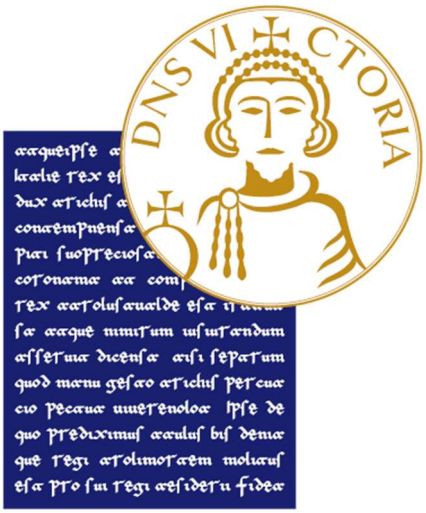
UNIVERSITÀ DEGLI STUDI DEL SANNIO



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

Capitolo 1: Il Morbo di Parkinson 1
1.1 Definizione ed epidemiologia 1
1.2 Fisiopatologia 1
1.3 Sintomatologia 3
1.3.1 Sintomi motori 3
1.3.2 Sintomi non motori 4
1.4 Fattori di rischio 5
1.5 Diagnosi 5
1.6 Scale di misura 6
1.7 Trattamenti 7
Capitolo 3: Caso di studio 9
3.1 Introduzione 9
3.2 Descrizione e pre-processing dei Dataset 9
3.2.1 DataSet UCI 9
3.2.1 DataSet NIATS 23
3.3 Analisi e classificazione 26
3.3.1 Classificazione DataSet UCI 26
3.3.2 Classificazione DataSet NIATS 29
3.4 Risultati 42
Bibliografia: 43

Indice delle figure

Figura 1 – Sistema Dopaminergico 2
Figura 2 – DaTscan di un paziente sano (a sinistra) 6
Figura 3 – Scala di Hoehn and Yahr modificata 7
Figura 4 – Struttura del dataset UCI 10
Figura 5 – Funzione per il calcolo del numero di strokes 16
Figura 6 – Funzione per il calcolo dell'accellerazione 16
Figura 7 – Funzione per il calcolo del vettore velocità 17
Figura 8 – Funzione per il calcolo del vettore jerk 18
Figura 9 – Funzione per il calcolo del numero di variazioni di accelerazioni per semicerchio 19
Figura 10 – Funzione per il calcolo del numero di variazioni di velocità per semicerchio 19
Figura 11 – Funzione per il calcolo della velocità istantanea in un punto 20
Figura 12 – Funzione per il calcolo del tempo passato non in contatto con il tablet 20
Figura 13 – Funzione per il calcolo del tempo passato in contatto con il tablet 20
Figura 14 – Funzione per l'estrazione di tutti gli attributi da spiral test del dataset UCI 21
Figura 15 – Funzione di data cleaning per le features estratte dal dataset UCI 22
Figura 16 – Struttura del dataset NIATS 23
Figura 17– Risultati dataset NIATS per diversi valori di data augmentation 24
Figura 18 – Funzione di data augmentation 25
Figura 19 – Matrice di correlazione tra le features dataset UCI 26
Figura 20 – risultati testing dataset UCI 28
Figura 21 – risultati cross validation dataset UCI 28
Figura 22 – funzione per l'estrazione di feature e label dataset NIATS 29
Figura 23 – Immagine in input (a sinistra) HOG estratto (a destra) 30
Figura 24 – Risultati testing dataset NIATS 31
Figura 25 – Risultati cross validation dataset NIATS 31
Figura 26 – Definizione rete neurale con base convolutiva ResNet50 32
Figura 27 – Definizione rete neurale con base convolutiva VGG16 33
Figura 28 – Definizione rete neurale custom 33
Figura 29 – Struttura della rete neurale realizzata sfruttando ResNet50 34
Figura 30 – Struttura della rete neurale realizzata sfruttando VGG16 34
Figura 31 – Struttura della rete neurale custom 35
Figura 32 – Training ResNet50 37
Figura 33 – Dettaglio training epoch e validation ResNet50 37
Figura 34 – Training VGG16 38
Figura 35 – Dettaglio training epoch e validation VGG16 38
Figura 36 – Training custom CNN 39
Figura 37 – Dettaglio training epoch e validation custom CNN 39
Figura 38 – ROC ResNet50 40
Figura 39 – ROC VGG16 40
Figura 40 – ROC custom CNN 41
Figura 41– Risultati reti neurali su dataset NIATS 41

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].

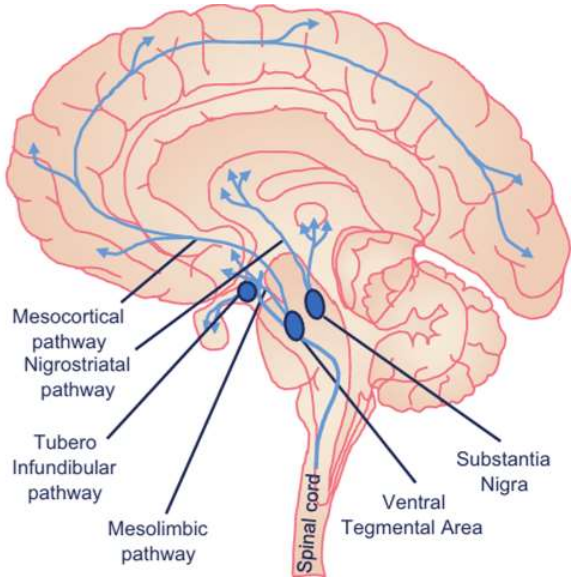


Figura 1 – Sistema Dopaminergico

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.

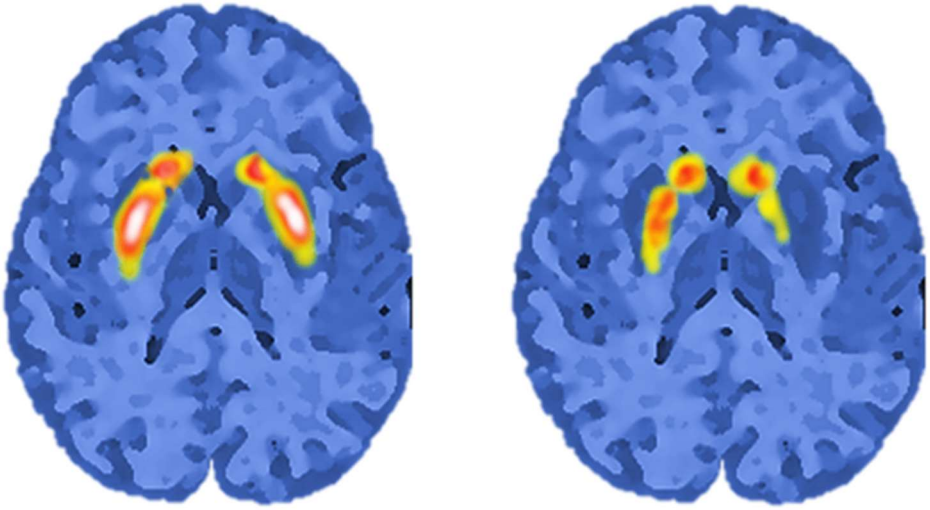


Figura 2 – DaTscan di un paziente sano (a sinistra)
e di un paziente con malattia di Parkinson (a destra)

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 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 cartaceo e dati numerici di test eseguiti attraverso supporto digitale.
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.

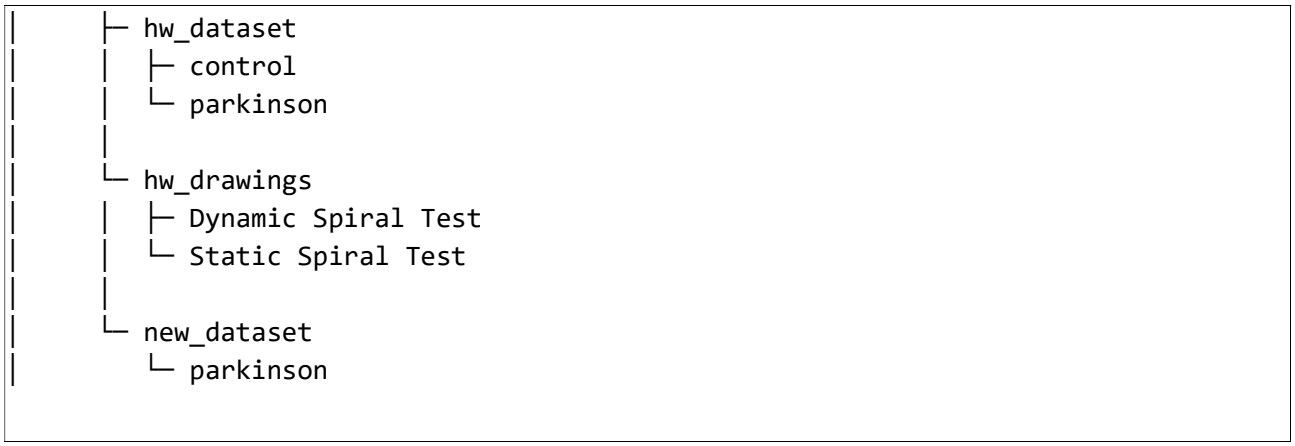


Figura 4 – Struttura del dataset UCI

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 i punti del test, aventi le seguenti caratteristiche:

- *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.

Dai test presenti nelle directories /parkinson e /control sono stati estratti i seguenti attributi per ognuno di essi:

- *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):

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

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:

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

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:

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

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:

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

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:

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

Vettore Jerk orizzontale:

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

Vettore Jerk verticale:

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

Modulo del vettore Jerk:

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

Modulo del vettore Jerk orizzontale:

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

Modulo del vettore Jerk verticale:

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

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

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

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

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

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

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

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

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

Gli attributi sono stati estratti attraverso lo script data_extraction.py, nel quale sono definite le seguenti funzioni:

```
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 5 – 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 6 – Funzione per il calcolo dell'accellerazione

```
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 7 – 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 8 – 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 9 – 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 10 – 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 11 – 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 12 – 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 13 – 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 14 – 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 15 – 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.

3.2.1 DataSet NIATS

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

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

Figura 16 – 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

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. Il primo passo è stato quello di ricercare empiricamente il valore 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 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.

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 17– Risultati dataset NIATS per diversi valori di data augmentation

Per ognuna delle immagini sono state quindi prodotte 20 variazioni, applicando una serie di filtri e trasformazioni. In particolare, le immagini generate sono versioni roteate, leggermente shiftate e tagliate 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.

Il dataset generato risulta così suddiviso:

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

```
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 == 50:
                break
```

Figura 18 – Funzione 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 è stata calcolata la *correlazione* delle features con l'attributo target, in modo da poter individuare quali siano gli attributi più correlati. Utilizzando il package *seaborn* è stata calcolata una *heatmap* dove per ogni coppia di attributi viene mostrato il *grado di correlazione*.

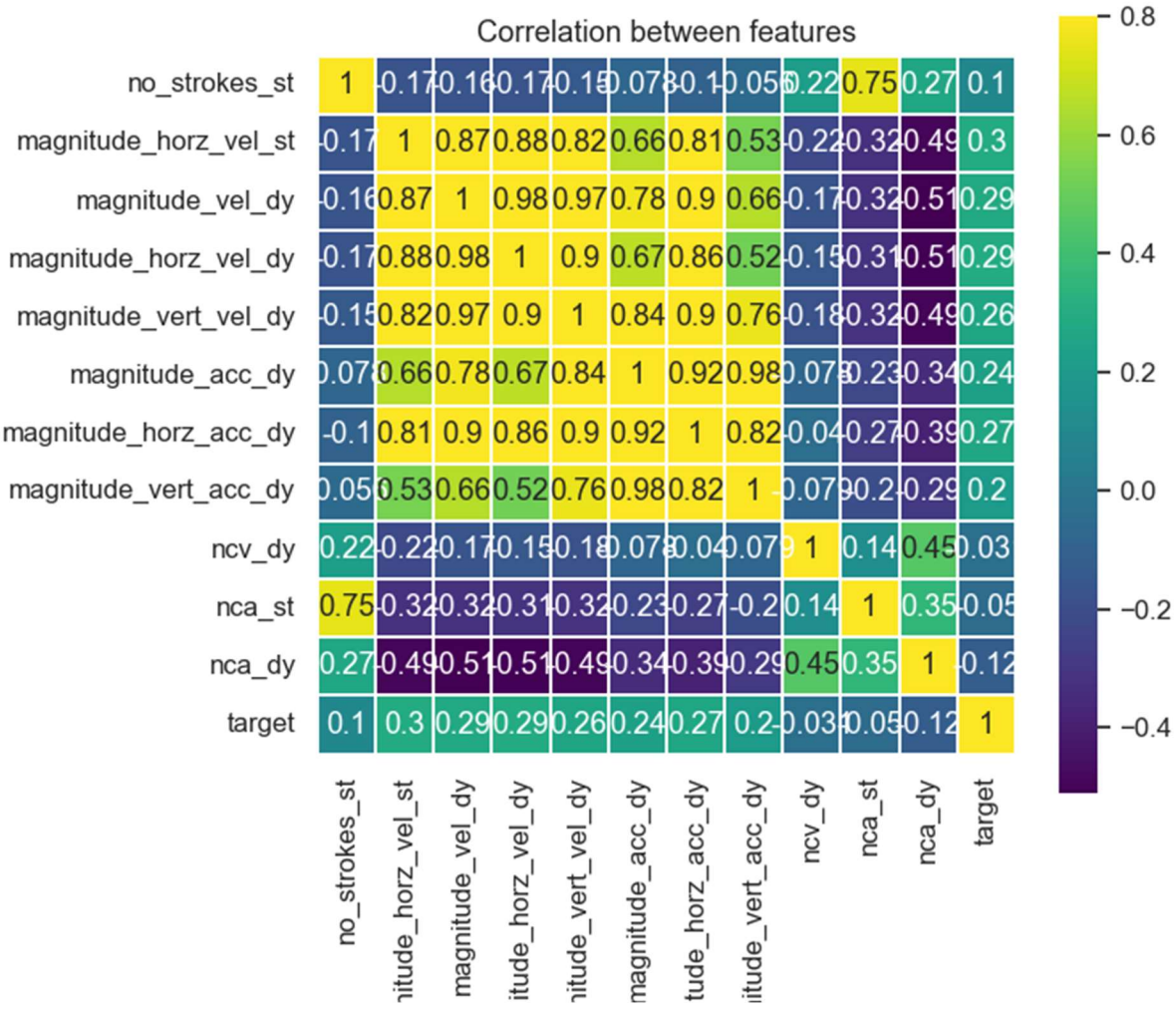


Figura 19 – Matrice di correlazione tra le features dataset UCI

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:

- 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 (k = 2)	Accuracy	Precision	Recall	F1
Logistic Regression	$\begin{bmatrix} 18 & 5 \\ 56 & 6 \end{bmatrix}$	95%	74%	100%	85%
Random Forest	$\begin{bmatrix} 18 & 5 \\ 19 & 43 \end{bmatrix}$	54%	33%	85%	55%
Decision Tree	$\begin{bmatrix} 23 & 0 \\ 0 & 62 \end{bmatrix}$	100%	79%	100%	67%
K-Nearest Neighbors	$\begin{bmatrix} 12 & 11 \\ 16 & 46 \end{bmatrix}$	54%	44%	81%	57%
Gradient Boosting	$\begin{bmatrix} 8 & 0 \\ 0 & 31 \end{bmatrix}$	100%	80%	100%	88%

Figura 20 – risultati testing dataset UCI

Classificatore	Accuracy media (k = 2)	Precision media (k = 2)	Recall media (k = 2)	F1 media (k = 2)
Logistic Regression	87%	83%	84%	84%
Random Forest	71%	75%	70%	65%
Decision Tree	73%	73%	76%	72%
K-Nearest Neighbors	68%	62%	63%	62%
Gradient Boosting	74%	97%	76%	88%

Figura 21 – risultati cross validation dataset UCI

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 la k-cross validation 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%, attraverso la funzione *load_split* si caricano tutte le immagini presenti in una directory, estraendo l'HOG come descrittore dell'immagine e recuperando l'attributo target dal nome della sub-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 22 – funzione per l'estrazione di feature e label dataset NIATS

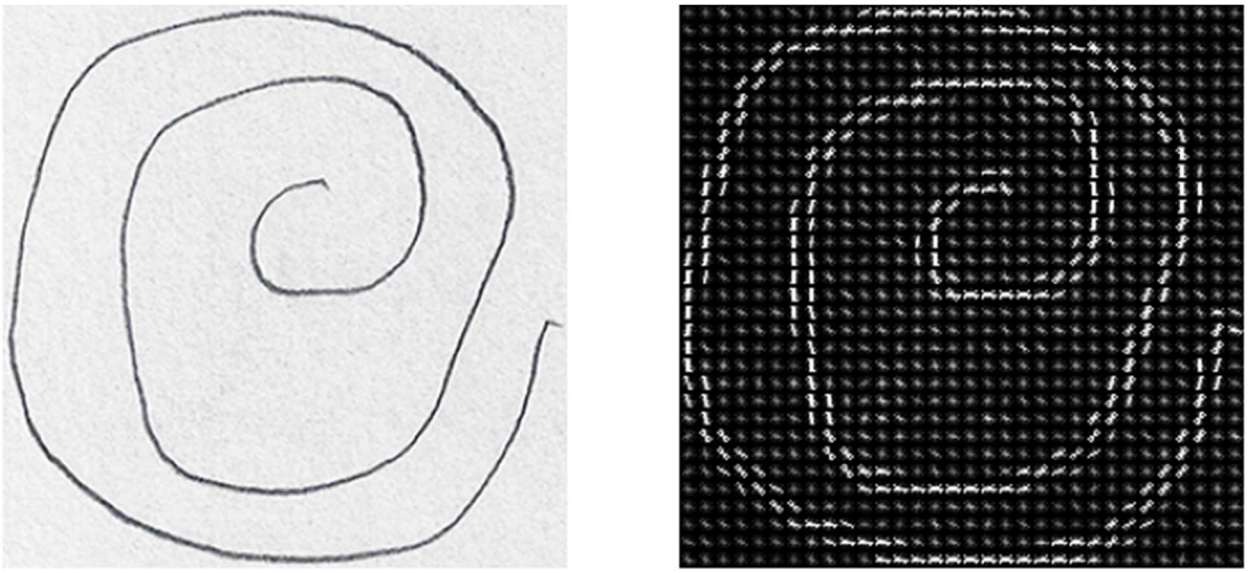


Figura 23 – Immagine in input (a sinistra) HOG estratto (a destra)

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:

- 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 (k =5)	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 24 – Risultati testing dataset NIATS

Classificatore	Accuracy media (k = 5)	Precision media (k = 5)	Recall media (k = 5)	F1 media (k = 5)
Logistic Regression	63%	63%	63%	62%
Random Forest	68%	69%	70%	65%
Decision Tree	61%	60%	59%	60%
K-Nearest Neighbors	59%	71%	59%	53%
Gradient Boosting	67%	67%	66%	67%

Figura 25 – Risultati cross validation dataset NIATS

I risultati di testing sono molto scoraggianti, gli algoritmi presentano accuracy massimo del 50%, fornendo quindi la stessa validità di una scelta randomica. La validazione attraverso k-cross validation (con k = 5) ha migliorato i risultati, i quali 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. 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.

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.

```
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 26 – 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 27 – 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 28 – Definizione rete neurale custom

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 29 – Struttura della rete neurale realizzata sfruttando ResNet50

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 30 – Struttura della rete neurale realizzata sfruttando VGG16

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 31 – 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 difatti eseguita una prima run di training di 100 *epoch* in cui si è potuto notare che il modello, intorno alla quindicesima *epoch*, iniziava ad andare in *overfitting* presentando *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 si è notato che il modello andava 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.*

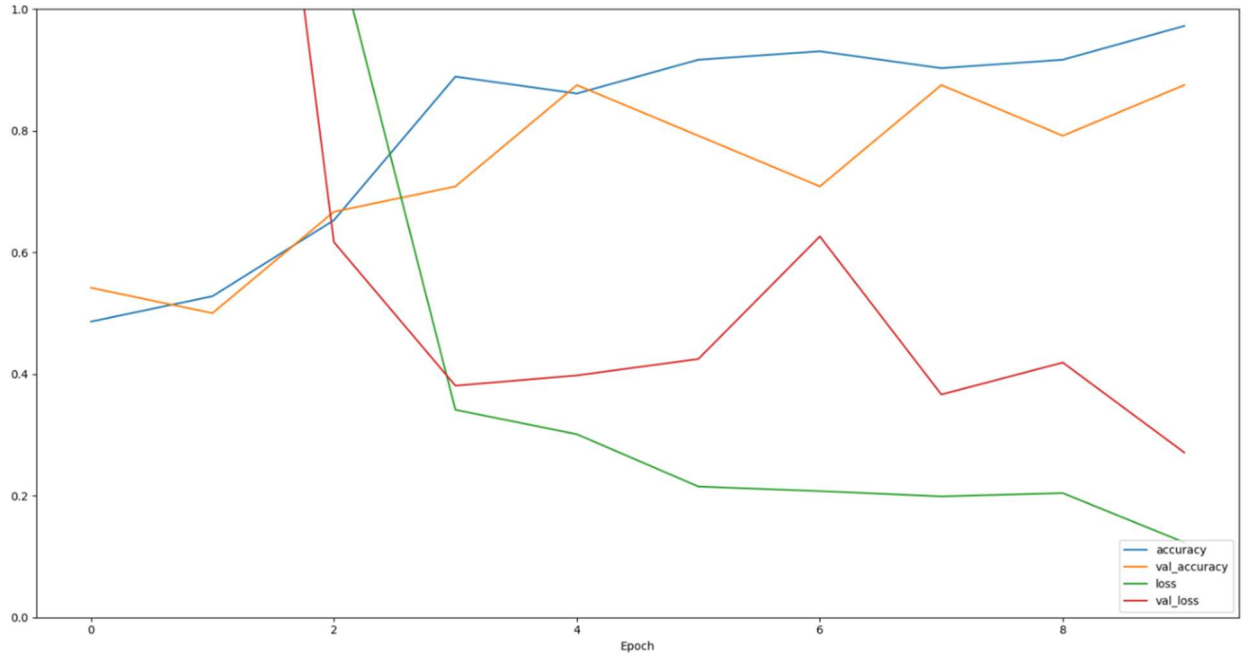


Figura 32 – Training ResNet50

```
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 33 – Dettaglio training epoch e validation ResNet50

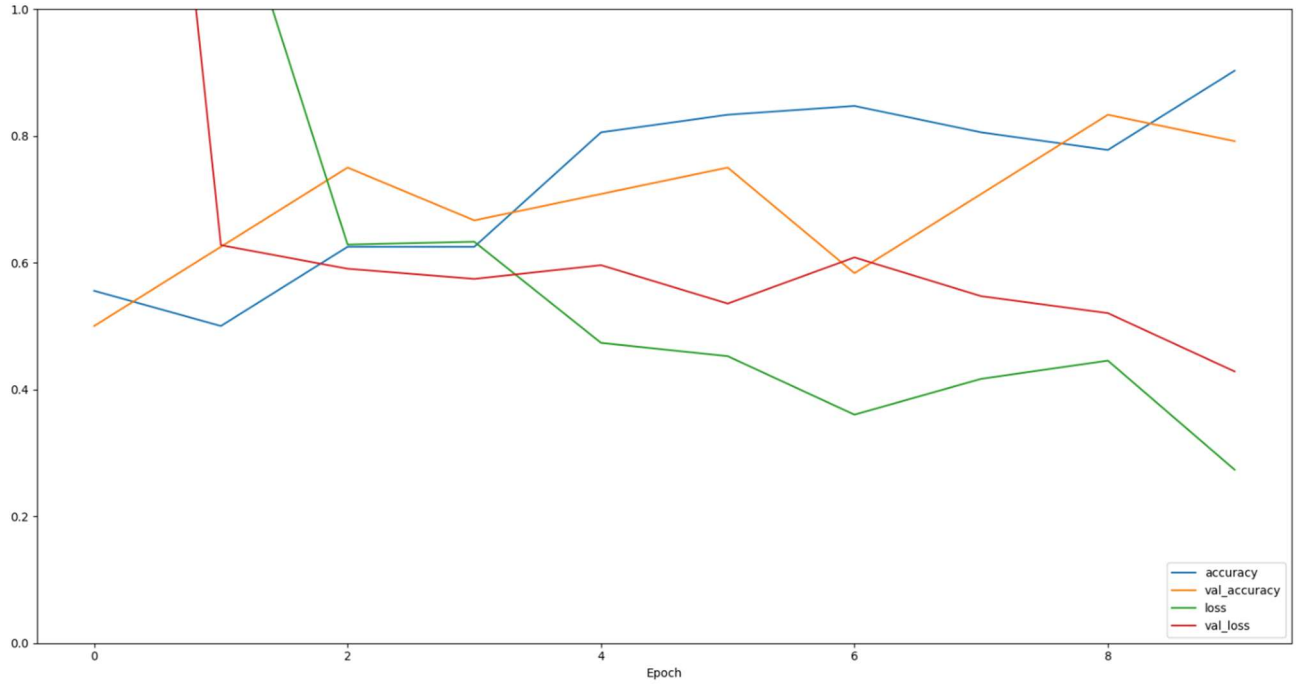


Figura 34 – Training VGG16

```
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 35 – Dettaglio training epoch e validation VGG16

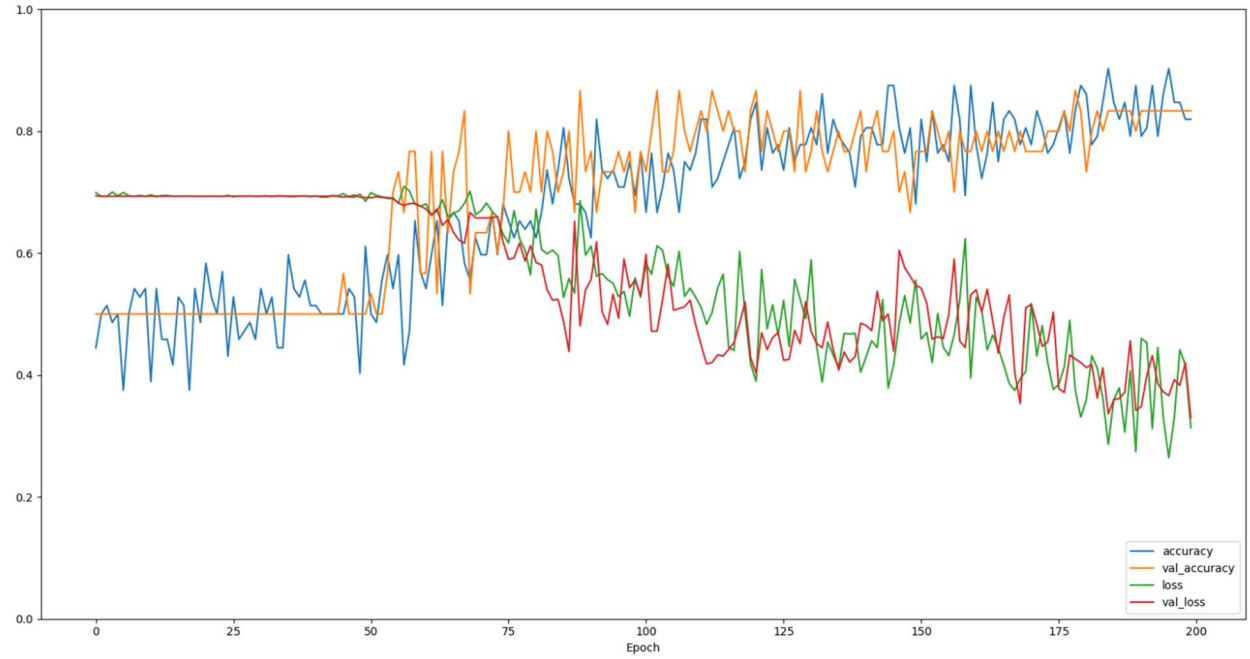


Figura 36 – Training custom CNN

```
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 37 – 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.

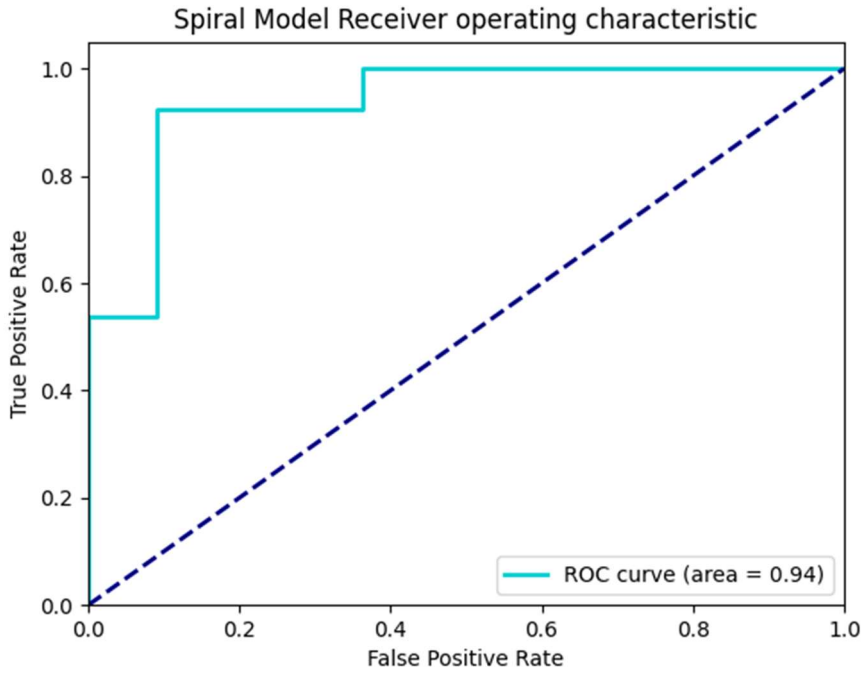


Figura 38 – ROC ResNet50

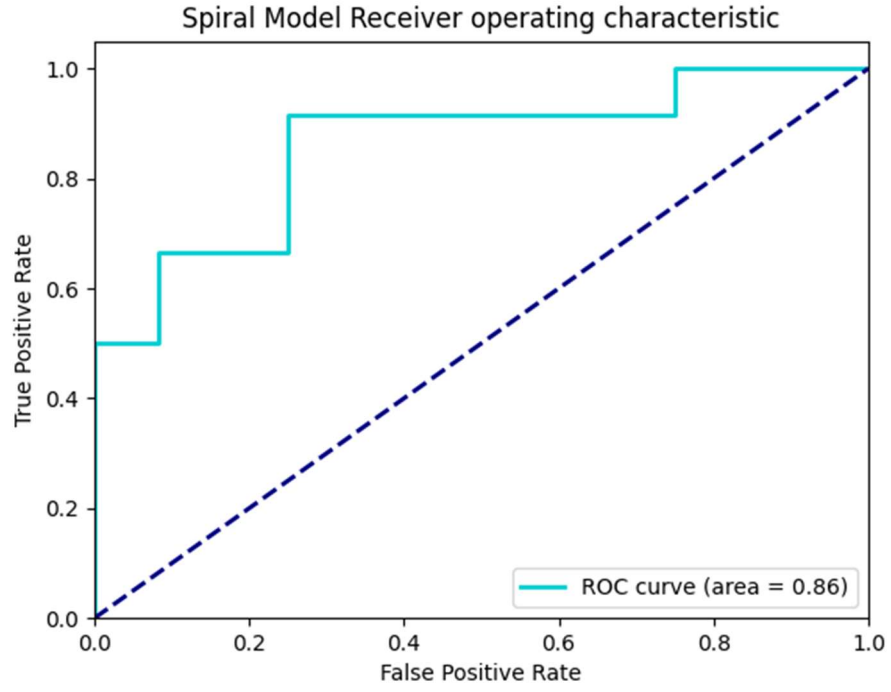


Figura 39 – ROC VGG16

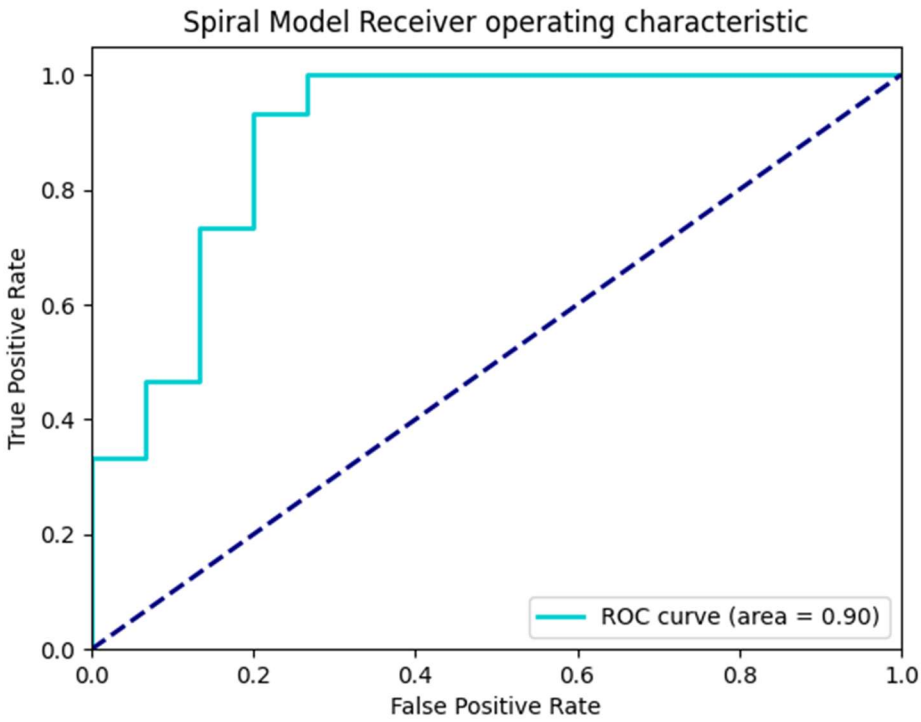


Figura 40 – ROC custom CNN

Rete Neurale	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	AUC
ResNet50 (10 *epoch*)	0.97	0.12	0.88	0.27	0.94
VGG16 (10 *epoch*)	0.90	0.27	0.80	0.42	0.86
Custom (200 *epoch*)	0.81	0.31	0.83	0.33	0.90

Figura 41– Risultati reti neurali su dataset NIATS

I risultati ottenuti sono validi per tutte le reti neurali. Quella che presenta 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 molto simile a ResNet50 e presenta risultati leggermente inferiori. La rete VGG16 presenta i valori peggiori di validation accuracy e validation loss. Inoltre, il comportamento della rete è risultato il meno lineare durante la fase di training.

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 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.