Progetto di Sistemi Digitali M

Daniele Menchetti

Anno Accademico 2020/2021

Indice

Capitolo 1

Introduzione

In questo progetto ci si è posti l'obiettivo di creare un sistema software (*Dog Breed Recognizer*) in grado di analizzare immagini raffiguranti cani ed identificare la razza di quest'ultimi. Il sistema riceverà in ingresso un'immagine in cui è presente un cane e, attraverso un processo di analisi, produrrà in uscita un responso che identifichi la razza canina dell'animale se puro oppure le due razze da cui esso proviene se incrocio.

Per sviluppare tale software si fa ricorso ad una rete neurale di tipo convoluzionale (**CNN**) e quindi si ricorre ad una branchia dell'intelligenza artificiale: il deep learning.
Essa si occuperà di risolvere il problema di classificazione di un'immagine in una delle classi (o razze) sulle quali è stata precedentemente addestrata. Inoltre, se la seconda classe predetta con valore più alto risulta essere sufficientemente grande, allora il cane verrà interpretato come incrocio in quanto la rete ha riconosciuto tratti dell'animale inerenti anche ad un'ulteriore razza.

Questa rete, sviluppata tramite il framework **TensorFlow 1.15** e addestrata con **NVIDIA GeForce GTX 850M**, costituirà il motore dell'intero sistema software sviluppato sia come applicazione mobile **Android** che come applicativo client-server (implementato su **Raspberry Pi 3B**) fruibile attraverso browser.

La scelta di adoperare una rete neurale piuttosto che altri strumenti ricade sul fatto che esse hanno riscosso particolare successo negli ultimi anni[1] per risolvere problemi di classificazione in quanto riescono ad estrarre features di ciascuna classe in maniera autonoma senza che sia necessario impartire delle regole o conoscenza a priori sulle classi stesse (come accade invece nei sistemi simbolici).

[1] I primi segni di reti neurali risalgono alla fine della prima metà del XX secolo, ma l'evoluzione tecnologica non permetteva ancora possibili applicazioni.

Capitolo 2

Modello di rete neurale

In questo capitolo si affronta il cuore dell'intero sistema software: lo sviluppo della rete neurale.
Essa è stata in un primo momento progettata per classificare **120** razze diverse (problema difficile) e in un secondo momento per classificarne **10** (problema facile).

2.1 Modello con 120 classi

La rete neurale qui sviluppata viene addestrata su un dataset composto da 120 razze canine.
La classificazione di immagini in un numero di classi così elevato presenta un problema difficile e dunque, durante la fase di testing, si accosta al valore di **top-1**[2] il valore di **top-5**[3] per definire con maggior precisione il livello di accuratezza della rete.

Il **dataset** utilizzato è composto da **10362** immagini totali etichettate a

[2]top-1: percentuale di accuratezza ottenuto considerando una predizione corretta se il valore più elevato prodotto corrisponde alla classe effettiva dell'immagine.

[3]top-5: percentuale di accuratezza ottenuto considerando una predizione corretta se tra i 5 valori più elevati prodotti vi è un valore corrispondente alla classe effettiva dell'immagine.

3 canali (RGB) di risoluzione 128x128 pixels divise in 9326 (90%) per il training e 1036 (10%) per il testing.

In un primo momento si prende in esame la rete neurale qui di seguito rappresentata.

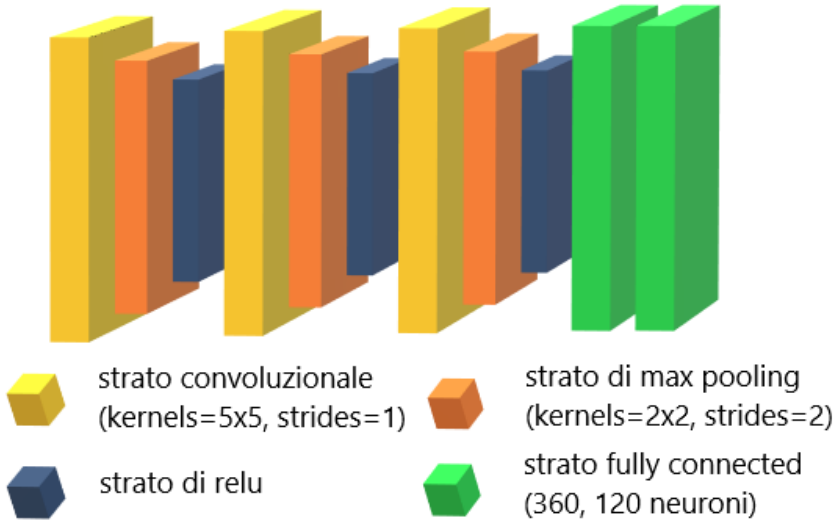

Figura 2.1: Architettura rete neurale

I valori adottati nella fase di training sono:

- training steps: 10 000
- learning rate: 10^{-3}
- batch size: 128

Di seguito sono riportati il log ed il grafico di training.

```
number of trainable parameters: 320844
step:1000/10000 | loss:0.488916 | elapsed time:  2m:58s
step:2000/10000 | loss:0.019615 | elapsed time: 12m:20s
step:3000/10000 | loss:0.060038 | elapsed time: 15m:12s
step:4000/10000 | loss:0.000603 | elapsed time: 18m: 4s
step:5000/10000 | loss:0.000949 | elapsed time: 20m:56s
step:6000/10000 | loss:0.000249 | elapsed time: 23m:48s
step:7000/10000 | loss:0.000174 | elapsed time: 26m:41s
step:8000/10000 | loss:0.000893 | elapsed time: 29m:33s
step:9000/10000 | loss:0.000422 | elapsed time: 32m:31s
Training ended!
```

Figura 2.2: Log di training

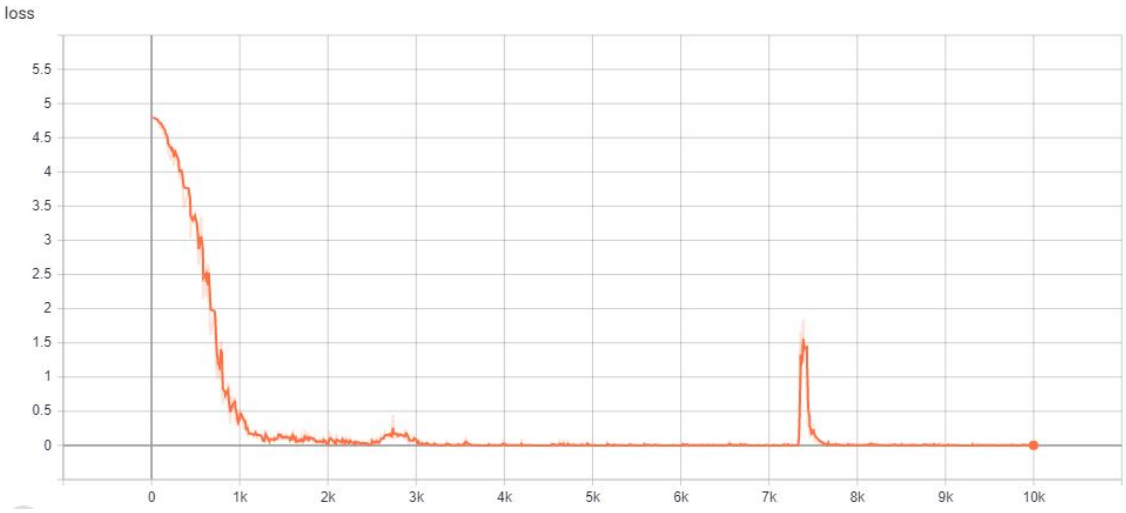

Figura 2.3: Grafico ottenuto tramite Tensorboard

Si può notare che il valore di loss tende diminuisce con l'aumentare del numero di steps: questo è il classico comportamento di una rete che sta imparando.

Tuttavia, testando la rete su immagini mai viste si ottengono percentuali di accuratezza basse di **top-1** (3%) e **top-5** (10%).
Inoltre, a cavallo degli steps 7k e 8k vi è un picco di loss.

Si cerca quindi di alterare il modello di rete neurale affinché si possa produrre un'accuratezza il più possibile elevata, riducendo eventuali picchi di loss.

Come prima modifica si è alterata la dimensione del kernel convoluzionali a **3x3**.
Tale scelta ha portato ad un lieve miglioramento (**top-1**: 4% e **top-5**: 10%).
Successivamente si sono sostituiti gli strati di **relu** con strati di **tanh** ottenendo una **top-1** di 5% ed una **top-5** di 11%.

Si sono poi fatti ulteriori esperimenti atti ad aggiungere-rimuovere-modificare strati (average pooling al posto di max pooling, variazione della dimensione dei filtri convoluzionali, numero di neuroni nei fully connected etc...), ma nessuno di essi ha portato benefici.
Inoltre, anche aumentando o diminuendo gli steps di training non si ottiene un miglioramento della rete.

In un secondo momento si è deciso di attuare una sorta di **image augmentation** andando ad aggiungere al dataset, per ciascuna immagine, le rotazioni di 90°, 180° e 270°.
Questo è stato fatto per aumentare i campioni di training e testing in modo da molte più immagini per ciascuna razza di cane (passando da 80-120 a 320-480 immagini per razza).
Purtroppo questo metodo contribuisce in maniera negativa al livello di accuratezza della rete andando a peggiorarne il valore.

Ritenendo tale situazione un possibile **overfitting** si prova ad aggiungere alcuni strati di **dropout** (strati che vanno a disattivare alcuni neuroni durante il training) e ridurre il learning rate, ma i risultati ottenuti non superano il livello di accuratezza precedente.

2.2 Modello con 10 classi

In un secondo momento si è deciso di ridurre fortemente il numero di classi in uscita così da passare da un problema difficile (100 o più classi) ad uno facile (decine di classi).

Tale decisione è stata presa considerando il dominio applicativo in esame: molte razze canine, spesso, non differiscono le une dalle altre in maniera significativa, bensì presentano **features comuni**, rendendo di fatto complesso il processo di classificazione della rete neurale.

Il **criterio** adottato per scegliere 10 razze da preservare nel dataset si basa sul numero di immagini per ciascuna razza presenti (si scelgono le razze che hanno più esempi e con un numero simile di immagini tra di loro) e sulle differenze visive di ciascuna razza (si scelgono le 10 razze più diverse tra di loro).

Le razze scelte sono dunque:

- Beagle
- Cane da pastore maremmano abruzzese
- Dalmata
- Leonberger

- Levriero afgano
- Levriero scozzese
- Maltese
- Pitbull
- Siberian Husky
- Volpino di Pomerania

Il **dataset** che si ottiene è composto da **1155** immagini etichettate divise in 1030 (90%) per il training e 125 (10%) per il testing.

Attraverso le informazioni precedentemente ottenute (modello con 120 classi) che hanno contribuito ad apportare benefici al modello, si è deciso di partire direttamente dall'architettura di rete qui di seguito raffigurata.

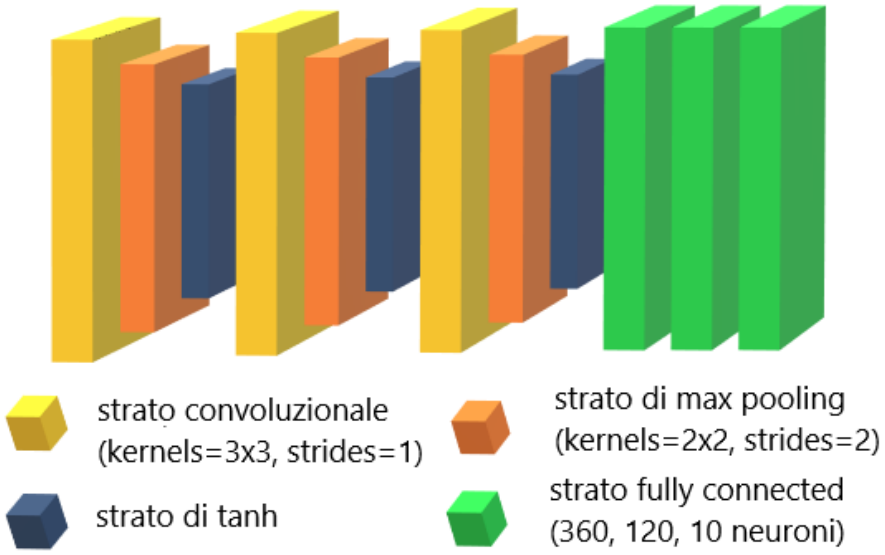

Figura 2.4: Architettura rete neurale

I valori adottati nella fase di training sono:

- training steps: 10 000
- learning rate: 10^{-3}
- batch size: 128

Di seguito sono riportati il log ed il grafico di training.

```
number of trainable parameters: 321622
step:1000/10000 | loss:0.031665 | elapsed time:  2m:18s
step:2000/10000 | loss:0.001445 | elapsed time:  4m:33s
step:3000/10000 | loss:0.000565 | elapsed time:  6m:46s
step:4000/10000 | loss:0.000218 | elapsed time:  8m:59s
step:5000/10000 | loss:0.000107 | elapsed time: 11m:13s
step:6000/10000 | loss:0.000062 | elapsed time: 92m:48s
step:7000/10000 | loss:0.000034 | elapsed time: 94m:59s
step:8000/10000 | loss:0.000020 | elapsed time: 97m:12s
step:9000/10000 | loss:0.000012 | elapsed time: 99m:23s
Training ended!
```

Figura 2.5: Log di training

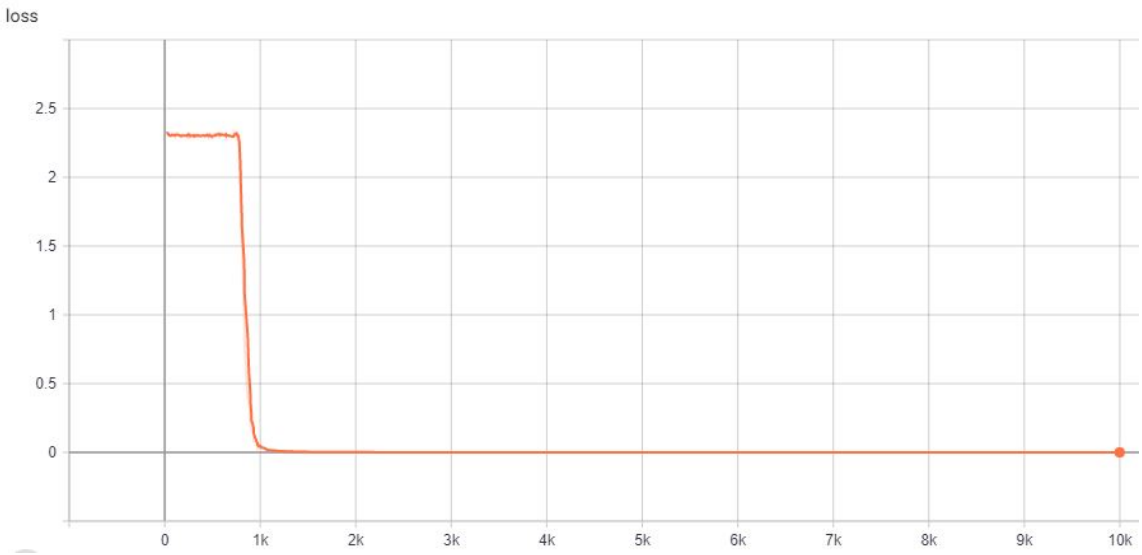

Figura 2.6: Grafico ottenuto tramite Tensorboard

Come è possibile notare il valore di loss tende vertiginosamente a 0: da ciò è possibile concludere che la rete abbia imparato a classificare pressoché tutti i campioni di training.

Tale affermazione sarà però valida anche nel caso delle immagini di testing? La risposta, come vedremo, è no.
Infatti testando il modello ottenuto su immagini che esso non ha mai visto otteniamo una percentuale di accuratezza del 37% per la **top-1** e 83% per la **top-5**.
Questo dimostra che la rete non riesce a generalizzare ed evidenzia una possibile situazione di **overfitting**.

Per risolvere tale situazione si prova ad aggiungere alcuni strati di **dropout** e ridurre il learning rate, ma i risultati ottenuti non superano comunque il 35% di accuratezza (**top-1**).

Perciò, come ultimo esperimento, si è deciso di ridurre il numero di training steps per constatare se la rete abbia raggiunto un livello di accuratezza maggiore già in precedenza.
Dopo alcuni tentativi si arriva a concludere che il modello migliore ottenuto è quello con **8000 steps**, il quale ha prodotto una percentuale di accuratezza del 48% per la **top-1** e 81% per la **top-5**.

Quest'ultimo modello sarà adottato per le successive fasi di conversione e sviluppo.

Capitolo 3

Sviluppo software su sistema embedded

In questo capitolo si procede a delineare i passi di sviluppo dell'intero software in due sistemi embedded: sistemi con **Android OS** e Raspberry Pi 3B con **Raspberry Pi OS**.

Come introdotto nel capitolo 1 il software dovrà riconoscere, oltre alla razza canina, i casi in cui nell'immagine è presente un cane di razza **pura**, **mista** oppure il caso in cui non vi sia proprio un cane o vi sia una razza non nota al sistema (situazione di **invalidità**).

L'algoritmo che si occupa di questa parte, di seguito riportato in codice Python, è stato modellato sperimentalmente facendo alcuni tentativi su immagini che rispecchiano i tre scenari precedentemente citati.

```python
dog = Cane(...)
#Classe Cane:
#    percent1 --> 1° valore maggiore prodotto dalla rete (percentuale)
#    razza1   --> razza associata a percent1
#    percent2 --> 2° valore maggiore prodotto dalla rete (percentuale)
#    razza2   --> razza associata a percent2
#    tipo     --> tipo di cane in esame {'invalido', 'puro', 'misto'}

soglia = 1
if (dog.percent1 < 0.7):
    dog.tipo = 'invalido'
else:
    if (dog.percent1 >= 0.7 and dog.percent1 < 0.8):
        soglia = 0.15
    else:
      if (dog.percent1 >= 0.8 and dog.percent1 < 0.85):
          soglia = 0.1
      else:
          if (dog.percent1 >= 0.85 and dog.percent1 < 0.9):
              soglia = 0.07
          else:
              if (dog.percent1 >= 0.9 and dog.percent1 < 0.95):
                  soglia = 0.04

    if (dog.percent2 >= soglia):
        dog.tipo = 'misto'
    else:
        dog.tipo = 'puro'
```

Figura 3.1: Algoritmo discriminante i tre scenari

3.1 Conversione modello TensorFlow in TensorFlow Lite

Il primo passo per sviluppare il software consiste nel convertire il modello TensorFlow, ottenuto a termine della fase di training, in un modello TensorFlow Lite.

Per raggiungere questo obiettivo si sono eseguite alcune conversioni intermedie: conversione del modello TensorFlow (.meta, .index, .data) nel file di testo protocol buffer(.pbtxt), conversione di pbtxt in modello frozen (.pb), conversione del modello frozen in modello TensorFlow Lite (.tflite).

Il modello TensorFlow Lite ottenuto pesa **1,2 MB**.
E' disponibile, inoltre, una rappresentazione degli strati del modello tramite il seguente link, ottenuto con Netron.

3.2 Sviluppo su Android OS

Lo sviluppo su Android OS consiste nell'implementazione di un'applicazione per sistemi mobile che riceva in input un'immagine, esegua una predizione della razza canina attraverso l'interprete TensorFlow Lite e restituisca all'utente il risultato a video.

L'applicazione realizzata, la quale è stata fatta girare su un **Samsung Galaxy S10**, è molto efficiente nel produrre risultati: infatti, limitandoci alla predizione dell'interprete TensorFlow Lite, essa richiede **8.4 ms** per essere completata.

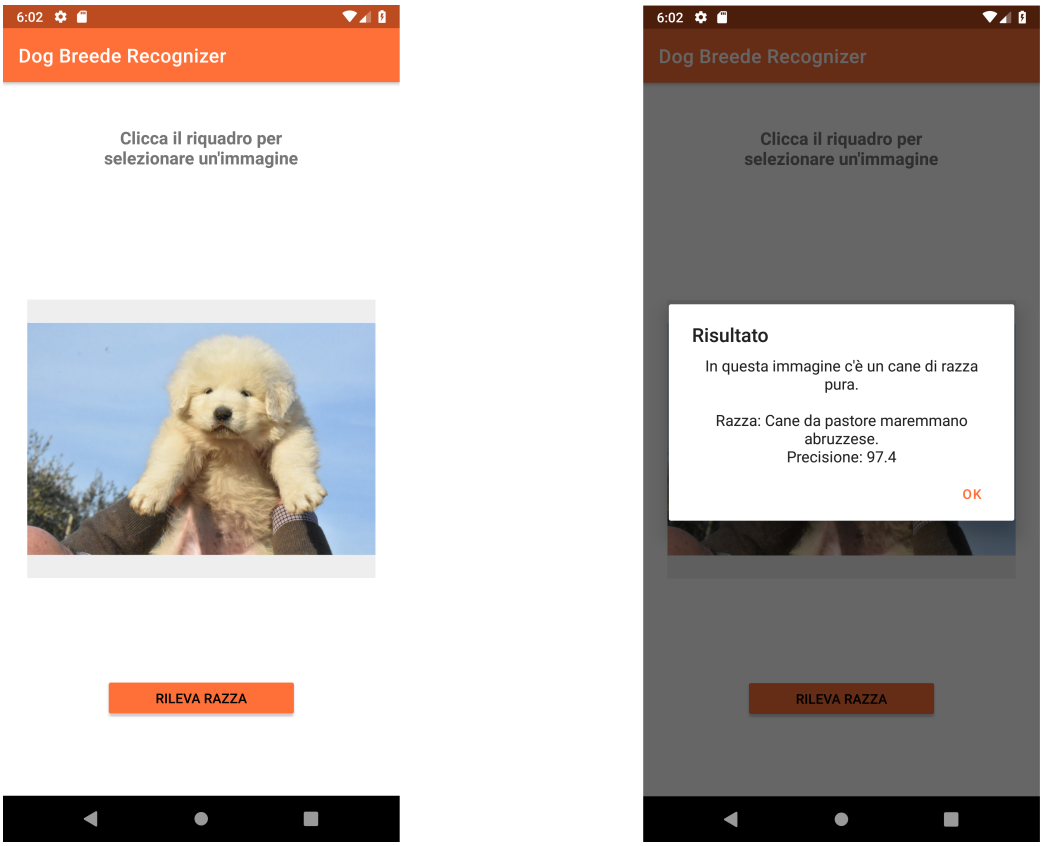

Figura 3.2: Screenshot applicazione Android

3.3 Sviluppo su Raspberry Pi OS

Lo sviluppo su Raspberry Pi OS consiste nell'implementazione di un architettura client-server la quale, lato server (**Raspberry Pi 3B**), si occupa di eseguire un software Python che a sua volta produca una predizione attraverso l'interprete TensorFlow Lite.

Nel dettaglio l'intero sistema è composto da una pagina web attraverso cui è possibile caricare un'immagine, inviarla al server con una richiesta AJAX e ricevere in un secondo momento una risposta da esso. Lato server Raspberry si occupa, tramite codice PHP, di salvare l'immagine sul server e avviare il software Python che a sua volta richiederà l'intervento dell'interprete TensorFlow Lite per generare una predizione sull'immagine. Infine il risultato verrà elaborato dall'algoritmo citato (Figura 3.1) e trasmesso al client affinché si possa visualizzare a video.

L'esecuzione, limitata al solo processo di predizione, richiede in questo caso **45.1 ms** per essere completata.

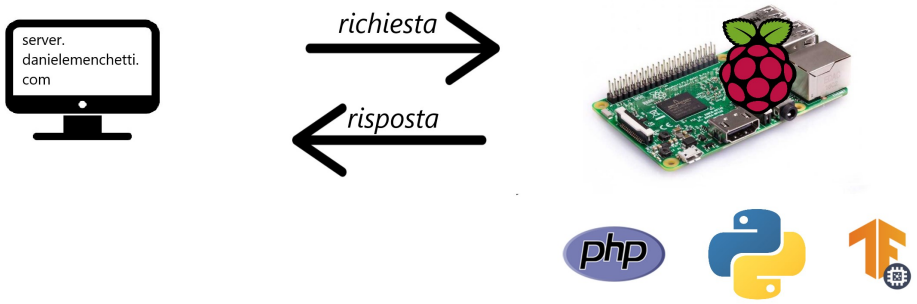

Figura 3.3: Architettura del sistema

Tale sistema è fruibile al seguente link.

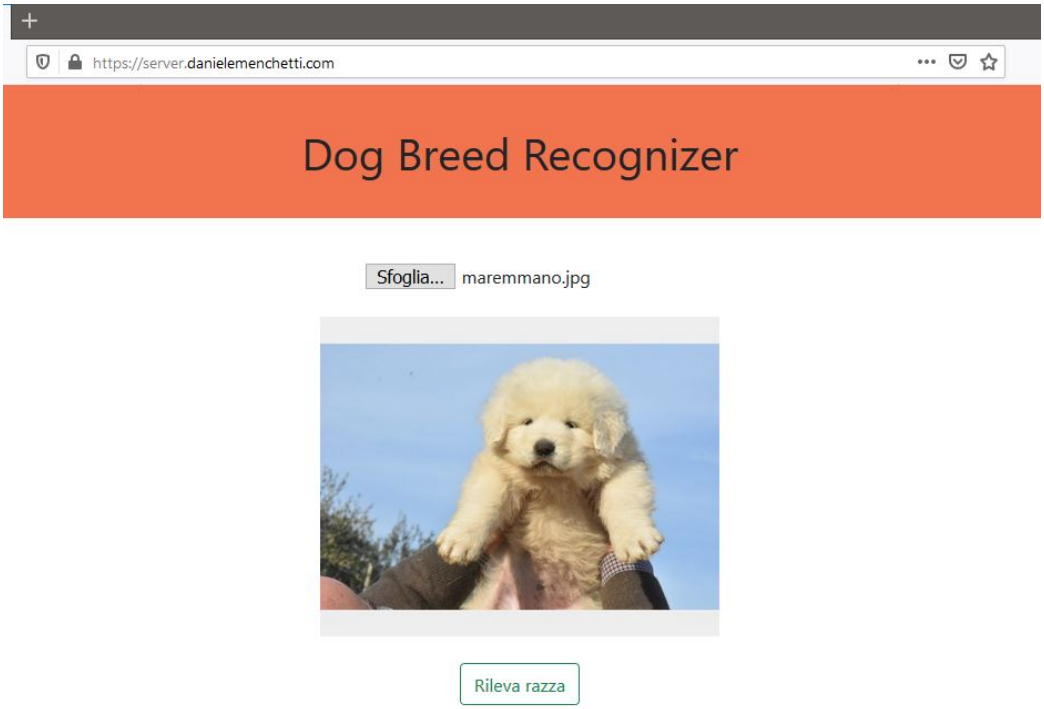

Figura 3.4: Screenshot sistema client-side (1)

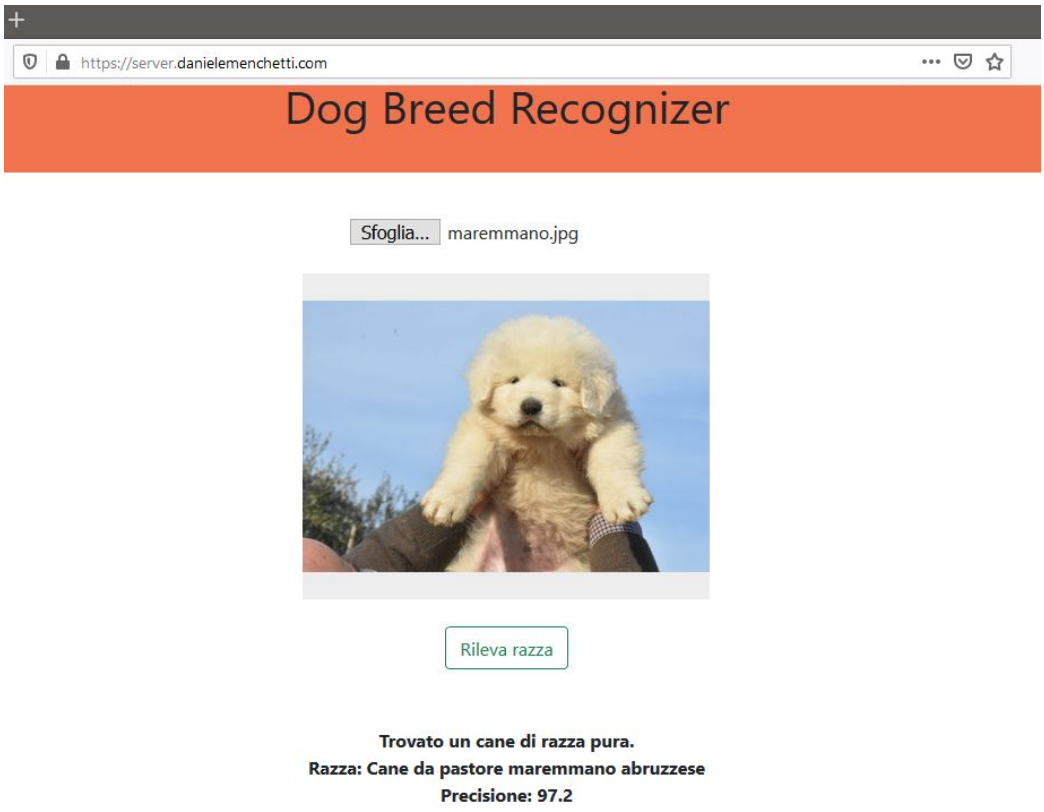

Figura 3.5: Screenshot sistema client-side (2)

Conclusioni

Il progetto è stato realizzato con successo, tuttavia il livello di accuratezza della rete neurale è ancora basso per permettere il suo utilizzo in scenari di applicazioni reali. Infatti, nonostante ci siano stati dei miglioramenti nella capacità di generalizzazione della rete, essa non riesce a produrre un'accuratezza superiore al 50% (top-1) nel modello con 10 classi.

Questi risultati sono però prevedibili in quanto il dominio in esame risulta essere complesso agli occhi di un calcolatore perché spesso le razze canine si differenziano tra di loro mediante pochi tratti distintivi e si accomunano per la maggior parte delle features estratte.

Si rende così necessario l'uso di modelli più complessi oppure l'uso di più reti neurali allo stesso tempo per avere una maggiore quantità di features identificative e produrre quindi valori di accuratezza migliori.