UNIVERSITÀ DEGLI STUDI DI SASSARI

Corso di laurea in ingegneria informatica

Utilizzo dei dati geografici per migliorare l'identificazione di serpenti da immagini mediante deep learning.

Relatore:
Prof. Piernicola Oliva

Laureando:
Matteo Comini **mat. 50035432**

Anno accademico 2021/2022

Indice

1 Introduzione

Il machine learning [1] è un campo di ricerca che sta avendo sempre più importanza nelle aree della scienza e dell'ingegneria. In particolare, le reti CNN (reti neurali convoluzionali) [2], sono diventate di fondamentale importanza nel campo della classificazione delle immagini. Un esempio di questo tipo di architettura è ResNet (Residual Network), una tipologia di convolutional neural network con grandi potenzialità nel campo della classificazione di immagini. Nel mio progetto ResNet-18 verrà utilizzato come estrattore dei dati dalle immagini, che definirò come feature, che in seguito verranno passati ad un classificatore per ottenere i risultati. Il classificatore in questione sarà SVM linear (Support Vector Machine), un algoritmo di apprendimento molto utilizzato per la sua semplicità e varsatilità.
In questo progetto andremo ad analizzare e confrontare le performance di questo sistema, addestrato per il riconoscimento di due classi di immagini di serpenti: la Vipera aspis [3] e la Vipera berus [4].
Inoltre, andremo ad esaminare le performance che la rete ottiene con o senza l'aggiunta delle coordinate geografiche dell'immagine.

1.1 Obiettivi

L'obiettivo sarà quello di valutare se, aggiungendo le coordinate geografiche, la rete sia agevolata nel riconoscere le specie di serpenti e quindi ottenere una maggiore accuratezza nella classificazione. Il giudizio verrà determinato in seguito a vari passaggi: inizialmente guardando le performance della rete addestrata solo con le immagini, in seguito quella con solo le coordinate geografiche delle immagini e infine, le performance della rete con le feature estratte dal modello con le sole immagini con in aggiunta le coordinate geografiche.
Il risultato che si intende raggiungere è quello che l'accuratezza del modello supportato da essi sia molto più performante e corretto nella classificazione, dato che le due specie di serpenti selezionate hanno una differenza sostanziale a livello territoriale ma non a livello morfologico.

1.2 Strumenti e librerie

L'analisi è stata effettuata utilizzando il linguaggio Python [5], attraverso l'uso di PyCharm [6]. Inoltre, ho usufruito di varie librerie per la stesura del codice. Questo è un breve elenco delle stesse:

- torch [7]
- torchvision [8]
- matplotlib [9]
- sklearn [10]
- pandas [11]
- plotly [12]

2 Analisi del dataset

Il primo passo all'inizio del progetto, cosa di fondamentale importanza nel campo del machine learning e del deep learning [13], è quello di creare un dataset che soddisfi i nostri bisogni. Per ottenere le immagini, con i relativi dati geografici, ho utilizzato un sito contenente un database molto fornito di dati correlati a immagini riguardanti animali, piante e molto altro. Il sito in questione si chiama iNaturalist [14]. Un requisito di fondamentale importanza è stato quello di trovare due specie di serpenti morfologicamente simili (come si può notare dalla *figura 1*), ma con differenti distribuzioni geografiche.
Da esso ho deciso di prendere i dati per le due specie europee di serpenti con i dataset tra i più forniti all'interno del sito per avere così un dataset più grande possibile; tutto ciò in modo da avere più possibilità che l'accuracy finale della classificazione sia più alta.
I dati esportati dal sito sono sottoforma di file csv, uno per ogni specie. Per ogni riga del file avevamo un url da cui grazie all'utilizzo di uno script Python abbiamo ottenuto le immagini, le coordinate geografiche e altri dati che non sono stati rilevanti al fine del progetto. Le immagini ottenute per classe erano 2500 e sono state divise in due cartelle differenti per le due specie, dentro il quale sono state poi nuovamente suddivise tra immagini di train e validation.

Figure 1: Immagini casuali estratte dal mio dataset prima della trasformazione: Vipera aspis [1], Vipera berus [2]

2.1 Trasformazione immagini

Per utilizzare le immagini all'interno di una rete ho la necessità di renderle omogenee a livello di grandezza, perciò ho applicato delle trasformazioni ai dataset di train e validation per le due classi: come prima cosa effettuo il `RandomResizedCrop()` [15] per tutte le immagini, ciò sta a significare che seziono le immagini di una grandezza da me stabilita in una porzione casuale dell'immagine originale. La grandezza della regione di taglio sarà quadrata di 224px, perché ResNet-18 accetta quel tipo di dato in input. In seguito, normalizzo i valori dei pixel dell'immagine in base a media e deviazione standard predefinite di ImageNet [16] per ogni canale di colore (pratica molto utilizzata perché ImageNet calcola media e deviazione standard sulla base di milioni di immagini), in modo da avere una distribuzione dei valori compatibile tra le diverse immagini.

2.2 Preparazione dei dati

Oltre alla ripartizione delle immagini nelle cartelle delle due specie (le immagini delle due classi sono situate in due cartelle differenti dove all'interno ci sarà un'altra suddivisione tra immagini di train e validation, questa ripartizione verrà approfondita nel *paragrafo 3.2*), viene effettuato un approfondimento dei dati contenuti all'interno dei file csv ottenuti dal sito. Vengono eliminate tutte le colonne non utili allo scopo del progetto e vengono lasciate solo quelle significative: `'specie'`, `'latitudine'`, `'longitudine'`, `'id'`. I dati di maggior importanza sono quelli di `'latitudine'` e `'longitudine'` che verranno associati attraverso il campo `'id'` della riga all'immagine presente nelle cartelle di train o validation. Una volta eliminate le colonne non utili ai fini del progetto ho concluso il mio lavoro con la pulizia dei file csv e ho proseguito con l'analisi delle immagini.

3 Machine learning

Come anticipato nell'analisi degli strumenti, per lo svolgimento del progetto ho utilizzato tecniche di machine learning (e più in particolare deep learning) che verranno affrontate in particolare nei successivi paragrafi.

3.1 ResNet-18

ResNet-18 [17][18] è una rete neurale convoluzionale che, come suggerisce il nome, utilizza 18 livelli di profondità nella fase di train delle immagini. Questa architettura viene molto utilizzata grazie alla sua versione preaddestrata che contiene un milione di immagini prese dal database ImageNet: essa è infatti capace di classificare 1000 classi di oggetti, tra cui molti animali. Questo mi ha portato a scegliere questo tipo di rete per compensare il fatto che il mio dataset di immagini non sia poi così tanto ampio.
Io ho utilizzato la tecnica del Fine-Tuning perchè in questo modo ho ottimizzato i pesi della rete in modo da far diventare le feature sempre più specifiche per l'attività che voglio svolgere.
Descrivo brevemente l'architettura, per capire un po' più da vicino cosa accade all'interno di un modello ResNet-18:

- `'conv1'`: il layer convoluzionale che prende in input le immagini e genera in output 64 mappe di feature. Il kernel è di 7x7 con stride 2x2 e padding 3x3 [19].
- `'bn1'`: il layer di normalizzazione batch che ha lo scopo di normalizzare le uscite di 'conv1', ha 64 feature.
- `'relu'`: una funzione che viene applicata ad ogni singolo elemento dell'output del layer precedente [20].
- `'maxpool'`: questo layer viene utilizzato per ridurre la dimensione dell'output di un fattore di 2.
- `'layer1'`, `'layer2'`, `'layer3'`, `'layer4'`: Essi sono i blocchi sequenziali e contengono i blocchi di base. Ogni blocco di base include due strati convoluzionali e due normalizzatori di batch, mentre tra di loro viene effettuato il `ReLu`.
- `'avgpool'`: è un livello di pooling impiegato per ridurre la dimensione spaziale dell'output [21].
- `'fc'`: è un layer lineare che, come input, prende le feature mandate in output dal livello precedente e trasforma il contenuto in un vettore di probabilità. Questo livello ha come input 512 ingressi e 2 uscite che stanno a rappresentare le probabilità di raffigurazione delle due classi [22].

3.2 Addestramento e validazione

Per la formazione del dataset ho optato per la suddivisione delle 2500 immagini che ho per ogni classe fra le cartelle di train e validation per permettere al modello di ottenere delle feature funzionanti per il classificatore. La ripartizione di esse è all'80% train e al 20% validation, come viene sempre consigliato. Inizialmente però, ho compiuto un passaggio per rendere il dataset delle immagini completo, ossia ho pesato perfettamente le due classi così da non renderne una predominante sull'altra: ho bilanciato le due classi per renderle con stesso numero perchè inizialmente partivo con 6000 immagini per la Vipera berus e 2500 per la Vipera aspis e quando si parla di classificazione di immagini è sempre consigliabile renderle equivalenti.

3.3 Feature extraction

Con il termine feature extraction [23] si intende il processo di estrazione dei pesi da un layer di una rete. In particolare, nel nostro caso, verrà utilizzato per prelevare le feature contenute nel penultimo layer della rete addestrata con le sole immagini per poi processare tutto attraverso un classificatore. Noi, per sfruttare questa tecnica, ci appoggeremo sugli ultimi dati che il modello può fornirci prima della classificazione, vale a dire i pesi contenuti nel livello fully connected (`'fc'`). I dati che otterrò dall'estrazione saranno sottoforma di vettore con 512 celle, perciò alla fine dell'estrazione avrò 512 parametri per ogni input che in origine avevo inserito all'interno della rete. Questo ci servirà in seguito per poter valutare con SVM inizialmente le prestazioni delle feature e successivamente le prestazioni delle feature combinate ai dati geografici.

3.4 SVM lineare

Support Vector Machine (SVM) [24] è un algoritmo di apprendimento con supervisione, utilizzato nell'ambito della classificazione. L'obiettivo di questo algoritmo è quello di suddividere nel modo migliore i punti appartenenti a una classe da quelli di un'altra classe attraverso la ricerca dell'iperpiano di separazione ottimale. La caratteristica che rende un iperpiano migliore di un altro è la massimizzazione del margine tra le due classi. Il margine, come si può notare dall'immagine sottostante (*figura 2*), è la distanza minima tra l'iperpiano e i punti delle classi più vicini ad esso. Questo tipo di algoritmo si rende efficiente solo per i problemi dove appunto i dati sono separabili in modo lineare. Se si volesse estendere SVM alla risoluzione di problemi non lineari si potrebbero optare per la modifica del kernel.

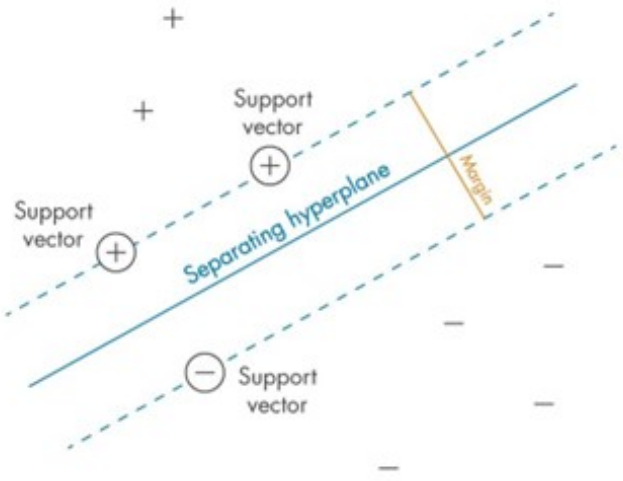

Figure 2: Separazione dell'iperpiano in modo lineare tra due classi evidenziando il significato di margine

3.5 Valutazione performance: ROC e AUC

Per la valutazione delle prestazioni dell'algoritmo SVM lineare ho deciso di utilizzare la valutazione `roc_auc_score()` [25] che si basa, appunto, sull'utilizzo della curva ROC AUC [26].

La curva ROC (Receiver Operating Characteristic) mostra le prestazioni di un modello di classificazione a diverse soglie di probabilità, in particolare rappresenta il tasso di vero positivo (True Positive Rate, TPR) al variare del tasso di falso positivo (False Positive Rate, FPR); visualizzando il tutto attraverso il grafico presente nella *figura 3*, noteremo che l'FPR si troverà nell'asse delle ascisse, mentre il TPR nell'asse delle ordinate. L'AUC (Area Under The Curve) rappresenterà invece l'area sottostante alla curva ROC, il suo valore indicherà le performance di classificazione che potranno oscillare tra 0.5 (caso peggiore) a 1 (classificatore ideale).

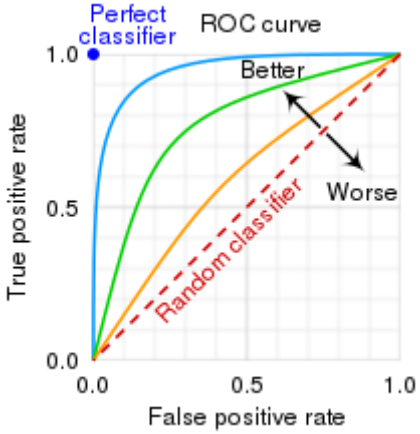

Figure 3: Rappresentazione grafica curva ROC e AUC

4 Risultati

In questo capitolo analizzeremo le performance ottenute dalle diverse reti addestrate: inizialmente vedremo i risultati conseguiti dalla rete ResNet-18 confrontando l'andamento per 40 epoche tra l'utilizzo del Fine-Tuning e Feature extractor, in seguito vedremo le performance di classificazione con SVM delle stesse feature prese dal penultimo layer della rete in Fine-Tuning; più avanti invece, avverrà il confronto tra una rete addestrata con solo le coordinate geografiche, infine addestreremo una rete che come input riceverà le feature combinate con i dati geografici. Vedremo se l'insieme dei dati porterà ad un aumento dell'accuratezza studiando quanto le coordinate abbiano impattato sui risultati finali.

4.1 Prestazioni delle sole immagini

Prima di "legare" le immagini con le coordinate geografiche, per capire se effettivamente le coordinate aumentino l'accuratezza del classificatore, dobbiamo valutare le prestazioni che esso ottiene con solo le feature ottenute attraverso l'estrazione dei pesi dal modello ResNet-18. I risultati ottenuti con ResNet-18 sono differenti utilizzando il metodo di Feature extractor o di Fine-Tuning. Nell'immagine sottostante infatti si può notare come con l'ottimizzazione dei pesi in tutti i livelli, addestrando la rete per 40 epoche, le prestazioni fossero maggiori rispetto all'addestramento solo dell'ultimo layer; perciò in seguito ho preferito estrarre le feature dal modello in Fine-Tuning piuttosto che da quello in Feature extractor [27].

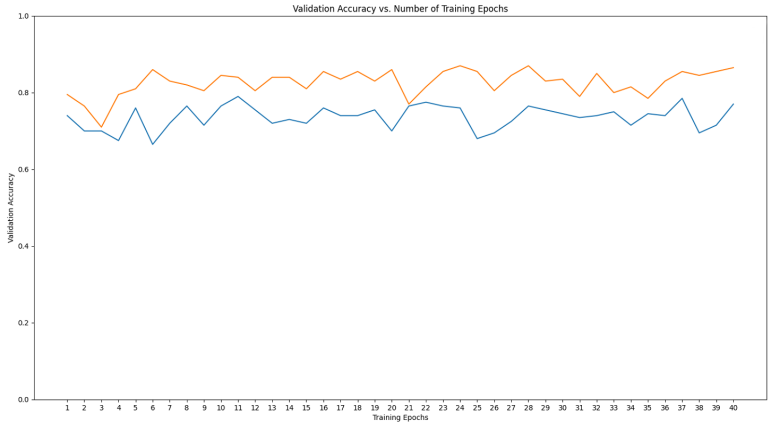

Figure 4: Training della rete ResNet-18 per 40 epoche: si nota come la rete in Fine-Tuning (arancione) abbia prestazioni migliori rispetto a quella in Feature extractor (blu)

Come si vede dalla *figura 4* l'accuratezza utilizzando come classificatore ResNet-18 in Fine-Tuning è di 0.85, mentre i risultati ottenuti classificando le feature estratte dalla rete precedente e analizzate attraverso SVM sono dell'0.875 di accuratezza, risultati già ottimi. Oltre all'accuratezza ho calcolato anche la feature importance: per stimare l'importanza dei dati utilizzati, in questo caso le feature estratte dal modello in Fine-Tuning, ho sfruttato la funzione `coef_()` [29] che, data una feature, darà in output il peso della stessa. Per il calcolo della feature importance di questa rete si può notare come le feature rappresentate (prime 25 della *figura 5*) abbiano pressochè quasi la stessa soglia d'importanza.
Successivamente, andremo a utilizzare queste stesse feature analizzate ed estratte che saranno

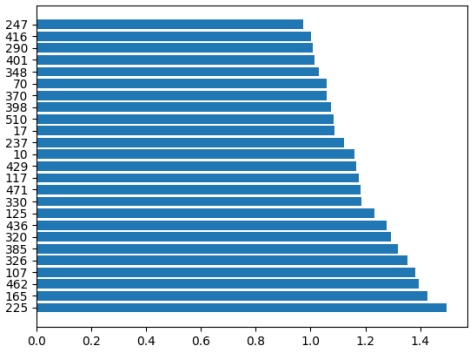

Figure 5: Feature importance della rete addestrata con le sole immagini (prime 25 feature)

combinate in un primo momento con le coordinate normalizzate reali e in seguito con le coordinate normalizzate e alterate.

4.2 Prestazioni delle sole coordinate geografiche naturali

Analizzando le prestazioni del classificatore delle coordinate, ho pensato fosse opportuno confrontare i risultati ottenuti con la distribuzione dei dati graficamente; non a caso i risultati sono ottimi, l'accuratezza con SVM lineare è di 0.89 considerando che ho utilizzato le sole coordinate normalizzate [28]. Ulteriormente, si può notare nell'immagine sottostante come le due classi di serpenti siano geograficamente molto differenti. In particolare, la classe Vipera aspis (colore blu nella cartina) si colloca principalmente nelle zone italiane, mentre la classe delle Vipera berus (colore giallo nella cartina) si dispone in tutta Europa, differenziandosi notevolmente dalla classe precedente.

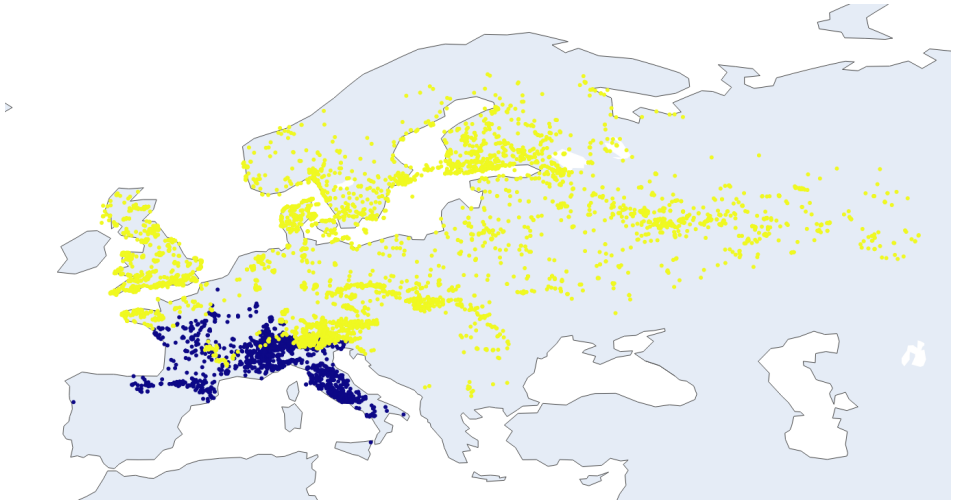

Figure 6: Distribuzione delle coordinate geografiche della classe aspis (colore blu) e della classe berus (colore giallo)

Parlando invece di feature importance, in questo caso specifico i risultati attribuiscono più importanza al dato latitudine anziché a quello della longitudine, nonostante il secondo sia comunque fondamentale ai fini di un'accuratezza alta.

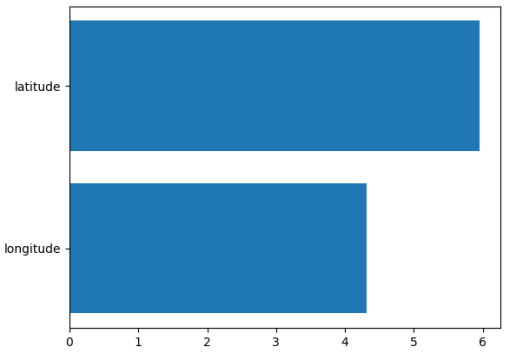

Figure 7: Feature importance della rete addestrata con solo coordinate geografiche reali

4.3 Prestazioni delle sole coordinate geografiche alterate

Oltre all'addestramento di una rete con l'utilizzo di coordinate reali, ho provato anche a modificare queste ultime, cercando di rendere meno riconoscibili le classi tra di loro, per vedere quanto le performance potessero peggiorare. Questi due test sono stati effettuati per mostrare che, anche se le due specie fossero geograficamente meno riconoscibili, le performance della rete con le feature combinate aumenterebbero comunque rispetto alle performance della rete con solo le immagini.
I due test effettuati sono avvenuti modificando le coordinate della classe aspis: in un primo momento ho modificato ogni valore della latitudine aggiungendo $+5°$ e ogni valore della longitudine aggiungendo $+4°$; infine, ho invece fatto un secondo test tenendo la latitudine ancora con $+5°$, ma modificando la longitudine da $+4°$ a $+5°$. Entrambi i test sono stati effettuati normalizzando i dati sfruttando una nuova media e deviazione standard. In questo modo, utilizzando come sempre un classificatore SVM lineare, ho ottenuto rispettivamente per il primo e per il secondo test 0.66 e 0.5 di accuratezza. Risultati notevolmente peggiori rispetto all'utilizzo di coordinate reali.
In seguito, mostrerò la distribuzione nel piano delle coordinate normalizzate in modo da far capire che si può notare anche a occhio nudo quale dataset ha dato risultati peggiori.
Come detto precedentemente si nota che, nella *figura 8* guardando da sinistra verso destra, le due classi di serpenti si mischiano sempre di più andando a peggiorare i risultati delle performance. Più avanti, andremo a determinare quanto queste due classi modificate, combinate con le feature estratte dal modello addestrato con sole immagini, incideranno sulle performance del classificatore.
Concentrandoci sull'importanza che possiedono le coordinate, possiamo notare come il coefficiente crolli per entrambi i dati a livello di peso con il peggioramento delle prestazioni. Visualizzando l'immagine riguardante le feature della rete addestrata con $+5°$ di latitudine e longitudine (*figura 9.2*), si può notare come la longitudine risulti più importante della latitudine; questa è sicuramente un'anomalia dettata dal fatto che le performance con queste specifiche coordi-

Figure 8: Distribuzione dei dati normalizzati: dati naturali [1], latitudine+5° e longitudine+4° [2], latitudine+5° e longitudine+5° [3]

nate raggiungono la soglia del minimo dell'accuracy consentita da SVM, non a caso l'ordine di grandezza del coefficiente rispetto alle altre feature confrontate è molto minore. In pratica nessuna delle due feature aiuta la classificazione.

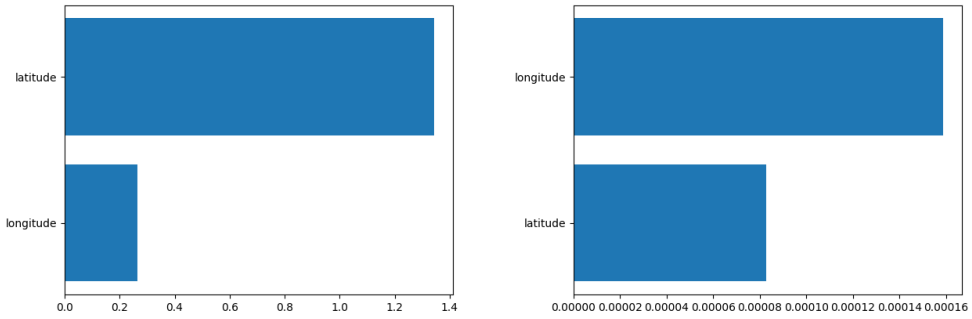

Figure 9: Feature importance della rete addestrata con coordinate alterate: latitudine+5° e longitudine+4° [1], latitudine+5° e longitudine+5° [2]

4.4 Prestazioni delle feature combinate

Facendo un'analisi con tutte e tre le tipologie di coordinate le prestazioni si migliorano: utilizzando le coordinate naturali si ottiene un'accuratezza che raggiunge 0.96; mentre utilizzando i dati modificati si raggiunge l'accuratezza prima del 0.924 e poi del 0.89 (*figura 11*) che, anche se minori rispetto al precedente risultato, comunque aumentano le prestazioni ottenute con le sole immagini, il che ci porta ad essere pienamente soddisfatti dei risultati. In queste tre situazioni si possono notare molte cose guardando la feature importance: l'aggiunta di uno o più dati che risultano importanti nella classificazione (come nel nostro caso l'aggiunta della latitudine e della longitudine), può causare un diverso ranking d'importanza di esse, come si può notare confrontando la rete senza coordinate (solo feature d'immagini) con le reti con anche le coordinate (rete con feature d'immagini e coordinate). Si può constatare però che alcune feature, nonostante l'aggiunta delle coordinate, rimangono comunque importanti anche se varia il loro peso, ciò è dovuto al fatto che i pesi si modificano leggermente e, poiché molte feature hanno la

stessa importanza, la lista cambia ordine. Fattore più importante da notare è che, utilizzando le coordinate naturali, le feature più importanti diventano latitudine e longitudine con un gap molto ampio rispetto alle precedenti. Nei due grafici, utilizzando invece le coordinate alterate (*figura 10.2* e *figura 10.3*), si può notare come la longitudine perda nettamente importanza scomparendo dalle 25 feature più importanti; mentre la latitudine, seppur peggiorando il proprio peso, continua ad essere di fondamentale importanza nell'accuratezza della classificazione.

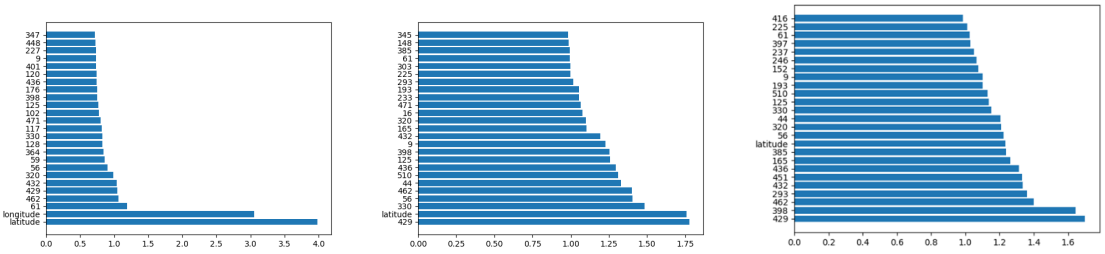

Figure 10: Feature importance (prime 25) della rete addestrata con le feature delle immagini e con le coordinate geografiche normalizzate: dati naturali [1], latitudine+5° e longitudine+4° [2], latitudine+5° e longitudine+5° [3]

5 Conclusioni

In questo progetto ho testato un metodo di classificazione di immagini e di confronto di dati con l'utilizzo del deep learning e del linguaggio di programmazione Python. Ho utilizzato una rete ResNet-18 e ho confrontato le prestazioni di essa in modalità Feature extractor e Fine-Tuning (dalla quale ho estratto le feature delle immagini dal penultimo layer) e di un classificatore SVM lineare che ho impiegato per ogni classificazione dei dati, che sono risultati sufficienti per l'intero progetto. I risultati rispettano a pieno le aspettative che mi ero prefissato: le coordinate geografiche (anche quelle alterate) migliorano le performance della rete. La rete con le sole immagini ci fa ottenere prestazioni elevate che, comunque combinate con i dati geografici, portano ad un innalzamento ulteriore delle performance da 0.875 (feature valutate attraverso SVM) a 0.96 con l'utilizzo delle coordinate reali, perciò possiamo ritenerci soddisfatti considerando il fatto che il progetto raggiunge i risultati prefissati. Un ulteriore sviluppo per questo progetto

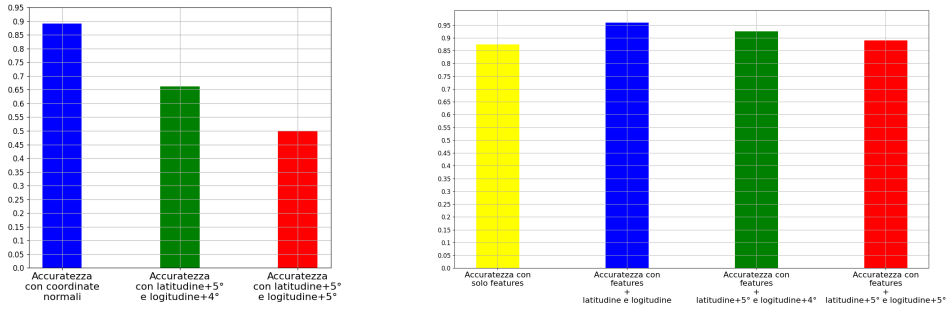

Figure 11: Istogrammi per rappresentare l'accuratezza delle varie reti: confronto reti addestrate con solo coordinate [1], confronto reti addestrate con le feature delle immagini [2]

potrebbe essere quello di utilizzare modelli più performanti di ResNet-18, come ad esempio VVG16 o anche lo stesso ResNet-50, per vedere come al cambiamento del modello cambiano le performance. Un'altra cosa che sarebbe interessante testare sarebbe il confronto con dataset di grandezze diverse e con l'utilizzo di un k-fold cross-validation per la suddivisione del dataset, per vedere come le prestazioni variano al variare di dataset diversi [30] [31].

6 Bibliografia

[1] Machine learning: https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-machine-learning/

[2] CNN: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] Vipera Aspis: https://it.wikipedia.org/wiki/Vipera_aspis

[4] Vipera Berus: https://it.wikipedia.org/wiki/Vipera_berus

[5] Python: https://www.python.org/

[6] PyCharm: https://www.jetbrains.com/pycharm/

[7] Torch: https://pypi.org/project/torch/

[8] Torchvision: https://pytorch.org/vision/stable/index.html

[9] Matplotlib: https://matplotlib.org/

[10] Scikit-learn: https://scikit-learn.org/stable/

[11] Pandas: https://pandas.pydata.org/

[12] Plotly: https://plotly.com/

[13] Deep learning: https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-deep-learning/

[14] iNaturalist: https://www.inaturalist.org/

[15] RandomResizedCrop(): https://pytorch.org/vision/main/generated/torchvision.transforms.RandomResizedCrop.html

[16] ImageNet: https://www.image-net.org/update-mar-11-2021.php

[17] ResNet-18:
1) https://it.mathworks.com/help/deeplearning/ref/resnet18.html
2) https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[18] Ou, Xianfeng and Yan, Pengcheng and Zhang, Yiming and Tu, Bing and Zhang, Guoyun and Wu, Jianhui and Li, Wujing - "Moving Object Detection Method via ResNet-18 With Encoder–Decoder Structure in Complex Scenes", in IEEE Access, vol. 7, pp. 108152-108160, 2019: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8781779

[19] Convolution, Padding, Stride, and Pooling in CNN: https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26

[20] ReLu layer: https://www.educative.io/answers/what-is-a-relu-layer

[21] MaxPool and AvgPool: https://iq.opengenus.org/maxpool-vs-avgpool/

[22] Fully Connected Layer: https://iq.opengenus.org/fully-connected-layer/

[23] Feature extraction: https://pytorch.org/vision/stable/feature_extraction.html

[24] SVM linear: https://scikit-learn.org/stable/modules/svm.html

[25] roc_auc_score(): https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

[26] curva ROC e AUC: https://it.mathworks.com/discovery/roc-curve.html

[27] Ignazio Barraco - "Valutazione delle prestazioni delle Reti Neurali Convoluzionali su immagini affette da distorsione", 2019: https://webthesis.biblio.polito.it/13125/1/tesi.pdf

[28] Normalizzazione: https://learn.microsoft.com/it-it/azure/machine-learning/component-reference/normalize-data

[29] coef_(): http://www.stat.yale.edu/Courses/1997-98/101/correl.htm

[30] Top 4 Pre-Trained Models for Image Classification with Python Code: https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

[31] Cross-validation: https://scikit-learn.org/stable/modules/cross_validation.html

Ringraziamenti

Sono giunto al termine di questo meraviglioso, quanto faticoso, percorso. Voglio ringraziare ogni singola persona che in questi ultimi tre anni e mezzo mi ha detto anche solo una parola d'incoraggiamento: probabilmente senza il vostro contributo non ce l'avrei mai fatta. Ci tengo a ringraziare il professore Piernicola Oliva per avermi accompagnato nell'ultima parte di questa esperienza. Voglio ringraziare in particolare tutta la mia famiglia, sarebbe riduttivo dire un grazie solo per gli anni universitari, probabilmente non smetterò mai di ringraziarvi a sufficienza per tutto quello che mi avete insegnato e dato a livello umano. Ringrazio la mia ragazza Anna per esserci sempre stata in qualsiasi momento, sia di felicità che di tristezza, mi hai reso una persona migliore. Ringrazio i miei amici di una vita per gli infiniti momenti passati assieme. Per ultimo ci tengo a ringraziare il mio amico e collega Andrea, mi hai fatto capire quanto il lavoro di squadra possa giovare nel percorso scolastico, un pezzo di laurea lo devo al lavoro svolto con te.