Progetto realizzato da Luca Vignali

*Report Progetto - “Laboratorio di Ottimizzazione, Intelligenza Artificiale e ML”*

# Introduzione

Questo progetto è stato realizzato con lo scopo di prevedere i fenomeni metereologici basandosi su un modello allenato con un dataset custom preparato con dati estratti dall’archivio de ilmeteo.it.

# Obiettivi del progetto

* Creazione di un modello per custom
* Allenamento di questo modello
* Ottimizzazione del modello
* Capacità di prevedere dei fenomeni basandosi su input di un utente

# Descrizione del Progetto

## Struttura

data/: cartella del dataset

data.csv: dataset

models/: cartella del modello e dizionario

meteo\_scaler.pkl: scaler del modello

meteo\_state\_dict.pth: dizionario del modello

meteo.pth: modello

runs/meteo\_model: cartella dei resoconti dell’allenamento visualizzabili con Tensorboard

src/: cartella dei file di programma

\_init\_.py: file di inizializzazione

create\_graphs.py: funzione per la creazione di grafici Tensorboard

data\_processing.py: preparazione del dataset e creazione del dataset

model.py: funzioni per la creazione del modello

predict.py: funzione di predizione del meteo basandosi su dati inseriti dall’utente

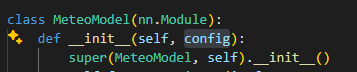
train\_evaluate: file per l’addestramento, la valutazione e il testing del modello

config.yaml: file contenente percorsi e dati per le funzioni

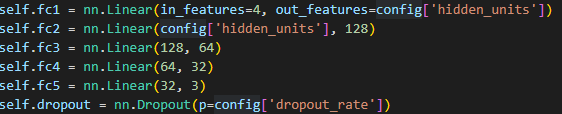
main.py: file principale di esecuzione

requirements.txt: file contenente le estensioni con annesse versioni necessarie per fa andare il programma

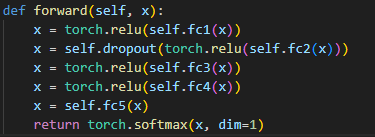
## Il modello

Il modello è stato realizzato utilizzando PyTorch e le funzioni che rende disponibili per la creazione e l’allenamento di reti neurali.  


Il costruttore \_\_init\_\_ inizializza la creazione del modello e accetta i dati preconfigurati nel file config.yaml e il metodo super serve per richiamare il costruttore della classe nn.Module.



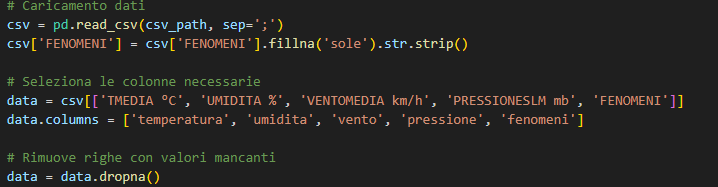
Questa parte serve a definire i layer del modello e il livello di dropout; self.fc1 definisce il numero di passaggi che il modello farà e il primo output che poi verrà passato a self.fc2 che a sua volta accetterà il numero di output di fc1 come input e produrrà un input che verrà usato da fc3 e così ripetendo fino ad arrivare a fc5 che, come output, darà le tre classi che il modello dovrà riconoscere.

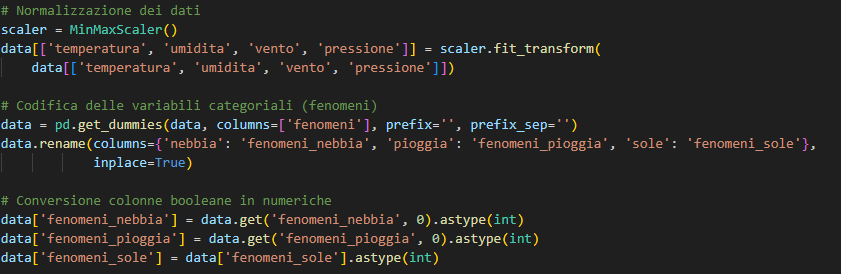


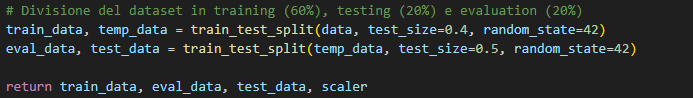
Il metodo forward serve per definire l’avanzamento della rete, passando la variabile x attraverso i 5 punti del modello utilizzando una funzione di attivazione ReLu, e infine l’output viene passato attraverso la funzione softmax di torch per ottenere le probabilità dei fenomeni.

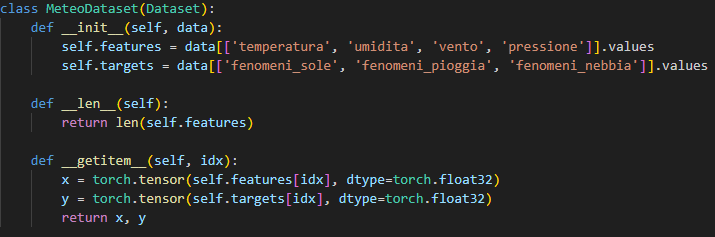
## Caricamento dati

Nel file data\_processing.py viene letto e preparato il dataset che poi verrà usato per addestrare il modello.

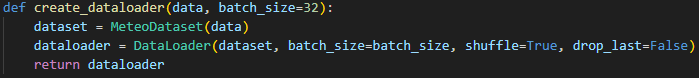
Per prima cosa viene letto il csv contenente tutti i dati, di seguito vengono riempite le caselle vuote con sole, e isolate e rinominate solo le colonne necessarie all’addestramento.

In seguito i dati vengono normalizzati e vengono codificati i fenomeni che si possono trovare all’interno della colonna *fenomeni* e vengono convertiti da valori booleani a numerici così da poter essere utilizzati nell’addestramento.

Infine tutti i dati vengono divisi in tre categorie, *dati per il training*, che verranno usati per l’addestramento del modello, *dati per la valutazione,* che saranno utilizzati per verificare che l’addestramento sia avvenuto correttamente e *dati di testing* per provare il modello.

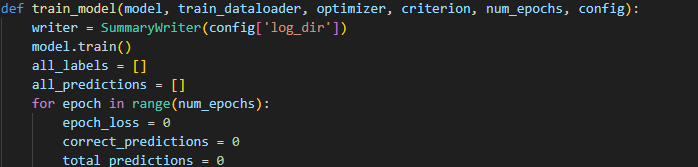
La classe *MeteoDataset*  serve per creare il dataset utilizzato dal modello; per prima cosa definisce temperatura, umidità, vento e pressione come caratteristiche del modello e fenomeni\_sole, fenomeni\_pioggia e fenomeni\_nebbia come gli obiettivi che il modello deve trovare.

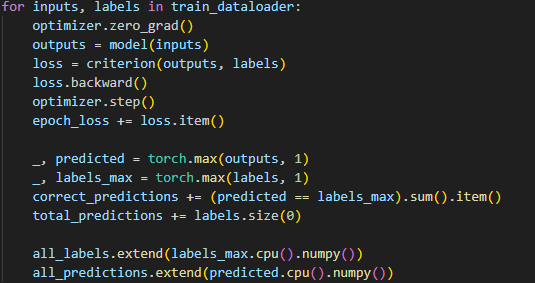
La funzione \_\_len\_\_ restituisce la quantità di campioni che sono presenti nel dataset e la funzione \_\_getitem\_\_ restituisce il campione all’indice idx e converte x e y in tensori pytorch.

Infine la funzione create\_dataloader crea un’istanza di MeteoDataser e poi un dataloader con il dataset, la grandezza del batch, se il dataset dev’essere mischiato e se l’ultimo batch deve essere scartato.

## Processo di allenamento

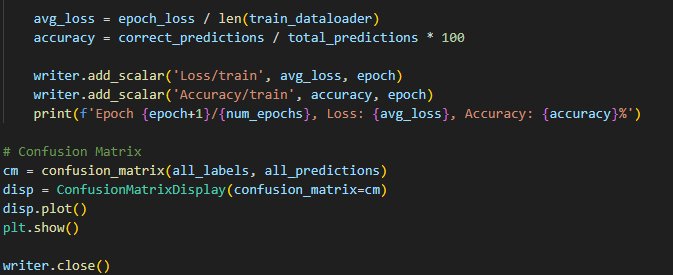
Prima di iniziare l’allenamento, la valutazione e il testing del modello vengono definiti il *criterion*, ovvero una funzione di loss utilizzata per misurare quanto bene o male il modello stia performando, e *l’optimizer*, un algoritmo che minimizza la funzione di loss aggiornando i pesi del modello; come ottimizzatore ho scelto *Adam,* che i vantaggi di due altri ottimizzatori, *AdaGrad* e *RMSProp*, ed è noto per essere efficiente e richiedere meno memoria.

Prima di iniziare l’addestramento creo un oggetto di SummaryWriter per registrare i dati di addestramento per Tensorboard, successivamente imposto il modello in modalità train e inizializzo delle liste per etichette e previsioni che verranno usate per creare una confusion matrix.  
Successivamente entro nel ciclo delle epoche, che si ripete seguendo il numero specificato nel file di config, e inizializzo delle variabili per la perdita e l’accuratezza.

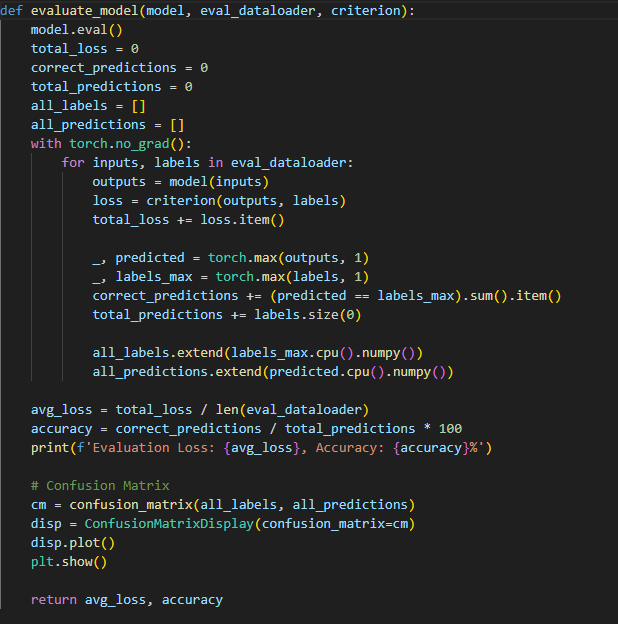


Dopo l’inizializzazione del modello e la creazione di variabili il programma entra nel ciclo dell’addestramento effettivo; per prima cosa vengono azzerati i gradienti accumulati negli addestramenti precedenti, poi vengono fatte delle predizioni passando una batch di dati presi dal dataset; successivamente i risultati vengono analizzati da criterion per calcolare la loss, confrontando i risultati con le label vere e con loss.backwards i gradienti indicano la direzione e le modifiche da fare per ridurre la perdita e vengono applicati all’optimizer per minimizzare la perdita.

In seguito, vengono aggiornate le metriche per la monitorizzazione del modello.

Per ultimo vengono calcolate le media di perdita e accuratezza, registrate le metriche su TensorBoard, stampate le metriche dell’epoca corrente, calcolato e mostrato la confusion matrix e chiuso il metodo del SummaryWriter.

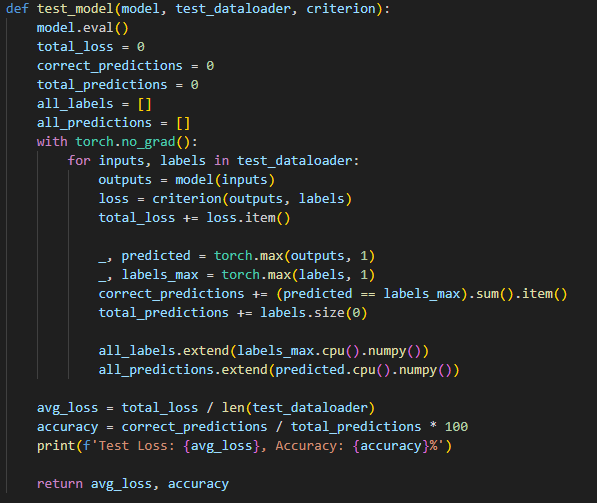
## Processo di valutazione



La parte di valutazione del modello è relativamente simile come codice alla parte di addestramento, con alcune differenze:

* Il modello viene inizializzato in modalità eval invece di train
* Non vengono calcolati i gradienti
* Non vengono aggiornati i pesi

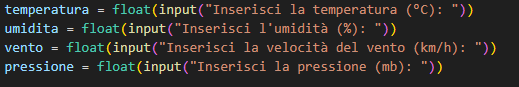
## Processo di test



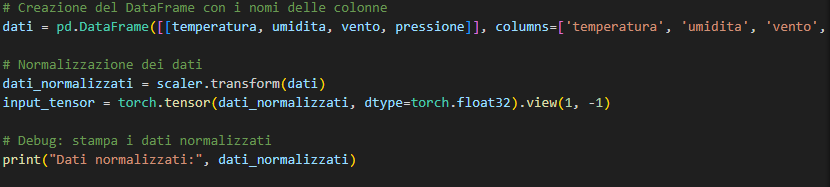
Similarmente il processo di test non differisce molto dal processo di valutazione a parte la mancanza di una confusion matrix.

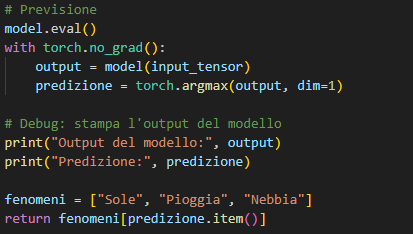
## Previsione dei fenomeni

L’ultima funzione importante del codice è la funzione per prevedere uno dei fenomeni su cui si è addestrato il modello basandosi su dei dati di input dell’utente.



La prima parte del codice chiede semplicemente dei dati all’utente.

Successivamente i dati vengono inseriti in un DataFrame di pandas e normalizzati attraverso lo scaler per renderli comprensibili dal modello e mostrati all’utente.

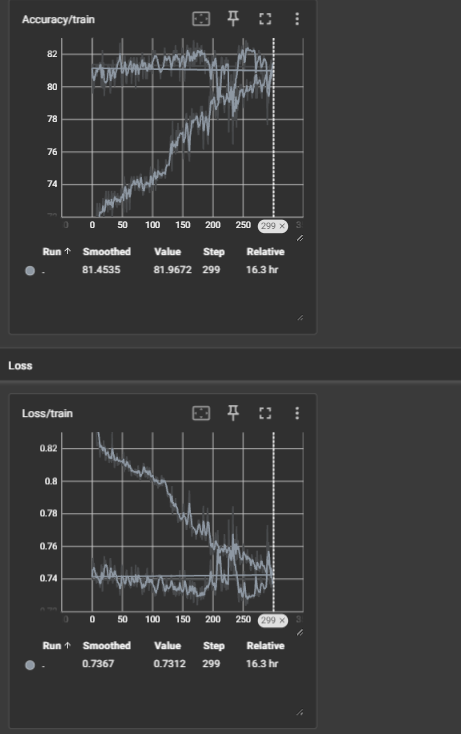


Il modello viene impostato in modalità valutazione, gli vengono dati i valori forniti dall’utente e tenta di predire quale dei fenomeni possibili possa essere presente.

Infine, vengono restituiti gli output del modello e mappato questo output ad uno dei fenomeni.

## Conclusioni

Nonostante il modello risulti allenato accuratamente, come mostrato dai grafici creati da TensorBoard:



Il problema principale che ho notato è la difficoltà del modello nel predire i fenomeni naturali sotto alcune caratteristiche; questo penso sia dovuto al fatto che ho allenato il modello utilizzando dati della mia città di Rimini, che essendo una città di mare ha caratteristiche diverse rispetto a città come Milano o Berlino, che si trovano distante dal mare.

Testando il programma su una mole maggiore di dati, provenienti da diverse aree ambientali potrebbe istruirlo meglio e produrre risultati migliori.

## Fonti

Dati metereologici presi dall’archivio del meteo.it: [Archivio Meteo Storico](https://www.ilmeteo.it/portale/archivio-meteo/)