UNIVERSITÀ DEGLI STUDI DI SALERNO

Fondamenti di Intelligenza Artificiale

Corso di Laurea Triennale in Informatica

BEEHAVE

Professore:

Fabio Palomba

Studenti:

Luigi Bacco (0512110510)

Maria Lucia Fede (0512111995)

Irene Gaita (0512110411)

Gianluca Ronga (0512111008)

ANNO ACCADEMICO 2022/2023

Indice

Capitolo 1

Introduzione

Solo in Europa, in trent'anni il numero di api si è ridotto notevolmente e la durata media della loro vita è diminuita. La scomparsa degli impollinatori, in particolare delle api, non è da sottovalutare: gran parte delle principali culture agrarie e delle piante selvatiche da fiore si riproducono grazie a questi insetti.
Da alcuni decenni ormai, numerosissime ricerche hanno messo in luce un declino diffuso e generalizzato dell'ape domestica (Apis mellifera) ed hanno individuato due principali cause:

- La diffusione dell'acaro parassita Varroa Destructor
- L'uso sconsiderato di pesticidi (agrofarmaci)

Nel corso degli anni molteplici sono state le segnalazioni a livello mondiale di mortalità o spopolamenti di alveari con presenza di principi attivi di pesticidi rinvenuti nelle api. La nostra soluzione ha previsto lo sviluppo di una piattaforma web riservata al miele e sull'interesse di tale tematica con l'integrazione di un piano di notificazione concerne la presenza di Varroa Destructor sul corpo delle api. Al seguente link è disponibile l'intero progetto realizzato che contiene la realizzazione della piattaforma dalle fasi di Requirements Analysis fino all'implementazione.

`https://github.com/gianwario/BeeHave`

In particolare, l'implementazione effettiva del modulo di Machine Learning può essere raggiunta tramite il seguente link:

`https://github.com/GameWame/beehaveIA`

1.1 Obiettivo del Sistema

Se chiedete ad un apicoltore quale sia la malattia delle api maggiormente temuta, è probabile che risponderà "la varroasi", causata dal Varroa Destructor.
Il Varroa Destructor è un acaro parassita presente in scala mondiale da circa 30 anni, ed è il principale nemico dell'Apis mellifera, la specie di ape più diffusa al mondo.
Spesso una sua infestazione, se non trattata adeguatamente, può portare alla morte dell'alveare nella stagione autunnale, ovvero quando l'allevamento dei fuchi cessa ed i parassiti iniziano ad attaccare le api operaie.
Il sistema che si intende realizzare ha come obiettivo la classificazione di immagini di api (Apis mellifera in particolare), basandosi sulla presenza o meno di tale parassita.

Come individuare l'acaro varroa?
La Varroa è un parassita che si attacca alle api e può essere visibile come un piccolo corpo rotondo o oblungo, di colore marrone o bianco, sul dorso dell'ape. I non esperti tendono a pensare che l'acaro varroa sia difficile da riconoscere, ma in realtà grazie al nostro sistema l'apicoltore potrà facilmente capire se il parassita ha attaccato l'alveare.

1.2 Obiettivo dell'Agente Intelligente

L'agente intelligente si occuperà della classificazione delle immagini, riconoscendo la presenza, o l'assenza, del parassita Varroa Destructor sul carapace di alcune api.

1.3 Struttura del documento

Nelle sezioni successive si procederà con l'analisi del problema individuato, procedendo con la sua formalizzazione e descrizione dell'ambiente in cui opera. Una sezione sarà dedicata alle modalità scelte per il raccoglimento dei dati e alla creazione e struttura del Dataset risultante. In seguito, si descriverà la soluzione proposta, analizzando modelli, tecnologie e architetture utilizzati per tale scopo.

Capitolo 2

Analisi del problema

In questa sezione viene analizzato il problema individuato, la sua formalizzazione e descrizione mediante il modello PEAS. Oltre ciò, viene descritto l'ambiente in cui l'agente opera, e le sue principali caratteristiche.

2.1 Specifica PEAS

2.1.1 Performance

La misura di performance del modulo si basa sulla bravura dell'agente ad individuare il parassita Varroa Destructor.

2.1.2 Environment

L'ambiente consiste nella foto dell'ape inviata al classificatore.

2.1.3 Actuators

Gli attuatori consistono in un sistema di classificazione e notifica per quanto riguarda la presenza del parassita.

2.1.4 Sensors

I sensori consistono in un sistema di ricezione di immagini di api da classificare.

2.2 Proprietà dell'ambiente

- **Completamente osservabile:** L'agente riceverà sempre lo stato completo dell'ambiente poiché si tratta di un'immagine.
- **Stocastico:** Lo stato successivo dell'ambiente non è determinato, in quanto, non si conosce la nuova immagine passata in input all'agente.
- **Episodico:** Ogni classificazione dell'agente non influenzerà quelle successive.
- **Dinamico:** Ogni immagine fornita può raffigurare un'ape differente.
- **Discreto:** L'ambiente fornisce un numero limitato di percezioni e azioni distinte, definite in modo chiaro, poiché lavora su un numero limitato di pixel e può classificare l'immagine solamente in due modi.
- **Singolo:** L'ambiente consente la presenza di un unico agente.

Capitolo 3

Dataset

Il dataset scelto è composto da immagini di api in formato 300x150 pixel.

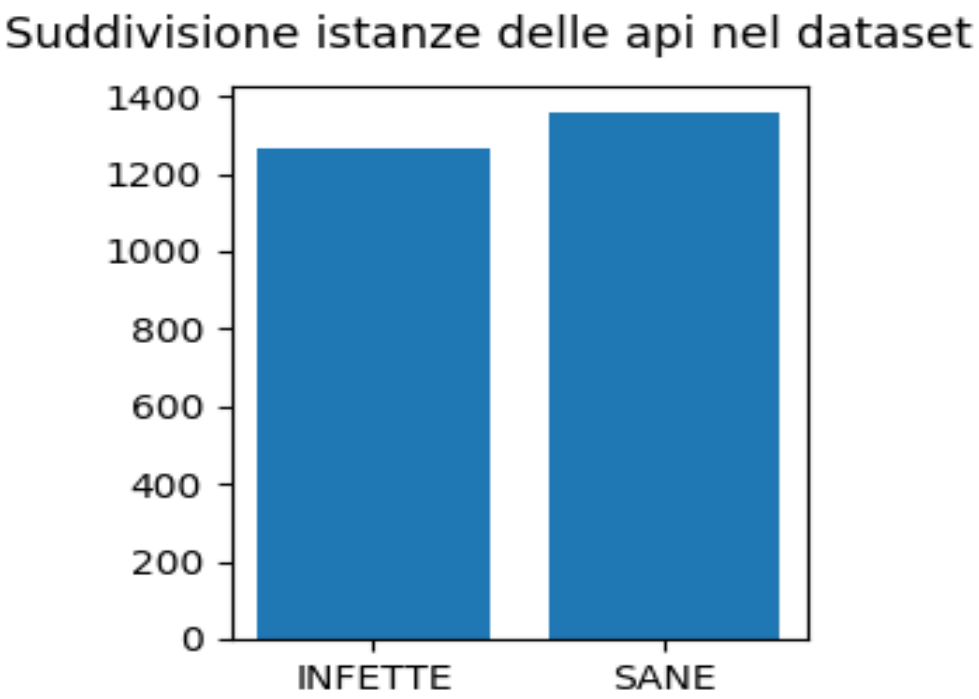
3.1 Raccolta dei dati

Il primo scoglio da superare, per il team, è stato sicuramente la ricerca di un dataset che rispecchiasse perfettamente la natura del problema. Dopo svariate ricerche nei database pubblici, si è scelto di utilizzare il dataset trovato su TensorFlow al seguente link:
`https://www.tensorflow.org/datasets/catalog/bee_dataset`

3.2 Struttura del Dataset

```python
IMG_SIZE = (50, 50)


class VarroavsNoVarroa():
    SI_VARROA = "SiVarroa"
    NO_VARROA = "NoVarroa"
    LABELS = {NO_VARROA: 0, SI_VARROA: 1}
    si_varroa_count = 0
    no_varroa_count = 0
    training_data = []

    def make_training_data(self):
        print("Preparando i dati...\n")
        for label in self.LABELS:
            for f in tqdm(os.listdir(label)):
                try:
                    path = os.path.join(label, f)
                    img = cv2.imread(path)
                    resized = cv2.resize(img, IMG_SIZE)
                    self.training_data.append([resized, np.eye(2)[self.LABELS[label]]])

                    if label == self.SI_VARROA:
                        self.si_varroa_count += 1
                    elif label == self.NO_VARROA:
                        self.no_varroa_count += 1
                except Exception as e:
                    pass
        np.random.shuffle(self.training_data)
        np.save("training_data.npy", self.training_data)
```

Per implementare la struttura del dataset è stato utilizzato un modulo open source messo a disposizione dal fornitore del dataset, il quale ha permesso la categorizzazione delle immagini di api in cartelle apposite: Varroa, Cooling, Wasp, Pollen e None. Essendo il problema incentrato sulla presenza o meno del parassita, si è scelto di utilizzare la cartella contenente le immagini di api con il parassita, categorizzata come Varroa. Dato lo sbilanciamento delle immagini (5000 circa: senza Varroa vs 1000 circa: con Varroa) è stata scelta esclusivamente la cartella categorizzata come None, ovvero una categoria neutra, diminuendo il numero di immagini da 5000 a 3000 circa. Un'ulteriore operazione di undersampling è stata la rimozione di alcune immagini di api, prediligendo la rimozione di immagini con all'interno un numero di api maggiore o uguale a due ed immagini vuote o non nitide, portando ad un bilanciamento quasi perfetto del dataset. Ogni immagine è stata ridimensionata in un formato 50x50 pixel, successivamente aggiunta ad una lista, contenente immagini trasformate in una struttura dati composta da un array multidimensionale di grandezza 50x50x3 e la relativa etichetta. Infine è stato effettuato lo shuffle della lista e poi salvata in un file con estensione `npy`.



Capitolo 4

Soluzione Proposta

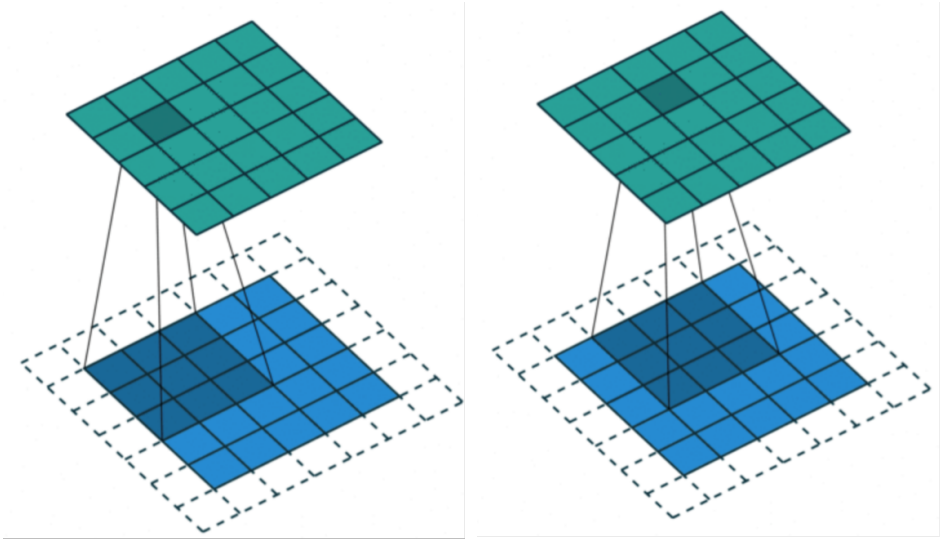
La soluzione proposta è un modello di Machine Learning capace di predire la presenza o meno del parassita posto sopra il carapace dell'ape, in modo da segnalarne la presenza all'apicoltore. La nostra soluzione non solo porterebbe a facilitare le azioni di monitoraggio dell'alveare, ma mirerebbe soprattutto a favorire la qualità del miele prodotto offrendo costantemente un alto livello di affidabilità al cliente.

4.1 Modello

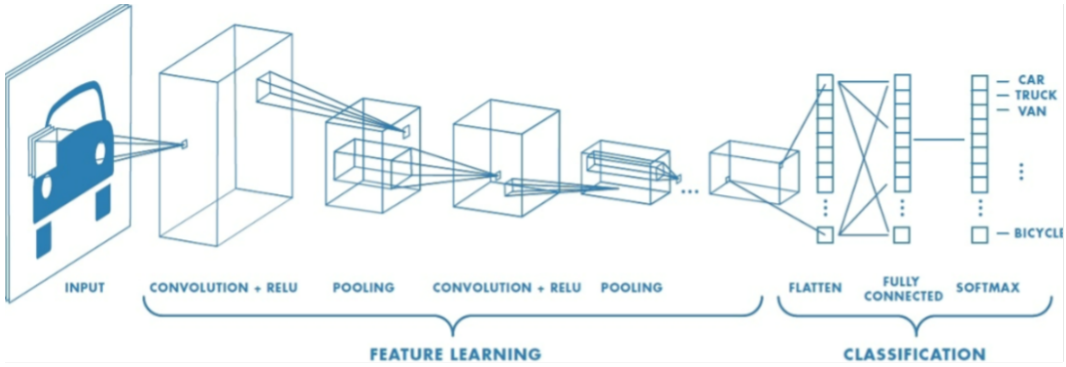
Nel presente paragrafo verrà descritto il modello e verrà illustrato come le proprietà e le caratteristiche di quest'ultimo sono state rispettate. Inoltre, saranno discusse le motivazioni che hanno portato alla scelta di questo modello.

4.1.1 Modello utilizzato

Il modello scelto si basa su una rete neurale convoluzionale, la quale classifica, effettuando varie operazioni, immagini simili.
Una rete neurale convoluzionale presenta un layer di input, uno o più layer nascosti (hidden layer), e un layer di output. Ogni layer ospita quella che viene chiamata "feature map", ovvero la specifica feature che i nodi si preoccupano di cercare. Nell'esempio qui sotto il primo layer potrebbe occuparsi di codificare le linee diagonali; quindi, si "fa scorrere" un apposito filtro (qui nell'esempio un 2×2) sull'immagine, e moltiplicandolo (prodotto scalare) per l'area sottostante. In sostanza significa che all'interno del layer convoluzionale ciascun nodo è mappato solo su un sottoinsieme di nodi di input (campo recettivo), e di fatto moltiplicare il filtro per il campo recettivo di ciascun nodo equivale concettualmente a "far scorrere" il filtro lungo l'immagine di input (windowing).



Il risultato sarà un'altra matrice, leggermente più piccola dell'immagine originale. Ad ogni layer di neuroni si possono applicare eventualmente più filtri, generando quindi più feature map.



Tipicamente ogni layer convoluzionale viene fatto seguire da uno di Pooling, riducendo via via la dimensione della matrice, ma aumentando il livello di astrazione. Si passa quindi da filtri elementari, come appunto linee verticali e orizzontali, a filtri via via più sofisticati, in grado di riconoscere parti dell'immagine, fino all'ultimo livello, fully connected, dove è in grado di classificare, secondo una certa probabilità, l'immagine fornita in input.

4.1.2 Tecnologie ed implementazione del Modello

Il modulo è stato implementato in Python 3.10 utilizzando le librerie OpenCV (`https://docs.opencv.org/4.7.0/`), NumPy (`https://numpy.org/doc/stable/`) e PyTorch (`https://pytorch.org/docs/stable/torch.html`).
Dopo aver effettuato la raccolta dei dati e creato il dataset si è passati alla sua acquisizione ed è stata effettuata l'operazione di feature scaling. Essendo un' immagine, il range dei valori parte da 0 fino a 255, quindi è stata effettuata una semplice divisione per 255 in modo tale da avere un range di valori tra 0 e 1. Per addestrare il modello con più dati possibili è stato utilizzato il 90% del dataset, il restante 10% è stato utilizzato per la fase di test e il calcolo delle metriche. Si è passati poi all'istanziazione della rete neurale ed alla scelta della funzione di ottimizzazione e della funzione di perdita.

Rete Neurale

La rete neurale è composta da un layer di input con 3 canali in ingresso, ognuno relativo al colore. La struttura interna della rete è formata da 13 strati:
Convolution->Batch Normalization->ReLu->Pooling->Convolution->Batch Normalization->ReLu->Pooling->Convolution->Batch Normalization->Relu->Pooling- >Fully Connected.

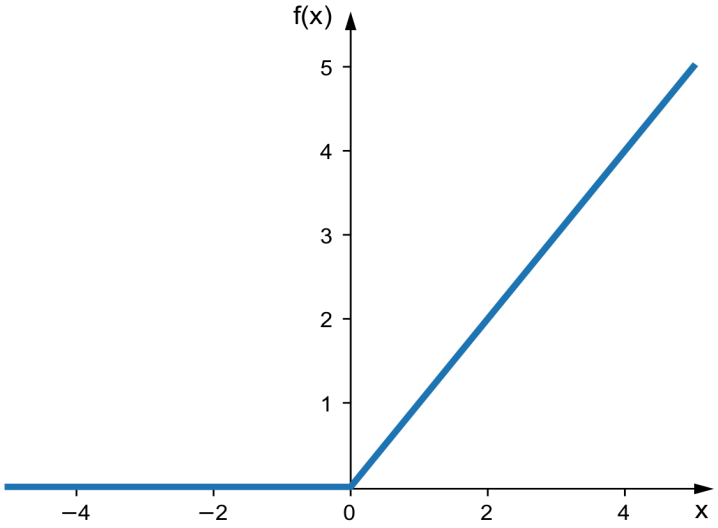
```python
class Net(nn.Module):

    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=128, kernel_size=3, stride=1)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1)
        self.conv3 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=2, stride=1)

        self.bn1 = nn.BatchNorm2d(128)
        self.bn2 = nn.BatchNorm2d(256)
        self.bn3 = nn.BatchNorm2d(512)
        self.fc1 = nn.Linear(512, 2)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.bn1(self.conv1(x))), 3)

        x = F.max_pool2d(F.relu(self.bn2(self.conv2(x))), 3)

        x = F.max_pool2d(F.relu(self.bn3(self.conv3(x))), 3)

        x = x.view(-1, 512)

        x = self.fc1(x)

        return F.softmax(x, dim=1)
```

Dal codice possiamo notare il numero di canali in input, rappresentano l'immagine con ognuno dei tre colori. Attraverso una sperimentazione empirica sono stati scelti i vari canali in output e in input dei vari strati. Ogni canale di output rappresenta un filtro, ogni filtro si occupa di ricercare uno specifico pattern all'interno dell'immagine, la grandezza del filtro è data dal valore del parametro `kernel_size`. La scelta della grandezza del kernel è stata basata sulla seguente formula matematica:

$$(i - k) + 1$$

Prendendo come esempio il primo layer convoluzionale:
i identifica la grandezza dell'immagine, in questo caso 50;
k identifica la grandezza del kernel, in questo caso 3;
il risultato della formula è 48, ovvero un'immagine 48x48 pixel in output. Questa operazione risulta estremamente utile ed importante per tenere traccia della grandezza delle immagini che andranno a formarsi, via via, ad ogni convoluzione.
La scelta di utilizzare la Batch normalization, subito dopo il processo di convoluzione, è data dal fatto di poter ridurre al minimo possibili rumori risultanti, appunto, dall'operazione di convoluzione.
Il layer di output ha una grandezza pari a 2, ovvero, le probabilità delle due possibili classificazioni dell'ape. Queste ultime vengono calcolate attraverso la funzione softmax, la quale permette che la somma delle due probabilità in output sia 1.

Funzione di attivazione



Come funzione di attivazione è stata utilizzata la Relu. Essa permette di mantenere i valori, risultanti dalla convoluzione, maggiori di 0 e di trasformare a 0 i valori negativi: $f(x) = max(0, x)$.
In questo modo il filtro non verrà influenzato da pattern diversi rispetto quello di cui si occupa.

Funzione di ottimizzazione

```
"""Ottimizzatore della rete"""
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)
```

Le reti più frequentemente utilizzate hanno diverse strutture in diverse parti di esse. Per esempio, la prima frazione di una CNN può essere molto spesso composta da strati convoluzionali su immagini grandi, mentre più in fondo alla rete possiamo avere convoluzioni di un gran numero di canali su immagini piccole. Queste due opreazioni sono molto diverse, cosicché un livello di apprendimento che funzioni bene per la prima parte della rete potrebbe non andare bene per le sezioni finali. Ciò implica i livelli di apprendimento adattivi potrebbero essere di una certa utilità. É stata utilizzata la funzione adattiva Adam la quale è, a differenza di altre funzioni, un buono stimatore poco rumoroso della soluzione. Inoltre, facendo un confronto con SGD, è risultato leggermente migliore.

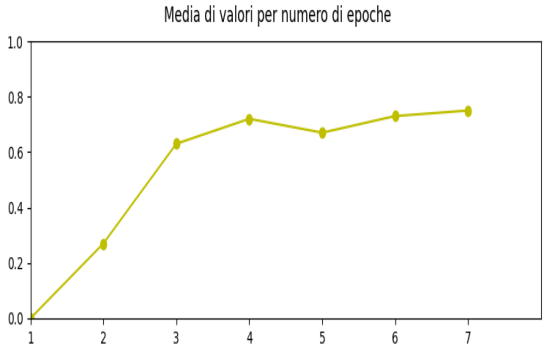
Funzione di perdita

```
"""Funzione di perdita"""
loss_function = torch.nn.L1Loss()
```

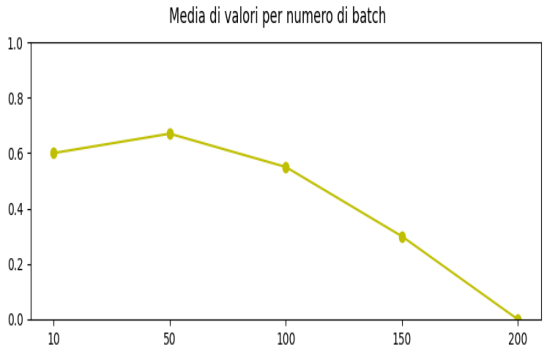
La funzione di ottimizzazione basa le sue modifiche in base alla funzione di perdita. La funzione utilizzata, ed ampiamente consigliata per avere una maggiore nitidezza nelle immagini, è la MAE, funzione basata sull'errore assouluto medio. Facendo ulteriori confronti con le restanti funzioni di perdita si è rivelata la scelta migliore.

Training

Il training dell'algoritmo è stato effettuato per un numero di epoche uguale a 4:



e lunghezza di batch uguale a 50:



- **Epoca**

  Il numero di epoche definisce il numero di volte che l'intero dataset viene visto dalla rete neurale.

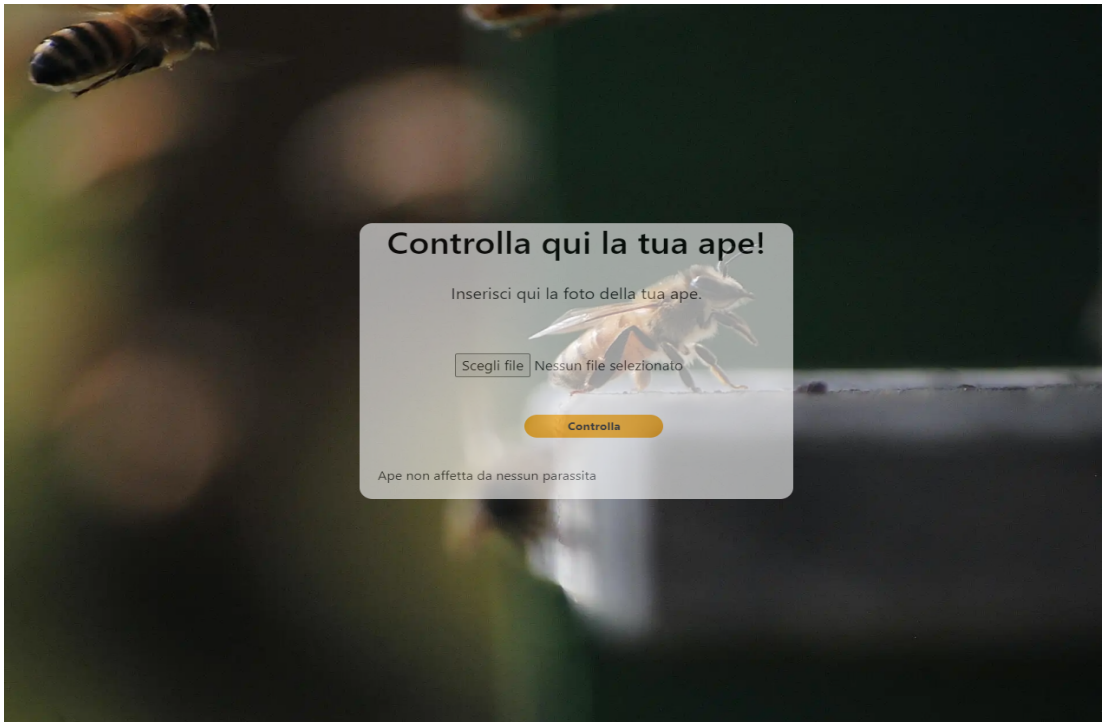
- **Batch**

  La lunghezza di batch definisce il numero di immagini, alla volta, passate all'algoritmo ad ogni iterazione.

La scelta del numero di epoche è stata effettuata tenendo conto del seguente trade-off: underfitting e minor tempo di calcolo per un valore basso, overfitting e maggior tempo di calcolo per un valore alto. La scelta della grandezza del batch è data dal fatto che, con un numero basso, il modello, avendo a disposizione poche immagini, non sarebbe stato in grado di imparare correttamente. Invece, con un numero alto, non avrebbe effettuato un numero soddisfacente di iterazioni, quindi di backwards, per poter migliorare.
Per poter selezionare un ottimo bilanciamento di questi due parametri, è stata utilizzata una metrica di valutazione simil Elbow point.

4.2 Integrazione in BeeHave

Non avendo la possibilità di integrare fisicamente i sensori all'interno dell'alveare, si è pensato di fornire una simulazione dei sensori attraverso una pagina web. L'Apicoltore, sulla piattaforma web BeeHave, avrà la possibilità di inserire, in una pagina dedicata, l'immagine dell'ape da controllare. La presenza/non presenza del parassita Varroa Distructor verrà successivamente notificata all'apicoltore tramite un messaggio di testo.
Il modulo è stato implementato utilizzando un adapter, il quale, oltre a richiamare il metodo del modulo, fornisce il relativo messaggio di testo in base alla predizione effettuata.



4.3 Risultati e conclusioni

Premessa:
Essendo un argomento di approfondimento e, soprattutto, nuovo per noi, tutte le scelte effettuate (iperparametri, struttura della rete, scelta delle funzioni etc.) si sono basate su suggerimenti e consigli forniti dalle fonti di studio. Eventualmente, è stata comunque effettuata, da tutti i membri del team, una continua sperimentazione empirica, tenendo conto dell'underfitting, overfitting, del tempo di esecuzione dell'algoritmo e dei vincoli che una rete neurale comporta. Un esempio di questi vincoli è dato dal fatto che il numero di input di un layer deve essere uguale al numero di output del layer precedente.

Il miglior modello ha avuto i seguenti risultati:

Accuracy	67%
Precision	62%
Recall	92%

Priorità maggiori sono state date a `precision` e a `recall`, in quanto indicano quanto bene il modello riesce a trovare api affette da questo parassita.
I risultati sono relativamente soddisfacenti, in quanto, all' inizio, le metriche sono state inaccettabili. Il modello forniva un'accuracy del 70%, ma una precision e recall pari allo 0%, ovvero, classificava tutte le api come sane. Analizzando il problema si è stati in grado di risolverlo. Si è arrivati ad un punto in cui il modello ha iniziato a convergere, in media, verso i valori mostrati precedentemente. Data la natura degli algoritmi non deterministici (si è cercato di togliere questa parte non deterministica ma, non è stato possibile), utilizzati dalla libreria durante le varie fasi, il risultato, per input identici, si è mostrato spesso differente; quindi, si è pensato di salvare il miglior modello risultante.
Il modello è stato addestrato per effettuare predizioni di immagini di api secondo determinati vincoli, ovvero:

- Lo sfondo deve essere di colore verde;
- L'immagine deve raffigurare il dorso dell'ape;
- L'immagine deve raffigurare interamente l'ape;
- La grandezza dell'immagine deve coincidere con la grandezza dell'ape;

Il mancato rispetto di questi vincoli porterebbe, quasi sicuramente, a fenomeni di data leakage, e quindi, a predizioni errate. In conclusione, approfondire un argomento di questo tipo è stato molto interessante, ed è stata, soprattutto, un'esperienza positiva esser riusciti ad implementare questo modello.

Capitolo 5

Sviluppi Futuri

In questo paragrafo mostriamo cosa prevedono gli sviluppi futuri.
Puntiamo ad apportare rilasci successivi mantenendo alta la qualità dei nostri servizi, così abbiamo previsto le seguenti migliorie da effettuare:

1. L'apicoltore avrà la possibilità di visualizzare la presenza o meno del parassita nella pagina 'Stato Alveare' e in caso di "varroasi" verrà reindirizzato ad una pagina informativa sull'uso accurato dei pesticidi (agrofarmaci).

2. Grazie all'ausilio di sensori all'interno dell'alveare, le foto verranno scattate ed inviate automaticamente al modello di machine learning. In questo caso l'alveare dovrà essere preparato seguendo i vincoli citati precedentemente.

Glossario

Underfitting

L'underfitting si verifica quando la classificazione si basa su pochi parametri; quindi, il suo Bias diventa troppo elevato.

Overfitting

L'overfitting si verifica quando la classificazione si basa su troppi parametri; quindi, la sua varianza diventa troppo elevata visto l'alta sensibilità del modello verso i dati.

Dataset

Un dataset è un insieme di dati strutturati, organizzati in forma relazionale.

Modello

Un modello di Machine Learning è un file sottoposto a training per riconoscere determinati tipi di modelli.

Rete Neurale

Le reti neurali simulano il comportamento del cervello umano, imitando il modo in cui i neuroni si inviano segnali.

Astrazione

Processo concettuale nel quale si isola un elemento da tutti gli altri ai quali era connesso e lo si considera quale particolare oggetto di interesse.

Bilanciamento

Al fine di evitare incongruenze nei calcoli svolti dall'algoritmo, è necessario che il numero di valori di una data classe debba essere simile al numero di valori dell'altra classe in esame.

Bibliografia

[1] https://atcold.github.io/pytorch-Deep-Learning/it/week05/05-2/

[2] https://atcold.github.io/pytorch-Deep-Learning/it/week11/11-1/

[3] https://learn.microsoft.com/it-it/windows/ai/windows-ml/tutorials/pytorch-train-model

[4] https://it.wikipedia.org/wiki/Varroa_destructor

[5] https://www.slideshare.net/ashraybhandare/deep-learning-cnn-and-rnn

[6] https://www.youtube.com/watch?v=r6QxSUHmWGg&list=PLlck-sAzCMzL5HtssT5o69kkWuUz153IO&index=13

[7] https://www.youtube.com/watch?v=nToH7FP-xV0&list=PLlck-sAzCMzL5HtssT5o69kkWuUz153IO&index=14

[8] https://www.youtube.com/watch?v=oy8rBOhYt5w

[9] https://github.com/BeeAlarmed/BeeAlarmed

[10] https://www.youtube.com/watch?v=37bhzGfk

[11] https://www.youtube.com/watch?v=D1LzRWG7LdA&t=16s

[12] https://www.salute.gov.it/portale/sanitaAnimale/dettaglioContenutiSanitaAnimale.jsp?lingua=italiano&id=256&tab=1