [40].

2.4.3 Reti neurali convoluzionali

Le *reti neurali convoluzionali* (CNN) sono un particolare tipo di rete neurale, composte anche esse da neuroni, pesi e bias. La peculiarità di tali reti è che l'input debba avere una precisa struttura-dati di riferimento, come ad esempio un'immagine. Altra differenza fondamentale rispetto alle normali reti neurali artificiali è che i diversi *layers* di una CNN hanno neuroni disposti lungo tre dimensioni: *larghezza, altezza* e *profondità*. Inoltre, si perde la caratteristica di rete neurale *completamente connessa*, in quanto i neuroni di un layer saranno connessi solo ad una piccola porzione del livello precedente [41].

Nelle CNN si distinguono tre tipologie di *layer* che vengono stratificati nella composizione:

- *Convolutional layer:* blocco portante di una CNN in cui vengono eseguite la maggioranza delle operazioni di computazione più dispendiose. I parametri di questo layer sono una serie di *filtri (o kernel),* molto piccoli dal punto di vista dimensionale (filtro tipico utilizzato 5x5x3 con 5 e 5 che individuano i pixel e 3 che indica la profondità, in questo caso RGB), ma che vengono estesi all'intero input del layer. Il risultato di tale applicazione è una *activation map* bidimensionale relativa al filtro applicato. La rete apprenderà, intuitivamente, quali sono i filtri che causano l'attivazione delle uscite del neurone quando viene osservato un certo tratto visuale distintivo. Tutte le *activation map* prodotte dai vari filtri vengono poi concatenate lungo la dimensione della profondità, producendo un *volume* di output. La quantità di volume prodotto dipende dal numero di filtri applicati e dallo stride (quanto viene traslato un filtro ad ogni passo). Buona norma è inserire degli zeri lungo il bordo dell'input in modo da tenerne sotto controllo la dimensione , questa tecnica è definita *zero padding.*
  La dimensione spaziale del *volume* di output può essere calcolata come segue, considerando W il volume di input, F la dimensione del filtro, S lo stride utilizzato e P il valore di zero padding:

$$\frac{W - F + 2P}{S} + 1$$

Solitamente dopo ogni *convolutional layer* si trova un *ReLU layer* che ha come obiettivo quello di aumentare la non linearità della funzione di attivazione senza intaccare la dimensione del filtro [41].

- *Pooling layer:* layer che solitamente viene disposto in mezzo a due *convolutional layer*. La sua funzione principale è quella di ridurre progressivamente la dimensione spaziale della rappresentazione in modo da poter avere un numero inferiore di parametri e conseguentemente abbattere i costi di computazione. L'algoritmo utilizzato in questo tipo di layer riduce l'input di un fattore N*N, dove N è la dimensionalità del filtro utilizzato. Le due varianti più utilizzate sono *Max Pooling* e *Average Pooling* che rispettivamente scelgono l'elemento massimo o l'elemento medio come rappresentazione di una porzione N*N dell'immagine di input [41].

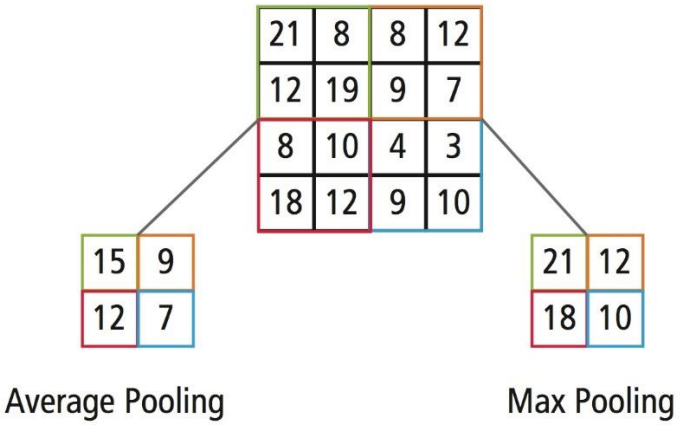

Figura 13 – Max Pooling e Average Pooling

- *Fully connected layer:* layer in cui i neuroni sono tutti interconnessi alle funzioni di attivazione del layer precedente, come nelle classiche reti neurali. L'output tipico di questo layer è un vettore 1x1xK dove K è il numero di neuroni di cui è composto il layer. Tutte le informazioni raccolte dalla rete fino ad ora vengono espresse attraverso un singolo numero utile per la classificazione finale. Tipicamente vengono posti più fully connected layer in serie, con dimensionalità decrescente, fino ad arrivare all'ultimo che fungerà da *output layer*. L'ultimo layer è caratterizzato da un numero di neuroni K pari al numero delle classi presenti nel dataset [41].

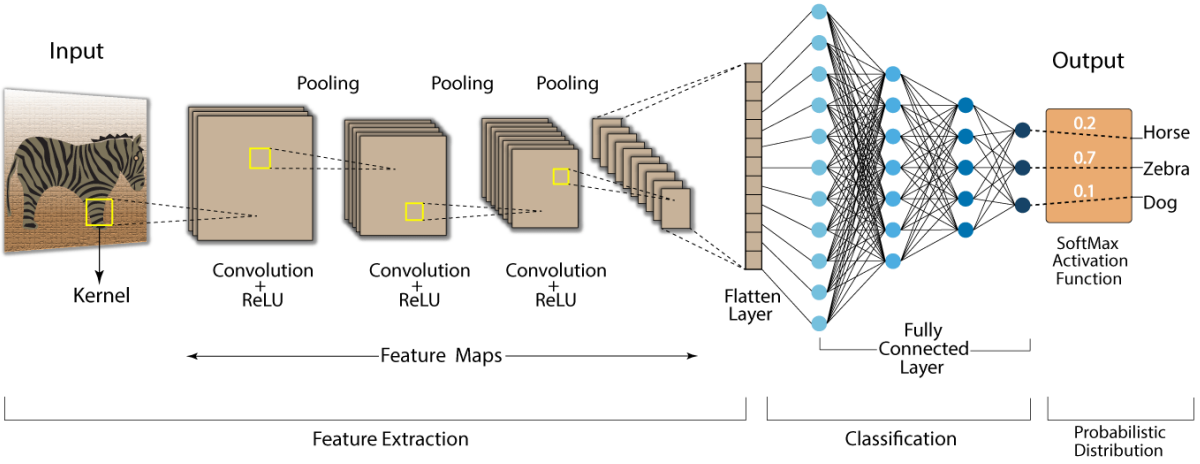

Figura 14 – Esempio Convolutional Neural Network

2.4.4 Image pre-processing

I primi layer di una CNN, o più in generale di una rete neurale, si occupano di eseguire l'estrazione di features dall'immagine in input. Si definisce *descrittore di features* una rappresentazione delle features più *descrittive* di un'immagine, il quale può essere utilizzato per eseguire classificazione. Un buon descrittore è compatto e riesce a rappresentare tutte le informazioni salienti, tralasciando quelle non significative. Uno dei descrittori più utilizzati in computer vision e image processing è l'*istogramma dei gradienti orientati* (HOG). L'idea di base di tale descrittore è che i *bordi* di ogni forma contenuta in un immagine possono essere descritti in funzione del gradiente di ogni pixel [43].

Il gradiente è una funzione matematica che consente di misurare la *direzione* e l'*ampiezza* di una variazione nei valori dei pixel. Può essere calcolato andando a considerare le derivate parziali del pixel rispetto a x e y [44].

$$\nabla F(w) = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}\right]$$

La direzione del gradiente rappresenta la direzione con cambiamento più veloce di intensità dei pixel [44]:

$$\theta = tan^{-1}\left[\frac{\partial F}{\partial x} / \frac{\partial F}{\partial y}\right]$$

L'ampiezza del gradiente rappresenta l'intensità del bordo [44]:

$$\|\nabla F(w)\| = \sqrt{\left(\frac{\partial F}{\partial x}\right)^2 + \left(\frac{\partial F}{\partial y}\right)^2}$$

L'HOG viene estratto da un immagine attraverso i seguenti step:

1. Per ogni pixel viene calcolato il gradient descriptor, utilizzando dei kernel. Attraverso tali kernel si va a calcolare la correlazione tra ogni pixel ed i pixel adiacenti.
2. Attraverso un ulteriore filtro viene calcolato il vettore gradiente, suddividendo l'immagine in sub-immagini.
3. Viene calcolato l'istogramma relativo alle ampiezze.
4. Viene calcolato l'istogramma relativo alle direzioni.

Infine, i due istogrammi vengono concatenati per ottenere l'HOG.

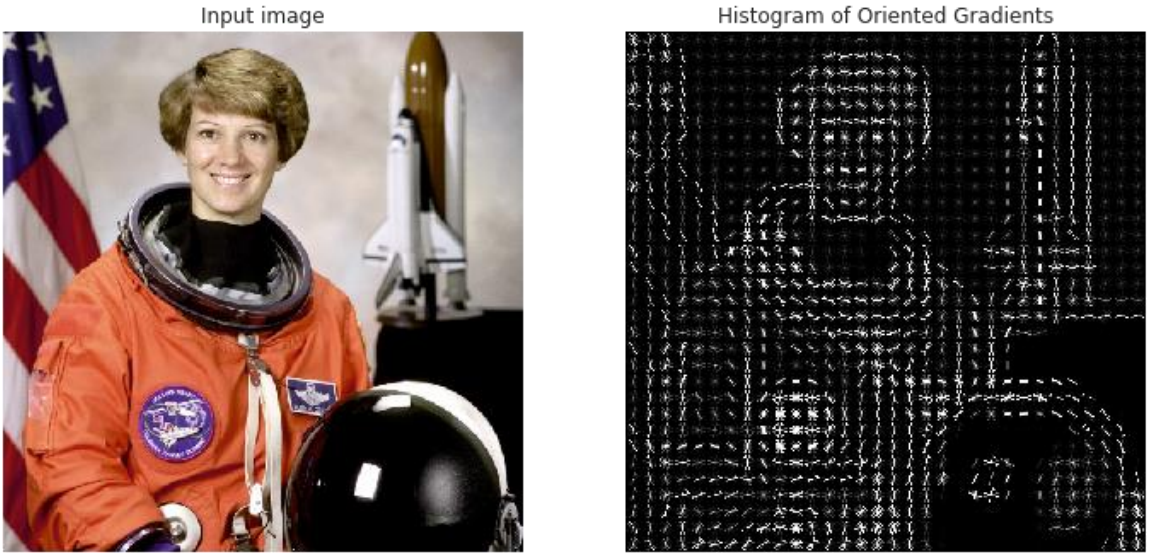

Figura 15 – Esempio di istogramma dei gradienti orientati

2.4 Tecniche di valutazione e di validazione

Sono numerose le metriche utilizzate per valutare la bontà e l'affidabilità di un classificatore, alcune delle più importanti sono *accuracy*, *precision*, *recall* ed F1, calcolabili a partire dalla *matrice di confusione* [41].

2.4.1 Matrice di confusione

Prendendo come esempio un problema di classificazione binaria:

$$Confusion\ matrix\ = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

TP e TN, sulla diagonale principale, indicano rispettivamente il numero di positivi e di negativi classificati correttamente; FT e TN, sulla diagonale secondaria, indicano rispettivamente il numero di positivi e di negativi classificati in modo non corretto [41].

Partendo dalla matrice di confusione è possibile ricavare le varie metriche:

- $Accuracy\ = \frac{TP + TN}{TP + TN + FP + FN}$ indica la percentuale di classificazioni eseguite correttamente rispetto al numero totale delle classificazioni.

- $Precision\ = \frac{TP + TN}{TP + FP}$ fornisce indicazioni circa la capacità del classificatore di non produrre falsi negativi.

- $Recall\ = \frac{TP}{TP+FN}$ fornisce indicazioni circa la capacità del classificatore di individuare tutti i campioni positivi.

- $F1\ = 2 * \frac{Precision * Recall}{Precision + Recall}$ media armonica ponderata di precision e recall, è un indicatore generale delle prestazioni del classificatore.

2.4.2 Curva ROC

La curva *ROC* (Receiver Operating Characteristics) è una metodologia ampiamente utilizzata in ambito medico, e non solo, per poter confrontare i risultati ottenuti da un test diagnostico. La curva ROC viene calcolata su due assi: *sensibilità* e *specificità*, rispettivamente la probabilità che un soggetto malato risulti test-positivo e la probabilità che un soggetto sano risulti test-positivo. In altre parole, andando a valutare la funzione

ROC si va a studiare il rapporto tra “allarmi veri” e “falsi allarmi” [45].

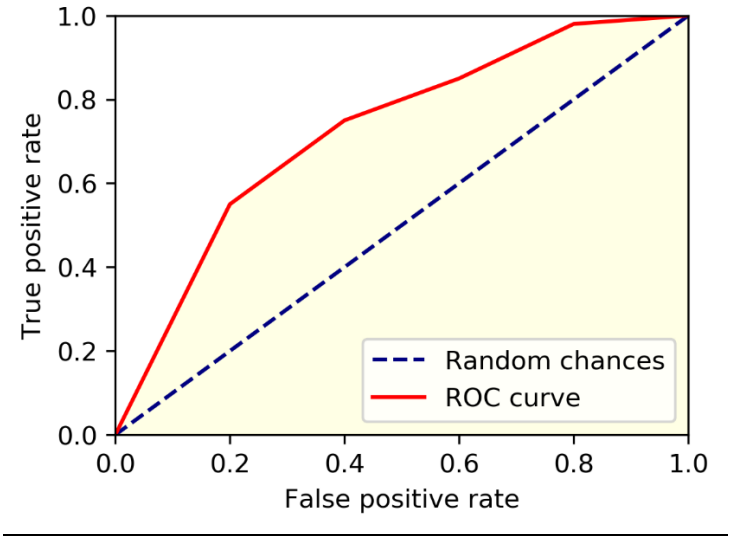

Figura 16 – Esempio di curva ROC

La capacità discriminante di un classificatore, ovvero la sua attitudine a separare propriamente la popolazione in studio in “malati” e “sani”, è proporzionale all’area sottesa alla curva ROC. L’AUC (Area Under Curve) equivale alla probabilità che il risultato di un test su individuo estratto a caso dal gruppo dei malati sia superiore a quelli di uno estratto a caso dal gruppo dei non malati. Un classificatore perfetto ha AUC pari ad 1 [45].

2.4.3 Tecniche di validazione

Le metriche presentate in precedenza, calcolate sull’insieme di training, perdono significato in quanto si andrebbe a testare un classificatore proprio sui dati con cui lo si è addestrato. Esistono diversi approcci per poter valutare le metriche su dati mai sperimentati dal classificatore [46]:

- *holdout:* approccio più frequentemente utilizzato, si dividono i dati tipicamente in 2/3 training set e 1/3 testing set in modo da poter utilizzare quest’ultimo per la valutazione delle metriche.
- *K-Cross-validation:* il training set viene diviso in K sottoinsiemi di uguale dimensione ed esclusivi tra loro. Vengono eseguite K iterazioni in cui un insieme viene utilizzato per la valutazione e i restanti K-1 per il training. Alla fine, viene calcolata la media sulle K iterazioni delle metriche.
- *Leave-one-out-validation*: variante della cross validation in cui viene scelto K pari al numero di istanze a disposizione, tra le tecniche esposte è sicuramente la più dispendiosa.

Capitolo 3: Caso di studio

3.1 Introduzione

L'obiettivo del caso di studio è quello di valutare e classificare la malattia di Parkinson applicando tecniche di Machine Learning a Spiral Test.

Gli Spiral Test sono dei test diagnostici in cui viene richiesto al paziente di disegnare delle spirali archimedee seguendo una traccia. Esistono due diverse tipologie di test:

- Static Spiral Test, in cui la traccia da seguire è fissa.
- Dynamic Spiral Test, in cui la traccia da seguire compare e scompare con periodo regolare.

Tali test sono in grado di valutare la presenza o meno di sintomi motori caratteristici, basandosi sulla forma della spirale disegnata dal paziente. Possono essere eseguiti su carta, oppure attraverso supporto digitale, il quale consente di ottenere informazioni più dettagliate sul disegno.

Nel lavoro proposto vengono utilizzati due diversi dataset, contenenti rispettivamente immagini di test eseguiti su supporto cartaceo e dati numerici di test eseguiti attraverso supporto digitale, in entrambi i casi i test sono stati eseguiti seguendo una traccia, digitale o realizzata interponendo la traccia tra il supporto cartaceo ed una fonte luminosa.

Una volta pre-elaborati i dati di entrambi i dataset verrà eseguita una classificazione binaria sugli Spiral Test in modo da poter distinguere pazienti affetti da malattia di Parkinson e pazienti sani; L'obiettivo è quello di valutare quali siano gli algoritmi di classificazione più efficaci per ogni dataset.

3.2 Descrizione e pre-processing dei Dataset

I dataset utilizzati sono due, in particolare:

- Parkinson Disease Spiral Drawings Using Digitized Graphics Tablet Data Set fornito da UCI, Center for Machine Learning and Intelligent Systems [31]
- Parkinson Spiral Test Drawings fornito dal NIATS of Federal University of Uberlândia.

I due dataset sono tra loro profondamente diversi, il primo presenta dati numerici descrittivi del test punto per punto, in quanto è stato eseguito su supporto digitale. Il secondo è composto da immagini di spiral test eseguiti su supporto cartaceo. [32]

3.2.1 DataSet UCI

Il DataSet UCI contiene dati di spiral test statici e dinamici eseguiti su tablet. Tale dataset è composto da tre directories:

- *hw_dataset*: contiene dati di spiral test di 25 persone malate e 15 sane.
- *hw_drawings*: contiene le immagini dei test dei 25 malati .
- *new_dataset*: contiene dati di spiral test di 37 malati.

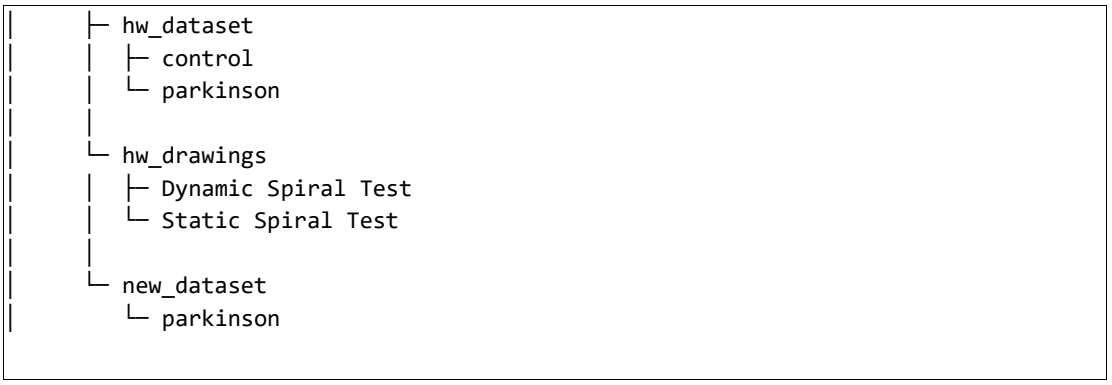

Figura 17 – Struttura del dataset UCI

Le immagini degli spiral test di questo dataset non sono state utilizzate, la directory hw_drawings contiene solo immagini corrispondenti ai test di pazienti malati ed è quindi incompleta rispetto a hw_dataset. Per questo motivo non verrà presa in considerazione.

Il dataset new_dataset/parkinson è stato integrato in hw_dataset/parkinson. In questo modo il numero di istanze contenute in hw_dataset è aumentato a 77: 62 malati e 15 sani.

Ogni test è rappresentato da un file di testo in formato CSV contenente i dati del test statico e del test dinamico. In ogni file sono descritti tutti i punti dei due test (statico e dinamico), aventi i seguenti attributi:

- *X*: posizione del punto rispetto all'asse x.
- *Y*: posizione del punto rispetto all'asse y.
- *Z*: posizione del punto rispetto all'asse z.
- *Pressure*: pressione esercitata in quel punto.
- *GripAngle*: angolo con cui è mantenuta la penna.
- *Timestamp*: tempo relativo in cui è stato registrato il punto.
- Test *ID*: 0 se static test, 1 se dynamic test.

In Figura 18 è riportato un estratto del dataset UCI.

X	Y	Z	Pressure	GripAngle	TimeStamp	TestID
201	200	0	20	880	68140147	0
201	200	0	86	900	68140154	0
201	200	0	123	900	68140161	0
201	200	0	146	900	68140168	0
201	200	0	158	900	68140175	0
201	200	0	159	900	68140182	0
201	199	0	161	880	68140189	0
201	199	0	163	880	68140196	0
201	199	0	165	900	68140203	0
200	199	0	188	900	68140210	0
200	199	0	217	900	68140217	0
200	199	0	244	900	68140224	0

Figura 18- Estratto Dataset UCI

Questi ultimi sono stati utilizzati per ricavare nuovi attributi descrittivi dei test, attraverso scripting Python:

- *no_strokes_st*: numero di variazioni di pressione registrati nel test statico.
- *no_strokes_dy*: numero di variazioni di pressione registrati nel test dinamico.
- *speed_st*: velocità media test statico.
- *speed_dy*: velocità media test dinamico.
- *magnitude_vel_st*: ampiezza del vettore velocità, test statico.
- *magnitude_horz_vel_st*: ampiezza del vettore velocità lungo x, test statico.
- *magnitude_vert_vel_st*: ampiezza del vettore velocità lungo y, test statico.
- *magnitude_vel_dy*: ampiezza del vettore velocità, test dinamico.
- *magnitude_horz_vel_dy* : ampiezza del vettore velocità lungo x, test dinamico.
- *magnitude_vert_vel_dy*: ampiezza del vettore velocità lungo y, test dinamico.
- *magnitude_acc_st*: ampiezza del vettore accelerazione, test statico.
- *magnitude_horz_acc_st*: ampiezza del vettore accelerazione lungo x, test statico.
- *magnitude_vert_acc_st*: ampiezza del vettore accelerazione lungo y, test statico.
- *magnitude_acc_dy*: ampiezza del vettore accelerazione, test dinamico.
- *magnitude_horz_acc_dy*: ampiezza del vettore accelerazione lungo x, test dinamico.
- *magnitude_vert_acc_dy*: ampiezza del vettore accelerazione lungo y, test dinamico.
- *magnitude_jerk_st*: ampiezza del vettore jerk (variazioni di accelerazione), test statico.
- *magnitude_horz_jerk_st*: ampiezza del vettore jerk lungo x, test statico.
- *magnitude_vert_jerk_st*: ampiezza del vettore jerk lungo y, test statico.

- *magnitude_jerk_dy*: ampiezza del vettore jerk, test dinamico.
- *magnitude_horz_jerk_dy*: ampiezza del vettore jerk lungo x, test dinamico.
- *magnitude_vert_jerk_dy*: ampiezza del vettore jerk lungo y, test dinamico.
- *ncv_st*: numero di variazioni di velocità, test statico.
- *ncv_dy*: numero di variazioni di velocità, test dinamico.
- *nca_st*: numero di variazioni di accelerazione, test statico.
- *nca_dy*: numero di variazioni di accelerazione, test dinamico.
- *on_surface_st*: tempo passato in contatto con il supporto, test statico.
- *on_surface_dy*: tempo passato in contatto con il supporto, test dinamico.
- *target*: 1 se il test è di un paziente con Parkinson, 0 altrimenti.

  Il numero di strokes in un test, che sia statico o dinamico, si può calcolare andando a studiare le variazioni di pressione tra un punto ed il successivo (valutando solo i punti in cui si è a contatto con il supporto, quindi con pressione > 600):

$$\sum \boldsymbol{p(t)} \; ! = \boldsymbol{p(t+1)} \quad \forall \, \boldsymbol{p(t) > 600}, \qquad con\; p = Pressure$$

  L'implementazione di tale funzione è realizzata in Figura 19.

  La velocità media in un test, statico o dinamico, si può calcolare andando a valutare il rapporto tra la distanza totale percorsa e il tempo totale passato:

$$\boldsymbol{V} = \frac{\sum \boldsymbol{S(t+1)} - \boldsymbol{S(t)}}{\sum \boldsymbol{T(t+1)} - \boldsymbol{T(t)}} \quad \forall t,$$

  L'implementazione di tale funzione è realizzata in Figura 25.

  Il calcolo della velocità in un test, statico o dinamico, si può eseguire andando a calcolare prima il vettore velocità, per poi decomporlo nelle sue componenti:

  Vettore velocità:

$$\vec{\boldsymbol{V}} = \sum \frac{\boldsymbol{S(t+10)} - \boldsymbol{S(t)}}{\boldsymbol{T(t+10)} - \boldsymbol{T(t)}} \;\; \forall \; t, \qquad con\; S = posizione(x, y, z)\; e\; T = Timestamp$$

  L'implementazione di tale funzione è realizzata in Figura 21.

Vettore velocità orizzontale:

$$\boldsymbol{V_x} = \sum \frac{\boldsymbol{S_x(t+10) - S_x(t)}}{\boldsymbol{T(t+10) - T(t)}} \ \forall\ t, \qquad con\ S_x = posizione(x)\ e\ t = Timestamp$$

Vettore velocità verticale:

$$\boldsymbol{V_y} = \sum \frac{\boldsymbol{S_y(t+10) - S_y(t)}}{\boldsymbol{T(t+10) - T(t)}} \ \forall\ t, \qquad con\ S_y = posizione(y)\ e\ t = Timestamp$$

Modulo del vettore velocità:

$$|\boldsymbol{V}| = \sqrt{\boldsymbol{V_x}^2 + \boldsymbol{V_y}^2}$$

Modulo del vettore velocità orizzontale:

$$|\boldsymbol{V_x}| = \sqrt{\boldsymbol{V_x}^2}$$

Modulo del vettore velocità verticale:

$$|\boldsymbol{V_y}| = \sqrt{\boldsymbol{V_y}^2}$$

Il calcolo dell'accelerazione in un test, statico o dinamico, si può eseguire andando a calcolare il vettore accelerazione, per poi decomporlo:

Vettore accelerazione:

$$\boldsymbol{A} = \sum \frac{\boldsymbol{V(t+10) - V(t)}}{\boldsymbol{T(t+10) - T(t)}} \ \forall\ t, \qquad con\ V = Velocità(x, y, z)\ e\ T = Timestamp$$

L'implementazione di tale funzione è realizzata in Figura 19.

Vettore accelerazione orizzontale:

$$\boldsymbol{A_x} = \sum \frac{\boldsymbol{V_x(t+10) - V_x(t)}}{\boldsymbol{T(t+10) - T(t)}} \;\; \forall \; t, \qquad con \; V_x = Velocità(x) \; e \; T = Timestamp$$

Vettore accelerazione verticale:

$$\boldsymbol{A_y} = \sum \frac{\boldsymbol{V_y(t+10) - V_y(t)}}{\boldsymbol{T(t+10) - T(t)}} \;\; \forall \; t, \qquad con \; V_y = Velocità(y) \; e \; T = Timestamp$$

Modulo del vettore accelerazione:

$$|\boldsymbol{A}| = \sqrt{\boldsymbol{A_x}^2 + \boldsymbol{A_y}^2}$$

Modulo del vettore accelerazione orizzontale:

$$|\boldsymbol{A_x}| = \sqrt{\boldsymbol{A_x}^2}$$

Modulo del vettore accelerazione verticale:

$$|\boldsymbol{A_y}| = \sqrt{\boldsymbol{A_y}^2}$$

Il calcolo delle variazioni di accelerazione (Jerk) in un test, statico o dinamico, si può eseguire andando a calcolare il vettore Jerk, per poi decomporlo nelle sue componenti:

Vettore Jerk:

$$\boldsymbol{J} = \sum \frac{\boldsymbol{A(t+10) - A(t)}}{\boldsymbol{T(t+10) - T(t)}} \;\; \forall \; \boldsymbol{t}, \qquad con \; A = Accelerazione \; (x, y, z) \; e \; T = Timestamp$$

L'implementazione di tale funzione è realizzata in Figura 21.

Vettore Jerk orizzontale:

$$\boldsymbol{J_x} = \sum \frac{\boldsymbol{A_x(t+10) - A_x(t)}}{\boldsymbol{T(t+10) - T(t)}} \ \forall \ \boldsymbol{t}, \quad con \ V_x = Accelerazione \ (x) \ e \ T = Timestamp$$

Vettore Jerk verticale:

$$\boldsymbol{J_y} = \sum \frac{\boldsymbol{A_y(t+10) - A_y(t)}}{\boldsymbol{T(t+10) - T(t)}} \ \forall \ \boldsymbol{t}, \quad con \ A_y = Accelerazione \ (y) \ e \ T = Timestamp$$

Modulo del vettore Jerk:

$$|\boldsymbol{J}| = \sqrt{\boldsymbol{J_x}^2 + \boldsymbol{J_y}^2}$$

Modulo del vettore Jerk orizzontale:

$$|\boldsymbol{J_x}| = \sqrt{\boldsymbol{J_x}^2}$$

Modulo del vettore Jerk verticale:

$$|\boldsymbol{J_y}| = \sqrt{\boldsymbol{J_y}^2}$$

Il numero di variazioni di velocità in un test, statico o dinamico, si può calcolare in questo modo:

$$\sum \boldsymbol{V(t)} \ ! = \boldsymbol{V(t+1)} \ \ \forall \boldsymbol{t}, \quad con \ V = Velocità$$

L'implementazione di tale funzione è realizzata in Figura 24.

Il numero di variazioni di accelerazione in un test, statico o dinamico, si può calcolare in questo modo:

$$\sum \boldsymbol{A(t)} \ ! = \boldsymbol{A(t+1)} \ \ \forall \boldsymbol{t}, \quad con \ A = Accelerazione$$

L'implementazione di tale funzione è realizzata in Figura 23.

Il tempo totale passato in contatto con il supporto può essere calcolato come:

$$\sum \boldsymbol{p(t) > 600} \quad \forall \boldsymbol{t}$$

L'implementazione di tale funzione è realizzata in Figura 27.

Il tempo totale non passato in contatto con il supporto può essere calcolato come:

$$\sum \boldsymbol{p(t) < 600} \quad \forall t$$

L'implementazione di tale funzione è realizzata in Figura 26.

I nuovi attributi sono stati estratti attraverso lo script data_extraction.py, nel quale sono definite le seguenti funzioni che realizzano le funzioni matematiche descritte in precedenza:

```
def get_no_strokes(df):
    pressure_data = df['Pressure'].to_numpy()
    on_surface = (pressure_data > 600).astype(int)
    return ((np.roll(on_surface, 1) - on_surface) != 0).astype(int).sum()
```

Figura 19 – Funzione per il calcolo del numero di strokes

```
def find_acceleration(f):

    Vel, magnitude, timestamp_diff, horz_Vel, vert_Vel, magnitude_vel, magni-
tude_horz_vel, magnitude_vert_vel = find_velocity(
        f )
    accl = []
    horz_Accl = []
    vert_Accl = []
    magnitude = []
    horz_acc_mag = []
    vert_acc_mag = []
    for i in range( len( Vel ) - 2 ):
        accl.append(
            ((Vel[i + 1][0] - Vel[i][0]) / timestamp_diff[i], (Vel[i + 1][1] -
Vel[i][1]) / timestamp_diff[i]) )
        horz_Accl.append( (horz_Vel[i + 1] - horz_Vel[i]) / timestamp_diff[i] )
        vert_Accl.append( (vert_Vel[i + 1] - vert_Vel[i]) / timestamp_diff[i] )
        horz_acc_mag.append( abs( horz_Accl[len( horz_Accl ) - 1] ) )
        vert_acc_mag.append( abs( vert_Accl[len( vert_Accl ) - 1] ) )
        magnitude.append( sqrt( ((Vel[i + 1][0] - Vel[i][0]) /
timestamp_diff[i]) ** 2 + (
                (Vel[i + 1][1] - Vel[i][1]) / timestamp_diff[i]) ** 2 ) )

    magnitude_acc = np.mean( magnitude )
    magnitude_horz_acc = np.mean( horz_acc_mag )
    magnitude_vert_acc = np.mean( vert_acc_mag )
    return accl, magnitude, horz_Accl, vert_Accl, timestamp_diff, magnitude_acc,
magnitude_horz_acc, magnitude_vert_acc
```

Figura 20 – Funzione per il calcolo dell'accelerazione

```
def find_velocity(f):
    data_pat = f
    Vel = []
    horz_Vel = []
    horz_vel_mag = []
    vert_vel_mag = []
    vert_Vel = []
    magnitude = []
    timestamp_diff = []

    t = 0
    for i in range( len( data_pat ) - 2 ):

        if t + 10 <= len( data_pat ) - 1:
            Vel.append( ((data_pat['X'].to_numpy()[t + 10] -
data_pat['X'].to_numpy()[t]) / (
                        data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[t]),
                            (data_pat['Y'].to_numpy()[t + 10] -
data_pat['Y'].to_numpy()[t]) / (
                                    data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[
                                t])) )
            horz_Vel.append( (data_pat['X'].to_numpy()[t + 10] -
data_pat['X'].to_numpy()[t]) / (
                        data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[t]) )

            vert_Vel.append( (data_pat['Y'].to_numpy()[t + 10] -
data_pat['Y'].to_numpy()[t]) / (
                        data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[t]) )

            magnitude.append( sqrt( ((data_pat['X'].to_numpy()[t + 10] -
data_pat['X'].to_numpy()[t]) / (
                        data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[t])) ** 2 + (((
                        data_pat['Y'].to_numpy()[t + 10] -
data_pat['Y'].to_numpy()[t]) /
                         (data_pat['Timestamp'].to_numpy()[t + 10] -data_pat[
'Timestamp'].to_numpy()[t])) ** 2) ) )

            timestamp_diff.append( data_pat['Timestamp'].to_numpy()[t + 10] -
data_pat['Timestamp'].to_numpy()[t] )

            horz_vel_mag.append( abs( horz_Vel[len( horz_Vel ) - 1] ) )
            vert_vel_mag.append( abs( vert_Vel[len( vert_Vel ) - 1] ) )
            t = t + 10
        else:
            break
    magnitude_vel = np.mean( magnitude )
    magnitude_horz_vel = np.mean( horz_vel_mag )
    magnitude_vert_vel = np.mean( vert_vel_mag )


return Vel, magnitude, timestamp_diff, horz_Vel, vert_Vel, magnitude_vel, magni-
tude horz vel, magnitude vert vel
```

Figura 21 – Funzione per il calcolo del vettore velocità

```
def find_jerk(f):

accl, magnitude, horz_Accl, vert_Accl, timestamp_diff, magnitude_acc, magni-
tude_horz_acc, magnitude_vert_acc = find_acceleration(
        f )
    jerk = []
    hrz_jerk = []
    vert_jerk = []
    magnitude = []
    horz_jerk_mag = []
    vert_jerk_mag = []

    for i in range( len( accl ) - 2 ):
        jerk.append(
            ((accl[i + 1][0] - accl[i][0]) / timestamp_diff[i],
            (accl[i + 1][1] - accl[i][1]) / timestamp_diff[i]) )
              hrz_jerk.append( (horz_Accl[i + 1] - horz_Accl[i]) /
      timestamp_diff[i] )
              vert_jerk.append( (vert_Accl[i + 1] - vert_Accl[i]) /
      timestamp_diff[i] )
              horz_jerk_mag.append( abs( hrz_jerk[len( hrz_jerk ) - 1] ) )
              vert_jerk_mag.append( abs( vert_jerk[len( vert_jerk ) - 1] ) )
              magnitude.append( sqrt( ((accl[i + 1][0] - accl[i][0]) /
      timestamp_diff[i]) ** 2 + (
                      (accl[i + 1][1] - accl[i][1]) / timestamp_diff[i]) ** 2 )
      )

          magnitude_jerk = np.mean( magnitude )
          magnitude_horz_jerk = np.mean( horz_jerk_mag )
          magnitude_vert_jerk = np.mean( vert_jerk_mag )

return jerk, magnitude, hrz_jerk, vert_jerk, timestamp_diff, magnitude_jerk,
magnitude_horz_jerk, magnitude_vert_jerk
```

Figura 22 – Funzione per il calcolo del vettore jerk

```
def NCA_per_halfcircle(f):
    data_pat = f
    Vel, magnitude, timestamp_diff, horz_Vel, vert_Vel, magnitude_vel, magni-
tude_horz_vel, magnitude_vert_vel = find_velocity(
        f )
    accl = []
    nca = []
    temp_nca = 0
    basex = data_pat['X'].to_numpy()[0]
    for i in range( len( Vel ) - 2 ):
        if data_pat['X'].to_numpy()[i] == basex:
            nca.append( temp_nca )
            temp_nca = 0
            continue

        accl.append(
            ((Vel[i + 1][0] - Vel[i][0]) / timestamp_diff[i], (Vel[i + 1][1] -
Vel[i][1]) / timestamp_diff[i]) )
        if accl[len( accl ) - 1] != (0, 0):
            temp_nca += 1
    nca.append( temp_nca )
    nca = list( filter( (2).__ne__, nca ) )
    nca_Val = np.sum( nca ) / np.count_nonzero( nca )
    return nca, nca_Val
```

Figura 23 – Funzione per il calcolo del numero di variazioni di accelerazioni per semicerchio

```
def NCV_per_halfcircle(f):
    data_pat = f
    Vel = []
    ncv = []
    temp_ncv = 0
    basex = data_pat['X'].to_numpy()[0]
    for i in range( len( data_pat ) - 2 ):
        if data_pat['X'].to_numpy()[i] == basex:
            ncv.append( temp_ncv )
            temp_ncv = 0
            continue

        Vel.append( ((data_pat['X'].to_numpy()[i + 1] -
data_pat['X'].to_numpy()[i]) / (
                data_pat['Timestamp'].to_numpy()[i + 1] - data_pat['Time-
stamp'].to_numpy()[i]),
                    (data_pat['Y'].to_numpy()[i + 1] -
data_pat['Y'].to_numpy()[i]) / (
                        data_pat['Timestamp'].to_numpy()[i + 1] -
data_pat['Timestamp'].to_numpy()[i])) )
        if Vel[len( Vel ) - 1] != (0, 0):
            temp_ncv += 1
    ncv.append( temp_ncv )
    # ncv = list(filter((2).__ne__, ncv))
    ncv_Val = np.sum( ncv ) / np.count_nonzero( ncv )
    return ncv, ncv_Val
```

Figura 24 – Funzione per il calcolo del numero di variazioni di velocità per semicerchio

```
def get_speed(df):
    total_dist = 0
    duration = df['Timestamp'].to_numpy()[-1]
    coords = df[['X', 'Y', 'Z']].to_numpy()
    for i in range( 10, df.shape[0] ):
        temp = np.linalg.norm( coords[i, :] - coords[i - 10, :] )
        total_dist += temp
    speed = total_dist / duration
    return speed
```

Figura 25 – Funzione per il calcolo della velocità istantanea in un punto

```
def get_in_air_time(data):
    data = data['Pressure'].to_numpy()
    return (data < 600).astype( int ).sum()
```

Figura 26 – Funzione per il calcolo del tempo passato non in contatto con il tablet

```
def get_on_surface_time(data):
    data = data['Pressure'].to_numpy()
    return (data > 600).astype( int ).sum()
```

Figura 27 – Funzione per il calcolo del tempo passato in contatto con il tablet

```
def get_features(f, parkinson_target):
    global header_row
    df = pd.read_csv( f, sep=';', header=None, names=header_row )
    df_static = df[df["Test_ID"] == 0]  # static test
    df_dynamic = df[df["Test_ID"] == 1]  # dynamic test
    initial_timestamp = df['Timestamp'][0]
    df['Timestamp'] = df['Timestamp'] - initial_timestamp
    data_point = []

    data_point.append( get_no_strokes( df_static ) if df_static.shape[0] else 0 )
    data_point.append( get_no_strokes( df_dynamic ) if df_dynamic.shape[0] else 0 )

    data_point.append( get_speed( df_static ) if df_static.shape[0] else 0 )
    data_point.append( get_speed( df_dynamic ) if df_dynamic.shape[0] else 0 )

    Vel, magnitude, timestamp_diff, horz_Vel, vert_Vel, magnitude_vel, magni-
tude_horz_vel, magnitude_vert_vel = find_velocity(
        df ) if df_static.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_vel, magnitude_horz_vel, magnitude_vert_vel] )
    Vel, magnitude, timestamp_diff, horz_Vel, vert_Vel, magnitude_vel, magni-
tude_horz_vel, magnitude_vert_vel = find_velocity(
        df_dynamic ) if df_dynamic.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_vel, magnitude_horz_vel, magnitude_vert_vel] )

    accl, magnitude, horz_Accl, vert_Accl, timestamp_diff, magnitude_acc, magni-
tude_horz_acc, magnitude_vert_acc = find_acceleration(
        df_static ) if df_static.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_acc, magnitude_horz_acc, magnitude_vert_acc] )
    accl, magnitude, horz_Accl, vert_Accl, timestamp_diff, magnitude_acc, magni-
tude_horz_acc, magnitude_vert_acc = find_acceleration(
        df_dynamic ) if df_dynamic.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_acc, magnitude_horz_acc, magnitude_vert_acc] )

    jerk, magnitude, hrz_jerk, vert_jerk, timestamp_diff, magnitude_jerk, magni-
tude_horz_jerk, magnitude_vert_jerk = find_jerk(
        df_static ) if df_static.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_jerk, magnitude_horz_jerk, magnitude_vert_jerk] )
    jerk, magnitude, hrz_jerk, vert_jerk, timestamp_diff, magnitude_jerk, magni
tude_horz_jerk, magnitude_vert_jerk = find_jerk(
        df_dynamic ) if df_dynamic.shape[0] else (0, 0, 0, 0, 0, 0, 0, 0)
    data_point.extend( [magnitude_jerk, magnitude_horz_jerk, magnitude_vert_jerk] )

    ncv, ncv_Val = NCV_per_halfcircle( df_static ) if df_static.shape[0] else (0, 0)
    data_point.append( ncv_Val )
    ncv, ncv_Val = NCV_per_halfcircle( df_dynamic ) if df_dynamic.shape[0] else (0, 0)
    data_point.append( ncv_Val )
    nca, nca_Val = NCA_per_halfcircle( df_static ) if df_static.shape[0] else (0, 0)
    data_point.append( nca_Val )
    nca, nca_Val = NCA_per_halfcircle( df_dynamic ) if df_dynamic.shape[0] else (0, 0)
    data_point.append( nca_Val )
    data_point.append( get_on_surface_time( df_static ) if df_static.shape[0] else 0 )
    data_point.append( get_on_surface_time( df_dynamic ) if df_dynamic.shape[0] else 0 )
    data_point.append( parkinson_target )
    return data_point
```

Figura 28 – Funzione per l'estrazione di tutti gli attributi da spiral test del dataset UCI

Nel main dello script data_extraction.py viene chiamata la funzione get_features, per ogni file della directories /parkinson e /control, passando come parametro target rispettivamente 1 e 0.
Prima di salvare il nuovo dataset ottenuto è necessaria un'operazione di data cleaning. Andando ad eseguire un'analisi dei dati manuale (possibile in quanto le istanze sono solo 78) si può notare come le istanze 13-14-15-16-17-18 presentano alcuni valori a 0 per attributi in cui lo 0 non è ammesso. Questa è stata l'unica operazione di data cleaning effettuata in quanto il resto dei dati estratti risulta valido.

```
def data_cleaning(df):
    features = list( df.columns.values )
    for n in [13, 14, 15, 16, 17, 18]:
        for f in features:
            if int( df.loc[n][[f]] ) == 0 and str( f ) != 'target' and str( f )
!= 'no_strokes_st' and str( f ) != 'no_strokes_dy':
                df.loc[n][[f]] = df[[f]].mean()
    return df
```

Figura 29 – Funzione di data cleaning per le features estratte dal dataset UCI

A questo punto il dataframe risultante è stato salvato nel file extracted_data.csv, dove sono presenti tutti e 77 i pazienti con i dati estratti dai loro test.

3.2.2 DataSet NIATS

Il DataSet NIATS contiene immagini di spiral test statici eseguiti su supporto cartaceo ed è strutturato come segue:

```
└─ spiral
   ├─ testing
   │  ├─ healthy
   │  └─ parkinson
   │
   └─ training
      ├─ healthy
      └─ parkinson
```

Figura 30 – Struttura del dataset NIATS

Il dataset è già suddiviso in insieme di training e di testing, la ripartizione è circa 70-30 per un totale di 102 istanze:

- testing/healthy: 36 istanze
- testing/parkinson: 36 istanze
- training/healthy: 15 istanze
- training/parkinson: 15 istanze

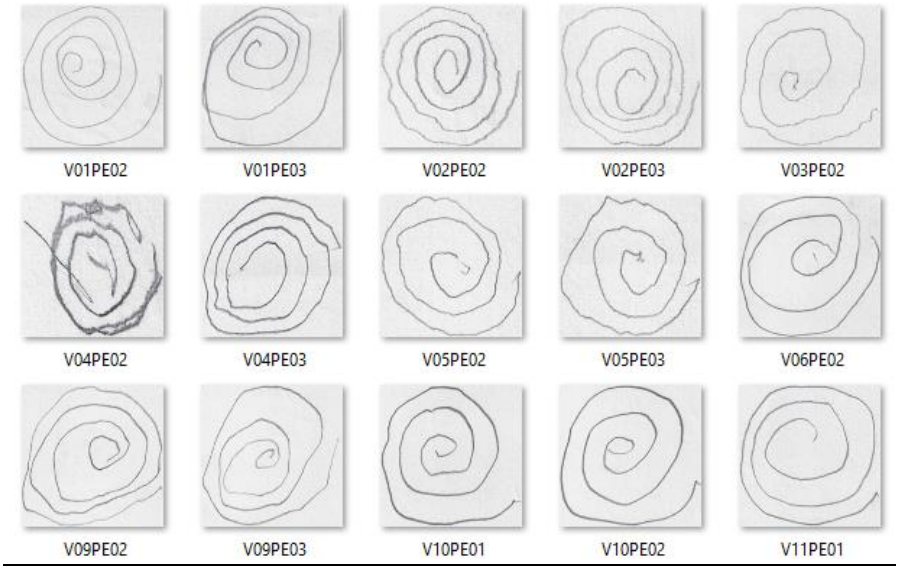

Figura 31 – Estratto dataset NIATS

A differenza del dataset UCI in questo caso sono presenti solo le immagini relative agli spiral test statici, senza nessun dato numerico. I test sono stati realizzati su supporto cartaceo, seguendo una traccia statica posta tra il supporto ed una fonte luminosa.

I dati di questo dataset sono stati processati attraverso tecniche di data augmentation al fine di poter incrementare il numero di istanze a disposizione. Il tutto è stato eseguito attraverso il package *keras.preprocessing*. In particolare, è stato utilizzato lo strumento *ImageDataGenerator*, il quale consente di generare delle copie modificate di un'immagine in input basandosi sulla variazione dei seguenti parametri, impostati come segue:

- Rotation Range = 30°, range massimo di rotazione
- Width shift = 0.1, valore massimo di shift orizzontale
- Height shift = 0.1, valore massimo di shift verticale
- Horizontal flip = True, possibilità di ribaltare orizzontalmente
- Shear range = 0.2, distorsione massima lungo un asse
- Zoom range = 0.2, valore massimo di zoom

In Figura 32 è rappresentata l'implementazione della funzione con cui si è eseguito data augmentation.

```
def data_augmentation(in_path, out_path):
    print( "Data augmentation..." )
    for filename in glob.glob( in_path ):

        image = load_img( filename )
        image = img_to_array( image )
        image = np.expand_dims( image, axis=0 )

        aug = ImageDataGenerator(
            rotation_range=30,
            width_shift_range=0.1,
            height_shift_range=0.1,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode="nearest"
        )

        total = 0

        imageGen = aug.flow( image, batch_size=1, save_to_dir=out_path,
                             save_prefix=uuid.uuid4(), save_format="jpg" )

        for image in imageGen:
            total += 1

            if total == 20:
                break
```

Figura 32 – Funzione di data augmentation

Le immagini generate sono versioni roteate, leggermente shiftate e zoomate dell'immagine iniziale, tali da preservarne le caratteristiche utili ai fini dell'analisi. L'immagine iniziale non verrà utilizzata in modo da evitare problemi di overfitting.
In seguito, è stato ricercato empiricamente il numero di immagini generate che garantisse migliori risultati. Sono stati eseguiti diversi test, andando a valutare l'andamento dell'accuracy di diversi classificatori. Dai test eseguiti (riportati in Figura 33) è emerso come all'aumentare delle immagini generate aumentino, per tutti i classificatori, i risultati di accuracy. Bisogna però porre molta attenzione alle prestazioni, le quali calano drasticamente all'aumentare dei dati generati, va quindi trovato un buon compromesso tra qualità dei risultati e le performance. Tra i vari test eseguiti risulta essere 20 il numero giusto di immagini da generare, si ha infatti un netto miglioramento delle metriche rispetto al dataset originale, mantenendo comunque dei tempi ragionevoli in rapporto alla capacità di elaborazione della macchina utilizzata.

Il dataset generato risulta così suddiviso:

- testing/healthy: 720 istanze
- testing/parkinson: 720 istanze
- training/healthy: 300 istanze
- training/parkinson: 300 istanze

Numero immagini generate	Accuracy	Runtime (min)
Logistic Regression		
1	60%	2
2	46%	3
5	63%	6
10	68%	11
20	69%	26
50	68%	50
100	72%	90
K-Neighbors		
1	50%	2
2	56%	3
5	56%	6
10	59%	11
20	62%	26
50	66%	50
100	70%	90
Decision Tree		
1	53%	2
2	53%	3
5	61%	6
10	60%	11
20	56%	26
50	61%	50
100	61%	90
Random Forest		
1	63%	2
2	60%	3
5	73%	6
10	76%	11
20	75%	26
50	75%	50
100	76%	90
Gradient Boosting		
1	50%	2
2	67%	3
5	71%	6
10	73%	11
20	72%	26
50	74%	50
100	77%	90

Figura 33 – Risultati dataset NIATS per diversi valori di data augmentation

3.3 Analisi e classificazione

L'obiettivo del lavoro di analisi e classificazione è quello di individuare quali siano gli algoritmi di machine learning più adatti per poter classificare pazienti affetti da Parkinson, utilizzando Spiral Test. I due dataset sono tra loro profondamente diversi, è quindi di nostro interesse valutare il funzionamento degli stessi classificatori su dati di natura dissimile.

3.3.1 Classificazione DataSet UCI

Come prima analisi, utilizzando il package *numpy*, è stato calcolato il *coefficiente di correlazione* di ogni attributo con l'attributo target. Valori di correlazione compresi tra -1 e 0 indicano come un attributo influenzi negativamente la presenza del valore 1 per l'attributo target, valori compresi tra 0 e 1 invece indicano come un attributo ne influenzi positivamente la presenza. In questo modo è stato possibile capire quali sono le features che più influenzano la classificazione.

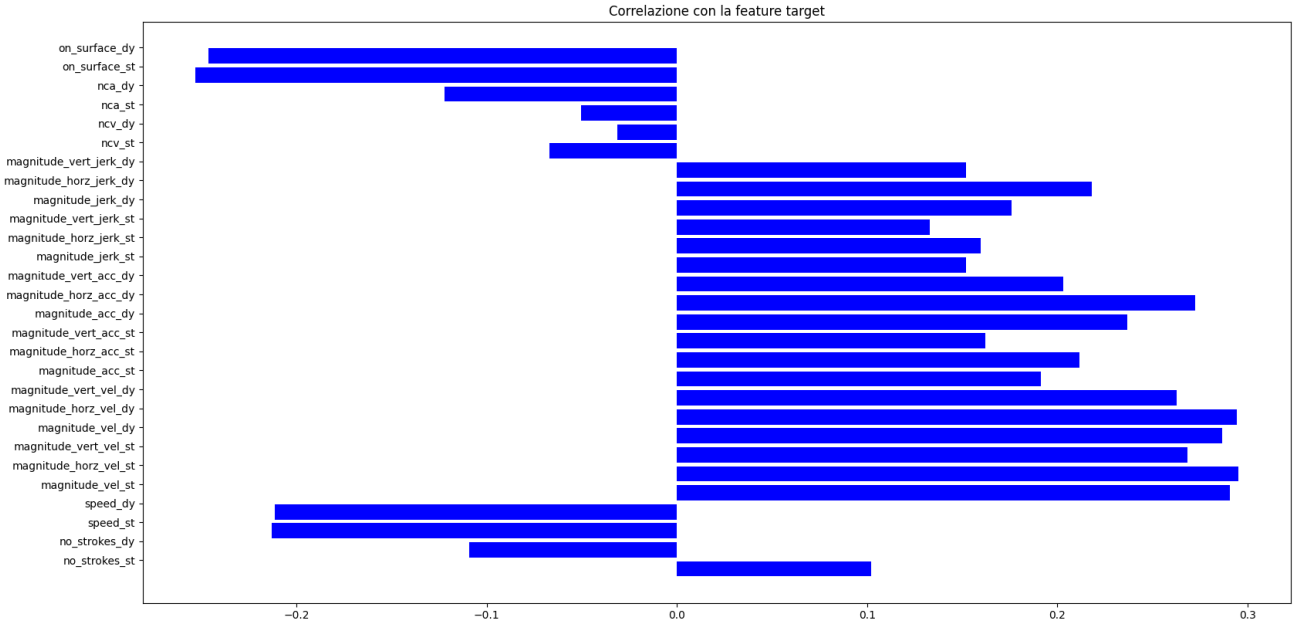

Figura 34 – Correlazione features-target dataset UCI

Gli attributi che presentano correlazione più elevata sono, in ordine decrescente, quelli relativi a *velocità*, *accelerazione* e *jerk* del test. Anche il numero di *strokes* calcolati nel test statico ha correlazione positiva con l'attributo target, a differenza del numero di strokes calcolati nel test dinamico. Questo dato trova corrispondenza nella struttura dei

due test, nel test dinamico è naturale prevedere dei cambi di pressione, dettati dal comparire e scomparire della traccia in background .

In seguito, è stata eseguita la divisione delle 77 istanze presenti nel file *extracted_data.csv* (62 Parkinson, 15 Sani) in training set 75% e testing set 25%, in modo da poter procedere con training, classificazione e validazione.

I classificatori utilizzati sono stati importati dalla libreria *sklearn* e sono i seguenti:

- Logistic Regression
- Random Forest
- Decision Tree
- K-Nearest Neighbours
- Gradient Boosting

Su ognuno dei modelli viene effettuato *training* e successivo *testing,* da cui vengono calcolate:

- Confusion matrix
- Accuracy
- Precision
- Recall
- F1

Attraverso *k-cross validation* (con k = 2, dato il basso numero di istanze) vengono calcolate:

- Confusion matrix
- Accuracy media
- Precision media
- Recall media
- F1 media

Classificatore	Matrice di confusione	Accuracy	Precision	Recall	F1
Logistic Regression	$\begin{bmatrix} 8 & 0 \\ 2 & 29 \end{bmatrix}$	95%	74%	100%	85%
Random Forest	$\begin{bmatrix} 8 & 0 \\ 19 & 12 \end{bmatrix}$	54%	33%	85%	55%
Decision Tree	$\begin{bmatrix} 8 & 0 \\ 0 & 31 \end{bmatrix}$	100%	79%	100%	67%
K-Nearest Neighbors	$\begin{bmatrix} 4 & 4 \\ 14 & 17 \end{bmatrix}$	54%	44%	81%	57%
Gradient Boosting	$\begin{bmatrix} 8 & 0 \\ 0 & 31 \end{bmatrix}$	100%	80%	100%	88%

Figura 35 – risultati testing dataset UCI

Classificatore	Matrice di confusione (k = 2)	Accuracy media (k = 5)	Precision media (k = 5)	Recall media (k = 5)	F1 media (k = 5)
Logistic Regression	$\begin{bmatrix} 18 & 5 \\ 56 & 6 \end{bmatrix}$	87%	83%	84%	84%
Random Forest	$\begin{bmatrix} 18 & 5 \\ 19 & 43 \end{bmatrix}$	71%	75%	70%	65%
Decision Tree	$\begin{bmatrix} 23 & 0 \\ 0 & 62 \end{bmatrix}$	73%	73%	76%	72%
K-Nearest Neighbors	$\begin{bmatrix} 12 & 11 \\ 16 & 46 \end{bmatrix}$	68%	62%	63%	62%
Gradient Boosting	$\begin{bmatrix} 8 & 0 \\ 0 & 31 \end{bmatrix}$	74%	97%	76%	88%

Figura 36 – risultati cross validation dataset UCI

Dalla tabella riportata in Figura 35 si può notare come i classificatori che presentano migliori risultati di *accuracy* sul testing set sono *Logistic Regression*, *Decision Tree* e *Gradient Boosting*. I risultati di *recall* sono piuttosto alti per tutti i classificatori, la *precision* risulta invece relativamente bassa. Questo dato indica come i classificatori sono capaci di trovare la maggior parte dei positivi del dataset, a discapito però della presenza di molti falsi positivi. Andando a valutare i risultati della k-cross validation, riportati in Figura 36, possiamo notare che il classificatore che presenta i risultati più "bilanciati" è *Logistic Regression*, con valore di *accuracy* pari all'87% e circa dell' 84% per le altre

metriche. I valori di *Decision Tree* e *Gradient Boosting* invece diminuiscono leggermente per tutte le metriche e sono molto simili tra loro. Questo risultato non ci sorprende in quanto il *Gradient Boosting* è un classificatore che utilizza in maniera combinata ed ottimizzata più *Decision Tree*, i risultati sono quindi giustamente molto simili.

3.3.2 Classificazione DataSet NIATS

I dati presenti in questo dataset sono già divisi in training set 70% e testing set 30%. In Figura 37 sono riportate due funzioni *quantify_image* e *load_split*. La prima viene utilizzata per estrarre l'HOG da un'immagine attraverso il metodo *feature.hog* presente nel package *Scikit_image*, la seconda estrae HOG e label da tutte le immagini presenti in una directory, ricavando la label dal nome della directory di appartenenza.

```
def quantify_image(image):

    features = feature.hog( image, orientations=9,
                            pixels_per_cell=(10, 10), cells_per_block=(2, 2),
                            transform_sqrt=True, block_norm="L1" )

    return features



def load_split(path):

    imagePaths = list( paths.list_images( path ) )
    data = []
    labels = []


    for imagePath in imagePaths:

        label = imagePath.split( os.path.sep )[-2]

        image = cv2.imread( imagePath )
        image = cv2.cvtColor( image, cv2.COLOR_BGR2GRAY )
        image = cv2.resize( image, (200, 200) )

        image = cv2.threshold( image, 0, 255,
                               cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU )[1]

        features = quantify_image( image )

        data.append( features )
        labels.append( label )


    return (np.array( data ), np.array( labels ))
```

Figura 37 – funzione per l'estrazione di HOG e label dal dataset NIATS

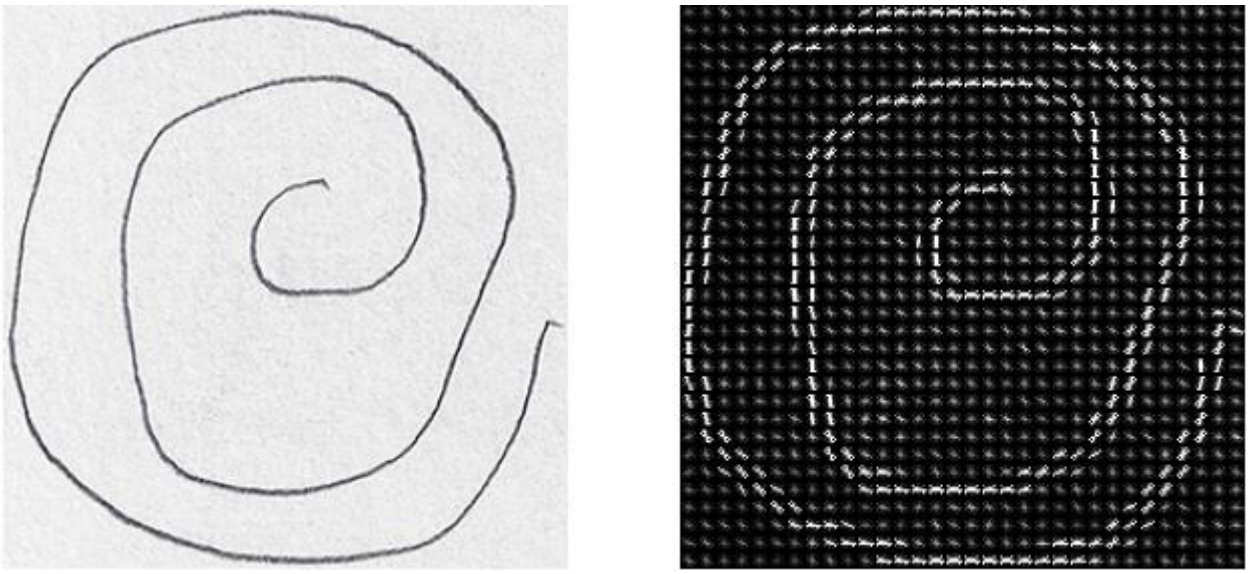

Figura 38 – Immagine in input (a sinistra) HOG estratto (a destra)

A questo punto i dati sono pronti per essere classificati. I classificatori utilizzati sono stati importati dalla libreria *sklearn* e sono i seguenti:

- Logistic Regression
- Random Forest
- Decision Tree
- K-Nearest Neighbours
- Gradient Boosting

Su ognuno dei modelli viene effettuato *training* e successivo *testing*, da cui vengono calcolate:

- Confusion matrix
- Accuracy
- Precision
- Recall
- F1

Attraverso *k-cross validation* (con k = 5) vengono calcolate:

- Confusion matrix
- Accuracy media
- Precision media
- Recall media
- F1 media

Classificatore	Matrice di confusione	Accuracy	Precision	Recall	F1
Logistic Regression	$\begin{bmatrix} 192 & 108 \\ 114 & 186 \end{bmatrix}$	50%	50%	100%	67%
Random Forest	$\begin{bmatrix} 223 & 77 \\ 120 & 180 \end{bmatrix}$	40%	40%	86%	55%
Decision Tree	$\begin{bmatrix} 172 & 128 \\ 123 & 176 \end{bmatrix}$	50%	50%	100%	67%
K-Nearest Neighbors	$\begin{bmatrix} 69 & 231 \\ 21 & 279 \end{bmatrix}$	48%	48%	67%	56%
Gradient Boosting	$\begin{bmatrix} 220 & 80 \\ 108 & 192 \end{bmatrix}$	50%	50%	100%	67%

Figura 39 – Risultati testing dataset NIATS

Classificatore	Matrice di confusione (k =5)	Accuracy media (k = 5)	Precision media (k = 5)	Recall media (k = 5)	F1 media (k = 5)
Logistic Regression	$\begin{bmatrix} 214 & 85 \\ 91 & 209 \end{bmatrix}$	63%	63%	63%	62%
Random Forest	$\begin{bmatrix} 240 & 59 \\ 86 & 214 \end{bmatrix}$	68%	69%	70%	65%
Decision Tree	$\begin{bmatrix} 165 & 134 \\ 98 & 202 \end{bmatrix}$	61%	60%	59%	60%
K-Nearest Neighbors	$\begin{bmatrix} 103 & 196 \\ 17 & 283 \end{bmatrix}$	59%	71%	59%	53%
Gradient Boosting	$\begin{bmatrix} 216 & 83 \\ 78 & 222 \end{bmatrix}$	67%	67%	66%	67%

Figura 40 – Risultati cross validation dataset NIATS

I risultati di testing, rappresentati nella tabella in Figura 39, sono molto scoraggianti, gli algoritmi presentano accuracy massimo del 50%, fornendo quindi la stessa validità di una scelta randomica. I risultati ottenuti attraverso k-cross validation (con k = 5), riportati nella tabella in Figura 40, sono migliori dei precedenti ma restano comunque lontani da quelli ottenuti sul dataset UCI. I classificatori che presentano i migliori risultati sono Logistic Regression, Random Forest e Gradient Boosting, con valori per tutte le metriche di poco inferiori al 70%. Il migliore in assoluto è Random Forest, il quale presenta le migliori accuracy, precision e recall. Possiamo quindi affermare che i classificatori impiegati hanno lavorato molto meglio sul dataset numerico, anche in assenza di grandi quantità di dati.

A questo punto, per poter migliorare i risultati ottenuti sul dataset NIATS sono state realizzate tre reti neurali. Due di esse sono state costituite utilizzando come nucleo convolutivo delle reti pre-addestrate sul dataset Imagenet [33] :

- VGG16: rete neurale convolutiva costituita da 16 layer e 134,268,738 parametri, in input accetta immagini RGB 224x224 e come output restituisce una tra mille classi.
- ResNet50: rete neurale convolutiva costituita da 50 layer e 25,636,712 parametri, in input accetta immagini RGB 224x224 e come output restituisce una tra mille classi.

Entrambe le reti sono state modificate attraverso layer addizionali per poter affrontare al meglio la classificazione binaria. Nelle figure 41 e 42 è riportata l'implementazione delle due reti. Il terzo modello è stato realizzato da zero, utilizzando la libreria keras. L'architettura ottenuta è una rete neurale convoluzionale, composta da 21 layers e avente 1,200,137 parametri addestrabili, l'implementazione è riportata in Figura 43.

```
def ResNet_model():

    base_model = ResNet50(include_top = False, weights = 'imagenet', input_shape
= (256,256, 3))

    for layer in base_model.layers[:-8]:
        layer.trainable = False

    model = Sequential()
    model.add(base_model)
    model.add( Flatten() )
    model.add( Dense( 500 ) )
    model.add( Activation( 'relu' ) )
    model.add( Dropout( 0.5 ) )
    model.add( Dense( 1 ) )
    model.add( Activation( 'sigmoid' ) )


    model.compile(
        loss='binary_crossentropy',
        optimizer=tf.keras.optimizers.SGD( lr=0.001),
        metrics=['accuracy']
    )

    return model
```

Figura 41 – Definizione rete neurale con base convolutiva ResNet50

```
def VGG_model():

    base_model = vgg16.VGG16(weights='imagenet', include_top=False, in-
put_shape=(256, 256, 3) )

    for layer in base_model.layers[:-8]:
        layer.trainable = False

    model = Sequential()
    model.add(base_model)
    model.add( Flatten() )
    model.add( Dense( 500 ) )
    model.add( Activation( 'relu' ) )
    model.add( Dropout( 0.5 ) )
    model.add( Dense( 1 ) )
    model.add( Activation( 'sigmoid' ) )


    model.compile(
        loss='binary_crossentropy',
        optimizer=tf.keras.optimizers.SGD(lr =0.001),
        metrics=['accuracy']
    )

    return model
```

Figura 42 – Definizione rete neurale con base convolutiva VGG16

```
def cnn_model():
    if K.image_data_format() == 'channels_first':
        input_shape = (3, model_params['img_width'], model_params['img_height'])
    else:
        input_shape = (model_params['img_width'], model_params['img_height'], 3)

    model = Sequential()

    for i, num_filters in enumerate( model_params['filters'] ):
        if i == 0:
            model.add( Conv2D( num_filters, (3, 3), input_shape=input_shape ) )
        else:
            model.add( Conv2D( num_filters, (3, 3) ) )
        model.add( Activation( 'relu' ) )
        model.add( MaxPooling2D( pool_size=(2, 2) ) )

    model.add( Flatten() )
    model.add( Dense( 500 ) )
    model.add( Activation( 'relu' ) )
    model.add( Dropout( 0.5 ) )
    model.add( Dense( 1 ) )
    model.add( Activation( 'sigmoid' ) )
    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(),
        metrics=['accuracy']
    )

    return model
```

Figura 43 – Definizione rete neurale custom

In Figura 44 viene riportato il *summary* della rete realizzata sfruttando ResNet. Il modello presenta 89,125,713 parametri di cui 68,124,713 *trainable*. I Layer successivi al primo sono stati aggiunti per poter rendere possibile la classificazione binaria. In particolare, è significativa l'aggiunta dell'ultimo layer Dense(1) con funzione di attivazione Sigmoid, il quale restituisce in output valori compresi tra 0 e 1.

Layer (type)	Output Shape	Trainable params
resnet50 (Functional)	(None, 8, 8, 2048)	23587712
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 500)	65536500
activation (Activation)	(None, 500)	0
dropout (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 1)	501
activation_1 (Activation)	(None, 1)	0

Total params: 89,124,713
Trainable params: 68,952,553
Non-trainable params: 20,172,160

Figura 44 – Struttura della rete neurale realizzata sfruttando ResNet50

In Figura 45 viene riportato il *summary* della rete realizzata sfruttando VGG16. Il modello presenta 31,099,689 parametri di cui 29,364,201 *trainable*. I Layer successivi al primo sono stati aggiunti per poter rendere possibile la classificazione binaria. In particolare, è significativa l'aggiunta dell'ultimo layer Dense(1) con funzione di attivazione Sigmoid, il quale restituisce in output valori compresi tra 0 e 1.

Layer (type)	Output Shape	Trainable params
VGG16(functional)	(None, 8, 8, 512)	14.681.919
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 500)	16384500
activation (Activation)	(None, 500)	0
dropout (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 1)	501
activation_1 (Activation)	(None, 1)	0

Total params: 31,099,689
Trainable params: 29,364,201
Non-trainable params: 1,735,488

Figura 45 – Struttura della rete neurale realizzata sfruttando VGG16

In Figura 46 viene riportato il *summary* della rete Custom. Il modello presenta 1,200,137, tutti *trainable*. La rete è realizzata stratificando per cinque volte una sequenza di layer *convolutivi* e layer di max *pooling*, gli ultimi layer sono invece di tipo completamente connesso ed hanno come output un layer di due neuroni, in modo da consentire la classificazione binaria.

Layer (type)	Output Shape	Trainable params
conv2d (Conv2D)	(None, 254, 254, 32)	896
activation (Activation)	(None, 254, 254, 32)	0
max_pooling2d	(MaxPooling2D (None, 127, 127, 32))	0
conv2d_1 (Conv2D)	(None, 125, 125, 32)	9248
activation_1 (Activation)	(None, 125, 125, 32)	0
max_pooling2d_1	(MaxPooling2D (None, 62, 62, 32))	0
conv2d_2 (Conv2D)	(None, 60, 60, 32)	9248
activation_2 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_2	(MaxPooling2D (None, 30, 30, 32))	0
conv2d_3 (Conv2D)	(None, 28, 28, 32)	9248
activation_3 (Activation)	(None, 28, 28, 32)	0
max_pooling2d_3	(MaxPooling2D (None, 14, 14, 32))	0
conv2d_4 (Conv2D)	(None, 12, 12, 64)	18496
activation_4 (Activation)	(None, 12, 12, 64)	0
max_pooling2d_4	(MaxPooling2D (None, 6, 6, 64))	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 500)	1152500
activation_5 (Activation)	(None, 500)	0
dropout (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 1)	501
activation_6 (Activation)	(None, 1)	0

Total params: 1,200,137

Trainable params: 1,200,137

Non-trainable params: 0

Figura 46 – Struttura della rete neurale custom

Per le reti pre-addestrate è stato eseguito un addestramento di 10 epoch, utilizzando come ottimizzatore SGD del package keras. La scelta delle epoch non è casuale: è stata eseguita una prima run di training di 100 epoch in cui si è potuto notare che i modelli, intorno alla quindicesima epoch, iniziavano ad andare in overfitting presentando valori di validation_loss sempre maggiore. Per la rete custom è stato invece eseguito un addestramento di 200 epoch, utilizzando come ottimizzatore Adam, importato dallo stesso package. Anche in questo caso è stata eseguita una run iniziale di 800 epoch in cui il modello è andato in overfitting intorno alla duecentocinquantesima epoch. La scelta degli ottimizzatori è stata influenzata dalla quantità dei parametri addestrabili presenti nella rete, per le reti VGG16 e ResNet50 è stato scelto il metodo di discesa stocastica del gradiente attraverso SGD, adatto per modelli con molti parametri. Per la rete custom è stato utilizzato Adam, in quanto il numero di parametri è molto inferiore. Entrambi gli ottimizzatori utilizzano come parametro l'accuracy e come funzione di loss binary cross entropy.

In Figura 47 è riportato il grafico relativo al training della rete ResNet50 ed in Figura 48 sono riportati i valori di accuracy, loss, validation accuracy e validation loss per tutte le epoch di training. L'ultima riga di tale figura riporta i valori di accuracy e loss ottenuti sul testing set. Possiamo notare come l'addestramento di ResNet è risultato molto lineare, presentando valori di accuracy sempre crescenti e loss quasi sempre decrescenti.

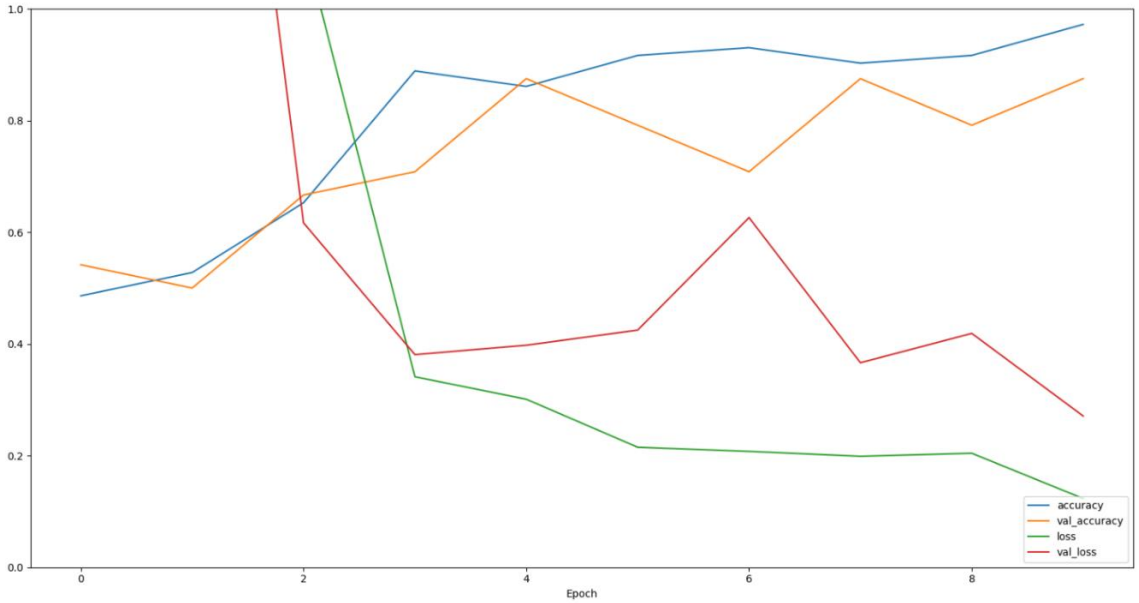

Figura 47 – Training ResNet50

```
Epoch 1/10
3/3 [==============================] - 9s 3s/step - loss: 6.7344 - accuracy: 0.4861 - val_loss: 1.4173 - val_accuracy: 0.5417
Epoch 2/10
3/3 [==============================] - 6s 2s/step - loss: 1.7249 - accuracy: 0.5278 - val_loss: 2.1817 - val_accuracy: 0.5000
Epoch 3/10
3/3 [==============================] - 6s 2s/step - loss: 1.1219 - accuracy: 0.6528 - val_loss: 0.6168 - val_accuracy: 0.6667
Epoch 4/10
3/3 [==============================] - 6s 2s/step - loss: 0.3411 - accuracy: 0.8889 - val_loss: 0.3807 - val_accuracy: 0.7083
Epoch 5/10
3/3 [==============================] - 6s 2s/step - loss: 0.3009 - accuracy: 0.8611 - val_loss: 0.3976 - val_accuracy: 0.8750
Epoch 6/10
3/3 [==============================] - 6s 2s/step - loss: 0.2149 - accuracy: 0.9167 - val_loss: 0.4246 - val_accuracy: 0.7917
Epoch 7/10
3/3 [==============================] - 6s 2s/step - loss: 0.2074 - accuracy: 0.9306 - val_loss: 0.6261 - val_accuracy: 0.7083
Epoch 8/10
3/3 [==============================] - 6s 2s/step - loss: 0.1987 - accuracy: 0.9028 - val_loss: 0.3662 - val_accuracy: 0.8750
Epoch 9/10
3/3 [==============================] - 6s 2s/step - loss: 0.2041 - accuracy: 0.9167 - val_loss: 0.4188 - val_accuracy: 0.7917
Epoch 10/10
3/3 [==============================] - 6s 2s/step - loss: 0.1230 - accuracy: 0.9722 - val_loss: 0.2708 - val_accuracy: 0.8750
2/2 [==============================] - 2s 342ms/step - loss: 0.3564 - accuracy: 0.8667
1/1 [==============================] - 2s 2s/step
```

Figura 48 – Dettaglio training epoch e validation ResNet50

In Figura 49 è riportato il grafico relativo al training della rete VGG16 ed in Figura 50 sono riportati i valori di accuracy, loss, validation accuracy e validation loss per tutte le Epoch. L'ultima riga di tale figura riporta i valori di accuracy e loss ottenuti sul testing set. In questo caso il training risulta meno lineare rispetto a ResNet50, presentando valori oscillatori.

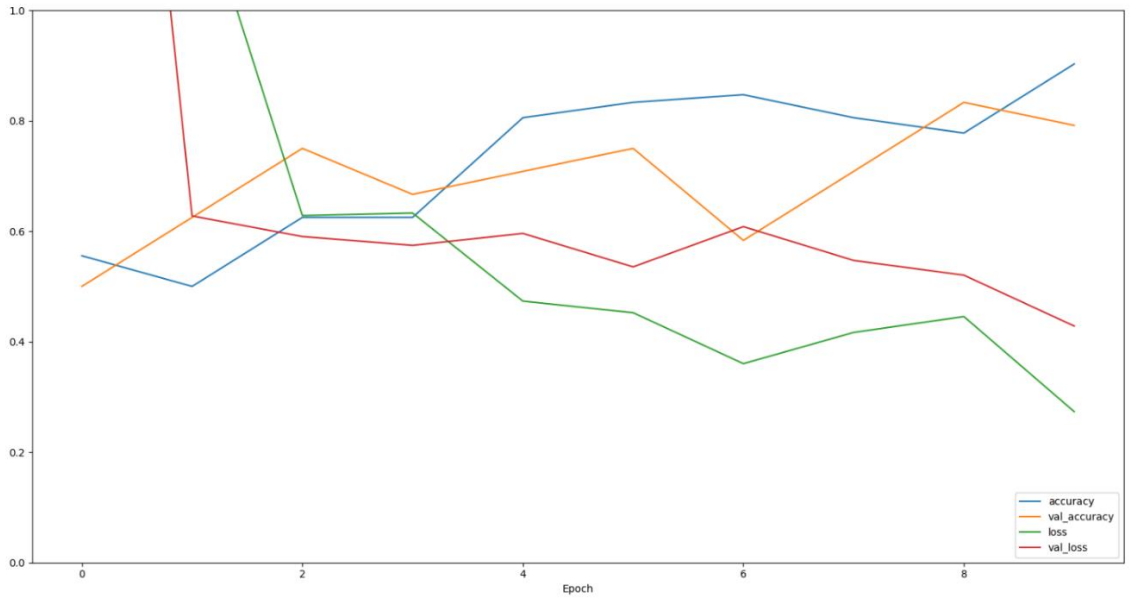

Figura 49 – Training VGG16

```
Epoch 1/10
3/3 [==============================] - 16s 6s/step - loss: 4.7974 - accuracy: 0.5556 - val_loss: 2.5262 - val_accuracy: 0.5000
Epoch 2/10
3/3 [==============================] - 15s 5s/step - loss: 1.2545 - accuracy: 0.5000 - val_loss: 0.6276 - val_accuracy: 0.6250
Epoch 3/10
3/3 [==============================] - 15s 6s/step - loss: 0.6284 - accuracy: 0.6250 - val_loss: 0.5903 - val_accuracy: 0.7500
Epoch 4/10
3/3 [==============================] - 15s 5s/step - loss: 0.6329 - accuracy: 0.6250 - val_loss: 0.5744 - val_accuracy: 0.6667
Epoch 5/10
3/3 [==============================] - 15s 5s/step - loss: 0.4734 - accuracy: 0.8056 - val_loss: 0.5960 - val_accuracy: 0.7083
Epoch 6/10
3/3 [==============================] - 16s 6s/step - loss: 0.4524 - accuracy: 0.8333 - val_loss: 0.5354 - val_accuracy: 0.7500
Epoch 7/10
3/3 [==============================] - 15s 5s/step - loss: 0.3601 - accuracy: 0.8472 - val_loss: 0.6082 - val_accuracy: 0.5833
Epoch 8/10
3/3 [==============================] - 15s 5s/step - loss: 0.4166 - accuracy: 0.8056 - val_loss: 0.5470 - val_accuracy: 0.7083
Epoch 9/10
3/3 [==============================] - 15s 5s/step - loss: 0.4454 - accuracy: 0.7778 - val_loss: 0.5203 - val_accuracy: 0.8333
Epoch 10/10
3/3 [==============================] - 15s 5s/step - loss: 0.2732 - accuracy: 0.9028 - val_loss: 0.4283 - val_accuracy: 0.7917
2/2 [==============================] - 3s 546ms/step - loss: 0.4111 - accuracy: 0.8000
1/1 [==============================] - 2s 2s/step
```

Figura 50 – Dettaglio training epoch e validation VGG16

In Figura 51 è riportato il grafico relativo al training della rete Custom ed in Figura 52 sono riportati i valori di accuracy, loss, validation accuracy e validation loss per le ultime 10 Epoch. L'ultima riga di tale figura riporta i valori di accuracy e loss ottenuti sul testing set. Si può notare dal grafico come il comportamento della rete Custom è stato piuttosto lineare dalla settantacinquesima epoch. Da questo punto in poi, infatti, l'accuracy è sempre aumentata e la loss è sempre andata a diminuire, fino ad andarsi ad assestare intorno alla centottantesima epoch.

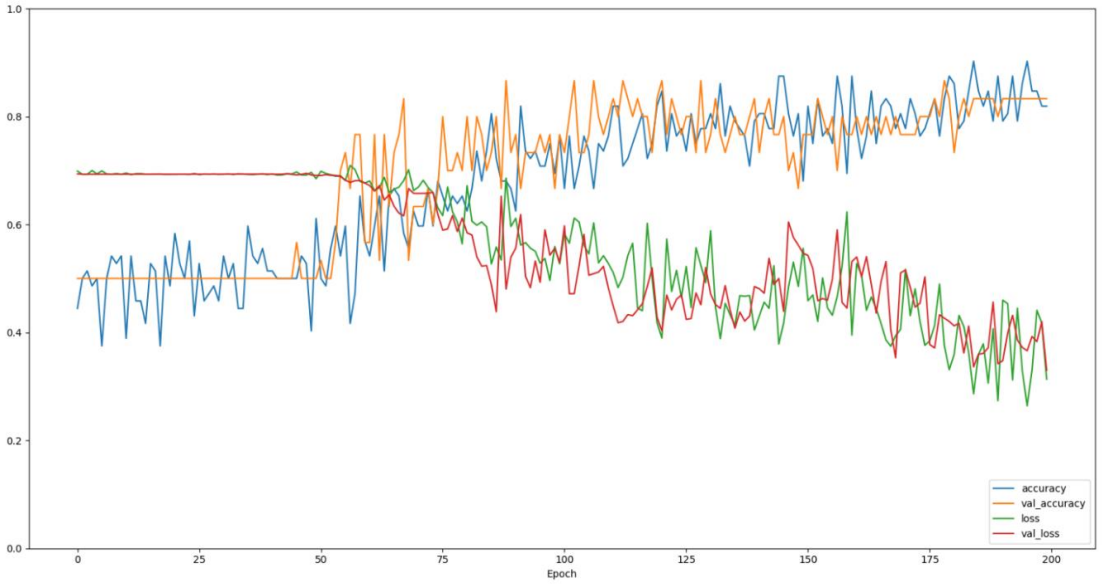

Figura 51 – Training custom CNN

```
Epoch 190/200
3/3 [==============================] - 2s 774ms/step - loss: 0.2736 - accuracy: 0.8750 - val_loss: 0.3418 - val_accuracy: 0.8000
Epoch 191/200
3/3 [==============================] - 2s 761ms/step - loss: 0.4598 - accuracy: 0.7917 - val_loss: 0.3478 - val_accuracy: 0.8333
Epoch 192/200
3/3 [==============================] - 3s 773ms/step - loss: 0.4532 - accuracy: 0.8056 - val_loss: 0.3990 - val_accuracy: 0.8333
Epoch 193/200
3/3 [==============================] - 3s 773ms/step - loss: 0.3120 - accuracy: 0.8750 - val_loss: 0.4317 - val_accuracy: 0.8333
Epoch 194/200
3/3 [==============================] - 2s 779ms/step - loss: 0.4450 - accuracy: 0.7917 - val_loss: 0.3858 - val_accuracy: 0.8333
Epoch 195/200
3/3 [==============================] - 3s 784ms/step - loss: 0.3292 - accuracy: 0.8611 - val_loss: 0.3723 - val_accuracy: 0.8333
Epoch 196/200
3/3 [==============================] - 2s 760ms/step - loss: 0.2640 - accuracy: 0.9028 - val_loss: 0.3660 - val_accuracy: 0.8333
Epoch 197/200
3/3 [==============================] - 2s 749ms/step - loss: 0.3300 - accuracy: 0.8472 - val_loss: 0.3923 - val_accuracy: 0.8333
Epoch 198/200
3/3 [==============================] - 2s 776ms/step - loss: 0.4412 - accuracy: 0.8472 - val_loss: 0.3830 - val_accuracy: 0.8333
Epoch 199/200
3/3 [==============================] - 2s 764ms/step - loss: 0.4169 - accuracy: 0.8194 - val_loss: 0.4201 - val_accuracy: 0.8333
Epoch 200/200
3/3 [==============================] - 2s 751ms/step - loss: 0.3134 - accuracy: 0.8194 - val_loss: 0.3297 - val_accuracy: 0.8333
1/1 [==============================] - 0s 284ms/step - loss: 0.3297 - accuracy: 0.8333
```

Figura 52 – Dettaglio training epoch e validation custom CNN

Per valutare le performance delle tre reti neurali, oltre training e validation, per ognuna di esse è stata calcolata la curva ROC e l'area sottesa da tale curva, l'AUC, per poi utilizzarla come metrica di confronto tra i vari modelli. Possiamo notare dalle figure 53, 54 e 55 come tutti i modelli presentino delle curve ROC valide. In particolare, l'AUC minimo è di 0.86 ed è relativo al modello VGG, mentre l'AUC massimo è 0.94, realizzato relativo al modello ResNet. L'AUC della rete custom si assesta su 0.90. Questo vuol dire che tutti e tre i modelli hanno un buon rapporto tra sensibilità e sensitività, ovvero tra il tasso di "veri positivi" e "falsi positivi" classificati.

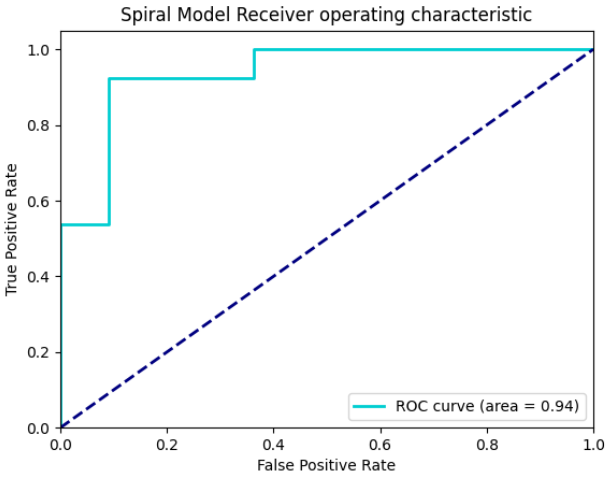

Figura 53 – ROC ResNet50

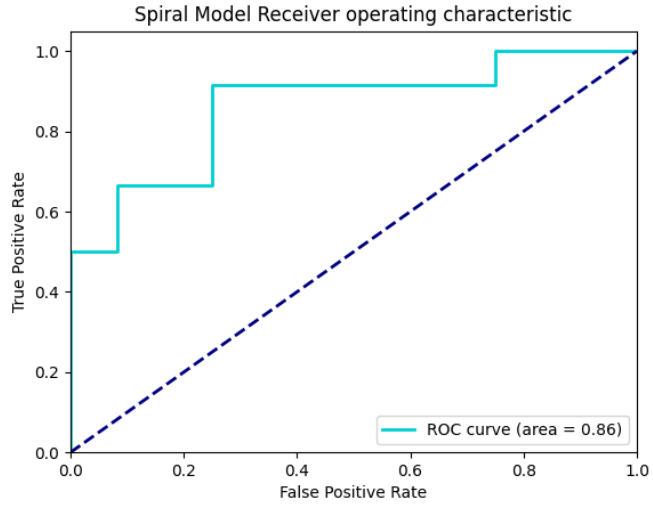

Figura 54 – ROC VGG16

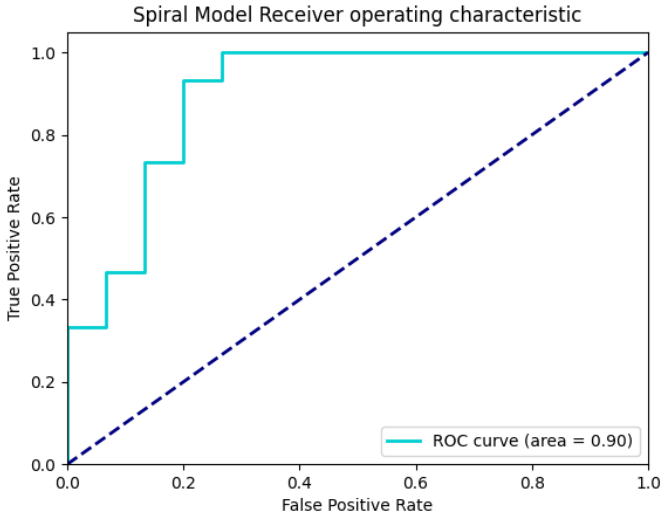

Figura 55 – ROC custom CNN

I risultati ottenuti dai 3 modelli sono riassunti nella tabella in Figura 56. La rete che presenta i migliori risultati è ResNet50, con AUC pari a 0.94, accuracy 0.87 e loss 0.35. La rete custom ha un grafico di training con andamento lineare, molto simile a ResNet50 e presenta risultati leggermente inferiori a quest'ultima. La rete VGG16 presenta i valori peggiori di testing accuracy e testing loss. Inoltre, il comportamento della rete è risultato il meno lineare durante la fase di training. È possibile notare come la rete custom presenta il valore di loss più basso: 0.31. Questo vuol dire che tra le tre reti è quella che ha commesso errori "meno gravi", andando a realizzare il valore più basso per la somma di tutti gli errori commessi in fase di testing.

Rete Neurale	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss	AUC
ResNet50 (10 *epoch*)	0.97	0.12	0.88	0.37	0.94
VGG16 (10 *epoch*)	0.90	0.27	0.80	0.41	0.86
Custom (200 *epoch*)	0.81	0.31	0.83	0.33	0.90

Figura 56– Risultati reti neurali su dataset NIATS

3.4 Conclusioni

In conclusione, possiamo affermare che i risultati ottenuti dal caso di studio sono molto soddisfacenti. L'indagine eseguita sulla classificazione di Spiral Test ha dato ottimi risultati per entrambi i dataset presi in considerazione, nonostante la natura dissimile dei dati a disposizione.

Per quanto riguarda il dataset UCI è stato dimostrato come l'utilizzo di dati raccolti attraverso supporto digitale sia di grande aiuto per l'analisi, consentendo l'estrazione di dati ben strutturati e molto significativi. Malgrado la poca quantità di dati a disposizione sono stati ottenuti risultati di accuracy 87% utilizzando Logistic Regression, con valori per precision, recall e F1 prossimi all'84%. Sicuramente, avendo avuto a disposizione più dati i risultati sarebbero stati migliori, questo lascia ben sperare per future applicazioni su dataset più numerosi.

Per il dataset NIATS invece il discorso è diverso. I dati di quest'ultimo non sono ben strutturati, in quanto il dataset è composto da sole immagini raccolte su supporto cartaceo. In questo caso, per poter sopperire alla mancanza di dati ben strutturati, si è ben pensato di aumentare i dati a disposizione, utilizzando tecniche di data augmentation. Così facendo i risultati di accuracy dei vari classificatori sono aumentati del 10-20% rispetto al dataset originale, non riuscendo però ad eguagliare i risultati ottenuti in precedenza dagli stessi classificatori. L'adozione di reti neurali convolutive ha permesso di ottenere risultati simili a quelli ottenuti sul dataset UCI. I migliori si hanno con un modello che sfrutta come nucleo convolutivo la rete ResNet50 pre-addestrata su imagenet, con valori di validation accuracy pari all'88%. È stata anche realizzata e allenata una rete neurale convolutiva da zero sul dataset NIATS, garantendo prestazioni prossime a quelle di ResNet.

Il lavoro svolto è stato molto stimolante ed ha sollecitato molti spunti di riflessione; utilizzare dataset molto piccoli è stata una grande sfida considerando il contesto applicativo Machine Learning, disciplina che basa le sue fondamenta sull'estrazione di informazioni da grandi quantità di dati. L'utilizzo di tecniche di estrazione e data augmenting è risultato quindi appropriato ed ha garantito il successo del caso di studio.

Bibliografia:

1. J. Parkinson, “Parkinson’s disease,” no. April, pp. 43–47, 2010
2. P. D. Entity and R. Factors, “MSW and its management,” Munic. Solid Waste
3. Parkinson’s Disease Foundation: Statistics on Parkinson’s. Retrieved from http://www.pdf.org/en/parkinson_statistics. (2013))
4. Wright Willis A, B Evanoff, M Lian, S Criswell, B Racette: Geographic and ethnic variation in Parkinson Disease: A population-based study of US Medicare Beneficiaries. Neuroepidemiology, 34, 143-151 (2010)
5. Occasional review the relevance of the Lewy body to the pathogenesis of idiopathic Parkinson’s disease,” pp. 745–752, 1988.
6. W. Dauer and S. Przedborski, “Parkinson’s disease: Mechanisms and models,” Neuron, vol. 39, no. 6, pp. 889–909, 2003. COVID-19 Prevention and Control.
7. K. F. Winklhofer and C. Haass, “Biochimica et Biophysica Acta Mitochondrial dysfunction in Parkinson’s disease,” BBA - Mol. Basis Dis., vol. 1802, no. 1, pp. 29–44, 2010.
8. C. A. Davie, “A review of Parkinson’s disease,” Br. Med. Bull., vol. 86, no. 1, pp. 109–127, 2008.
9. A. A. Moustafa et al., “Motor symptoms in Parkinson’s disease: A unified framework” Neurosci. Biobehav. Rev., vol. 68, pp. 727–740, 2016.
10. J. Massano and K. P. Bhatia, “Clinical approach to Parkinson’s disease: Features, diagnosis, and principles of management,” *Cold Spring Harb. Perspect. Med.*, vol. 2, no. 6, pp. 1–15, 2012
11. J. Jankovic, “Parkinson’s disease: clinical features and diagnosis Parkinson’s disease: clinical features and diagnosis,” J. Neurol. Neurosurg. Physiatry, no. May, pp. 368–376, 2008.

12. T. Simuni and K. Sethi, “Nonmotor manifestations of Parkinson’s disease,” Ann. Neurol., vol. 64, no. SUPPL. 2, pp. 65–80, 2008
13. W. Poewe, “Non-motor symptoms in Parkinson Õ s disease,” Eur. J. Neurol., vol. 15, pp. 14–20, 2008.
14. A. Park, M. A. Stacy, A. Park, and Æ. M. Stacy, “Non-motor symptoms in Parkinson’s disease non-motor symptoms in Parkinson’s disease,” vol. 256, no. May, pp. 293–298, 2015
15. E. Iannilli, L. Stephan, T. Hummel, H. Reichmann, and A. Haehner, “Olfactory impairment in Parkinson’s disease is a consequence of central nervous system decline,” J. Neurol., vol. 264, no. 6, pp. 1236–1246, 2017.
16. C. R. Baumann, “Sleep–wake and circadian disturbances in Parkinson disease: a short clinical guide,” J. Neural Transm., vol. 126, no. 7, pp. 863–869, 2019
17. A. Ascherio *et al.*, “Pesticide exposure and risk for Parkinson’s disease,” *Ann. Neurol.*, vol. 60, no. 2, pp. 197–203, 2006
18. K. Wirdefeldt, H. O. Adami, P. Cole, D. Trichopoulos, and J. Mandel, “Epidemiology and etiology of Parkinson’s disease: A review of the evidence,” Eur. J. Epidemiol., vol. 26, no. SUPPL. 1, 2011.
19. T. C. Booth, M. Nathan, A. D. Waldman, A. M. Quigley, A. H. Schapira, and J. Buscombe, “The role of functional dopamine-transporter SPECT imaging in parkinsonian syndromes, part 2,” *Am. J. Neuroradiol.*, vol. 36, no. 2, pp. 236–244, 2015.
20. K. D. Seifert and J. I. Wiener, “The impact of DaTscan on the diagnosis and management of movement disorders: A retrospective study,” *Am. J. Neurodegener. Dis.*, vol. 2, no. 1, pp. 29–34, 2013.

21. C. G. Goetz *et al.*, "Movement Disorder Society Task Force report on the Hoehn and Yahr staging scale: Status and recommendations," *Mov. Disord.*, vol. 19, no. 9, pp. 1020–1028, 2004
22. C. G. Goetz *et al.*, "Movement Disorder Society-Sponsored Revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): Scale presentation and clinimetric testing results," *Mov. Disord.*, vol. 23, no. 15, pp. 2129–2170, 2008
23. C. G. Goetz, P. Martinez-martin, G. T. Stebbins, M. B. Stern, B. C. Tilley, and A. E. Lang, "MDS-UPDRS," vol. 1, no. 414, 2008
24. C. B.S. and L. A.E., "Pharmacological treatment of Parkinson disease: A review," *JAMA - J. Am. Med. Assoc.*, vol. 311, no. 16, pp. 1670–1683, 2014.
25. A. Medical and S. Ontario, "00002792-201611010-00011 (1)," vol. 188, no. 16, pp. 1157–1165, 2016
26. P. J. Loehrer, "Current approaches to the treatment of thymoma," *Ann. Med.*, vol. 31, no. SUPPL. 2, pp. 73–79, 1999
27. N. Current and I. Finally, "Surgery for Parkinson's," pp. 2–8, 1997.
28. W. C. Koller, R. Pahwa, K. E. Lyons, and A. Albanese, "Surgical treatment of Parkinson's disease," vol. 167, pp. 1–10, 1999.
29. F. Fröhlich, "Deep Brain Stimulation," Netw. Neurosci., vol. 210002, pp. 187–196, 2016
30. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1125458/
31. https://archive.ics.uci.edu/ml/datasets/Parkinson+Disease+Spiral+Drawings+Using+Digitized+Graphics+Tablet#
32. http://www.niats.feelt.ufu.br/en/node/81
33. https://image-net.org/about.php
34. Machine Learning, Tom Mitchell, McGraw Hill, 1997.

35. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, part I, cap 5, 2016
36. M. K. Jiawei Han, Data Mining: Concepts and Techniques, 2006
37. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, part I, cap 1, 2016
38. M. Abeles. Corticonics: Neural circuits of the cerebral cortex. Cambridge Univ Press, 1991.
39. Wikibooks, Hebbian Learning, URL: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Hebbian_ Learning
40. D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, Nature, 1986.
41. Convolutional Neural Networks for Visual Recognition, URL: http://cs231n.github.io/convolutional-networks/#overview.
42. Kohavi R., A study of cross validation and bootstrap for accuracy estimation and model selection. Proc. 14th Int. Joint Conf. Artificial Intelligence, 338345, 1995.
43. P. H. Kassani and A. B. j. Teoh, "A new sparse model for traffic sign classification using soft histogram of oriented gradients," Applied Soft Computing, vol. 52, no. C, pp. 231-246, 2017.
44. Udacity, "Computational Photography by Georgia Tech," Udacity, 23 February 2015. URL: https://sa.udacity.com/course/computational-photography--ud955.
45. 1. Swets JA. ROC analysis applied to the evaluation of medical imaging techniques. *Invest Radiol.* 1979;14:109–21.
46. https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks