UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Laurea Triennale in Informatica

.

U-ProBE

A graphical Python interface to handle uncertainties in deep learning models

Relatore:

Prof: Davide Bacciu

Phd: Claudia Caudai

Phd: Giulio Del Corso

Candidato:

Lorenzo Bandini

ANNO ACCADEMICO 2023/2024

Indice

Elenco delle figure

Capitolo 1

Introduzione

In un mondo in cui il Machine Learning ed il Deep Learning sono sempre più utilizzati, in molti ambiti della scienza e anche della vita quotidiana, come ad esempio la domotica, l'ingegneria elettronica, l'edilizia, la medicina e la biologia, diventa sempre più importante essere in grado di poter valutare l'affidabilità dei modelli utilizzati, e soprattutto poter valutare quando il modello è "sicuro" degli outputs che propone.

La piattaforma U-Probe mira ad essere un supporto ad un crescente numero di utilizzatori che hanno necessità di valutare le performances di un modello e soprattutto la sua incertezza. Per questo motivo la piattaforma è dotata di una intuitiva interfaccia grafica semplice da utilizzare anche per i non addetti ai lavori.

L'analisi dell'incertezza delle predizioni di un modello di Machine Learning o Deep Learning può essere effettuata utilizzando varie tecniche. Alcune di queste sono intrusive (anche dette by design), tali tecniche vanno a modificare l'architettura introducendo strumenti probabilistici che possono fornire importanti indicazioni sulle caratteristiche delle predizioni, a livello di affidabilità e incertezza. Tali tecniche comprendono ad esempio le Bayesian Neural Networks, i Variational Autoencoders ed i Deep Gaussian Processes. Sono tecniche molto performanti, sia nel mitigare l'overfitting che nell'uncertainty quantification, di contro sono però molto costose e richiedono molte risorse di calcolo e di tempo per l'allenamento dei modelli. Esistono poi le tecniche semi-intrusive, i cui più conosciuti rappresentanti sono i Deep Ensemble; esse rappresentano una ampia classe di approcci che in generale combinano più modelli secondo criteri specifici in modo da valutare l'efficienza, l'incertezza e l'affidabilità delle predizioni senza interferire troppo con le architetture di partenza, ma richiedendo comunque un ampio dispendio di risorse.

In questo lavoro abbiamo deciso di utilizzare per i nostri scopi esclusivamente metodi post-hoc, cioè non intrusivi, come il Trust Score ed il Monte Carlo Dropout, che sono in grado di fare efficaci valutazioni sull'incertezza delle predizioni quando il modello è stato già allenato, senza andare a interferire con le fasi di apprendimento o a modificare i parametri già imparati dal modello durante la back propagation. Tali metodi sono leggermente meno performanti dei metodi intrusivi, ma hanno il vantaggio di essere estremamente più rapidi e meno costosi.

1.1 Contesto del Tirocinio

Il tirocinio è stato svolto presso il CNR-ISTI (Consiglio Nazionale delle Ricerche - Istituto di Scienza e Tecnologie dell'Informazione), uno degli istituti del principale ente pubblico di ricerca in Italia, in particolare, il lavoro si è svolto all'interno del SILAB (Signals and Images Laboratory), un laboratorio di ricerca interdisciplinare specializzato in computer vision, analisi dei segnali, sistemi di visione intelligente e interpretazione dei dati multimediali.

L'intero periodo di sviluppo del progetto è stato svolto sotto la supervisione di Claudia Caudai e Giulio Del Corso, entrambi Postdoctoral Researchers in Machine Learning and Uncertainty Quantification, i quali hanno fatto da guida per le varie scelte, prima teoriche, poi architetturali e di sviluppo, per la realizzazione dell'applicativo.

1.2 Obiettivi e Motivazioni

Il progetto, intitolato "U-ProBE (Uncertainty Probabilistic Bayesian Estimate): A graphical Python interface to handle uncertainties in deep learning models", riguarda lo studio, la progettazione e lo sviluppo di una GUI (Graphical User Interface) in Python per l'analisi di modelli di Deep Learning con previsioni affette da incertezza (i.e., Bayesian Probabilistic Models), nello specifico intende permettere agli utenti di applicare in modo semplice tecniche all'avanguardia per la valutazione dell'affidabilità (reliability) dei modelli attraverso metodologie non intrusive (MC-Dropout 2.2.1, Trust Score 2.2.2, ecc.) con particolare attenzione all'elaborazione di immagini mediche (MRI/ecografie) [21].

L'obiettivo finale è creare un applicativo semplice e accessibile che renda le tecniche per la valutazione dell'affidabilità più fruibili anche per le persone non specializzate nell'ambito del Deep Learning in modo da migliorare la qualità delle analisi in altri settori come la medicina, la finanza e l'ingegneria.

Questo progetto mi ha coinvolto particolarmente per le sue connessioni con la Statistica e l'Intelligenza Artificiale, due discipline che hanno suscitato in me un forte interesse durante il mio percorso di studi in Informatica. Tuttavia, la mia motivazione è cresciuta ulteriormente quando ho compreso il collegamento tra queste aree e il potenziale impatto reale che potrebbero avere nel campo medico, un tema a cui tengo profondamente, influenzato dalle esperienze e dai valori della mia famiglia.

1.3 Struttura della Relazione

La sezione 2 fornisce una panoramica delle basi teoriche e dei concetti preliminari necessari per comprendere il lavoro svolto. In particolare vengono introdotti i concetti di Reti Neurali, Uncertainty Quantification e Analisi Statistica.

La sezione 3 presenta le principali tecnologie impiegate nel progetto, tra cui linguaggi di programmazione, librerie e strumenti utilizzati per lo sviluppo e l'implementazione del sistema.

La sezione 4 analizza le fasi operative del tirocinio e le attività svolte, dalla progettazione, alle scelte implementative, fino alla realizzazione e valutazione del prodotto finale, esponendo i problemi riscontrati e le soluzioni adottate. Fornisce i dettagli di come è stato testato il progetto e spiega come preparare i dati necessari e poter utilizzare l'applicativo.

La sezione 5 offre una sintesi del lavoro svolto, analizzando in modo critico il processo di sviluppo, i principali ostacoli incontrati e le soluzioni adottate. Propone anche spunti di riflessione per possibili sviluppi futuri e riassume le competenze acquisite durante il progetto.

Capitolo 2

Basi di Partenza

Per portare a termine il progetto, pur avendo una conoscenza di base di Statistica e Intelligenza Artificiale, acquisita durante il mio percorso di studi in Informatica, ho dedicato un periodo iniziale allo studio e alla comprensione dei principali concetti di Deep Learning affinché potessi acquisire una solida base teorica per manipolare consapevolmente gli strumenti necessari nelle diverse fasi di realizzazione del progetto.

In particolare, le mie basi includevano competenze in statistica descrittiva, probabilità, inferenza statistica, e test statistici. Per quanto riguarda l'intelligenza artificiale, avevo già acquisito familiarità con agenti risolutori e ricerca euristica, agenti logici, e diverse tecniche di Machine Learning, tra cui modelli lineari, metodi supervisionati come SVM e K-NN, e metodi non supervisionati come alberi di decisione [2].

In particolare, i concetti approfonditi, che verranno introdotti in questo capitolo, includono:

- **Reti Neurali**: principi fondamentali e architetture principali utilizzate nel DL
- **Uncertainty Quantification**: metodi per valutare l'incertezza nei modelli predittivi
- **Analisi Statistica**: tecniche per analizzare i dati e interpretare i risultati ottenuti dai modelli

2.1 Concetti Necessari

Le reti neurali costituiscono il fulcro delle tecniche di DL, la quantificazione dell'incertezza ne amplia la robustezza, e l'analisi statistica fornisce le basi per comprendere i dati e valutare i risultati. Questo capitolo introduce e descrive proprio parte dei concetti teorici fondamentali, necessari per una piena comprensione del lavoro svolto.

Reti Neurali

Le reti neurali rappresentano il cuore delle moderne tecniche di DL, capaci di apprendere autonomamente rappresentazioni complesse dai dati. La loro architettura è composta da strati di nodi, anche detti neuroni, organizzati in layer, in modo da elaborare i dati in ingresso attraverso una serie di trasformazioni non lineari [7]. Queste trasformazioni vengono applicate attraverso le funzioni di attivazione, come ReLU o sigmoid, che introducono non linearità nei modelli, permettendo alle reti neurali di apprendere relazioni complesse tra input e output, poiché senza di esse il modello si ridurrebbe a semplici combinazioni lineari [2].

I layer sono organizzati in sequenza, con il risultato di un layer che diventa l'input per il successivo. Esistono numerosi tipi di layer. Tra i più comuni ci sono i layer densi (o fully-connected), che connettono ogni neurone a tutti quelli del layer successivo, i layer convoluzionali, particolarmente adatti per l'analisi di immagini e dati strutturati e i layer ricorsivi, utili per quando si lavora con dati temporali in quanto catturano le dipendenze e il contesto da input precedenti, ma ne esistono tanti altri [30]. Le differenze tra questi layer rispondono alla necessità di adattarsi a differenti tipi di dati. Le reti neurali, infatti, sono progettate per analizzare vettori numerici, ma le immagini, ad esempio, sono troppo grandi per essere trattate direttamente come vettori. Una semplice immagine a bassa risoluzione (ad esempio, 128x128) può avere fino a 16.384 elementi se trattata come un vettore in $\mathbb{R}^n$. Per questo motivo, le reti convoluzionali (CNN) utilizzano i layer convolutivi per ridurre la dimensionalità dell'immagine, concentrandosi su caratteristiche e strutture elementari, come bordi e forme [18].

Le reti hanno una varietà molto elevata e pertanto per scegliere la migliore rispetto ad un determinato dataset occorre una scelta degli iperparametri che la caratterizzano quali il numero di layer, la dimensione dei filtri delle CNN, la velocità di apprendimento e la batch size [27].

Allo stesso modo, la scelta della funzione di loss è cruciale [16], poiché determina come il modello valuta l'errore tra le sue previsioni e i valori target, e può variare a seconda del problema: per esempio, la cross-entropy è spesso usata per problemi di classificazione, mentre MSE (Mean Squared Error) è preferibile per la regressione. Una funzione di loss appropriata guida il processo di ottimizzazione, consentendo al modello di adattarsi progressivamente ai dati e migliorare le proprie prestazioni dando al modello un criterio chiaro per correggere gli errori durante l'addestramento.

Inoltre, per rendere l'addestramento più veloce ed efficace, si ricorre a metodi di ottimizzazione come Adam [16], che regola automaticamente il tasso di apprendimento per ogni parametro, e SGD (Stochastic gradient descent), che rende l'addestramento più rapido aggiornando i pesi dopo ogni singolo campione.

Infine, per evitare fenomeni di overfitting, cioè quando il modello si adatta troppo ai dati di addestramento, imparando anche il rumore e le anomalie, si ricorre a tecniche di regolarizzazione, come Dropout, che disattiva casualmente neuroni per impedire che il modello diventi troppo dipendente da particolari unità, e early stopping [25], che interrompe l'addestramento non appena le prestazioni sui dati di validazione

smettono di migliorare, prevenendo l'overfitting e migliorando di conseguenza la generalizzazione.

Uncertainty Quantification

Nel contesto del DL, la quantificazione dell'incertezza è fondamentale per capire quanto ci possiamo fidare delle previsioni restituite da un modello [5]. I tipi di incertezze che possiamo valutare sono due: l'incertezza aleatoria che è legata alla variabilità dei dati in ingresso come il rumore, e l'incertezza epistemica, che deriva dalla conoscenza incompleta del modello nel comprendere il problema a causa della sua complessità o dalla carenza di dati [14] [9].

Per stimare queste incertezze e integrare la probabilità all'interno del modello i principali metodi utilizzati sono gli approcci bayesiani, come i Gaussian Processes [10] e le Bayesian Neural Networks [24], che riescono ad integrare la probabilità nella struttura del modello. Alternative meno onerose sono gli approcci come l'Ensemble [17] o il Dropout che restituiscono stime indirette dell'incertezza combinando più previsioni. In aggiunta, possono essere utili indici di fiducia basati su euristiche, come la Topological Data Analysis, il Trust Score o gli indici interni del modello, che forniscono ulteriori indicazioni sulla robustezza delle previsioni del modello [1].

Queste incertezze vengono quantificate attraverso metriche come gli intervalli di confidenza, che indicano con una certa probabilità dove sono i valori plausibili delle previsioni del modello rispetto ai dati osservati, la varianza predittiva che misura la dispersione delle previsioni e l'entropia che misura quanto sono disperse le previsioni di un modello.

Analisi Statistica

L'analisi statistica è la scienza che si occupa di raccogliere, analizzare e visualizzare grandi quantità di dati per scoprire modelli e tendenze. Nel DL, ci viene in aiuto nella comprensione delle previsioni dei modelli e a valutare la loro incertezza [1]. La statistica descrittiva è il primo passo per sintetizzare e organizzare i dati, utilizzando misure come la media, la varianza e le distribuzioni per ottenere una visione complessiva delle caratteristiche fondamentali dei dati.

Successivamente, attraverso l'inferenza statistica [3], ci si concentra nell'estrapolazione di conclusioni a partire dai dati campionati attraverso tecniche come gli intervalli di confidenza, che ci indicano con quale probabilità possiamo aspettarci che i valori osservati appartengano a un certo intervallo e i test statistici, che verificano se una ipotesi sia valida o meno [20].

Spesso nell'intelligenza artificiale ci può servire sapere se due variabili sono correlate o meno e che relazione ci sia tra di esse. Per analizzare questa relazione utilizziamo delle tecniche di regressione (e.g., lineare, quadratica, monotona, etc.), che ci permettono di determinare se esista o meno una correlazione tra le variabili e in caso positivo, possiamo utilizzare il modello per fare previsioni sui dati [19].

Per problemi più complessi, ad esempio quando i dati hanno molte variabili, utilizziamo metodi di analisi multivariata come la PCA(Principal Component Analysis) [13],

che riduce la complessità dei dati, semplificando l'analisi e la ricerca di correlazioni tra più variabili.

L'analisi statistica si fonde alle reti neurali quando si vuole tenere conto dell'incertezza dei dati nei risultati dei modelli, ottenendo quindi quelli che vengono chiamati modelli bayesiani.

2.2 Metodologie Post-Hoc utilizzate

Le metodologie post-hoc sono dei metodi non intrusivi che, a differenza di quelli intrusivi come le Reti Bayesiane mediante Variational Inference [26] o la modifica delle funzioni di Loss per approssimare la posterior distribution [15], vengono applicati solo quando il modello è stato già addestrato. Ne consegue che queste metodologie risultino molto più leggere a livello computazionale dell'addestramento o della modifica dell'architettura di una rete, rendendoli quindi portatili e eseguibili su macchine di potenza computazionale limitata senza la necessità di un accellerazione GPU [17]. In questa sezione esploreremo le basi teoriche che stanno dietro ai metodi di MC-Dropout e Trust Score, successivamente implementati all'interno dell'applicativo, che rappresentano due gold standard di tecniche non intrusive per la quantificazione delle incertezze.

2.2.1 MC-Dropout

Introduzione

Il Dropout nasce come tecnica per ridurre l'overfitting dei modelli e viene applicato in fase di training [28]. La tecnica in sé consiste nel "disattivare" in maniera casuale alcuni nodi di un layer. Attraverso l'applicazione iterativa di questa tecnica, continuando a spegnere in maniera casuale i nodi e applicando la retropropagazione, la rete tende ad equilibrarsi, evitando nodi pivot con valori troppo alti e riducendo il rischio di overfitting. L'MC-Dropout, abbreviazione di Monte Carlo Dropout, non è altro che l'applicazione della tecnica del Dropout in fase di inferenza e la motivazione per cui è divento popolare nella pratica è la sua capacità di adattarsi a grandi quantità di dati e il fatto di richiedere modifiche minori dell'architettura per applicarlo [5].

Incertezza delle previsioni

L'approccio dell'MC-Dropout è motivato dai principi bayesiani, sebbene la tecnica stessa potrebbe non essere bayesiana, e dal fatto che è semplice, efficiente ed anche scalabile. Quando effettuiamo una misurazione o una previsione tenendo presente le incertezze, il risultato viene solitamente fornito come distribuzione di possibili valori. Le proprietà di questa distribuzione codificano il dubbio che abbiamo sul risultato. Ma le reti neurali generiche non producono distribuzioni, bensì un singolo output. Idealmente, in una rete neurale vogliamo ottimizzare un set specifico di parametri in modo che rimangano fissi dopo l'addestramento e che se valutiamo l'input più volte, otterremo sempre lo stesso output. Applicando questa idea all'MC-Dropout,

valutando più volte lo stesso input sulla rete, otterremmo una distribuzione di valori che sarà composta dagli output di queste iterazioni [14].

Dettagli tecnici

L'MC-Dropout è una tecnica che si applica in fase di inferenza. In particolare, data una generica rete neurale dove l'output del layer i-esimo può essere espresso come una funzione non lineare di una combinazione lineare dei valori del livello precedente, con la regolarizzazione tramite MC-Dropout, ogni livello i ad eccezione dell'ultimo è associato a una variabile casuale binaria multivariata e seguendo una regola di distribuzione (solitamente Bernoulli) ma si può scegliere anche tra altre, vengono scelti i neuroni da spegnere con una certa probabilità p.

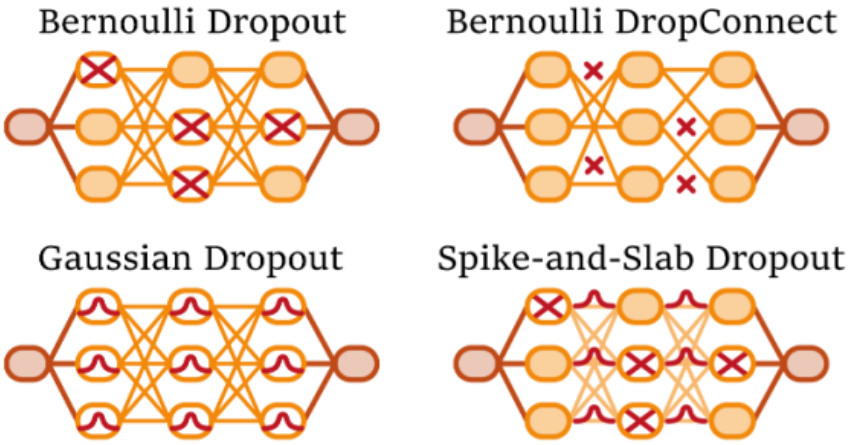

Figura 2.1: Varianti MC-Dropout

Quel che viene fatto successivamente è eseguire lo stesso procedimento di inferenza sullo stesso dato di test per un numero elevato S di volte. Al termine delle iterazioni, i risultati della rete verranno combinati per avere una distribuzione e per estrapolarne una media. Allo stesso modo, la varianza predittiva del modello può essere approssimata come la varianza campionaria degli S passaggi stocastici eseguiti.

Attualmente la scelta di p, come fattore di spegnimento dei neuroni, è arbitraria e possiamo solamente notare che su grandi dataset, un valore di Dropout di 0.5 risulta ottimale, ma anche valori più piccoli possono portare a un notevole aumento di prestazioni [28]. Su dataset di piccole dimensioni invece, un tasso di Dropout elevato rischia di rallentare eccessivamente la convergenza e potrebbe essere subottimale.

Solitamente 50 iterazioni vengono considerate una scelta sicura per stimare l'incertezza [28], ma non sempre avere un numero fisso di iterazioni è la scelta voluta. A volte può capitare di volersi fermare a poche iterazioni, mentre a volte vorremmo tenere il numero di iterazioni alto data la bassa velocità di convergenza, con il lato negativo di rendere l'operazione molto costosa. Esistono però metodi più eleganti ed avanzati come l'Halting Criterion [6] che permettono di definire in maniera adattiva il minimo numero di iterazioni [11].

$$\text{Halting Criterion} = \frac{\text{Median}\left(\text{Var}\left(\mu\right)\right)}{n} \tag{2.1}$$

dove:

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\text{predictions}_i, \quad \text{Var}(\mu) = \frac{1}{n}\sum_{i=1}^{n}(\mu_i - \bar{\mu})^2$$

Complessità e Ottimizzazioni

Lo svantaggio principale dell'MC-Dropout è la sua complessità computazionale, che può essere proporzionale al numero di valutazioni del modello che si devono compiere. Per ottimizzare, le propagazioni in avanti possono essere eseguite contemporaneamente, con conseguente riduzione dei tempi di calcolo fino a un massimo proporzionale al numero di unità di calcolo disponibili. Inoltre, quando i layer di Dropout si trovano vicino all'uscita della rete, è possibile salvare l'output del primo layer di Dropout alla prima iterazione e riutilizzarlo nelle iterazioni successive, evitando di calcolare inutilmente gli stessi valori più volte. Di conseguenza, la complessità computazionale dell'MC-Dropout può essere significativamente ridotta, rendendolo adatto anche a applicazioni in tempo reale.

2.2.2 Trust Score

Introduzione

Il Trust Score è un'altra tecnica non intrusiva, pubblicata dal reparto di ricerca di Google [12], per valutare l'incertezza dei risultati di output operando però da un altra prospettiva. Mentre gli altri metodi cercano una stima senza ulteriori fasi di addestramento, il Trust Score studia proprio i comportamenti della rete sui dati di training con una determinata GT, per poi confrontarli con i dati ottenuti dall'inferenza per derivarne l'incertezza del risultato, assegnandogli un punteggio di fiducia (i.e., Trust Score).

Dettagli tecnici

Il Trust Score si basa sull'idea che una previsione è più affidabile quando il campione di test è vicino a dati di training simili appartenenti alla classe predetta, e lontano dai dati delle altre classi, utilizzando proprio la distanza per calcolare il punteggio di affidabilità [22].

Formalmente, dato un campione di test x, un classificatore M, con un insieme di classi possibili K e un sottoinsieme altamente rappresentativo $H_\alpha(K_i)$ del dataset di addestramento per ciascuna classe K_i, si considera anche $O^{2nd}(x)$, la seconda classe più vicina diversa dalla classe predetta $M(x)$ [8]. Il Trust Score è definito come il rapporto tra:

- la distanza del campione di test dall'insieme dei punti più rappresentativi della classe predetta dal classificatore M(nominatore)

- la distanza del campione di test dall'insieme dei punti più rappresentativi della classe più vicina diversa da quella predetta $O^{2nd}(x)$, poiché quella più probabile secondo il modello dopo la classe predetta (denominatore)

$$TS(x) := \frac{d(x, H_\alpha(M(x)))}{d(x, H_\alpha(O^{2nd}(x)))} \tag{2.2}$$

Il metodo per calcolare la distanza che utilizziamo può essere deciso dell'utilizzatore della tecnica, quelle più utilizzate sono la distanza dal centroide dell'insieme [2] e K-NN, che calcola la distanza mediata fra i K rappresentanti più vicini della classe [4].

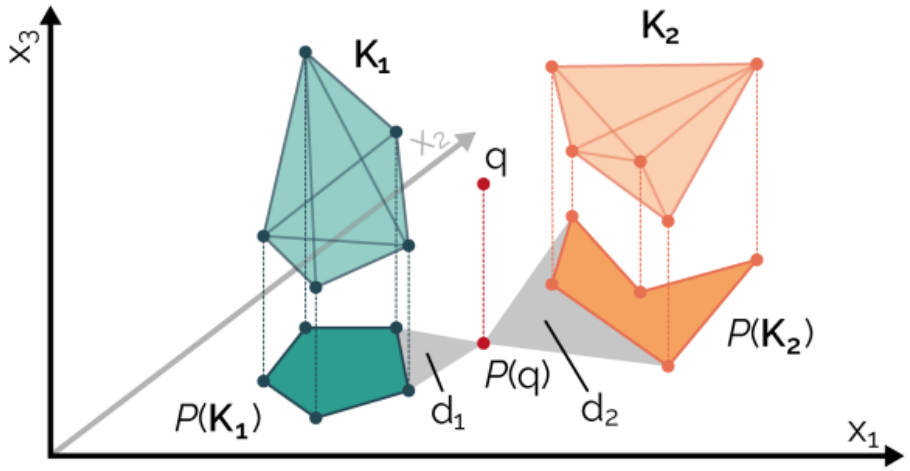

Figura 2.2: Esempio di Trust Score

In maniera intuitiva, il Trust Score utilizza delle distanze in una rappresentazione dell'input (numerico, immagini, etc.) in uno spazio $\mathbb{R}^s$ di deep features, corrispondente ai valori del layer selezionato. Con riferimento alla figura 2.2, questo corrisponde a proiettare uno spazio ad elevata dimensionalità di input in un sottospazio a dimensione ridotta, calcolato usando i primi layers della rete come un Deep Encoder per distillare l'informazione. La ridotta dimensionalità ottenuta mediante questa proiezione permette di calcolare la distanza in uno spazio più semplice e con, auspicabilmente, una maggiore separabilità fra i rappresentanti delle classi.

Capitolo 3

Tecnologie Utilizzate

3.1 Python

Python è un linguaggio di programmazione interpretato, orientato agli oggetti, di alto livello con semantica dinamica [29]. La sintassi semplice e facile da apprendere di Python enfatizza la leggibilità e, pertanto, riduce il costo della manutenzione del programma. Python supporta moduli e pacchetti, il che incoraggia la modularità del programma e il riutilizzo del codice. Inoltre, Python è un linguaggio interpretato, il che elimina la necessità di una fase di compilazione prima dell'esecuzione del codice. Questo permette un ciclo di sviluppo rapido, facilitando le prove, le correzioni di errori e l'esecuzione del codice, rendendo il processo di sviluppo di applicazioni veloce, flessibile e a basso costo.

In ambito di intelligenza artificiale e Machine Learning, Python si è affermato come il linguaggio di riferimento, grazie alla sua facilità d'uso e alla disponibilità di librerie specializzate. La rapidità di scrittura del codice lo rende particolarmente adatto per progetti complessi, permettendo agli sviluppatori di concentrarsi sulla logica e sull'implementazione delle soluzioni piuttosto che sui dettagli del linguaggio stesso. Oltre all'AI e alla Data Science, Python è utilizzato anche nello sviluppo di API, applicazioni web e interfacce grafiche, offrendo strumenti robusti per la creazione di software di alta qualità.

3.1.1 Librerie Utilizzate

Una libreria Python è un insieme di moduli, cioè file contenenti codice predefinito, che racchiude funzioni, classi e variabili che possono essere riutilizzate in vari programmi. L'impiego delle librerie rende la programmazione Python più semplice e comoda per il programmatore poiché elimina la necessità di scrivere lo stesso codice più volte per programmi diversi.

Qui in elenco le librerie più rilevanti utilizzate per lo sviluppo del progetto:

PyTorch

PyTorch è un framework per il Machine Learning basato sulla libreria Torch, sviluppato inizialmente da Meta AI e ora parte della Linux Foundation, utilizzato principalmente per creare reti neurali [23]. Pytorch è una delle piattaforme preferite per la ricerca sul DL e il suo obiettivo è accelerare il processo tra la prototipazione della ricerca e la distribuzione. Le principali funzionalità chiave che PyTorch apporta allo sviluppo in Python sono il calcolo tensoriale, TorchScript, che permette di esportare modelli PyTorch in ambienti senza dipendenze Python, il calcolo con grafi dinamici e la differenziazione automatica.

CustomTkinter

CustomTkinter è una libreria UI (User Interface) desktop Python basata su Tkinter, che è il toolkit standard di Python per la creazione di interfacce grafiche. Questa libreria fornisce widget dall'aspetto moderno e completamente personalizzabili rimanendo coerente su tutte le piattaforme desktop.

Numpy

NumPy è la libreria fondamentale per l'elaborazione scientifica in Python, dove sono previsti utilizzo di array e matrici multidimensionali. È una libreria Python che fornisce un oggetto array multidimensionale, vari oggetti derivati e un assortimento di funzioni per operazioni rapide sugli array, tra cui matematica, logica, manipolazione delle forme, ordinamento, selezione, I/O, trasformazioni di Fourier discrete, algebra lineare di base, operazioni statistiche di base, simulazione casuale e molto altro. Al centro del pacchetto NumPy c'è l'oggetto `ndarray`. Questo incapsula array n-dimensionali con tipo di dati omogeneo, con molte operazioni eseguite in codice compilato prestazioni superiori.

Pandas

Pandas è una libreria fondamentale, basata su Numpy, per la manipolazione e l'analisi dei dati. Ciò è possibile grazie a strutture dati flessibili simili a fogli di calcolo, i DataFrame, che permettono di lavorare con dati tabulari in righe e colonne, con funzionalità avanzate di indicizzazione, selezione, raggruppamento e unione di dati in maniera efficiente e veloce. Inoltre semplifica operazioni comuni come la lettura e la scrittura di file in diversi formati, come CSV, SQL e JSON.

Matplotlib

Matplotlib è una libreria di visualizzazione dei dati che consente di creare grafici e diagrammi statici, dinamici o interattivi in Python per facilitarne l'analisi e la comprensione.

Sklearn

Scikit-learn, più brevemente sklearn, è una libreria di Machine Learning che fornisce una serie di algoritmi e metodi per la valutazione dei modelli di classificazione e regressione, clustering e l'analisi dei dati.

Visualtorch

VisualTorch è una libreria progettata per visualizzare le architetture delle reti neurali basate su Torch ed è in grado di rappresentarle utilizzando lo stile Layered, per una la visualizzazione gerarchica, Graph, per la una visualizzare più dettagliata delle connessioni tra i nodi o LeNet per i modelli ispirati ad essa.

Altre librerie

Oltre alle librerie sopracitate che sono state largamente usate e sono chiave del progetto di tirocinio, sono state utilizzate diverse altre librerie per aspetti secondari.

La libreria Threading costruisce interfacce per il multi-threading di alto livello sopra la libreria di basso livello `_thread`, offrendo anche possibilità di utilizzare ThreadPoolExecutor e queue per lo scambio di dati tra thread in esecuzione. Correlato a questo c'è concurrent che contiene solo il modulo `concurrent.futures` fornisce un'interfaccia di alto livello per l'esecuzione asincrona di funzioni chiamabili.

Traceback è una libreria utile per la gestione degli errori e il debugging, in quanto permette di tracciare l'origine delle eccezioni sollevate durante l'esecuzione del codice.

importlib facilita l'importazione dinamica di moduli.

PIL è una libreria utile per la manipolazione delle immagini, consentendo operazioni come il ridimensionamento, il ritaglio e la modifica del formato delle immagini. La libreria os fornisce funzioni per interagire con il sistema operativo, consentendo la gestione di file e directory.

collections implementa tipi di dati contenitore specializzati fornendo alternative ai contenitori incorporati per scopi generali di Python, dict, list, set e tuple.

3.1.2 Conda Environment

L'ambiente Conda è uno strumento avanzato per la gestione di pacchetti e ambienti virtuali utile per lo sviluppo di progetti complessi che richiedono molte dipendenze e necessità di tenere traccia delle versioni dei singoli pacchetti per garantire stabilità.

A differenza di venv, il sistema nativo di virtual environments di Python, Conda gestisce non solo i pacchetti Python, ma anche librerie e strumenti esterni al linguaggio, come librerie C/C++ e altre dipendenze di sistema utilizzando il packet manager conda che distribuisce i pacchetti in formato binario, precompilati e ottimizzati a differenza di pip di venv che gestisce i pacchetti in formato sorgente da compilare, il che può portare a problematiche.

Conda permette la creazione di un file di configurazione `environment.yml` che definisce le dipendenze di un ambiente specifico e facilita quindi la condivisione e la replica.

3.2 Git

Git è un sistema di controllo di versione open-source creato da Linus Torvalds inizialmente per il Kernel Linux, che permette di tenere traccia delle modifiche al codice sorgente nel tempo, rendendo facile il ripristino di versioni precedenti, il monitoraggio delle modifiche e la gestione di diverse ramificazioni indipendenti per lo sviluppo di nuove funzionalità. Git risulta particolarmente utile quando più persone lavorano su uno stesso progetto offrendo un sistema per l'unione dei file modificati con una gestione dei conflitti nel caso di modifiche sovrapposte.

Git permette inoltre la sincronizzazione con una repository in remoto hostata su un servizio in cloud, ad esempio il più famoso Github, per facilitare la condivisione del codice e la collaborazione con tutto il mondo, dando la possibilità di inviare richieste di modifiche e segnalare problemi.

Capitolo 4

Lavoro Svolto

4.1 Progettazione del Sistema

Il progetto iniziale prevedeva la realizzazione di un'interfaccia grafica intuitiva, pensata per semplificare e velocizzare l'applicazione dei metodi post-hoc da parte dei ricercatori. L'applicativo avrebbe incluso strumenti per la personalizzazione delle tecniche e l'esportazione dei risultati in formati che che ne agevolassero la manipolazione.

Durante la prima fase di progettazione, l'intenzione era che l'utente potesse caricare solamente il modello, l'insieme dei dati necessari per il test in formato `.csv` e un dataloader, basato su di essi, per fare pre-processing dei dati e darli in pasto al modello. L'applicazione poi si sarebbe occupata di fare l'inferenza, applicare i metodi non intrusivi, visualizzare dei grafici e restituire i risultati con le relative statistiche dando all'utente la possibilità di esportarli e di poterli utilizzare per i propri scopi.

Queste idee sono state integrate in un mock-up preliminare dell'applicazione, mostrato in figura 4.1, che rappresenta le funzionalità richieste nella fase di progettazione.

Oltre a specificare i componenti dell'interfaccia, c'è stata una particolare premura e attenzione a quelli che erano gli input e gli output. Fin dall'inizio è stato stabilito che il modello ricevuto in input dovesse essere stato già allenato poiché le operazioni implementate, essendo non intrusive, sarebbero state applicate esclusivamente dopo la fase di training. Il modello però era necessario che venisse fornito nel formato che contenesse non solamente i pesi, ma anche con la geometria e l'architettura della rete, poiché le tecniche utilizzate le richiedevano. Il dataloader invece, a differenza del dataset, essenziale da importare, era stato progettato per poter essere un parametro opzionale e che l'applicazione potesse gestire o meno la sua mancanza tramite l'utilizzo di un dataloader di default studiato per coprire il maggior numero di casi.

L'applicazione, nel momento in cui riceveva tutti i file necessari, avrebbe elaborato e reso visibile la rappresentazione grafica dell'architettura della rete neurale, con relative indicazioni su quali fossero i layer su cui era stato applicato il Dropout in fase di addestramento e successivamente dando la possibilità all'utente di poter

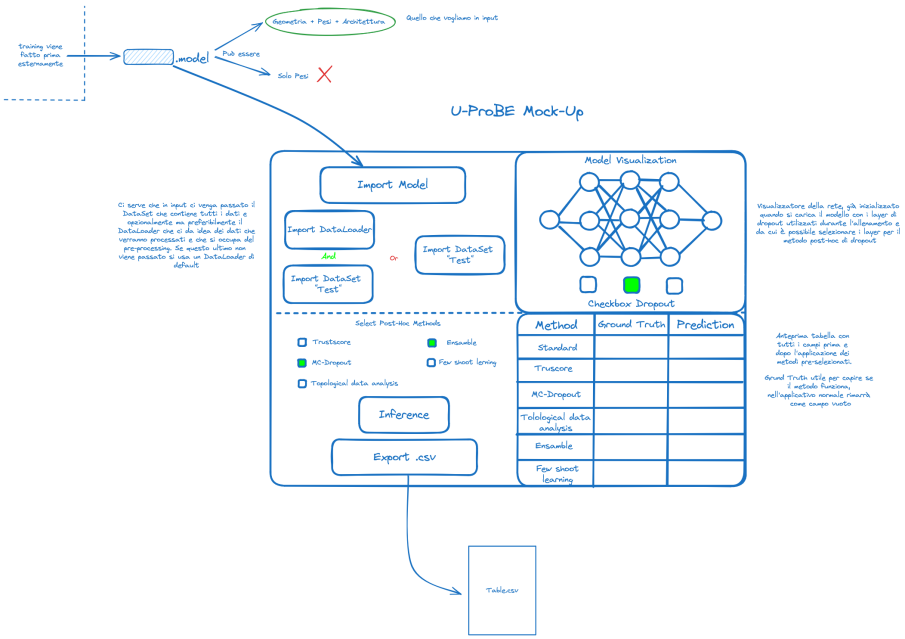

Figura 4.1: Primo Mock-Up di U-ProBE

specificare su quali layer fosse intenzionato ad applicare la tecnica dell'MC-Dropout in fase di inferenza.

Dopo aver preimpostato tutto, l'utente poteva a quel punto selezionare tramite delle apposite checkboxes i metodi post-hoc che desiderava applicare durante l'inferenza. Nella progettazione i metodi che si intendeva implementare erano il Trust Score, l'MC-Dropout, la Topological Data Analysis, l'Ensemble e il Few Shoot Learing.

Al termine dell'inferenza le statistiche più rilvevanti sarebbero state visualizzate all'interno della tabella della GUI mentre sarebbe stato possibile esportare su un file `.csv` i dettagli più specifici con i risultati per ogni test tramite l'apposito pulsante.

Durante lo sviluppo sono emersi comunque problemi tecnici e difficoltà di realizzazione di alcune funzionalità che hanno portato a modificare alcuni elementi decisi durante la fase di progettazione e intraprendere altre direzioni al progetto, mantenendo comunque il progetto coerente all'idea iniziale.

Data l'inesperienza anche dal punto di vista tecnologico, dopo aver progettato e visualizzato quello che c'era da fare, era il momento di scegliere la metodologia per realizzarlo.

Considerando che nel laboratorio di ricerca in cui si studiavano questi metodi e la mia totale inesperienza nell'utilizzo di framework di DL, è stato deciso di adottare PyTorch, guidati anche dal fatto che i primi utilizzatori dell'applicativo sarebbero stati proprio i membri del laboratorio stesso che avevano già familiarità con questo framework. PyTorch sulla carta risultava anche la scelta migliore poiché mettendolo a confronto con le sue principali alternative TensorFlow e Keras, poiché lavorando molto a basso livello, ci offriva la maggior flessibilità per fare operazioni complesse come quelle che erano necessarie per l'applicazione dei metodi post-hoc.

Un altro decisione cruciale da prendere prima di poter iniziare era quella della tecnologia da utilizzare per lo sviluppo dell'interfaccia grafica. La gamma di scelta per questo necessità era molto ampia ma per cercare di abbassare la barriera all'ingresso e per mantenere una coerenza con le altre tecnologie coinvolte, la scelta intrapresa è stata quella di sviluppare anche l'interfaccia grafica con l'utilizzo di Python, senza spostarci su altri strumenti, anche se più flessibili e professionali. Un requisito fondamentale per semplificare il lavoro futuro era anche quello che la tecnologia desse la possibilità di rendere l'applicazione multipiattaforma e non avesse problemi nella creazione di un eseguibile adeguato ad ogni piattaforma desktop, come ad esempio `.exe` per Windows o `.dmg` per macOS, per rendere più veloce e semplice l'eventuale distribuzione. Grazie a queste scelte siamo riusciti a stringere molto il campo delle tecnologie utilizzabili per lo sviluppo dell'interfaccia grafica, in particolare la scelta era arrivata a comprendere solamente le tecnologie CustomTKinter, BeeWare, PyQt, Kivy e Flet, ognuna con punti di forza e debolezze:

- **Flet**: Framework che permette di creare interfacce in Python utilizzando Flutter come motore di rendering, multipiattaforma con particolare attenzione all'ambito mobile, utilizza un approccio web-oriented e tendenzialmente risulta più lento per applicazioni desktop complesse
- **Kivy**: Framework per lo sviluppo di interfacce con focus principale sul touch e le gesture, supportando dispositivi desktop, mobili e anche embedded e con conseguente rischio di cali di prestazioni in applicazioni desktop complesse
- **BeeWare**: Strumento interessante che mette insieme più librerie per la creazione di applicazioni Python native ma ancora in fase di sviluppo attivo, non garantendo quindi la stabilità e la documentazione di cui il nostro applicativo ha bisogno
- **PyQt**: Strumento consolidato dai tanti anni di esistenza, basato sul potente framework Qt e di conseguenza ricco di funzionalità ma con una barriera all'ingresso relativamente alta rispetto agli altri framework e la necessità di installare strumenti come Qt Designer per progettare interfacce visivamente, fattore che disincentiva la possibilità di contribuire da parte di altri
- **CustomTkinter**: Libreria grafica che estende il toolkit grafico Tkinter, incluso di default in Python, il che offre come vantaggio il non dover installare dipendenze aggiuntive, con conseguente utilizzo di meno risorse, maggiore leggerezza e performance. Offre inoltre molti strumenti per lo sviluppo multipiattaforma e la distribuzione semplice su tutti i sistemi operativi desktop

Dopo aver analizzato i vantaggi e svantaggi delle tecnologie valutate, la scelta è ricaduta su CustomTkinter, ritenuto più adatto per quelle che erano le esigenze dell'applicativo, contraddistinto per la facilità nell'utilizzo, l'efficienza e lo stile grafico moderno delle componenti che mette a disposizione.

Definite queste scelte, l'unica cosa che rimaneva da fare era acquisire praticità con le tecnologie decise e iniziare a scrivere codice. Anche in questa fase l'approccio adottato è stato strutturato, cercando di creare l'applicativo seguendo una strut-

tura modulare e facilmente intercambiabile, con sezioni isolate le une dalle altre, comunicanti solamente attraverso delle interfacce dedicate.

4.2 Struttura della Cartella

La struttura delle directory è organizzata per facilitare la gestione modulare e rendere intuitivo l'accesso alle componenti principali del progetto. In questo capitolo indicheremo come **U-ProBE_Root** la directory principale del progetto, contenente tutte le cartelle e i file che lo compongono.

La struttura è organizzata come segue:

- **U-ProBE_Root**: Directory principale, al cui interno risiedono tutti i file e le cartelle necessarie per il progetto.
- **App**: Include il codice principale e le componenti modulari dell'applicazione.
- **Test**: Contiene al suo interno tutti i file necessari per eseguire i test dell'applicativo.
- **Assets**: Contiene al suo interno tutte le risorse grafiche che vediamo poi visualizzate nell'applicativo.

Di seguito, viene fornita una descrizione di ciascuna directory e dei file principali.

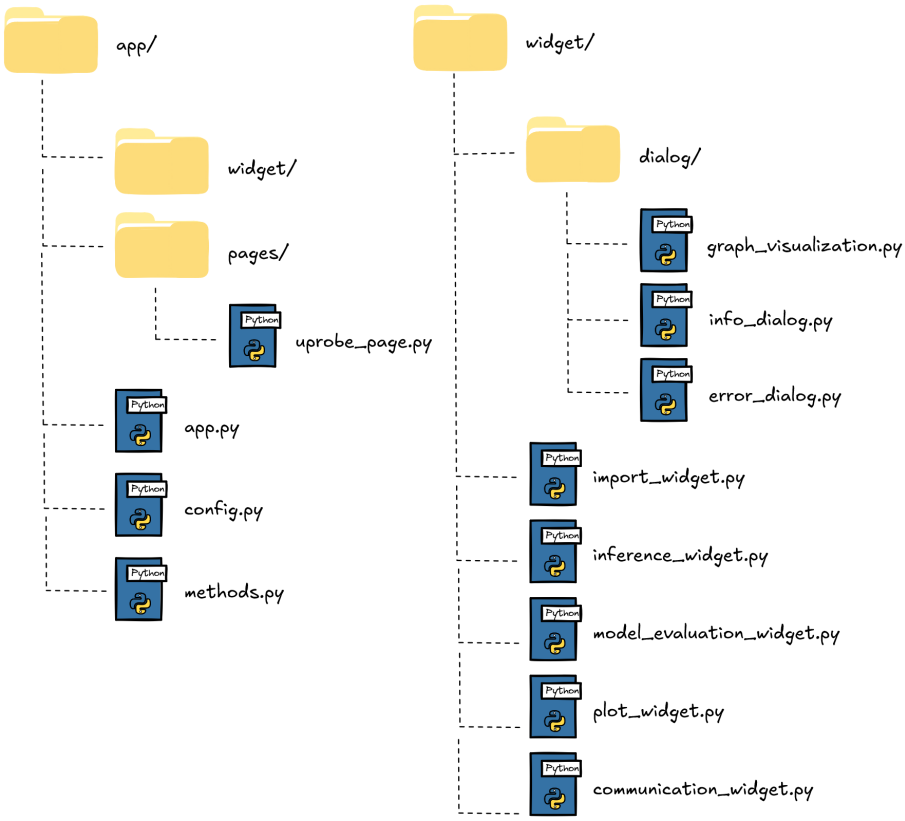

Figura 4.2: Struttura delle directory App e Widget di U-ProBE

4.2.1 Cartella `App`

La cartella `App`, come illustrato in figura 4.2, ha al suo interno solamente file Python con estensione `.py`, che comunicano per costruire e far funzionare l'interfaccia grafica. Questi file poi sono suddivisi tra di loro in cartelle per favorire una maggior comprensione del loro scopo, del loro funzionamento e della loro importanza.

- **App/**:
  - `app.py`: File di Boot per l'applicazione, si occupa di creare di definire gli iperparametri come dimensione della finestra principale, tema e icona per poi mostrare al suo interno la pagina di U-ProBE.
  - `config.py`: File di configurazione utilizzato per centralizzare le impostazioni principali del progetto.
  - `methods.py`: Contiene tutti i metodi e le relative funzioni che eseguono i metodi post-hoc.
- **App/Pages**: Questa cartella raccoglie tutte le pagine in cui è suddivisa l'applicazione. Attualmente contiene solo la pagina di U-ProBE ma con questa organizzazione sarà più facile in futuro aggiungere pagine e continuare ad ampliare le funzionalità dell'applicativo.
  - `uprobe_page.py`: Gestisce la visualizzazione della pagina principale dell'applicazione, contiene al suo interno tutti i widget relativi ad U-ProBE e si occupa di metterli in comunicazione tra di loro.
- **App/Widget**: Questa cartella raccoglie i vari widget che compongono la pagina di U-ProBE.
  - `import_widget.py`: Widget per l'importazione dei file necessari al funzionamento dell'applicativo.
  - `inference_widget.py`: Widget che si occupa di far selezionare ed eseguire i metodi post-hoc selezionati e successivamente esportare i risultati ottenuti.
  - `model_evaluation_widget.py`: Widget per la valutazione generale del modello e il riepilogo delle prestazioni dei metodi.
  - `plot_widget.py`: Widget per la visualizzazione grafica dei risultati delle inferenze e dei metodi applicati.
  - `comunication_widget.py`: Widget per la gestione delle comunicazioni tra componenti e utente.
- **App/Widget/Dialog**: Sottocartella contentente i componenti che aprono finestre esterne alla pagina principale.
  - `graph_visualization.py`: Finestra per visualizzazione grafica del modello e gestione di alcuni parametri per i metodi

– `error_dialog.py`: Mostra i messaggi di errore non banali con maggiori dettagli.

– `info_dialog.py`: Visualizza istruzioni e informazioni generali dell'applicativo.

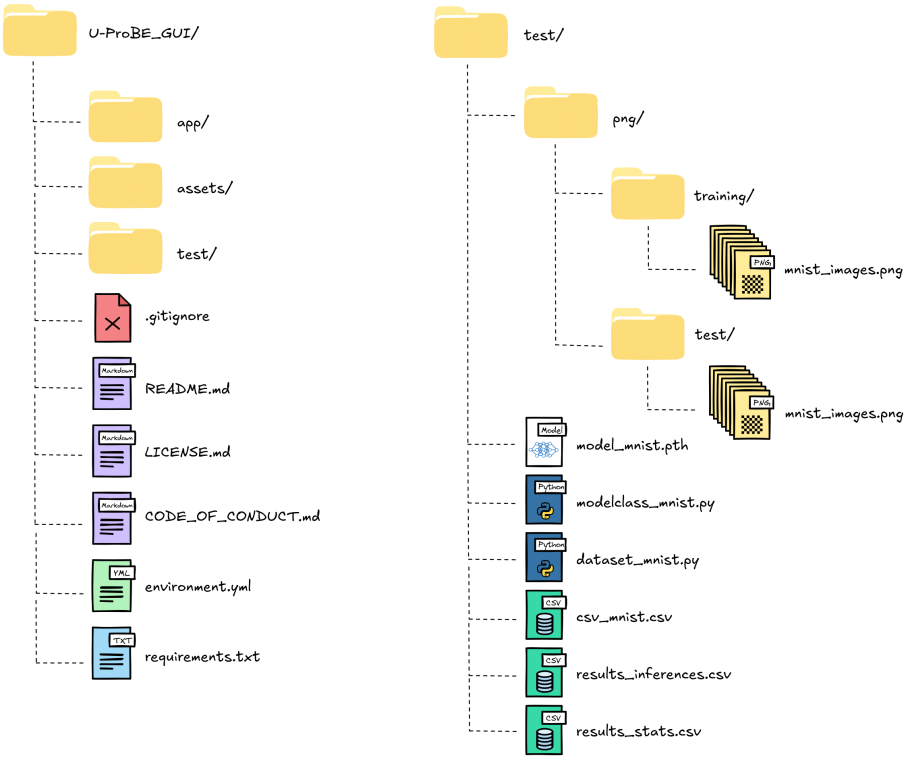

Figura 4.3: Struttura delle directory Root e Test di U-ProBE

4.2.2 Cartella `Test`

La cartella `Test` include tutti i file di test per verificare il corretto funzionamento del modello e dell'applicazione.

- `csv_mnist.csv`: Dataset di test per il modello MNIST. Contiene i path delle immagini poi contenute in **Test/png**.
- `modelclass_mnist.py`: File che definisce la struttura del modello.
- `dataset_mnist.py`: Scompone e pre-processa i dati a cui fare inferenza.
- `model_mnist.pth`: Pesi del modello addestrato.
- `results_inferences.csv` e `results_stats.csv`: File per i risultati delle inferenze e statistiche derivanti.
- **Test/png**: Contiene le immagini di training e di test in formato png per il dataset MNIST.

4.2.3 Directory Principale (Root)

La Root, o directory principale, oltre a tutte le cartelle già sopracitate, contiene file essenziali per la configurazione, il versionamento e la documentazione del progetto.

- `.gitignore`: Specifica quali file e cartelle escludere dal controllo di versione Git.
- `CODE_OF_CONDUCT.md`: Linee guida per il comportamento della comunità e la collaborazione.
- `LICENSE.md`: Termini di licenza per l'uso e la distribuzione del progetto. In particolare la licenza utilizzata è stata MIT, una licenza permissiva che consente l'uso, la modifica e la distribuzione del codice, anche per scopi commerciali, a condizione di mantenere il riconoscimento dell'autore originale.
- `environment.yml`: File di configurazione che definisce le dipendenze per l'ambiente virtuale con `conda`.
- `requirements.txt`: File per la gestione delle dipendenze Python.
- `README.md`: Documentazione principale, con istruzioni di installazione e uso.

4.3 Implementazione

4.3.1 Scelte Strutturali

CustomTkinter offre tre principali costrutti per la disposizione degli elementi nell'interfaccia grafica:

- **Place**: Permette di posizionare un widget a coordinate specifiche in pixel
- **Pack**: Dispone uno dopo l'altro in una direzione ad esempio da sinistra a destra oppure dall'alto verso il basso
- **Grid**: Fornisce una struttura più schematica e controllabile, dove è possibile decidere il numero di righe e colonne e decidere in quale cella posizionare gli elementi

Per una gestione più scalabile e seguendo i consigli della documentazione ufficiale di CustomTkinter, abbiamo optato per utilizzare esclusivamente il layout grid per tutto il progetto, trattandosi di un'applicazione di grandi dimensioni con la necessità di essere ben strutturata.

Un altro fondamentale di CustomTkinter da comprendere per utilizzare al meglio la grid è il concetto di weight, o peso, che definisce quanto una colonna si espanderà rispetto allo spazio disponibile e agli altri elementi con cui condivide la grid. Concetto simile è quello del column span che, a differenza del weight, specifica all'elemento su quante colonne può espandere il suo contenuto.

L'avvio dell'applicazione, la visualizzazione della pagina e la gestione degli iperparametri dell'app come dimensione della finestra e icona sono stati delegati tutti al file `app.py` che funge da Bootloader senza visualizzare direttamente alcun contenuto.

Il file che si occupa invece di contenere, organizzare e visualizzare i widget dell'applicazione è il file `uprobe_page.py`. Istanziando così tutti i widget all'interno di un singolo file, possiamo passare come parametro di un widget l'istanza di un altro, rendendo possibile, attraverso l'uso di interfacce predefinite, come semplici getter e setter, la comunicazione e lo scambio di informazioni o parametri.

La scelta di quali widget ricevevano il riferimento dell'istanza di altri widget come argomento è stata fatta seguendo il flusso di utilizzo delle risorse da parte dell'applicazione.

In aggiunta a queste componenti essenziali, è stato creato un widget per fornire comunicazioni all'utente, come feedback sulle azioni compiute dall'applicativo o segnalazioni di bug e problemi. Per facilitarne l'utilizzo, questo componente viene passato come parametro ad ogni widget della pagina ed utilizza un semplice metodo `set_message` per impostare il messaggio da far vedere all'utente con opzione per selezionare il colore e far capire in maniera più intuitiva la tipologia.

Infine per gestire in maniera centralizzata gli stili dell'applicazione così da poterli modificare semplicemente e velocemente, è stato utilizzato e importato negli altri moduli il file `config.py` che al suo interno contiene tutti i vari font e colori dell'applicativo. Questo file potrebbe tornare utile in futuro per la definizione di regole o metodi globali condivisi con tutti i file del progetto.

4.3.2 Problemi affrontati e soluzioni adottate

Estrazione delle classi necessarie dai file di input

Una delle prime sfide del progetto è stata quella di identificare correttamente la classe del dataloader dal file caricato dall'utente. Il problema principale consisteva nel fatto che la classe del dataloader era definita con un nome sconosciuto allo sviluppatore a priori, per cui non gli era possibile importare direttamente la classe in questo modo. L'unica soluzione trovata per risolvere questo problema è stata quella di porre come vincolo all'utente che nel file caricato, la classe che si voleva utilizzare venisse rinominata "CustomLoader". Sapendo questa informazione è stato possibile importare precisamente la classe desiderata, attraverso l'utilizzo della libreria `importlib`. La stessa soluzione è stata adottata successivamente al seguito della decisione di importare anche la classe del modello, ponendo come vincolo che si chiamasse "CustomModel".

Gestione dello spazio nell'interfaccia

Con il continuo sviluppo delle funzionalità e l'aggiunta di più sezioni, lo spazio disponibile è iniziato a risultare non sufficiente per quello che era da visualizzare. In particolare, nel momento in cui è nata l'intenzione di mostrare l'immagine della rete neurale, lo spazio rimanente all'interno della finestra non sarebbe bastato per sup-

portare questa funzionalità. Per ovviare al problema senza modificare le dimensioni della finestra e mantenendo l'applicazione su una singola pagina, è stato deciso di utilizzare una dialog, cioè un'altra finestra che si apriva all'occorrenza.

Nel corso dello sviluppo si sono presentate altre occasioni in cui era necessario ampliare lo spazio disponibile tramite dialog. Ciò è avvenuto per la realizzazione di una pagina per la visualizzazione dell'errore che ha semplificato e velocizzato il debug oltre ad aver aggiunto una funzionalità utile anche per gli utenti finali, e una pagina per la descrizione e le istruzioni dell'applicativo, che essendo testi di lunghe dimensioni, anche se all'interno di sezioni scrollabili, necessitavano di più spazio.

Visualizzazione della rete neurale

Una funzionalità che desideravamo implementare fin dall'inizio era quella della visualizzazione della rete neurale con annessa la possibilità di visualizzare su quali layer era stato applicato il Dropout in fase di training. Vista la quantità di funzioni da implementare e lo spazio già limitato dell'interfaccia, è stato deciso di adottare l'utilizzo di una dialog ausiliaria.

La visualizzazione del modello è stata realizzata grazie all'utilizzo di VisualTorch, un tool trovato su GitHub, apparentemente acerbo ma che comunque offre diverse possibilità visualizzazione delle reti come stratificato, grafo o LaNet. Per esaltare i layer e la loro divisione, la scelta presa è stata quella di utilizzare la visualizzazione con grafo. La sua implementazione però non è stata semplice e senza incorrere a problemi come pensato inizialmente: la dimensione dell'immagine variava molto in base al numero di layer, la soluzione è stata inserire un controllo per gestire la dimensione dell'immagine in base al numero di layer e scalarla conseguentemente per riuscirla a visualizzare interamente all'interno della finestra. Un secondo problema è stata la necessità imposta dalla libreria di inserire l'input shape del modello per visualizzare l'immagine.

Deduzione della `input_shape` per la visualizzazione del modello

Mentre con i modelli lineari era facile fare inferenza dell'input shape guardando il numero di feature in input, la situazione si complicava nel caso di CNN, nelle quali l'input shape è definita tramite la tupla (batch size, numero di canali, altezza immagine, larghezza immagine) dove il numero di canali assume valore 1 in caso di immagini su scala di grigi mentre 3 per immagini a colori RGB. Per ovviare a questo problema, dato che risultava impossibile inferire questi valori solamente osservando il modello, è stata implementata un'interfaccia che compare nel caso in cui l'applicativo non riesca a determinare la input shape, offrendo all'utente la possibilità di inserirla manualmente.

Estrapolazione degli hidden layer e identificazione dei layer con Dropout

Per soddisfare il requisito iniziale che il modello debba essere caricato non solo con i pesi, ma compreso di architettura e geometria, era stato scelto di richiedere in input il modello in formato TorchScript che, a differenza degli altri metodi di salvataggio,

ne rendeva possibile l'utilizzo in inferenza senza dover specificarne la classe. Per mostrare i layer che hanno ricevuto Dropout in training c'era bisogno di analizzare la struttura della rete. I primi problemi sono emersi durante il tentativo di manipolare il modello in formato TorchScript, che non consentiva di estrarre in modo semplice e privo di errori i layer su cui era stato applicato il Dropout. La soluzione ritenuta quindi più adatta, considerando soprattutto che in futuro sarebbe stato necessario intervenire sul modello in modo più approfondito e profondo, era quella di chiedere e ricevere dall'utente tutti gli strumenti per poter ricreare il modello internamente all'applicativo in PyTorch, così da poterlo manipolare con i suoi strumenti nativi. Il modello richiesto all'utente quindi passa da un formato TorchScript a un semplice modello con solamente i pesi, cioè salvato attraverso il comando `torch.save(model.state_dict(), PATH)`, e in aggiunta ad esso, doveva esser passata anche la sua classe che, analogamente a come era stato fatto per il dataloader, doveva essere rinominata "CustomModel" per poterla estrapolare correttamente dal file.

A questo punto erano presenti tutti i componenti necessari per ricostruire il modello, e da esso era possibile estrarre i layer su cui è stato applicato il Dropout in training. Per ogni layer, ad eccezione quello di input dove solitamente non si applica il Dropout per evitare di rimuovere feature in entrata, viene creata al di sotto del grafico una serie di checkboxes accompagnate dalle descrizioni dei layer, comprensive del tipo di layer, la dimensione dell'input e dell'output e in particolare dell'informazione relativa all'applicazione del Dropout o meno in fase di training. In caso positivo la checkbox risultava spuntata di default. L'utilità vera della checkbox infatti era quella di registrare l'intenzione dell'utente ad applicare l'MC-Dropout a uno o più determinati layer in fase di inferenza.

Gestione della tipologia di modello

Riguardo all'inferenza in sé, abbiamo riscontrato una difficoltà significativa legata alla generalizzazione delle varie parti del codice. Le situazioni possibili, determinate dalle scelte dall'utente, erano estremamente numerose e dipendevano da una varietà di variabili diverse. Gestire tutte le possibilità avrebbe richiesto uno sforzo non sostenibile. Anche semplificando al massimo, ci saremmo trovati davanti a trattare modelli di classificazione o modelli di regressione, che hanno già molti fattori diversi tra di loro come per esempio le statistiche che potevamo calcolare che per un classificatore potevano essere Accuracy e F1 Score, mentre per regressori MSE e R2 Score.

Tutto era reso più difficile dal fatto che in PyTorch non sia possibile provare a estrapolare questa informazione attraverso la sola osservazione del modello, quindi non ci è stata data neanche la possibilità di adattare il codice per le diverse occasioni. L'unica soluzione che rimaneva adottare era, anche in questo caso, quella di fissare uno standard e concentrarci su un tipo di modello alla volta. La scelta presa è stata l'utilizzo di modelli di classificazione di immagini, tipologia di modelli più utilizzata nel laboratorio di sviluppo del progetto e che quindi è stata ritenuta la scelta migliore.

Identificazione dei valori di output del dataloader

Un problema correlato alla varietà di modelli supportati è quello dei dataloader associati, che anch'essi possono restituire tramite `__getitem__` altrettanti output con shape differenti.

Anche in questo caso, la soluzione migliore è stata quello di adottare uno standard, in particolare dovevano essere gestite le colonne del dataset che erano state prefissate come obbligatorie e cioè quella della split, che serviva per indicare se un certo dato era di training oppure di test, e la GT. A questi dati si aggiunge l'immagine, restituita come tensore e ottenuta tramite una trasformazione. L'output finale sarà una tupla contenente, in ordine, l'immagine sotto forma di tensore, la stringa GT e la stringa split.

Consistenza dei dati tra i metodi

Un aspetto fondamentale che non era affatto scontato era la consistenza dei dati estratti. Infatti le incertezze venivano valutate sui risultati ottenuti tramite l'inferenza senza l'utilizzo di tecniche, e questi risultati dovevano essere condivisi tra tutti i metodi per calcolarne l'incertezza. Nel caso fossero stati necessari dati aggiuntivi da raccogliere per calcolare dei parametri utili per i metodi post-hoc, dovevano essere raccolti in quel momento. Questa motivazione ci ha fatto impostare il calcolo del `no_post-hoc_method` come obbligatorio prima di eseguire qualsiasi altro metodo. In aggiunta a questo, sono state create varianti che restituissero oltre che alle predizioni anche i dati di cui i metodi post-hoc selezionati avevano bisogno.

4.3.3 Implementazione metodi Post-Hoc

Prima di parlare delle tecniche post-hoc è necessario spiegare come avvengono le predizioni sulle quali stimare l'incertezza. La funzione che gestisce questa operazione, come tutte le altre metodologie, è situata in un file denominato `methods.py`, al di fuori delle classi che costruiscono l'interfaccia dell'applicazione. Le motivazioni principali sono quelle di separare i concetti dalla parte grafica dell'applicativo e soprattutto dando la possibilità quindi ai metodi di poter essere eseguiti in multi-threading.

La funzione in questione opera come una normale applicazione di un modello su un dataset. Riceve in input il modello e il dataloader, mette in modello in modalità valutazione con `model.eval()` e inizializza un array in cui verranno memorizzati i risultati finali. Successivamente, senza calcolare il gradiente, si itera sul dataloader che batch dopo batch, di dimensione imposta dall'utente, fornisce tutte le features.

Durante l'esecuzione si controlla la split prima di fare l'applicazione sul modello, in maniera da fare inferenza solo sui record che appartengono alla split di test. Infine viene finalmente eseguita la predizione e viene aggiunta all'array istanziato inizialmente, che verrà restituito al termine della funzione.

```
def no_post_hoc_method(model, dataloader):
    model.eval()
    inference_results = []

    with th.no_grad():
        for batch_features, _, split in dataloader:
            for feature, split_value in zip(batch_features, split
    ):
                if split_value == 'test':
                    outputs = model(feature.unsqueeze(0))
                    inference_results.extend(np.argmax(outputs,
    axis=1).cpu().numpy())

    return np.array(inference_results)
```

Come abbiamo già accennato nel capitolo precedente, anche in questo semplice metodo, le cose cambiano alla sola selezione di un altro metodo post-hoc poiché subentra la necessità di raccogliere altri dati durante l'inferenza. Nel caso venisse selezionato il Trust Score infatti, durante l'esecuzione della solita inferenza, viene realizzato un dataframe contenente tutti i valori dei nodi del primo layer fully-connected durante il passaggio dei nostri dati di test, utilizzando proprio la stessa funzione impiegata nel modulo del Trust Score per raccogliere gli stessi dati per i record di training.

```
def no_post_hoc_method_with_dataframe(model, dataloader):
    df_test = get_split_dataframe(model, dataloader, 'test')
    predicted = df_test['predicted'].to_numpy()
    num_tests = len(df_test)

    return predicted, df_test, num_tests
```

Dopo aver implementato il `no_post-hoc_method`, la concentrazione si è spostata sull'implementazione delle prime tecniche post-hoc.

Implementazione MC-Dropout

L'MC-Dropout è uno dei metodi di stima dell'incertezza più semplici e si basa sull'idea di applicare il Dropout durante la fase di inferenza, non solo durante il training, eseguendo molteplici passaggi forward con Dropout attivo per ottenere una distribuzione di predizioni da cui estrapolare l'incertezza osservando la variabilità dei risultati.

L'MC-Dropout è stato il primo metodo ad essere implementato ed è stato anche il metodo che ci ha fatto notare quanto questi metodi fossero onerosi nel loro piccolo e che migliorassero notevolmente con l'utilizzo di multi-threading.

Il metodo dell'MC-Dropout richiede in input il modello, il dataloader, l'array numpy contenente le predizioni precedentemente calcolate dal `no_post-hoc_method` e altri parametri utili per l'ottimizzazione dell'esecuzione che sono il numero di classi, utile per efficientare la creazione di numpy, e l'halting criterion threshold che imposta una soglia, decisa dall'utente dalla dialog per la visualizzazione della rete, utilizzata per terminare le iterazioni quando l'incertezza è successivamente bassa.

Anche se nella dialog con la visualizzazione della rete si offre la possibilità all'utente di poter scegliere i layer su cui poter applicare l'MC-Dropout, l'opzione non è attualmente funzionante, ma è pronta per implementazioni future. L'attuale metodo MC-Dropout applica il Dropout sui layer in cui è stato già utilizzato durante il training. L'attivazione di questi layer avviene tramite:

```
def enable_training_dropout(model):
    for m in model.modules():
        if isinstance(m, nn.Dropout):
            m.train()
```

Per migliorare le prestazioni, il metodo sfrutta il parallelismo, eseguendo molteplici `process_forward_pass()` in parallelo usando un `ThreadPoolExecutor` assegnando ad ogni thread un batch di dati.

All'interno di `process_forward_pass()`, il modello è impostato in modalità di valutazione tramite `model.eval()`, ma il Dropout viene forzato tramite la funzione `enable_training_dropout()`. Questo passaggio è cruciale, poiché, in fase di inferenza, il Dropout viene normalmente disabilitato per ottenere predizioni deterministiche ma nell'MC-Dropout, il Dropout deve essere mantenuto attivo, in modo da ottenere una distribuzione di predizioni diverse per ogni passaggio.

```
def process_forward_pass():
    predictions = np.empty((0, n_classes))
    model.eval()
    enable_training_dropout(model)

    softmax = nn.Softmax(dim=1)

    with th.no_grad():
        for images, _, splits in dataloader:
            test_indices = [i for i, split in enumerate(splits) if split == '
    test']
            test_images = [images[i] for i in test_indices]

            test_images_tensor = th.stack(test_images)

            outputs = model(test_images_tensor)
            outputs = softmax(outputs)

            predictions = np.vstack((predictions, outputs.cpu().numpy()))

    return predictions
```

Il codice itererà sui dati presenti nel dataloader e calcolando le predizioni solamente dei record di test. Le predizioni del modello vengono poi trasformate in probabilità tramite la funzione Softmax e accumulate su un array di predizioni che conterrà i risultati per ogni forward pass di ogni ciclo. Per arrestare il ciclo in anticipo valutiamo il valore dell'Halting Criterion che si basa sulla varianza delle medie delle predizioni e il numero di iterazione a cui ci troviamo.

```
1 def calculate_halting_criterion(predictions):
2     means = np.mean(predictions, axis=0)
3     variance_of_means = np.var(means, axis=0)
4     return np.median(variance_of_means) / len(predictions)
```

Per motivi di efficienza e per diminuire il rallentamento generale, questi calcoli vengono fatti solamente ogni 5 iterazioni saltando ovviamente la prima per impossibilità nel calcolare la varianza. Se l'Halting Criterion scende sotto una certa soglia allora il ciclo verrà arrestato.

Una volta che le predizioni sono state raccolte e il criterio di arresto è stato soddisfatto, si passa alla fase finale in cui vengono preparati i valori di ritorno. In questa fase, si calcola la media delle predizioni ottenute durante tutte le iterazioni, la quale rappresenta la probabilità media di ciascun campione di appartenere a ciascuna classe. A partire da queste medie, vengono estratti i valori corrispondenti alle classi predette dal `no_post-hoc_method`, in modo da mantenere coerenza con i dati già elaborati; questi ultimi verranno memorizzati in un array numpy.

Un altro valore che abbiamo voluto prendere in considerazione e che abbiamo ritenuto potesse essere rilevante, era la classe con la probabilità media più alta tra le iterazioni per ogni caso di test. Le classi scelte poi sono state inserite all'interno di un numpy array e restituite col nome di "MC-Dropout Predictions".

Il valore di ritorno del metodo consiste quindi in una tupla contenente questi due array numpy, che forniscono rispettivamente le probabilità per le classi predette e la classe con più probabilità per ogni campione.

Implementazione Trust Score

L'altro metodo implementato è stato quello del Trust Score, il cui codice è stato parzialmente tratto da un progetto di ricerca svolto all'interno di SiLab. La parte fornita definiva una classe denominata TrustScore, progettata per calcolare un punteggio di affidabilità associato alle predizioni di un modello. La classe prendeva in input i dati di riferimento, che nel nostro caso erano le predizioni di `no_post-hoc_method`, le classi corrette per ciascun dato, una metrica di distanza e, in caso di utilizzo della distanza k-nearest, il parametro k.

La classe TrustScore opera memorizzando i valori dei nodi di un determinato layer e le relative GT, che vengono utilizzati per creare insiemi strutturati per ciascuna classe. A seconda del metodo di distanza selezionato, queste strutture vengono

successivamente utilizzate per calcolare la vicinanza tra le predizioni del modello e i dati. La metrica di distanza è selezionabile tra quattro opzioni: `nearest` (la distanza minima), `average` (la distanza media), `centroid` (distanza dal centroide della classe), e `k-nearest` (media delle distanze dai k più vicini).

La funzione principale della classe, TrustScore, è progettata per calcolare il punteggio di affidabilità per una predizione specifica attraverso il metodo ausiliario `fun_TrustScore`, che implementa il calcolo del punteggio basandosi sul rapporto tra la distanza minima verso le classi non predette e la distanza verso la classe predetta. Se la classe predetta è più lontana rispetto alle altre, il punteggio di affidabilità sarà basso, segnalando una predizione potenzialmente meno affidabile.

Un aspetto mancante nel codice originario riguardava l'estrazione dei dati dai nodi del modello, necessaria per creare gli insiemi strutturati per ogni classe. Per implementare questa funzione abbiamo avuto bisogno di altri dati ed è il motivo dell'introduzione della colonna `split` che ha reso possibile distinguere i dati di training, che venivano utilizzati per creare i costrutti necessari al calcolo del Trust Score ma sui quali non veniva eseguita la predizione, dai dati di test sui quali calcolare il Trust Score. Inizialmente, per gestire la complessità di questa struttura, era stato preso in considerazione l'uso di un file `.csv`, in modo da poter consultare anche successivamente i valori dei nodi, ma le operazioni di lettura e scrittura sul file avrebbero rallentato eccessivamente le prestazioni. La soluzione scelta quindi è stata l'utilizzo di un dataframe per memorizzare i dati di training contenuto interamente in memoria interna.

Il layer scelto per l'estrapolazione dei dati dai nodi della rete è il primo layer fully-connected, solitamente contenente dati più rilevanti al fine delle buone performance del metodo, rispetto agli altri layer [12]. Le seguenti funzioni sono utilizzate per estrapolare i valori dai nodi di questo layer. La prima funzione, riportata di seguito, accetta come input il modello e un dato campione, restituendo i valori dei nodi del primo layer fully-connected:

```
def get_fc_output(model, x):
    for _, module in model.named_modules():
        if isinstance(module, th.nn.Linear):
            return module(x)
    return None
```

La seconda funzione, invece, elabora un'intera batch di dati e restituisce, per ciascun elemento, una tupla contenente la GT iniziale, la predizione del modello e i valori dei nodi del primo layer fully-connected:

```
def process_batch(model, feature, gt_value):
    feature = feature.unsqueeze(0)
    feature_flat = feature.view(feature.size(0), -1)

```

```
    fc_output = get_fc_output(model, feature_flat)
    if fc_output is not None:
        final_output = model(feature_flat)
        predicted = th.argmax(final_output, dim=1).item()

        return calculate_results_row(gt_value, predicted,
fc_output)
    return None
```

Mentre il dataframe dei dati di training deve essere creato, quello delle predizioni è disponibile poiché, nel momento in cui si seleziona il metodo Trust Score tra quelli da eseguire, al posto della funzione `no_post-hoc_method`, viene eseguita `no_post-hoc_method_with_dataframe`, che oltre alla normale inferenza, si occupa di raccogliere i dati e restituire il dataframe con i valori dei nodi del primo layer fully-connected

Per ottenere entrambi i dataframe viene utilizzata questa stessa funzione con unica differenza il campo `target_split` che in un caso sarà test e nell'altro sarà training:

```
def get_split_dataframe(model, dataloader, target_split):
    model.eval()
    results = []

    with th.no_grad():
        for batch_features, gt, split in dataloader:
            for feature, gt_value, split_value in zip(
batch_features, gt, split):
                if split_value == target_split:
                    row = process_batch(model, feature, gt_value)
                    if row is not None:
                        results.append(row)

    return pd.DataFrame(results)
```

Una volta che abbiamo entrambi i dati, la classe TrustScore viene istanziata e i dati vengono processati iterativamente per ciascun record di test. Al termine dell'elaborazione, la funzione restituisce un array numpy contenente i punteggi di Trust Score per ogni predizione.

Estrarre i valori dal primo layer fully-connected generalmente porta a risultati migliori. Tuttavia, in prospettiva futura, potrebbe essere interessante sviluppare una funzionalità, simile a quella per la selezione dei layer per l'MC-Dropout, che consenta all'utente di selezionare il layer specifico da cui estrarre i dati per il calcolo del Trust Score.

4.4 Analisi del Prodotto Finale

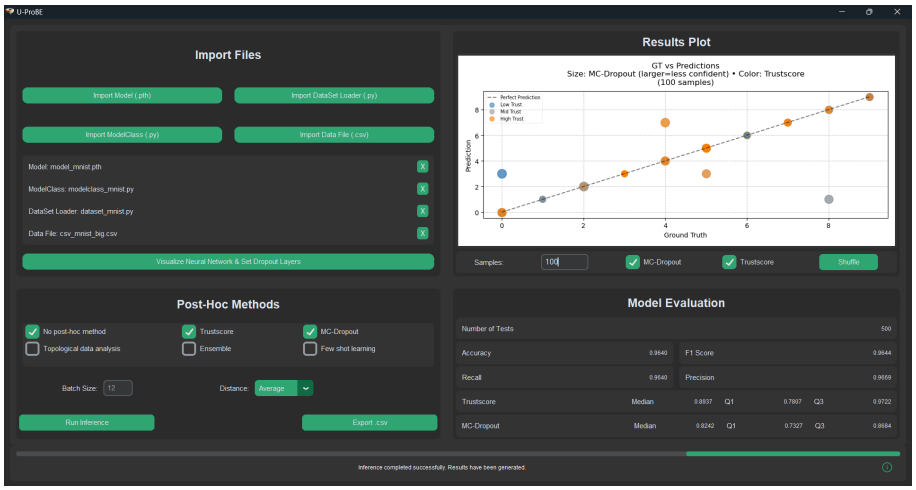

Figura 4.4: Immagine d'insieme U-ProBE

In questa sezione verrà presentato il prodotto finale, illustrandone il funzionamento e le principali componenti attraverso immagini e descrizioni delle funzionalità, adottando la prospettiva dell'utente finale per evidenziarne l'utilizzo pratico.

4.4.1 Sezioni Principali

L'applicativo, adottando il layout a griglia di CustomTkinter, si presenta con quattro sezioni ben chiare e delineate, ognuna con utilizzi e scopi differenti, capaci di comunicare tra di loro condividendo dati e risultati e aggiornandosi dinamicamente e in contemporanea all'avvenimento di eventi o azioni da parte dell'utente.

Import Section

L'import section, posizionata in alto a sinistra, è la prima sezione con cui l'utente avrà a che fare. Quasi autoesplicativa, permetterà all'utente di caricare tutti i file necessari per il corretto funzionamento dell'applicativo, in particolare:

- il file del modello PyTorch con estensione `.pt`, precedentemente salvato tramite il comando `torch.save(model.state_dict(), PATH)` che salva solamente i pesi del modello.
- il file contenente la classe del modello con estensione `.py`. Il nome della classe del modello deve essere impostato come "CustomModel" per essere utilizzato.
- il file contenente il dataloader per i dati con estensione `.py`. Il nome della classe del modello deve essere impostato come "CustomLoader" per essere utilizzato.
- il file con estensione `.csv` contenente i dati da processare tramite dataloader.

Durante l'importazione avvengono i controlli di importazione per cui non è possibile importare file con estensione diversa da quella specificata e i requisiti di esistenza

delle classi "Custom" vengono verificati all'importazione e in caso non esistessero nei file importati, quest'ultimi non verranno salvati.

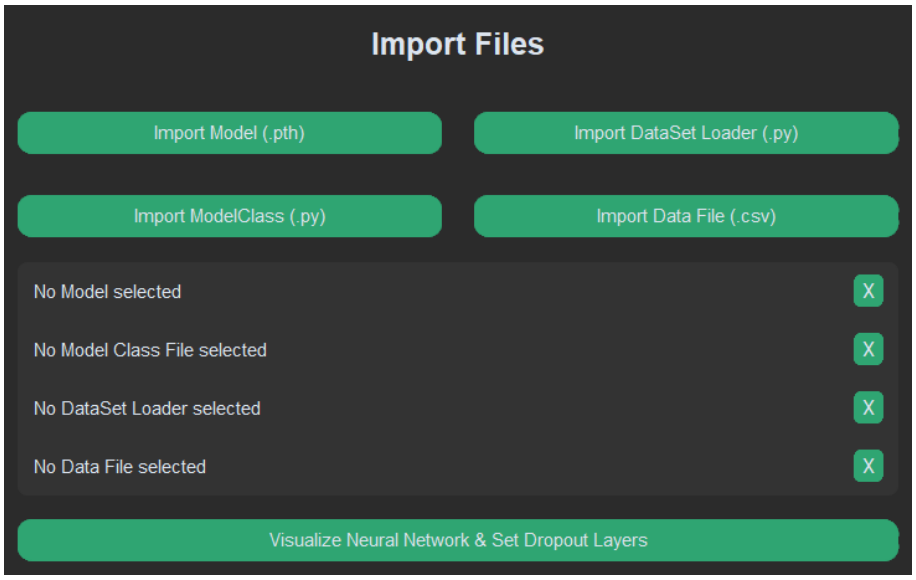

Figura 4.5: Import Widget

Al di sotto dei pulsanti per l'importazione abbiamo una sezione che indica gli attuali file selezionati, accompagnati da una "X" per l'eventuale riassegnamento di un determinato file.

Infine la sezione si conclude con un pulsante che ci permette, dopo aver caricato anche solo la classe del modello, l'apertura di una dialog per la visualizzazione della rete e la selezione di alcune operazioni specifiche.

Graph Visualization

La graph visulization è una dialog che si apre alla pressione del tasto di visualizzazione del grafo situato in fondo alla import section e si presenta come altra finestra rispetto a quella dell'applicativo principale.

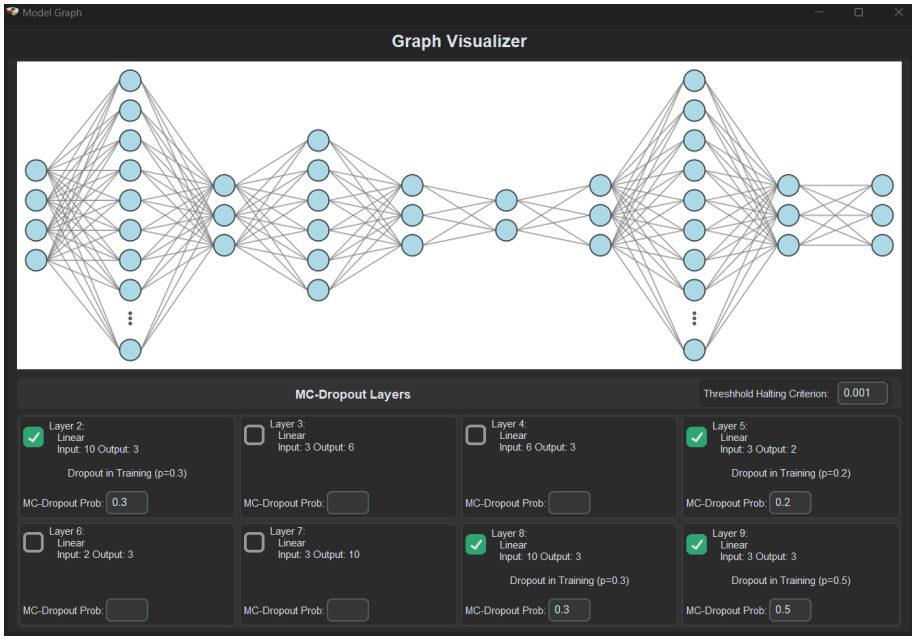

Figura 4.6: Graph Visualizer

Ha come scopo quello di visualizzare graficamente la rete neurale e analizzarne la struttura tramite le informazioni che vengono offerte. Infatti, oltre che visualizzarlo, viene eseguito uno scan del modello e create delle checkboxes per ogni hidden layer, accompagnate da una semplice descrizione. Le checkboxes, spuntate di default sulle caselle corrispondenti agli hidden layer che hanno ricevuto Dropout durante il training, consentono all'utente di selezionare i layer su cui applicare l'MC-Dropout e permettono inoltre di specificare la probabilità desiderata per ciascun layer; funzione che, pur essendo già stata implementata, non è ancora utilizzabile nell'applicazione dell'MC-Dropout. Oltre a questo, in questa pagina è possibile impostare il valore di Threshold per l'Halting Criterion che l'utente ha deciso di utilizzare durante l'esecuzione dell'MC-Dropout e che indica a che valore di Halting Criterion vogliamo terminare l'inferenza; più il valore è basso e più l'inferenza sarà dispendiosa e duratura, con probabili risultati più precisi.

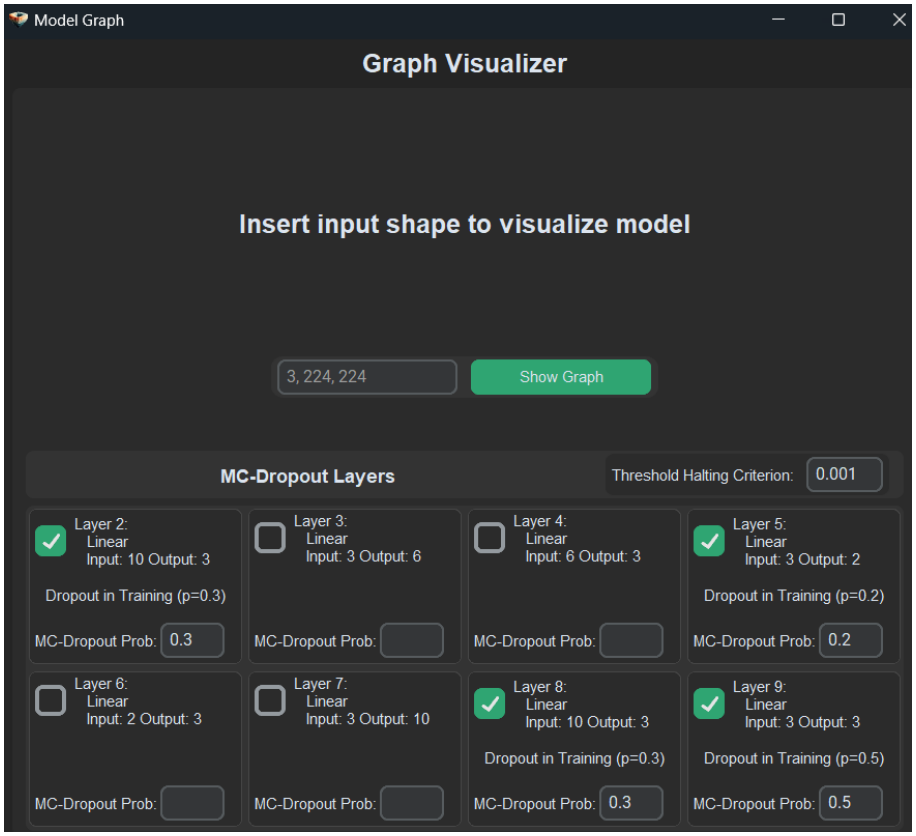

Figura 4.7: Graph Visualizer con inferenza della input shape senza successo

L'immagine viene creata a partire dal modello attraverso una libreria che necessita l'inserimento della input shape. Questo valore si tenta di dedurlo dalla struttura del modello, ma, se questo non fosse un successo, si chiede all'utente di inserirlo manualmente attraverso la sezione di input dedicata.

Inference Section

La sezione dedicata all'inferenza ha come funzione quella di permettere all'utente di selezionare i metodi post-hoc da utilizzare sugli input inseriti precedentemente nella import section.

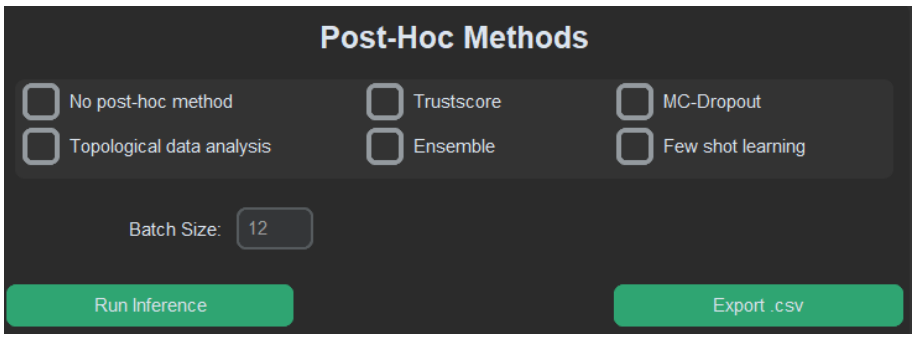

Figura 4.8: Inference Widget

I metodi post-hoc selezionabili sono il Trustscore, l'MC-Dropout, la Topological data analysis, Ensemble e Few shot learning. Tra questi, gli unici ad essere stati implementati e funzionare sono Trustscore e MC-Dropout, mentre gli altri, al momento, restituiscono valori casuali, in attesa di sviluppi futuri. Nessuno di questi metodi è selezionabile senza aver selezionato prima il No post-hoc method, che è semplicemente l'esecuzione normale dell'inferenza sulla quale gli altri metodi calcoleranno tutte le incertezze. Se si seleziona un altro metodo senza aver selezionato il No post-hoc method, quest'ultimo verrà selezionato automaticamente, e viceversa, se No post-hoc method viene deselezionato, tutte le checkboxes si deselezionano.

Selezionando la checkbox per il metodo Trustscore, comparirà sullo schermo un menù a tendina che permetterà all'utente di selezionare il metodo di distanza che preferisce utilizzare durante l'applicazione del Trustscore tra Nearest, Average, Centroid e K-Nearest.

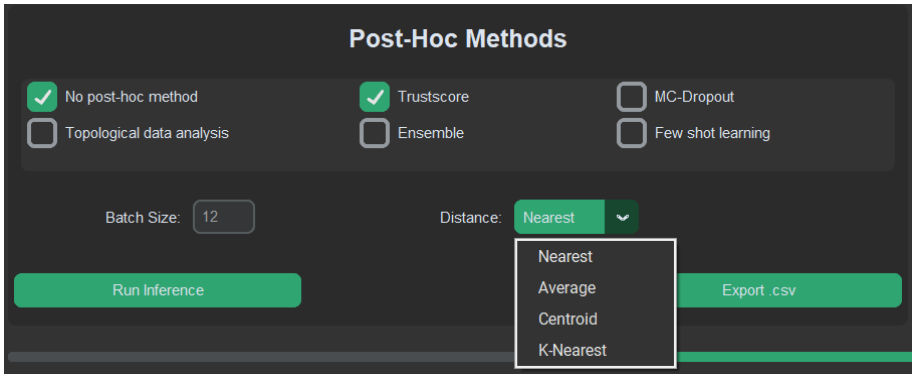

Figura 4.9: Opzioni Trustscore nell'Inference Widget

Nel caso in cui venisse selezionato quest'ultimo, comparirà un altro campo di input per poter impostare il valore k che più si preferisce per il K-Nearest.

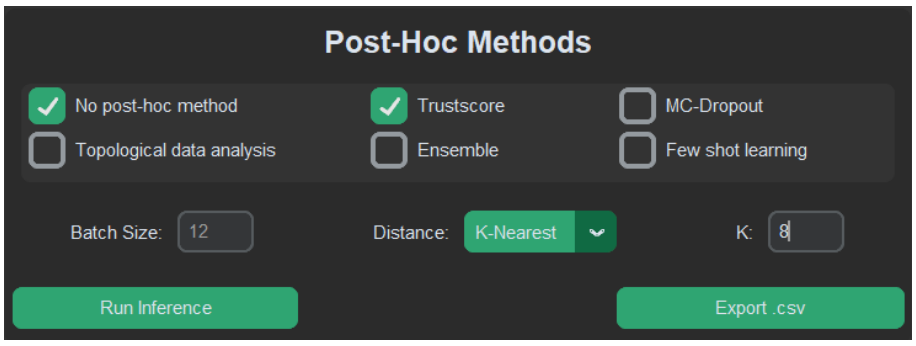

Figura 4.10: Opzione Trustscore K-Nearest nell'Inference Widget

Un parametro importante per l'inferenza è la grandezza della batch size che, essendo un valore altamente legato alla macchina su cui viene fatto girare il programma e al tipo di dato su cui dobbiamo lavorare, è stato reso modificabile per adattarlo ad ogni esigenza.

L'inferenza non può iniziare se non sono stati importati tutti i file necessari o non è stata selezionata alcuna modalità e il pulsante di esportazione darà errore in caso si provi ad esportare senza aver fatto prima l'inferenza.

Selezionati i metodi che vogliamo utilizzare e i vari parametri, possiamo far partire l'inferenza tramite l'apposito pulsante "Run inference" e potremo visionare il corretto andamento dell'inferenza tramite la barra di caricamento situata sulla Comunication Section in fondo all'applicativo. Al termine dell'inferenza la barra di caricamento si fermerà e comparirà un messaggio positivo in caso l'inferenza termini con successo. In caso di fallimento comparirà sullo schermo la Error Dialog che fornirà una spiegazione dettagliata dell'errore.

Al termine dell'inferenza sarà possibile esportare i risultati su file con estensione `.csv` tramite l'apposito pulsante. In particolare, i file che saranno esportati saranno due: uno con le statistiche di accuratezza, F1, recall e precisione in fase di inferenza normale e uno con le statistiche dettagliate dei risultati per ogni record di test passato in input.

Plot Section

La plot section è stata pensata come una sezione nella quale l'utente potesse visualizzare i risultati dell'inferenza in maniera veloce e intuitiva, per avere una panoramica sull'andamento del proprio modello e delle tecniche post-hoc utilizzate.

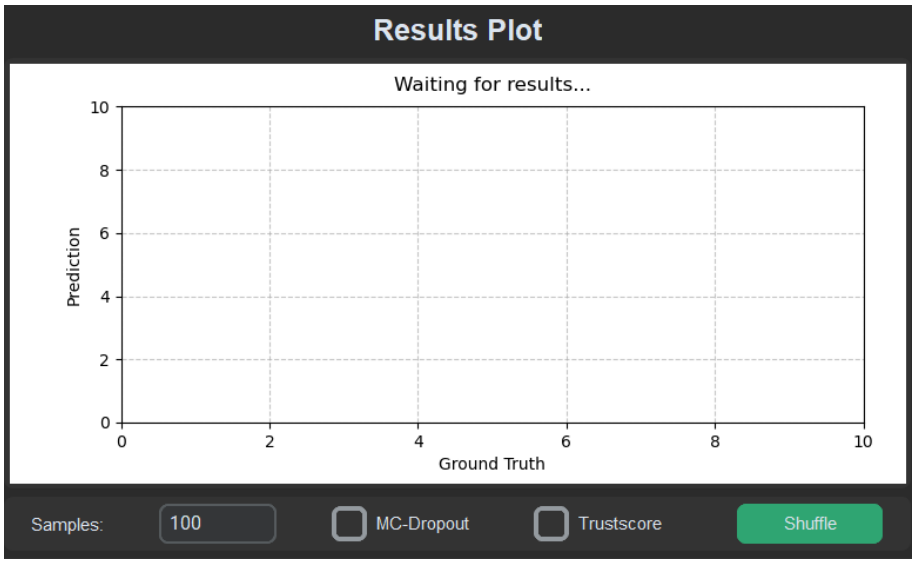

Figura 4.11: Plot Widget

Quando l'applicazione viene inizializzata il grafico risulta vuoto e non interagibile. Il grafico viene popolato automaticamente solo al completamento con successo dell'inferenza.

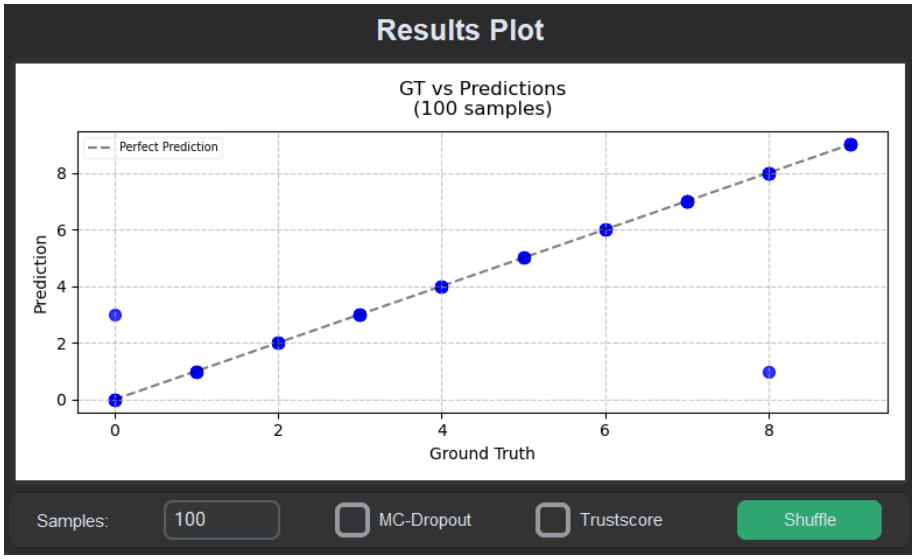

Figura 4.12: Plot Widget dopo inferenza

Il grafico presenta sull'asse delle ascisse la GT e sull'asse delle ordinate le predizioni del modello. Il nome e la lunghezza degli assi cambia dinamicamente all'occorrenza. Per caso singolo, si ha un buon risultato quando il valore della GT è uguale a quello della predizione, cioè il modello è riuscito a predire con successo la classe reale che è quindi uguale alla GT. Le predizioni giuste vengono posizionate sulla diagonale del grafico, dove la GT=Predizione, come indicato da legenda.

La parte alta del grafico è utilizzata per mostrare le informazioni essenziali per comprendere il grafico come ad esempio il numero di sample rappresentati.

Il numero di elementi che vogliamo rappresentare sul grafico è a discrezione dell'utente che può indicarlo tramite l'apposita sezione di input. I sample scelti saranno presi casualmente dall'insieme dei risultati dell'inferenza ed è possibile estrarne nuovi ogni volta che si vuole tramite il tasto "Shuffle".

Inoltre, se durante l'inferenza è stato applicato l'MC-Dropout oppure il Trustscore, è possibile applicare delle modifiche grafiche per visualizzare l'incertezza.

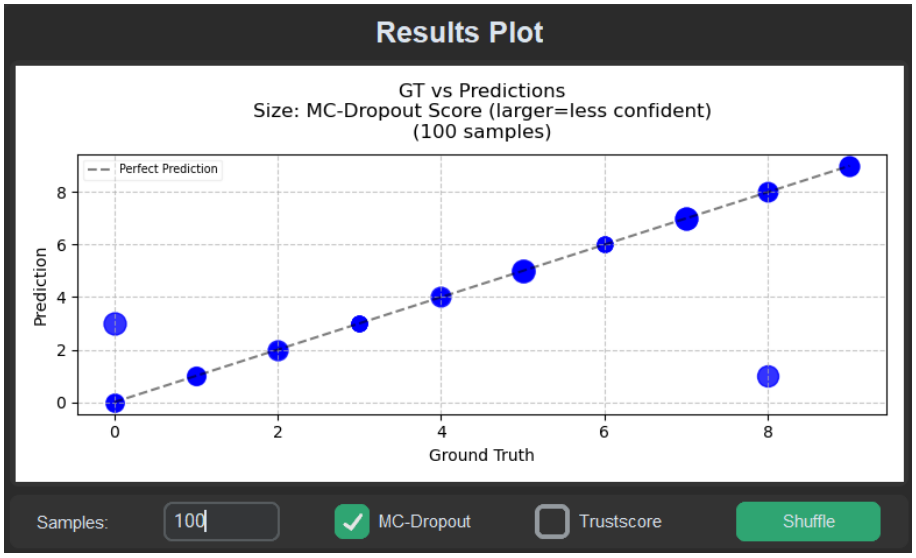

Figura 4.13: Plot Widget con visualizzazione MC-Dropout

Nel caso venisse selezionato di visualizzare l'MC-Dropout, l'incertezza calcolata da questo metodo è suggerita graficamente tramite la grandezza dei punti. In particolare più un punto è grande e meno siamo confidenti con la predizione.

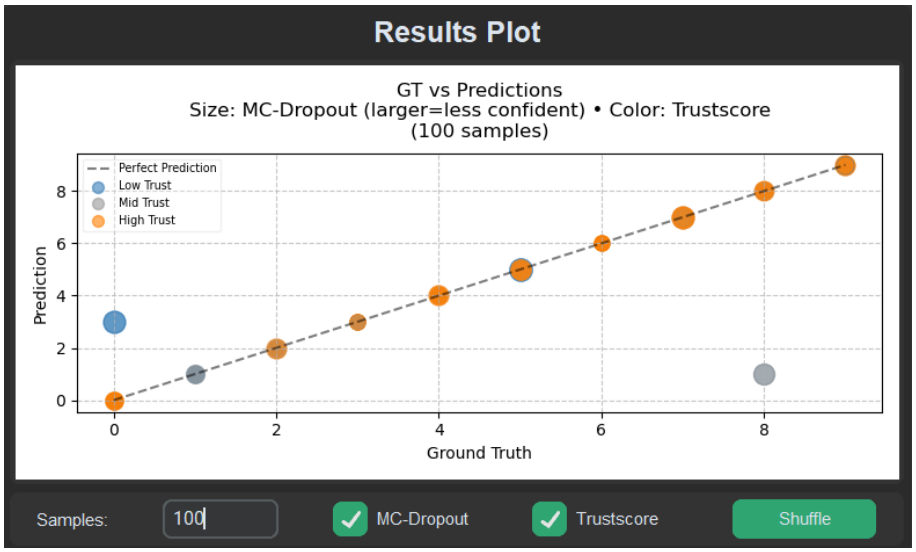

Figura 4.14: Plot Widget con visualizzazione Trustscore

Nel caso invece che venisse selezionato di visualizzare il Trustscore, l'incertezza calcolata è rappresentata sotto forma di un gradiente che parte dal blu, nel caso ci sia incertezza e si muove verso il arancione, dove il valore di incertezza è più basso, come mostrato nella legenda posizionata in alto a sinistra.

Model Evaluation Section

Per visualizzare sull'applicativo alcuni dei dati numerici più rilevanti riguardanti il modello e i metodi applicati è stata creata la sezione per la valutazione del modello.

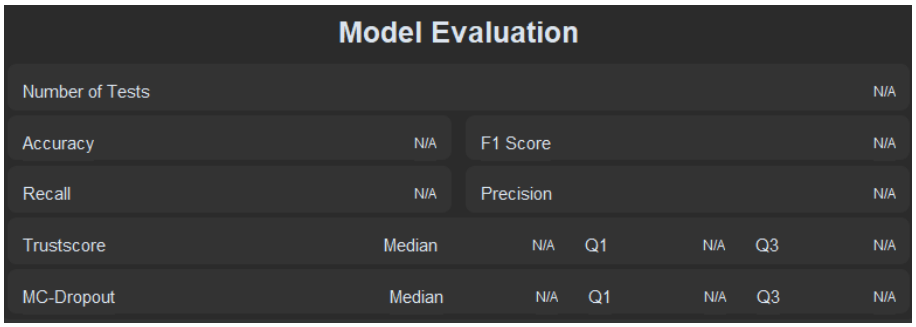

Figura 4.15: Model Evaluation Widget

Al termine dell'inferenza, la tabella viene automaticamente popolata, fornendo una panoramica immediata delle prestazioni in termini numerici del modello.

Le statistiche calcolate per il modello includono:

- **Accuracy**: rappresenta la percentuale di predizioni corrette rispetto al totale delle predizioni effettuate
- **Recall**: misura la capacità del modello di identificare correttamente tutti i casi positivi, evitando di perderne qualcuno
- **Precision**: indica la proporzione di predizioni corrette tra tutte quelle che il modello ha classificato come positive
- **F1 Score**: metrica che bilancia precisione e recall, utile quando è importante considerare sia gli errori di inclusione che quelli di esclusione

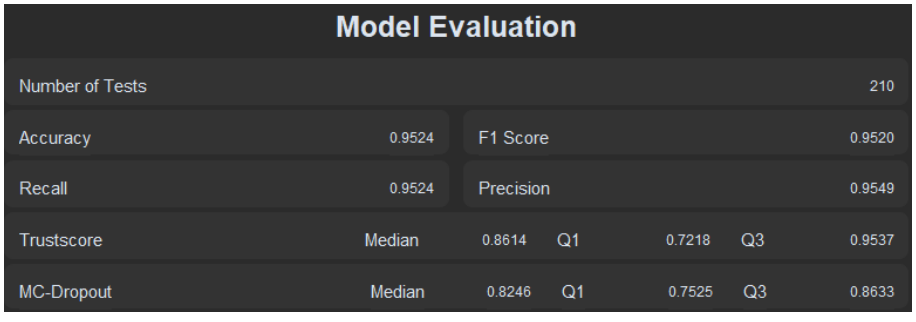

Figura 4.16: Model Evaluation Widget dopo l'inferenza

Per i metodi post-hoc, viene fornita una rappresentazione del range di variabilità dei risultati mostrando la mediana, il primo quartile e il terzo quartile.

Per contestualizzare le analisi, viene anche riportato il numero di dati di test utilizzati nel calcolo di queste statistiche.

4.4.2 Componenti di Comunicazione

I componenti di comunicazione sono elementi semplici ma essenziali all'interno dell'interfaccia, che si occupano di gestire lo scambio di informazioni tra l'applicativo e l'utente, mutando aspetto e comportamento in base alle sue azioni.

Comunication Widget

Il comunication widget è un elemento centrale utilizzato da tutte le componenti dell'applicativo per mostrare semplici messaggi a scopo di feedback o messaggi di errori non gravi se il comportamento dell'utente non segue le linee guida imposte.

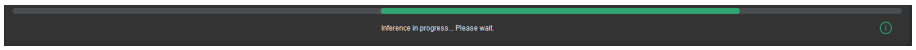

Figura 4.17: Communication Widget

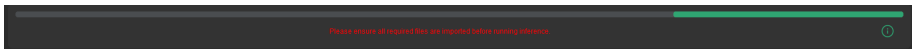

Figura 4.18: Communication Widget Errore

Al di sopra del widget è posizionata una barra di caricamento indeterminata, cioè che non rappresenta una percentuale precisa dello stato del caricamento ma comunica all'utente che è in corso un operazione. Tale viene azionata insieme all'inferenza e termina al suo completamento con l'intento di comunicare all'utente lo stato delle attività in corso.

Error Dialog

Nel caso si presentino errori inaspettati, che il Communication Widget non è in grado di gestire poiché esulano dagli errori comuni, viene aperta una dialog dedicata che ha lo scopo di fornire informazioni dettagliate sull'errore, aiutando l'utente a comprendere la natura del problema.

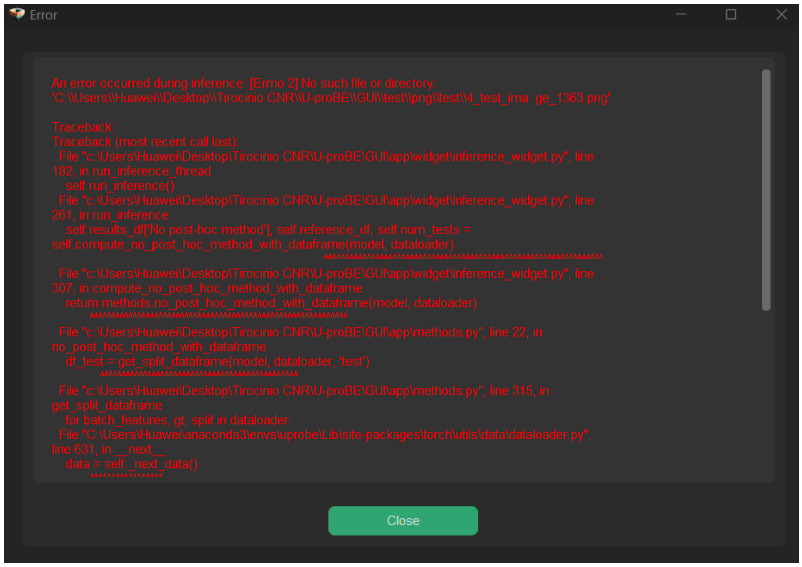

Figura 4.19: Error dialog

Info Dialog

Tramite il pulsante posizionato sul comunication widget, identificato dall'icona delle info, è possibile aprire una dialog per consultare lo scopo e il metodo di utilizzo dell'applicativo.

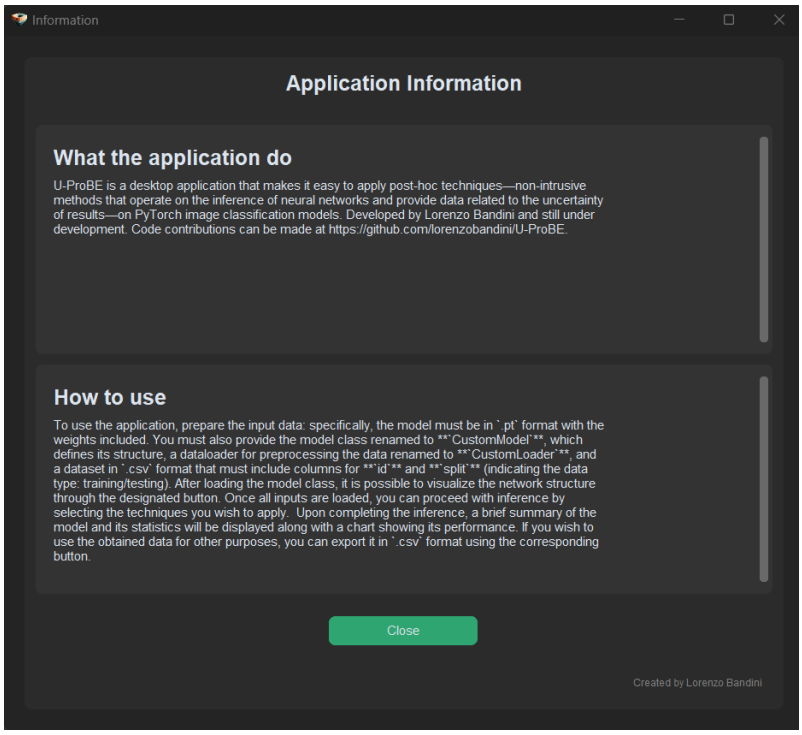

Figura 4.20: Info Dialog

4.5 Testing

4.5.1 Descrizione di MNIST

Il dataset MNIST è un insieme di dati che contiene 70.000 immagini di cifre scritte a mano, suddivise in 60.000 campioni di addestramento e 10.000 campioni di test. Ogni immagine è in scala di grigi, con dimensioni di 28x28 pixel, e rappresenta una cifra compresa tra 0 e 9. Questo dataset è stato creato per fornire una base di riferimento per la valutazione delle prestazioni degli algoritmi di riconoscimento delle immagini, come in Figura 4.21.

MNIST è ampiamente utilizzato per testare algoritmi di classificazione e riconoscimento, grazie alla sua facilità d'uso e alla disponibilità di un ampio numero di campioni [18]. Il dataset è perfetto per sviluppare e testare modelli di DL, in particolare reti neurali convoluzionali (CNN), poiché permette di valutare l'accuratezza dei modelli.

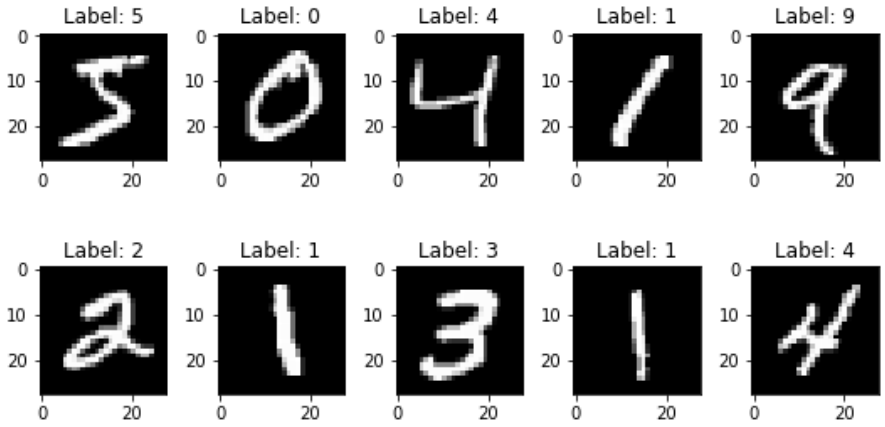

Figura 4.21: Esempio di immagini dal dataset MNIST.

4.5.2 Preparazione Modello e Dataloader

Per far funzionare l'applicativo abbiamo detto che gli dobbiamo fornire il modello con i pesi in formato `.pth`, per ciò ci servirà solamente salvare il codice con il comando:

```
1 torch.save(model.state_dict(), PATH)
```

Oltre a questo sarà indispensabile per il funzionamento dei metodi anche il caricamento del file contenente la classe del modello in formato `.py` in modo da poter ricostruire il modello e riutilizzarlo. Per poter selezionare la classe dall'interno di questo file è però necessario che la classe venga rinominata "CustomModel" in modo che il nostro applicativo possa importarla correttamente. Questo è la classe del modello MNIST utilizzata per fare testing:

```
import torch.nn as nn
import torch.nn.functional as F

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.dropout1 = nn.Dropout(p=0.2)
        self.fc3 = nn.Linear(64, 10)
        self.dropout2 = nn.Dropout(p=0.2)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.dropout2(x)
        return F.log_softmax(x, dim=1)
```

Oltre ai pesi e la classe del modello ci sarà bisogno di dare in input anche un dataloader che specifichi come estrapolare dal file `.csv` i dati e come preprocessarli per adattarli all'input richiesto dal modello. Analogamente alla classe del modello, anche qui sarà necessario chiamare la classe con un nome che sia noto all'applicativo. Il nome della classe scelto che dovrà essere usato è "CustomLoader". Il dataloader utilizzato è il seguente:

```
from torchvision import transforms
from PIL import Image
from torch.utils.data import Dataset

class CustomLoader(Dataset):
    def __init__(self, csv_file):
        self.data = csv_file
        self.transform = transforms.Compose([
            transforms.ToTensor(),
        ])

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        # Carica l'immagine e la trasforma in tensor
        img_path = self.data.iloc[idx]['image_path']
        image = Image.open(img_path)
        if self.transform:
            image = self.transform(image)

        # Ottieni l'etichetta come stringa
```

```
23          label = self.data.iloc[idx]['label']
24
25          split = self.data.iloc[idx]['split']
26
27          return image, label, split
```

Questo dataloader è progettato per leggere da un file `.csv` passato in input tutte le informazioni necessarie per l'inferenza. Il file deve includere una colonna ID, per identificare e riutilizzare successivamente il record, un path che specifica il percorso dell'immagine, la GT e una colonna split che indichi se il determinato record è di "test" o di "training". Qui un esempio:

ID	Path	Ground Truth	Split
1	/path/to/image1.png	7	test
2	/path/to/image2.png	3	training
3	/path/to/image3.png	5	test

Tabella 4.1: Esempio della struttura del file CSV per il dataloader.

4.5.3 Descrizione dei Dataset

Per la fase di testing, abbiamo utilizzato tre dataset distinti derivati da MNIST, che si differenziano unicamente per il numero di campioni destinati al training e al testing. L'adozione di dataset di diverse dimensioni è motivata esclusivamente per questioni di praticità e usabilità in quanto ci dava la possibilità di scegliere quale dataset utilizzare in base a ciò che volevamo fare.

È importante precisare che i dati di "training" in questo contesto non sono utilizzati per aggiornare i parametri dei modelli: i modelli rimangono invariati. Il termine "training" viene impiegato unicamente per distinguere questi dati da quelli di test, che invece vengono usati per l'effettiva inferenza. I dati di "training" sono necessari per l'implementazione del metodo TrustScore, il quale richiede di raccogliere i valori dei nodi del primo layer fully-connected della rete durante l'inferenza. Questi valori, registrati sui dati di "training", vengono successivamente confrontati con quelli calcolati sui dati di test per consentire il calcolo del TrustScore.

I tre dataset sono stati suddivisi in base al numero di campioni in modo da garantire la flessibilità necessaria per adattarsi alle diverse situazioni di testing. A seconda della disponibilità di tempo e delle risorse, è stato quindi possibile selezionare il dataset più adeguato per ottenere riscontri rapidi o per un'analisi più approfondita. Le configurazioni utilizzate sono le seguenti:

- **CSV Mini:** 90 campioni di test e 180 campioni di training
- **CSV Mid:** 200 campioni di test e 600 campioni di training
- **CSV Big:** 500 campioni di test e 1500 campioni di training

Prestazioni

Le specifiche della macchina su cui sono stati eseguiti i test sono:

- **Macchina:** Huawei MateBook D 15
- **Processore:** 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2.42 GHz
- **Memoria RAM:** 8,00 GB (7,80 GB utilizzabile)
- **Sistema Operativo:** Windows 11 Home

Risultati per MC-Dropout

Dataset	Threshold Halting Criterion	Forward Passes	Tempo (s)
CSV Mini	0.005	10	2.91
	0.001	40	8.55
	0.0005	75	15.09
CSV Mid	0.005	15	10.20
	0.001	55	35.64
	0.0005	110	71.77
CSV Big	0.005	15	30.44
	0.001	55	112.33
	0.0005	110	212.26

Tabella 4.2: Risultati per il metodo MC-Dropout su diversi dataset

Per i vari dataset creati appositamente a scopo di testing, è stato condotto uno studio per analizzare come le performance dell'MC-Dropout variassero in base al numero di test da eseguire e valore di soglia dell'Halting Criterion. La tabella 4.2 rappresenta i risultati estrapolati sotto forma di forward passes e in tempo di esecuzione. Notiamo che il numero di forward passes è fortemente influenzato dal valore di soglia. Di fatti, al diminuire della soglia, il numero di forward passes aumenta, mentre cresce meno significativamente al crescere delle dimensioni del dataset. Al contrario, la dimensione del dataset ha un impatto rilevante sul tempo di esecuzione, che cresce notevolmente con l'aumento del numero di test.

Risultati per TrustScore

Il metodo per la distanza utilizzato nel TrustScore è stato `average`, con una batch size di 12.

Dataset	Tempo (s)
CSV Mini	1.59
CSV Mid	4.02
CSV Big	26.99

Tabella 4.3: Risultati per il metodo TrustScore su diversi dataset

A differenza dell'MC-Dropout, Il Trust Score tiene in considerazione anche dei dati etichettati come di training e dai risultati in tabella 4.2 notiamo un aumento quasi esponenziale all'aumentare del numero di dati.

4.5.4 Installazione e Utilizzo Personale

Per utilizzare l'applicazione, è necessario seguire i passaggi seguenti:

- **Scaricare il codice:** È possibile clonare il repository pubblico utilizzando il seguente comando:

  ```
  git clone https://github.com/lorenzobandini/U-ProBE
  ```

- **Preparare l'ambiente:** Si consiglia di utilizzare Anaconda per gestire l'ambiente di sviluppo. È possibile creare un nuovo ambiente con il seguente comando:

  ```
  conda create -n uprobe_env python=3.8
  ```

  Attivare l'ambiente creato con:

  ```
  conda activate uprobe_env
  ```

- **Installare i requisiti:** Una volta attivato l'ambiente, è necessario installare i pacchetti richiesti. Questo può essere fatto eseguendo il seguente comando nella cartella del progetto:

  ```
  pip install -r requirements.txt
  ```

- **Avviare l'applicazione:** Dopo aver installato i requisiti, puoi avviare l'applicazione eseguendo il file `app.py` con il seguente comando:

  ```
  python app.py
  ```

Preparare i propri input

L'applicativo per funzionare, richiede di seguire alcuni vincoli sui file che l'utente deve rispettare.

- **Modello**: il file deve contenere i pesi di un modello PyTorch, salvato in precedenza attraverso il comando `torch.save(model.state_dict(), PATH)`.
- **Classe del modello:** file Python contenente la classe del modello in PyTorch congrua con il modello caricato. Il nome della classe deve essere "CustomModel"
- **Classe del dataLoader**: file Python contenente la classe del dataLoader per fare pre-processing dei dati e fornire al modello i dati nel formato e ordine come

seguente: immagine sotto forma di tensore, label in formato stringa e split in formato stringa. Il nome della classe deve essere "CustomLoader"

- **Dataset**: il file in formato `.csv` deve essere strutturato per essere scannerizzato dal dataLoader senza errori, con tutti i campi richiesti inseriti. Il campo split deve avere i valori "training" o "test" e nel caso di utilizzo del Trust Score, è consigliato che i dati di training riescano a coprire con qualche caso ogni classe del modello.

Analizzare gli output

Al termine dell'inferenza, direttamente all'interno dell'applicativo, è possibile visualizzare alcuni dei dati più salienti riguardanti il modello e l'inferenza appena avvenuta.

È possibile visualizzare le prestazioni del modello attraverso le diverse statistiche, come accuratezza, F1, recall e precisione, nonché quelle dei metodi utilizzati descritti tramite la loro mediana, il primo e il terzo quartile.

Insieme alle statistiche è possibile consultare la visualizzazione grafica delle prestazioni attraverso il plot widget, confrontando i punti rappresentati con la diagonale tratteggiata che indica le predizioni andate a buon fine.

Nel caso fossero stati attivate anche alcune delle tecniche post-hoc è possibile vedere graficamente le loro prestazioni. In particolare, quelle dell'MC-Dropout, sono indicate tramite la dimensione del punti; più un punto è grande e maggiore sarà la sua incertezza. Il Trust Score invece è rappresentato attraverso l'uso dei colori, più precisamente utilizza un gradiente che parte dal blu quando si ha un basso livello di fiducia sul punto fino ad arrivare ad arancione.

Se vogliamo analizzare in maniera più dettagliata i risultati di ogni singola inferenza possiamo esportare i risultati tramite l'apposito pulsante contenuto nel widget per l'inferenza. Ci verranno esportati due file al percorso indicato tramite l'apposita dialog; uno conterrà le statistiche sopracitate riguardante il modello mentre l'altro file conterrà i dettagli nello specifico delle inferenze su ogni dato di test.

Le colonne di questo file saranno:

- **Id**: Identificatore univoco del record, utile per poter confrontare i risultati con il file di test passato in input
- **GT**: Etichetta che indica la vera classe di appartenenza del dato
- **No post-hoc method**: Risultato dell'inferenza normale, le incertezze verranno calcolate su questa predizione
- **Trustscore**: Valore di incertezza stimato tramite il metodo Trust Score. Maggiore è il valore e maggiore sarà l'incertezza. Colonna presente solo se selezionato il metodo Trust Score

- **MC-Dropout Score**: Valore di incertezza stimato tramite il metodo MC-Dropout. Può assumere valori da 0 a 1. Maggiore è il valore, migliore è la predizione. Colonna presente solo se selezionato il metodo MC-Dropout
- **MC-Dropout Prediction**: Classe che è comparsa più volte durante l'esecuzione dell'MC-Dropout. Colonna presente solo se selezionato il metodo MC-Dropout

Capitolo 5

Conclusioni

5.1 Valutazione Critica

Il progetto iniziale era decisamente ambizioso e nessuno, compresi i tutor che operano quotidianamente con queste tecnologie, si aspettava di dover affrontare le difficoltà che invece sono emerse durante lo sviluppo dell'interfaccia. In fase di ideazione e progettazione, infatti, non si era pienamente consapevoli della reale complessità di ciò che volessimo realizzare, riscontrando successivamente varie problematiche legate alla gestione di molteplici tipi di modelli, ognuno dei quali richiedeva approcci e analisi specifiche. Le difficoltà che si sono presentate hanno dimostrato che ciò che può sembrare scontato durante lo sviluppo di un'applicazione specifica non lo è affatto quando si cerca di creare una soluzione in grado di generalizzare una grande vastità di casi. Infatti, non si dispone di alcuna informazione a priori sulle intenzioni dell'utente ed è difficile dedurle anche successivamente dai suoi input.

Nonostante le difficoltà, non ci siamo mai lasciati abbattere, anche quando è stato necessario rivedere completamente il nostro approccio all'applicazione. Insieme, siamo riusciti a risolvere con successo tutti i problemi che si sono presentati lungo il percorso di sviluppo, e possiamo essere soddisfatti del lavoro svolto. Il risultato finale si avvicina molto all'idea originale del progetto, e possiamo considerarlo un successo sotto molti aspetti.

Tuttavia, le opportunità di miglioramento e sviluppo sono ancora numerose. Mi auguro che questo progetto continui a evolversi in futuro, ampliandosi sempre di più grazie a persone che abbiano il mio stesso entusiasmo e passione. Un'applicazione come questa ha un enorme potenziale, e sarebbe auspicabile che venga portata avanti da qualcuno che ne comprenda l'importanza e le potenzialità future.

5.2 Sviluppi Futuri

Il progetto è stato concepito con un'architettura modulare, con l'obiettivo di facilitarne l'espansione.

Attualmente il progetto supporta solo modelli di classificazione di immagini ma l'obiettivo principale per il futuro sarebbe quello estendere le funzionalità della GUI a tutti i tipi di modello, anche con complessità maggiori, rimuovendo progressivamente tutte le limitazioni che abbiamo dovuto inserire durante questo primo periodo di sviluppo.

Riguardo a questo, la progettazione iniziale richiedeva in input solamente modello, dataloader e dataset ma durante lo sviluppo abbiamo dovuto scendere al compromesso di richiedere anche la classe del modello e quindi passare dal caricamento del modello da TorchScript Format (`.pt`), che rendeva possibile l'inferenza senza la definizione della classe del modello, a caricare separatamente i pesi del modello (`.pth`) ed insieme anche la classe del modello (`.py`), che non è una delle migliori pratiche, risultando anche complessa, ma almeno ci ha semplificato l'applicazione delle manipolazioni necessarie per tutte le funzionalità implementate.

Per l'MC-Dropout, nella dialog della graph visualization è già stato implementato tutto il sistema per selezionare i layer su cui inserire il Dropout e impostare le loro probabilità durante l'applicazione del metodo non intrusivo ma al momento questa funzione non è attiva poiché le tempistiche dello sviluppo non lo hanno permesso. Attualmente l'MC-Dropout viene eseguito con il Dropout attivo negli stessi layer sui quali abbiamo avuto Dropout anche in allenamento.

Riguardo invece il Trustscore, durante la nostra implementazione abbiamo deciso, guidati dai risultati di alcuni paper, di selezionare come layer da cui estrarre i valori dei nodi sempre il primo layer fully-connected ma questo potrebbe non essere sempre la scelta desiderata, soprattutto quando si vuole fare ricerca. Quindi sicuramente un miglioramento futuro potrebbe includere la possibilità di scegliere liberamente il layer da cui estrarre i valori da utilizzare nel Trustscore, in modo analogo alla selezione dei layer per l'MC-Dropout.

Sicuramente guardando invece verso il fronte dei nuovi sviluppi, i metodi già sviluppati e implementati nell'applicativo sono solo due di tanti altre metodologie non intrusive che sono solo teoriche o fanno parte di progetti assestanti, per fare alcuni nomi a titolo di esempio possiamo nominare sicuramente Topological Data Analysis, Ensemble e Few Shot Learning.

Infine, un sogno condiviso da tutto il team, che ha alimentato la volontà di realizzare questo applicativo, è che la piattaforma diventi un punto di riferimento per ricercatori, utenti e appassionati. Essendo open-source, ci auguriamo che si sviluppi una comunità di contributori che ne garantisca la crescita, integrando metodologie sempre nuove e perfezionando quelle esistenti.

Ci auguriamo che il nostro lavoro sia solo il primo passo di un effetto "butterfly" che contribuirà, attraverso l'uso dei modelli di DL, a valutare l'incertezza nei risultati di inferenza in modo sempre più preciso, trasformandola in un'alleata per comprendere meglio la realtà complessa e incerta in cui viviamo.

5.3 Competenze Acquisite

La mia esperienza all'interno del CNR-ISTI non è riassumibile in un semplice svolgimento ordinato di task al fine di sviluppare un prodotto. Il percorso durato qualche mese è riuscito a darmi molto, non solo a livello di competenze, ma anche a livello personale, immergendomi all'interno di un contesto di ricerca che non conoscevo e a cui non ero abituato, con persone eccezionali e fuori dal comune, da cui mi è stato trasmesso un grande spirito di curiosità e voglia di fare, dove la condivisione di idee e pensieri erano essenziali e fortemente incentivati.

Sul piano teorico, sono riuscito ad accrescere le mie conoscenze entrando più in dettaglio nell'ambito del Machine Learning e del Deep Learning, come mi ero prefissato prima di iniziare, e sono riuscito a conoscere argomenti di cui prima non ero a conoscenza prima come l'Uncertainty Quantification.

Anche dal punto di vista pratico ho avuto l'occasione di imparare nuove tecnologie ed esplorare nuovi ambiti di applicazione dell'informatica. Non avevo mai realizzato un interfaccia grafica per desktop, né avevo mai utilizzato Python, con annesse tutte le sue librerie. Ad oggi, posso affermare di aver acquisito praticità nel suo uso e di conoscere alcuni strumenti fondamentali come PyTorch, pandas e matplotlib, che sono stati indispensabili per il progetto ma che mi torneranno sicuramente utili in futuro.

Infine, nello sviluppo di un interfaccia, mi sono trovato per la prima volta nella posizione di dovermi immedesimare nell'utente finale, che non possiede nessuna delle conoscenze da me acquisite durante lo sviluppo, per cui non potevo dare niente per scontato. Ho cercato quindi di intuire quali potessero essere comportamenti di un utente che apre per la prima volta l'applicazione, in modo da poterlo guidare nell'uso e aiutarlo nel caso di insorgenza di problemi, comunicando attraverso i giusti feedback.

Tabella degli Acronimi

Acronimo	Descrizione
CNR-ISTI	Consiglio Nazionale delle Ricerche - Istituto di Scienza e Tecnologie dell'Informazione.
SI-Lab	Signals and Images Laboratory, un laboratorio di ricerca focalizzato sull'analisi di segnali e immagini.
MC-Dropout	Monte Carlo Dropout, una tecnica per stimare l'incertezza nei modelli di machine learning.
TDA	Topological Data Analysis, un approccio per analizzare la struttura dei dati.
BNN	Bayesian Neural Network, una rete neurale che integra la probabilità nella struttura del modello.
GP	Gaussian Processes, un metodo per modellare funzioni utilizzando distribuzioni gaussiane.
AI	Artificial Intelligence, intelligenza artificiale, simulazione di processi intelligenti da parte delle macchine.
ML	Machine Learning, un sottoinsieme dell'intelligenza artificiale che usa algoritmi per apprendere dai dati.
DL	Deep Learning, una branca del machine learning che utilizza reti neurali profonde.
GT	Ground Truth, i dati reali e veritieri utilizzati per confrontare i risultati di un modello.
NN	Neural Network, una rete neurale, un sistema di modelli matematici ispirato al cervello umano.
CNN	Convolutional Neural Network, una tipologia di rete neurale particolarmente utile per il riconoscimento di immagini.
UI	User Interface, interfaccia utente, il punto di interazione tra l'utente e il sistema.
GUI	Graphical User Interface, interfaccia utente grafica, un tipo di interfaccia che usa elementi visivi per l'interazione.
SVM	Support Vector Machine, una tecnica di machine learning per la classificazione e regressione.

K-NN	K-Nearest Neighbors, un algoritmo di machine learning basato sulla ricerca dei "vicini più prossimi".
SGD	Stochastic Gradient Descent, un algoritmo di ottimizzazione usato per addestrare i modelli.
PCA	Principal Component Analysis, una tecnica statistica per ridurre la dimensionalità dei dati.
API	Application Programming Interface, un'interfaccia che permette la comunicazione tra software diversi.
MSE	Mean Squared Error, errore quadratico medio, una misura di accuratezza di un modello di regressione.

Supporto Finanziario

Questo lavoro è stato parzialmente supportato dai due progetti finanziati dall'UE ProCAncer-I (GA 952159) e FAITH (GA 101135932).

Bibliografia

[1] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[2] Christopher M Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:1122–1128, 2006.

[3] George Casella and Roger Berger. *Statistical inference*. CRC Press, 2024.

[4] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[6] Michael J Gilman. A brief survey of stopping rules in monte carlo simulations, 1968.

[7] Ian Goodfellow. Deep learning, 2016.

[8] Trevor Hastie. The elements of statistical learning: data mining, inference, and prediction, 2009.

[9] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

[10] Kalvik Jakkala. Deep gaussian processes: A survey. *arXiv preprint arXiv:2106.12135*, 2021.

[11] Hans Janssen. Monte-carlo based uncertainty analysis: Sampling efficiency and sampling convergence. *Reliability Engineering & System Safety*, 109:123–132, 2013.

[12] Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. *Advances in neural information processing systems*, 31, 2018.

[13] Ian T Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

[14] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[15] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[16] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Douglas C Montgomery and George C Runger. *Applied statistics and probability for engineers*. John wiley & sons, 2010.

[20] David S Moore and George P McCabe. *Introduction to the practice of statistics*. WH Freeman/Times Books/Henry Holt & Co, 1989.

[21] Eva Pachetti, Giulio Del Corso, Serena Bardelli, and Sara Colantonio. Few-shot conditional learning: Automatic and reliable device classification for medical test equipment. *Journal of Imaging*, 10(7), 2024.

[22] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[24] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[25] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[26] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[27] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[29] Guido van Rossum. The python language reference: Release 2.6. 4. *Python Software Foundation*, 2009.

[30] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349, 2018.