

# Poké-Pi-Dex

Classificazione di Pokémon tramite l'uso di CNN su dispositivi embedded



Sistemi Digitali M

A.A. 2021-2022

**Membri del gruppo:**

Karina Chichifoi

Michele Righi

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Classificazione</b>	<b>5</b>
1.1 Dataset . . . . .	5
1.2 Preprocessing dei Dati . . . . .	5
1.2.1 Data Augmentation . . . . .	5
1.3 Addestramento della Rete . . . . .	6
1.4 Descrizione della Rete . . . . .	7
1.5 Allenamento del Modello . . . . .	9
1.6 Risultati e Prove Effettuate . . . . .	10
1.7 Conversione del Modello . . . . .	11
<b>2 Architettura</b>	<b>13</b>
2.1 Requisiti . . . . .	13
2.2 Sistema Operativo . . . . .	15
2.3 Installazione dei componenti . . . . .	15
2.3.1 Display . . . . .	15
2.3.2 PiCamera . . . . .	16
2.3.3 Speaker Audio . . . . .	16
2.3.4 Bottoni . . . . .	19
2.3.5 Levetta Analogica . . . . .	20
2.4 Case e Prototipo . . . . .	21
2.4.1 Base . . . . .	22
2.4.2 Sezione Intermedia . . . . .	22
2.4.3 Coperchio . . . . .	23
2.4.4 Rifiniture . . . . .	23
2.4.5 Verniciatura . . . . .	24
<b>3 Applicazione</b>	<b>25</b>
3.1 Installazione Package . . . . .	25
3.2 Architettura Logica . . . . .	26
3.2.1 Main . . . . .	26
3.2.2 Grafica e Interfacce Utente . . . . .	26
3.3 Video . . . . .	29
3.4 Dati Pokémon . . . . .	29
3.5 Impostazioni . . . . .	30
3.5.1 Lingua . . . . .	31
3.6 Audio . . . . .	31
3.7 Input . . . . .	32
3.8 Classificatore . . . . .	33
3.9 Distorsione nelle Fotocamere . . . . .	34

3.9.1	Calibrazione PiCamera . . . . .	35
<b>4</b>	<b>Demo</b>	<b>39</b>
4.1	Peluche . . . . .	39
4.2	Action Figure . . . . .	41
4.3	Carte da Gioco . . . . .	43
4.4	Immagini dal Telefono . . . . .	44
<b>5</b>	<b>Conclusioni</b>	<b>45</b>
<b>6</b>	<b>Sviluppi Futuri</b>	<b>46</b>
	Riferimenti bibliografici	47



# Introduzione

## Pokémon

Pokémon è un franchise di videogiochi giapponese creato da Satoshi Tajiri nel 1996 [1]. È iniziato come una serie di videogiochi e poi è cresciuto fino a diventare un cartone animato e un fumetto manga. Pokémon è stato anche adattato in altri media come carte collezionabili e film.

Il nome deriva da un'abbreviazione di *Pocket Monsters* (mostri tascabili).

Sono attualmente<sup>1</sup> presenti 898 specie di Pokémon, che possono essere catturati, allenati e usati nelle battaglie contro altri avversari. Queste creature si dividono in vari *tipi* che hanno diversi punti di forza e di debolezza, nonché mosse uniche da usare durante il combattimento.



Figura 1: Prima generazione di Pokémon

## Sistema di Combattimento

Le *lotte Pokémon* sono l'elemento principale del gameplay dei videogiochi della serie Pokémon. Sono strettamente necessarie in ogni gioco per potere avanzare nella trama della storia, per catturare e allenare i Pokémon e su di esse si basa il competitivo. Infatti,

---

<sup>1</sup>in data 5 dicembre



ogni giocatore viene chiamato *allenatore di Pokémon*, in quanto si costruisce una propria squadra catturando Pokémon nuovi e allenandoli per renderli più forti.



Figura 2: Esempio di lotta Pokémon nel videogioco

## Statistiche nei Videogiochi

I Pokémon hanno delle statistiche base fisse, che variano in base alla specie. Sono costituite da valori specifici e permanenti nel tempo, e sono in totale sei [2]:

- punti salute (HP, Health Points);
- attacco (Atk);
- difesa (Def);
- attacco speciale (Sp.Atk);
- difesa speciale (Sp.Def);
- velocità (Speed).

La forza di un Pokémon si basa anche su molti altri parametri non fissi, che non tratteremo in questo progetto.



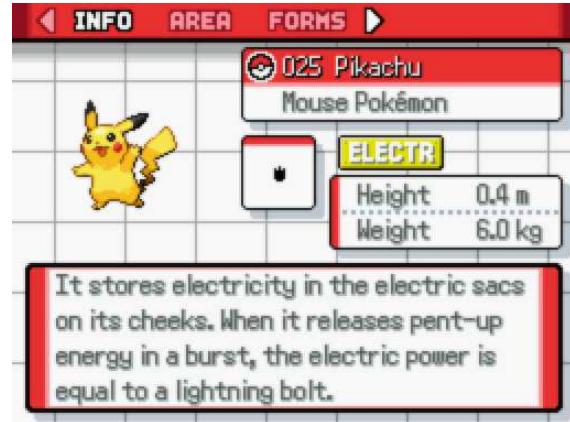
## Pokédex

Il Pokédex è un dispositivo che viene utilizzato dagli allenatori di Pokémon per identificare e tenere traccia dei Pokémon catturati.

- Registra le statistiche del Pokémon catturato, inclusi nome, specie, descrizione, tipo, verso e posizione.
- Scansiona e riconosce il Pokémon catturato nell'erba alta o in battaglia.
- Viene usato per mostrare i punti di forza e di debolezza del proprio Pokémon e degli avversari, così come altre statistiche come altezza e peso.



(a) Pokédex nell'anime



(b) Esempio voce Pokédex nel videogioco

Figura 3: Pokédex

## Scopo del Progetto

L'obiettivo del nostro progetto è quello di riconoscere in modo automatico i Pokémon sotto forma di peluche, carte da gioco e immagini prese dal cartone animato, tramite l'utilizzo di una rete neurale convoluzionale (CNN). A tal proposito, abbiamo realizzato un dispositivo fisico simile a un Pokédex, su cui verrà effettuato il deployment della rete neurale.

La relazione è suddivisa nei seguenti capitoli:

- nel capitolo 1 descriviamo il modello della rete, il preprocessing dei dati, le diverse scelte effettuate in fase di training e post training e i risultati ottenuti;
- nel capitolo 2 presentiamo l'architettura del dispositivo fisico utilizzato, l'installazione dei vari componenti e la costruzione del prototipo del case;
- nel capitolo 3 mostriamo l'architettura logica dell'applicazione, documentando il codice dei componenti software, come abbiamo effettuato il deployment e come abbiamo calibrato la PiCamera per rettificare le immagini;



- nel capitolo 4 mostriamo una demo di utilizzo del progetto e il riconoscimento di diversi oggetti;
- nel capitolo 5 ricapitoliamo i risultati ottenuti, traendo le conclusioni relative;
- nel capitolo 6 accenniamo come potremmo migliorare l'applicazione, ad esempio in vista dell'attività progettuale, proponendo dei possibili sviluppi futuri.

Il codice completo del progetto può essere recuperato su GitHub, al seguente link:  
<https://github.com/TryKatChup/Poke-Pi-Dex>



# 1 Classificazione

## 1.1 Dataset

L'obiettivo principale del progetto consiste nel classificare i primi 151 Pokémon presenti sotto forma di carte, peluche, immagini tratte dalla serie animata, dai videogiochi e disegni. A questo scopo, abbiamo inizialmente utilizzato un dataset misto costituito da 6837 immagini, ricavate dalle fonti sopra riportate<sup>2</sup>. Abbiamo osservato che il dataset, contrariamente a quanto riportato dall'autore, non ha immagini con lo stesso aspect ratio, e sono presenti anche numerosi errori. Abbiamo quindi creato un nuovo dataset da 11943 immagini selezionate e tagliate manualmente<sup>3</sup>. Discuteremo i risultati di entrambi i modelli nella sezione 1.6.

Successivamente, abbiamo suddiviso i dataset in tre parti.

- **Training set (80% del dataset)**: utilizzato dalla rete per imparare i Pokémon.
- **Validation set (10% del dataset)**: stesso insieme di distribuzione del training set, importante per effettuare *fine tuning* degli iperparametri della rete, che avviene durante il training.
- **Test set (10% del dataset)**: serve ad avere degli esempi che siano della stessa distribuzione di training e validation set, in questo modo abbiamo una valutazione più affidabile e coerente con i risultati dell'allenamento.

## 1.2 Preprocessing dei Dati

A causa delle dimensioni ridotte dei dataset:

- abbiamo diviso ciascuna immagine rappresentante un Pokémon in training set, validation set e test set. In questo modo abbiamo garantito la presenza di ogni Pokémon nei tre set citati. Ciascun Pokémon viene identificato da un `LabelEncoder`, successivamente esportato in un file `classes.npy`;
- abbiamo effettuato operazioni di *data augmentation*, descritte in seguito.

### 1.2.1 Data Augmentation

Le tecniche di *data augmentation* consistono nell'aumentare la dimensione del training set, aggiungendo varianti delle foto presenti in quest'ultimo. In questo modo:

- preveniamo l'overfitting della rete;
- aumentiamo la dimensione del dataset, nel caso in cui esso contenga poche immagini;

---

<sup>2</sup><https://www.kaggle.com/lantian773030/pokemonclassification>

<sup>3</sup><https://www.kaggle.com/unexpectedscepticism/11945-pokemon-from-first-gen>



- miglioriamo la performance del modello, aumentando i dati a nostra disposizione [3] [4] [5].

Abbiamo applicato le seguenti operazioni ai dati iniziali:

- **random rotation**: rotazioni casuali delle immagini;
- **random horizontal flip**: specchiamento dell'immagine lungo l'asse y;
- **random brightness**: modifica casuale della luminosità;
- **random contrast**: modifica casuale del contrasto entro un range  $\delta \in [-0.2, 0.199]$ ;
- **random hue**: modifica casuale del colore entro un range  $\delta \in [-0.01, 0.009]$ .

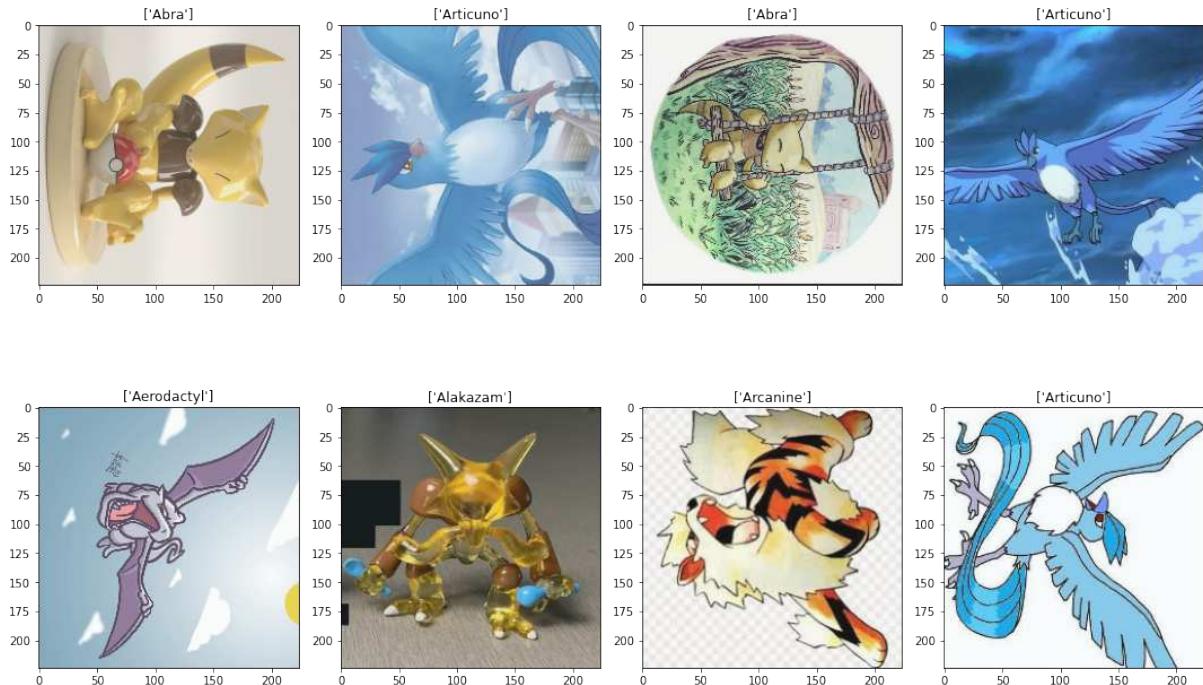


Figura 4: Esempi di data augmentation

Successivamente, abbiamo deciso di togliere la variazione di luminosità, in quanto in seguito alla modifica, a prescindere dal range usato, le immagini presentavano degli artefatti.

### 1.3 Addestramento della Rete

Per lo sviluppo e l'addestramento del modello abbiamo utilizzato TensorFlow 2.6 e Keras.



## 1.4 Descrizione della Rete

I motivi per cui abbiamo scelto di implementare una rete convoluzionale sono i seguenti:

- capacità di analizzare un'immagine e impararne una rappresentazione semanticamente ricca [6];
- elevata flessibilità: in questo caso, a causa del dataset poco ampio, risulta molto utile avere un'architettura molto semplice e configurabile.

I dettagli relativi all'architettura della nostra rete sono illustrati nella Figura 5.

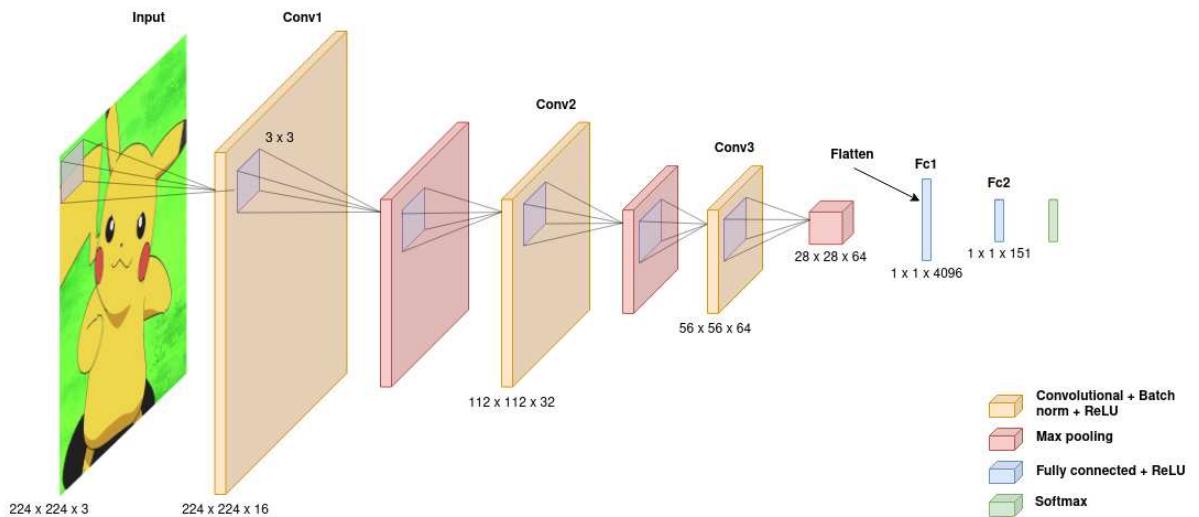


Figura 5: Architettura della nostra CNN

1. **Input layer:** processa i valori dei pixel dell'immagine; in questo caso l'immagine è di dimensioni  $224 \times 224$  e tre canali RGB.
2. **Tre layer convoluzionali:** caratterizzati da stride<sup>4</sup> unitario, filtri convoluzionali di dimensione crescente (16, 32, 64) e kernel size<sup>5</sup> pari a 3.
  - **Livello convoluzionale:** elabora l'output dei neuroni collegati alle singole finestre di input, effettuando un prodotto scalare tra i pesi della rete e una piccola porzione collegata all'input. Si avrà quindi un output pari a  $224 \times 224 \times 16$  in uscita dal primo layer convoluzionale (dato che sono stati applicati 16 filtri).

<sup>4</sup>Di quanti pixel il filtro convoluzionale si sposta sull'input

<sup>5</sup>Dimensione del filtro convoluzionale



Lo scopo dei filtri convoluzionali è di effettuare *feature extraction* di un'immagine: i pesi della rete neurale vengono ricalibrati in modo da estrarre le informazioni principali che caratterizzano un'immagine.

- **Batch Normalization:** usato per normalizzare l'output di un layer durante l'allenamento, in modo che abbia media nulla e varianza unitaria. Si è visto che ciò porta a un allenamento più veloce e consente l'utilizzo di diversi learning rate, senza compromettere la convergenza del training [7]. Tuttavia, l'efficacia del batch normalization è ancora ampiamente discussa in ambito accademico [8].
- **ReLU:** funzione di attivazione, che restituisce 0 se l'input risulta negativo, altrimenti restituisce il valore in ingresso.

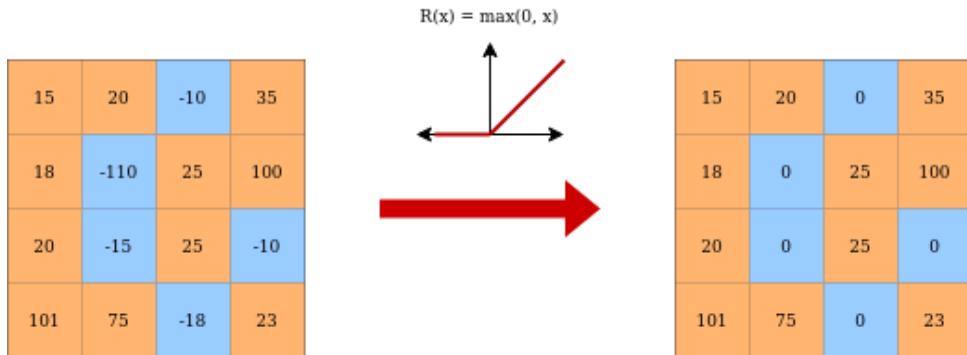


Figura 6: Funzionamento di ReLU

- **Max Pooling:** livello che riduce le dimensioni del dato in uscita dei livelli precedenti, in modo da diminuire il carico computazionale. Risulta utile per estrarre le caratteristiche principali di un'immagine e si ottiene dal valore massimo dalla porzione dell'immagine coperta dal kernel.

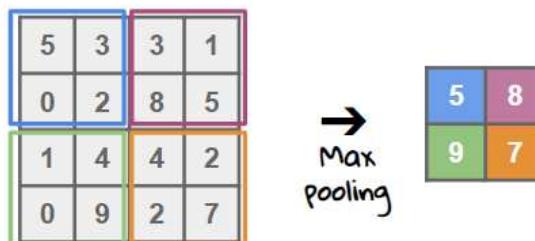


Figura 7: Max pooling

3. **Flatten:** il numero di dimensioni dell'input viene ridotto a uno. In questo caso in uscita avremo un tensore monodimensionale con 50176 elementi, risultato dal prodotto delle dimensioni dell'output dell'ultimo *pooling layer* (pari a  $28 \times 28 \times 64$ ).



4. **Classifier con 2 livelli fully connected (FC)**: tutti gli input di un layer sono collegati a ciascuna unità di attivazione del layer successivo. Dopo avere effettuato *feature extraction*, grazie ai livelli convoluzionali, dobbiamo classificare i dati in diverse classi e a questo scopo vengono utilizzati i livelli FC. I livelli fully connected utilizzano un numero inferiore di FLOPS<sup>6</sup> rispetto ai livelli convoluzionali, tuttavia richiedono più parametri da memorizzare.
- (a) **Dense con 2048 neuroni**: la dimensione dell'output del livello denso è condizionata dal numero di neuroni specificati (in questo caso 2048).
  - (b) **BatchNorm**: vedi prima.
  - (c) **ReLU layer**: vedi prima.
  - (d) **Dense con 151 neuroni (pari al numero di classi)**: i dati in uscita devono avere dimensione pari a 151, che corrisponde al numero di classi.
  - (e) **Softmax**: viene utilizzato come ultima funzione di attivazione e converte gli score di ciascuna classe (non normalizzati, arbitrariamente maggiori di uno o negativi) in una probabilità (numero reale compreso tra 0 e 1). La somma dei valori prodotti dalla funzione softmax è pari a 1.

## 1.5 Allenamento del Modello

Per compilare il modello abbiamo scelto come ottimizzatore *Adam* (ADaptive Moment estimation), che viene utilizzato come alternativa al SGD (Stochastic Gradient Descent) per aggiornare i pesi della rete durante l'allenamento. Ha numerose applicazioni in computer vision e in natural language processing. Adam è uno degli ottimizzatori più robusti: infatti, una rete allenata con Adam tende a convergere anche a fronte di piccole variazioni negli iperparametri. Di conseguenza, non richiede un tuning preciso degli iperparametri, che sarebbe altrimenti oneroso sia in termini computazionali che temporali [9].

Per calcolare il valore di *loss* abbiamo usato *SparseCategoricalCrossEntropy* di Keras, che consente di utilizzare un valore intero per il label che identifica la vera classe di appartenenza. Ha il vantaggio di essere meno costosa in termini di computazione e di memoria, poiché viene utilizzato un intero per classe, anziché un vettore.

```

1 model.compile(
2     optimizer='adam',
3     loss=tf.keras.losses.SparseCategoricalCrossentropy(),
4     metrics=['accuracy']
5 )

```

Successivamente, abbiamo utilizzato l'*early stopping* per il training del modello, in quanto risulta molto utile per fermare l'allenamento in caso di peggioramento delle performance del modello [10].

<sup>6</sup>Floating-point Operations Per Second.



- `monitor='val_loss'` consente di monitorare il *validation loss* a ogni epoca, per osservare eventuali peggioramenti.
- `patience = 10` indica quante epoche consecutive di peggioramento vengono tollerate.
- `restore_best_weights=True` è necessario per memorizzare i pesi della rete prima delle 10 epoche di peggioramento. Questa strategia previene il salvataggio del modello durante la fase di overfitting e allo stesso tempo non richiede di modificare manualmente il numero di epoche per ottenere le migliori performance.

```
1 callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10,
→ restore_best_weights=True)
```

Infine, il modello inizia il training (`model.fit()`), con `epochs=100` e `batch_size=64`. L’allenamento è stato eseguito su GPU NVIDIA GTX 1060 6 GB, con durata pari a circa 17 minuti.

```
1 history = model.fit(
2     train_dataset,
3     epochs=100,
4     callbacks=[callback],
5     steps_per_epoch=train_len // 64,
6     validation_data=val_dataset,
7     validation_steps=val_len // 64
8 )
```

## 1.6 Risultati e Prove Effettuate

Abbiamo effettuato i seguenti tentativi.

- Allenamento con il vecchio dataset e dropout: il dropout è una tecnica di regolarizzazione che “spegne” casualmente alcuni neuroni di un certo layer. Abbiamo notato un peggioramento pari a 8%, sia su training accuracy, che validation accuracy, rispetto al modello senza dropout. Il motivo per cui il dropout non risulta necessario è grazie all’utilizzo del batch normalization, che si comporta da regolarizzatore e aumenta la velocità del training [7]. Per questo motivo abbiamo rimosso il dropout.
- Allenamento con il nuovo dataset e 55 foto per classe: il vecchio dataset presentava diversi errori, pertanto abbiamo ritagliato 11943 nuove foto a mano, mantenendo lo stesso aspect ratio. Di queste, abbiamo considerato 8.238 foto (circa 55 foto per classe) e abbiamo riallenato la rete. Abbiamo notato un netto miglioramento con dei sample fatti dal vivo.



Nella tabella sottostante abbiamo riportato i dati relativi alle performance dei due modelli.

	Train loss	Train acc	Val loss	Val acc	Test loss	Test acc	Epoche
Vecchio	0.08	0.9810	0.0186	0.9980	0.0176	0.9983	75
Nuovo	0.0465	0.9901	0.0266	0.9929	0.0262	0.9933	72

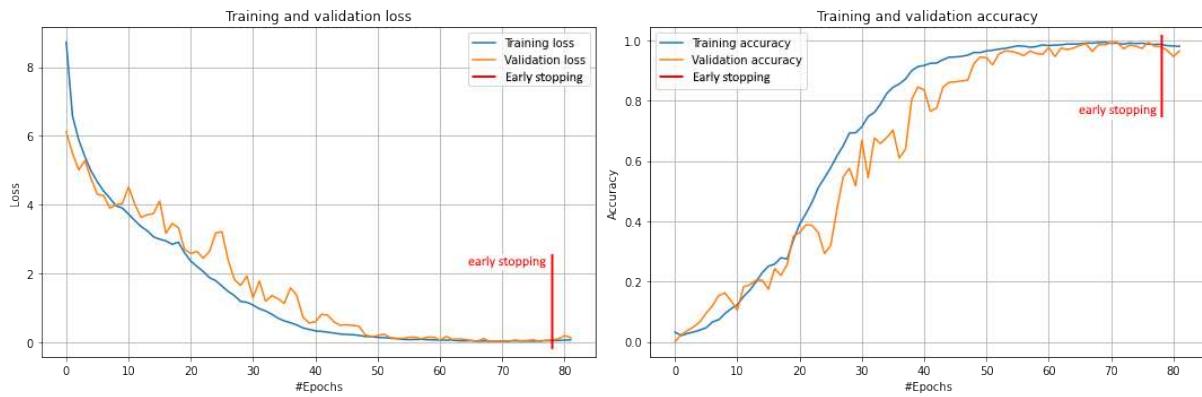


Figura 8: Performance del modello con 55 foto per classe

## 1.7 Conversione del Modello

Il modello allenato ha un peso complessivo pari a 1.2 GB, e ciò comporta diversi problemi.

- L'utilizzo del modello è limitato su sistemi con vincoli di memoria o di spazio, come ad esempio il Raspberry Pi 4 che andremo a utilizzare.
- È difficile utilizzare il modello in contesti distribuiti, poiché non è facilmente trasportabile.

Per utilizzare il modello su Raspberry Pi4 abbiamo deciso di convertirlo in formato *Tensorflow Lite*.

```

1 converter = tf.lite.TFLiteConverter.from_keras_model(model)
2 converter.target_spec.supported_ops = [
3     tf.lite.OpsSet.TFLITE_BUILTINS,
4     tf.lite.OpsSet.SELECT_TF_OPS
5 ]
6 converter.optimizations = [tf.lite.Optimize.DEFAULT]
7 tflite_quant_model = converter.convert()
8 open("modello.tflite", "wb").write(tflite_quant_model)

```



Un modello di Tensorflow Lite è rappresentato in un formato efficiente e facilmente trasportabile noto come *FlatBuffers* (identificato dall'estensione `.tflite`). Ciò costituisce diversi vantaggi rispetto al modello in Tensorflow, come ad esempio le dimensioni ridotte e inferenza più veloce, dato che l'accesso ai dati avviene senza ulteriori operazioni di *parsing* o di *deserializzazione* [11].

Abbiamo compresso il modello tramite tecniche di quantizzazione, ovvero riducendo la precisione dei numeri utilizzati per rappresentare i parametri di un modello (di default sono espressi in floating point a 32 bit) [12]. Ciò porta ad avere un modello con:

- velocità maggiore;
- dimensioni ridotte;
- minori consumi;
- un peggioramento di accuracy pari a circa 1% rispetto al modello originale (che risulta un buon compromesso).

Nel nostro caso, abbiamo convertito il modello da FP32 a INT8. Il modello compresso ottenuto dalle operazioni di quantizzazione ha una dimensione di 93 MB.



## 2 Architettura

Un altro aspetto importante del nostro progetto è la realizzazione di un Pokédex fisico. Per questo motivo abbiamo deciso di eseguire il deployment dell'applicazione su un dispositivo embedded. Così facendo, il prodotto finale ottenuto sarebbe stato il più possibile simile a una console, analogamente a quello presente nella serie animata e nei vari giochi.

Essendo già a disposizione di un Raspberry Pi4 Model B con [starter kit](#), e poiché si prestava bene al nostro scopo, sia per le dimensioni ridotte che per l'hardware piuttosto efficiente, abbiamo deciso di utilizzarlo come base per la nostra architettura, su cui eseguire il software dell'applicazione.

### 2.1 Requisiti

Per la realizzazione della console erano necessari diversi componenti.

- **Calcolatore:** [Raspberry Pi4 Model B](#).
- **Memoria secondaria:** abbiamo utilizzato la scheda MicroSD classe 10 da 32GB inclusa nello starter kit.
- **Alimentazione:** lo starter kit comprende un alimentatore ufficiale con porta USB type-C. Tuttavia, volendo realizzare una console portatile, avevamo bisogno di una batteria. Per semplicità abbiamo optato per un [powerbank ultrasottile](#).
- **Output video:** abbiamo scelto un [display LCD 3.5" HDMI con resistive touch screen](#). Questo schermo ha stesse dimensioni del Raspberry ed è collegabile direttamente ai pin, senza parti troppo sporgenti (vi è solo un piccolo adattatore HDMI che occupa meno di un centimetro di spazio su un lato).
- **Input video:** abbiamo scelto una [Pi Camera Rev 1.3 \(5MP, 1080p\)](#).
- **Output audio:** un [mini speaker](#) da inserire nella console.
- **Input alternativo:** il touchscreen in una console potrebbe essere scomodo da utilizzare, soprattutto quando bisogna impugnarla saldamente per scattare una foto. Pertanto, abbiamo deciso di aggiungere dei controlli alternativi: una levetta analogica per muovere il cursore e due pulsanti per emulare il click o eventualmente qualche altra funzionalità.  
Eravamo già a disposizione del joystick analogico ma, poiché il Raspberry dispone di soli GPIO digitali, per poterlo utilizzare abbiamo deciso di acquistare un [convertitore A/D ADS1115](#), mentre per i pulsanti abbiamo optato per dei semplici [push button](#).
- **Case:** avevamo bisogno della scocca in cui inserire i vari componenti, e abbiamo pensato di crearla con una stampante 3D. Tuttavia, per le nostre esigenze serviva un riferimento fisico, soprattutto per prendere le misure degli spazi con accuratezza.



Di conseguenza, per il momento abbiamo optato per un prototipo in cartoncino riciclato.



Figura 9: Componenti a disposizione



(a) Insieme dei componenti



(b) Bozza Progetto del Case

Figura 10: Struttura dell'architettura fisica



## 2.2 Sistema Operativo

Come sistema operativo abbiamo scelto Raspberry OS 32-bit: è la distribuzione Linux ufficiale e comprende firmware e driver ufficiali Raspberry, i quali assicurano stabilità e supporto nella comunicazione con periferiche esterne. Inoltre, tale distribuzione supporta l'interprete di Tensorflow Lite 2.4, necessario per poter utilizzare il classificatore.

## 2.3 Installazione dei componenti

Il Raspberry Pi4 Model B possiede una testata con 40 pin GPIO<sup>7</sup> (vedi Figura 11), che consentono di collegare componenti hardware direttamente al microprocessore attraverso MMIO, rendendo semplice la loro integrazione.

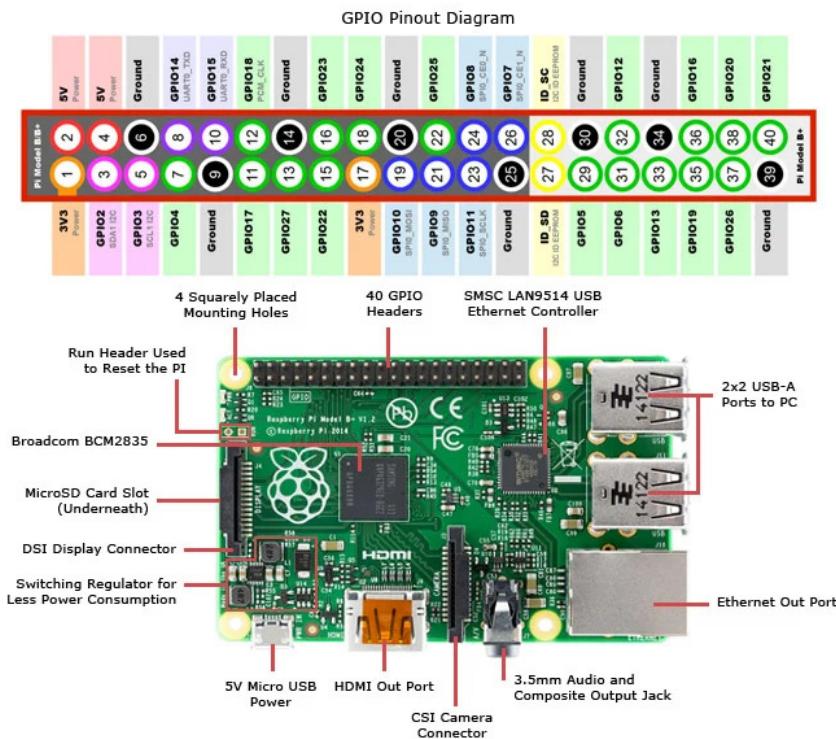


Figura 11: Mappa dei Pin del Raspberry Pi4

### 2.3.1 Display

Il display che abbiamo acquistato necessita dei primi 26 pin e un input HDMI per il funzionamento. Ciò significa che tali pin non sono utilizzabili da altri componenti e, come vedremo, sarà un problema per l'utilizzo della levetta analogica.

<sup>7</sup>GPIO: General-Purpose Input/Output



L'installazione è stata molto semplice: seguendo la guida presente nel [repository GitHub di Waveshare](#), abbiamo un sistema pronto all'uso, con risoluzione  $480 \times 320$  e il touchscreen funzionante.

### 2.3.2 PiCamera

La Picamera sfrutta la porta MIPI CSI. Una volta collegata,abbiamo utilizzato il comando **raspi-config** per abilitarla (Figura 12).

Con **raspistill** abbiamo scattato una foto, e con **raspivid** abbiamo registrato un video di test.

```
1 raspistill -o image.jpg # take a picture
2 raspivid -o myvideo.h264 --timeout 10000 # record a video
```

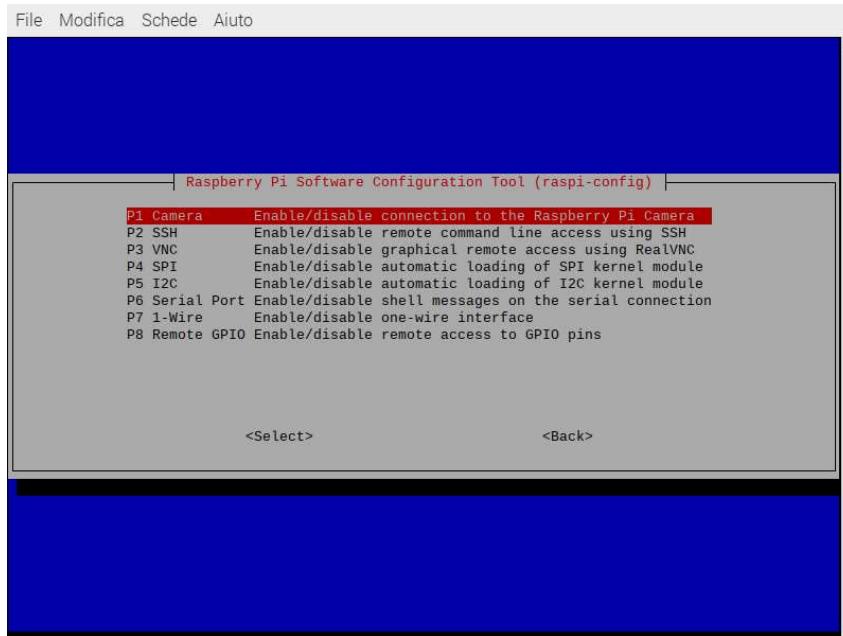


Figura 12: Abilitazione Picamera

### 2.3.3 Speaker Audio

Per semplicità e per risparmiare pin, abbiamo deciso di collegare lo speaker alla porta audio 3.5mm del Raspberry. A tal scopo abbiamo utilizzato dei vecchi auricolari rotti:

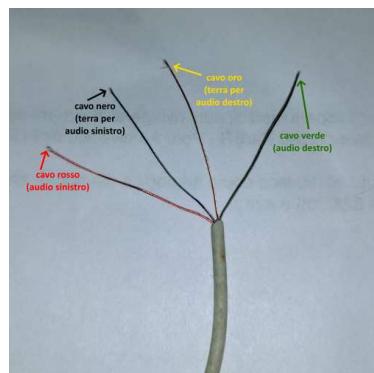
1. abbiamo tagliato il cavo vicino all'estremità del jack (Figura 13a);
2. abbiamo scoprendo i cavetti interni, separandoli gli uni dagli altri (Figura 13b);
3. abbiamo rimosso da ciascuno l'isolante che avevano all'interno (Figura 13c);



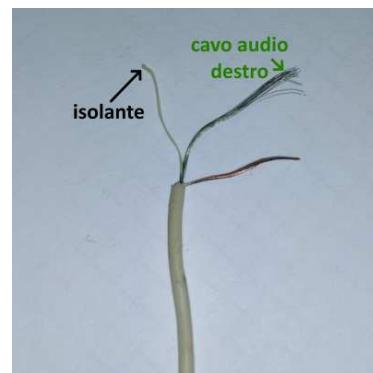
4. abbiamo unito il cavo del canale sinistro a quello del destro (Figura 14a), e i due cavi della messa a terra fra di loro, in quanto il nostro progetto utilizzerà un solo speaker, dunque un solo canale audio;
5. abbiamo collegato i due canali audio (rosso e verde) al cavo positivo dello speaker (rosso), e quelli della messa a terra fra di loro (Figura 14b);
6. abbiamo effettuato dei test per controllare che non ci fossero cortocircuiti fra i cavi;
7. verificato il funzionamento, utilizzando una saldatrice a stagno abbiamo quindi saldato i cavi (Figura 14c);
8. infine, li abbiamo coperti con del nastro isolante (Figura 15).



(a) 1. Auricolari tagliati



(b) 2. Cavi scoperti



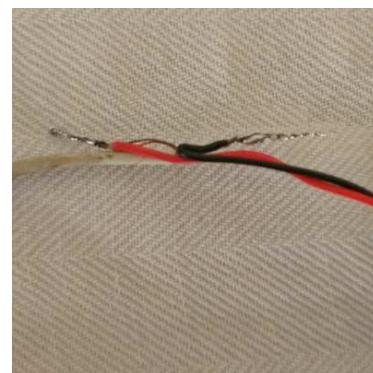
(c) 3. Separazione isolante



(a) 4. Cavi uniti



(b) 5. Collegamento cavi



(c) 6. Saldatura cavi

Per verificarne il funzionamento sul Raspberry, abbiamo riprodotto i file audio utilizzando **pygame**, in quanto è una libreria Python che gestisce audio e video, sviluppata appositamente per i videogiochi. Essendo basata su **SDL**, è multipiattaforma per numerose architetture; ciò ha permesso di effettuare diversi test sia su Windows che su Raspberry OS, senza alcuna modifica del codice.

Con un semplice script di test, ne abbiamo verificato il funzionamento:



Figura 15: Risultato finale speaker

---

```
1 import pygame
2 from signal import pause
3
4 # Pygame init
5 pygame.mixer.pre_init(44100, -16, 1, 4096)
6 pygame.init()
7 pygame.mixer.init()
8
9 # Get channel
10 channel = pygame.mixer.find_channel(True)
11
12 # Play audio file from Scratch
13 channel.play(pygame.mixer.Sound("Laugh-male1.wav"))
14
15 pause()
```

---

L'audio si sente, tuttavia il volume rimane piuttosto basso, in quanto lo speaker è sprovvisto di un amplificatore. Pensavamo quindi di integrarlo nel prototipo come attività progettuale (vedi Capitolo 6).

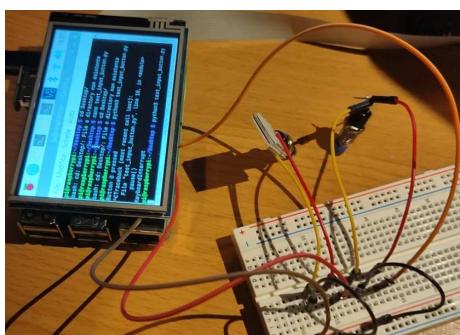


### 2.3.4 Bottoni

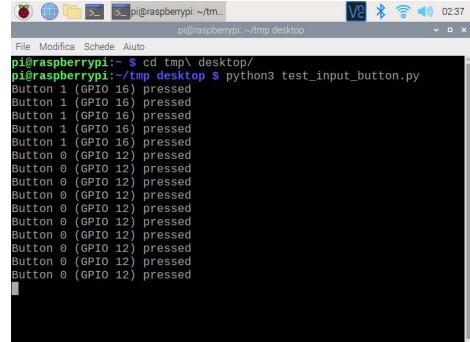
Per i bottoni sono necessari almeno 3 pin: uno di ingresso per ciascun push button, e almeno uno per la messa a terra, se questa viene condivisa. Per semplicità, abbiamo deciso di separare i collegamenti a terra dei due pulsanti, occupando di conseguenza 4 pin in totale: 30, 32, 34, 36.

Per effettuare i test dei push button a livello software abbiamo creato un circuito utilizzando dei jumper e una breadboard (Figure 16a). Tramite la libreria `gpiozero` e uno script di prova (vedi Figura 16b) ricavato facendo riferimento alla [documentazione](#), siamo riusciti a verificarne il funzionamento. Dopodiché, abbiamo accorciato i cavi per risparmiare spazio nel case che avremmo creato, e li abbiamo fissati con del nastro isolante.

```
1 from gpiozero import Button
2 from signal import pause
3
4 button0 = Button(12)
5 button1 = Button(16)
6
7 def test0():
8     print("Button 0 (GPIO 12) pressed")
9
10 def test1():
11     print("Button 1 (GPIO 16) pressed")
12
13 button0.when_pressed = test0
14 button1.when_pressed = test1
15
16 pause()
```



(a) Collegamento bottoni



(b) Test bottoni



### 2.3.5 Levetta Analogica

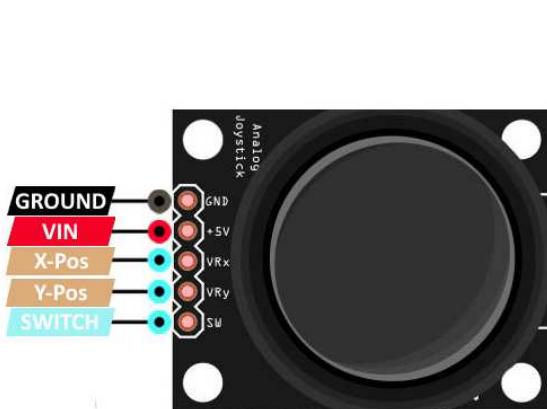
Il modulo joystick ha 5 pin:

- messa a terra;
- ingresso alimentazione (5V);
- tensione di uscita per l'asse X;
- tensione di uscita per l'asse Y;
- switch di uscita per il click del bottone (vedi Figura 17a).

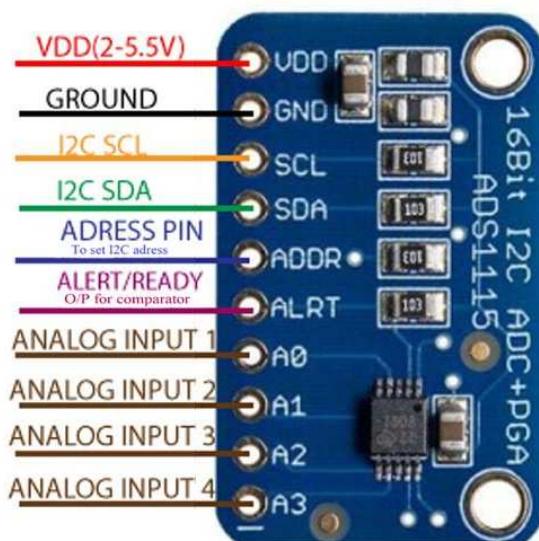
Questo componente sfrutta segnali analogici per comunicare, in quanto le uscite degli asse X e Y possono assumere valori da 0 a 1023. Tuttavia, poiché il Raspberry non dispone di pin analogici, ma solo digitali, per utilizzarlo è necessario un convertitore A/D come ADS1115.

Il convertitore ADS1115 possiede 10 pin e, sfruttando l'interfaccia I2C del Raspberry, cattura fino a 4 segnali analogici (vedi Figura 17b) per indicare correttamente il proprio stato. Tuttavia, nella nostra configurazione i pin dell'interfaccia I2C (GPIO2 per SDA e GPIO3 per SCL), necessari al convertitore, sono già occupati dal display; per poterlo utilizzare sarebbe necessario emulare tale interfaccia via software.

Vista la complessità di questo problema, abbiamo pensato di lasciarlo come sviluppo futuro.



(a) Modulo joystick



(b) Modulo ADS1115

Figura 17: Componenti joystick



## 2.4 Case e Prototipo

Per la realizzazione del prototipo del case ci siamo ispirati ad un [modello](#) trovato su Amazon (Figura 18).



Figura 18: Modello Pokédex

Come prima cosa abbiamo raccolto dettagli su come i componenti sarebbero dovuti essere posizionati all'interno della scocca. Successivamente, abbiamo misurato le dimensioni dei componenti e realizzato un progetto (Figura 19). Come materiale abbiamo pensato di utilizzare dei fogli di cartoncino riciclato, presi da diversi quaderni Quablock finiti.

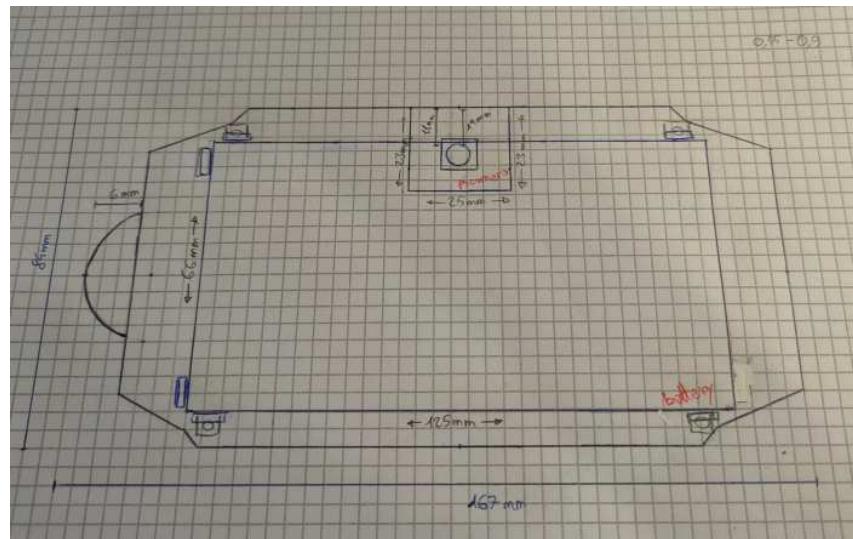


Figura 19: Progetto prototipo del case

Il prototipo sarebbe stato diviso in 3 parti separate e chiudibili a incastro:



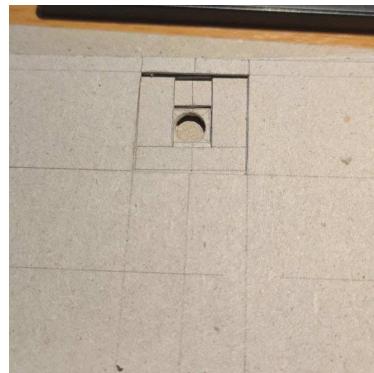
- una **base** solida e resistente, in cui avremmo ricavato un alloggio per inserire la Picamera e un foro per l'obiettivo;
- una **sezione intermedia**, che avrebbe avuto uno slot in cui inserire la batteria e appoggiarci sopra il Raspberry;
- un **coperchio**, in cui avremmo ricavato i buchi per il display del Raspberry, per i bottoni, per la levetta analogica e dei fori per lo speaker. Questa parte è la più complessa, in quanto deve comprendere dei sostegni per chiudere in modo il più ermetico possibile il resto della scocca, e un alloggio per l'inserimento dello speaker.

#### 2.4.1 Base

Abbiamo iniziato prendendo misure sui cartoncini e ritagliando le sagome per la base: in totale sono nove sezioni, in ciascuna delle quali abbiamo inciso una forma diversa (Figura 20a) in modo tale da incastrare perfettamente la Picamera (Figura 20b), e lasciando spazio per il cavo. Effettuate le dovute prove, abbiamo incollato fra di loro i pezzi, ottenendo un'unica base solida (Figura 20c).



(a) Forme Picamera



(b) Alloggio Picamera



(c) Base completata

#### 2.4.2 Sezione Intermedia

Per la sezione intermedia, essendo più alta e di conseguenza difficile da modellare, abbiamo cambiato metodo: abbiamo realizzato una base su cui appoggiare i cartoncini che avremmo utilizzato come pareti. Abbiamo usato la colla vinilica per fissarli e, per rendere questa sezione più resistente, abbiamo creato le pareti lungo i quattro lati utilizzando due strati di cartoncino sovrapposti (Figura 21a).

Per realizzare l'alloggio della batteria, in quanto questa sarebbe dovuta rimanere appoggiata appena sopra la Picamera, abbiamo ritagliato una sagoma nella base, e aggiunto dei rinforzi laterali per fare in modo che non si muovesse (Figura 21c). La parte più complicata di questa sezione è stata la realizzazione della parte circolare, per la quale abbiamo dovuto incidere diversi tagli paralleli fra di loro, per piegare il cartoncino senza romperlo e ottenere un effetto curvilineo (Figura 21b).



(a) Pareti lati finite



(b) Parete curvilinea



(c) Sezione intermedia

#### 2.4.3 Coperchio

Il coperchio è stata la parte più complessa da ricavare, ed è costituito da tre strati sovrapposti: due di struttura ed uno per i dettagli e le rifiniture.

Per i livelli di struttura abbiamo ritagliato due sagome quasi identiche, in cui abbiamo inciso in entrambe il buco per la levetta analogica, per il display, per i bottoni e per gli speaker. L'unica differenza è che per lo strato superiore abbiamo utilizzato delle dimensioni per il display inferiori, in modo tale che i bordi dello schermo venissero nascosti. Abbiamo realizzato lo strato dei dettagli, che, oltre a servire come rinforzo per la struttura, avevano uno scopo decorativo, al fine di accentuare le forme sulla parte superiore (Figura 22a).

Dopodiché, abbiamo realizzato un cassetto in cui poter posizionare e fissare lo speaker e delle sporgenze per incastrare il coperchio nella sezione intermedia (Figura 22b).

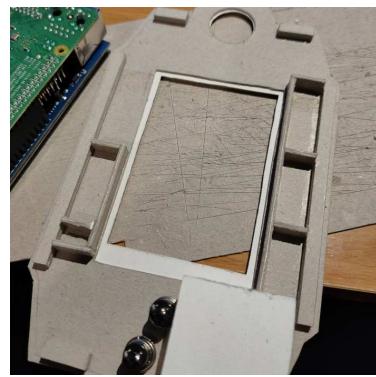
Il coperchio è stato infine rinforzato con dei sostegni (Figura 22c).



(a) Strato decorativo



(b) Slot speaker



(c) Coperchio rinforzato

#### 2.4.4 Rifiniture

Abbiamo aggiunto: un foro per il cavo dell'alimentatore, in modo tale che il Pokédex potesse essere usato anche senza batteria; un cerchio composto da due strati sotto la



base, in modo che la PiCamera non venisse a contatto con le superfici su cui il Pokédex veniva appoggiato.

#### 2.4.5 Verniciatura

Per la colorazione abbiamo utilizzato i colori acrilici, seguendo la stessa paletta dei Pokédex della serie animata e dei giochi: arancione come colore predominante, nero e bianco per i dettagli (Figura 23).



Figura 23: Verniciatura finita



Figura 24: Risultato finale prototipo del case



## 3 Applicazione

L'applicativo fornisce un'interfaccia grafica semplice e intuitiva, che consente all'utente di inquadrare il Pokémon con la fotocamera, scattare una foto e visualizzare le informazioni rilevanti riguardo al Pokémon riconosciuto (Figura 25). Viene garantita inoltre trasparenza sul suo funzionamento, tra cui i dettagli realizzativi del classificatore e i processi che permettono il funzionamento del programma.

Come linguaggio di programmazione abbiamo scelto Python, in quanto dispone di librerie ideali per data science e computer vision.

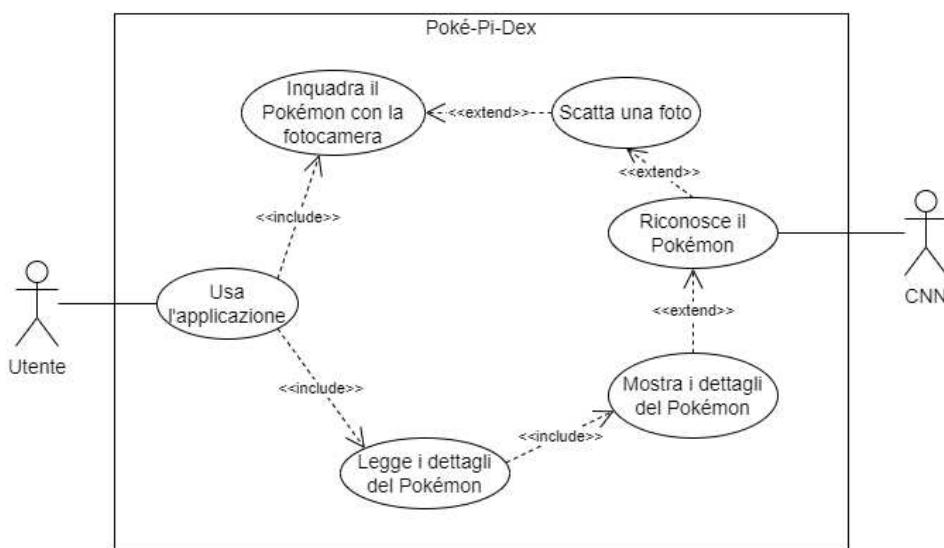


Figura 25: Caso d'uso

### 3.1 Installazione Package

Per il funzionamento dell'applicazione sono stati installati i seguenti package:

```
1 sudo apt install python3-tkinter # tkinter
2 sudo apt install python3-pil # pillow
3 sudo apt install python3-pil.imagetk # pillow-imagetk
4 pip3 install pygame # pygame
5 pip3 install -U scikit-learn # sklearn
6 pip3 install https://github.com/iCorv/tflite-runtime/raw/master/ \
→ tflite_runtime-2.4.0-cp37-cp37m-linux_armv7l.whl # tf-lite
7 sudo apt install python3-gpiozero # gpiozero
```



## 3.2 Architettura Logica

L'applicazione si basa sul pattern MVC, ed è costituita da diversi componenti, come mostrato in Figura 26. Il controller si occupa di costruire le varie interfacce e permettere all'utente di interagire col dominio tramite i componenti di input presenti nelle viste.

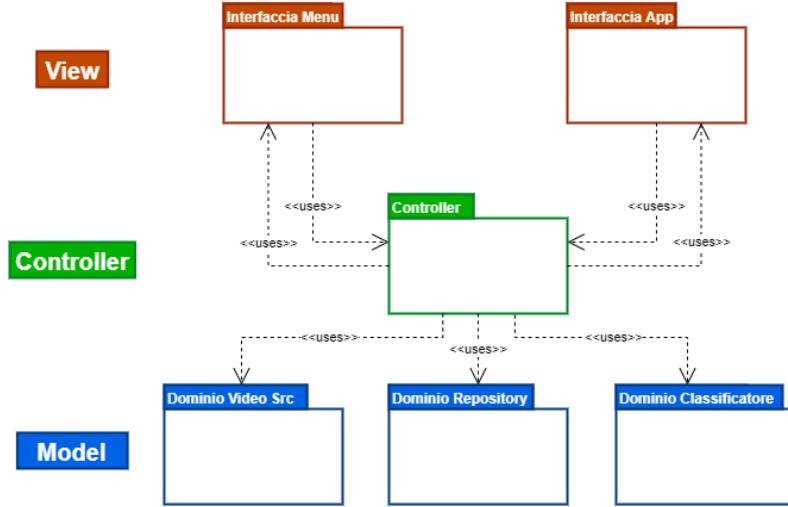


Figura 26: Diagramma dei package

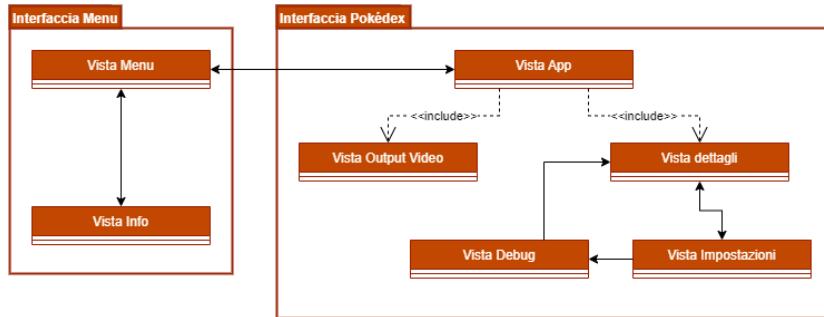


Figura 27: Diagramma di navigazione delle interfacce

### 3.2.1 Main

Il componente principale `poke-pi-dex.py` presenta il controller della nostra applicazione basata su MVC. Al suo interno viene costruita l'interfaccia grafica, con cui l'utente può interagire e spostarsi tra le varie viste, e richiama le funzioni fornite dagli altri componenti.

### 3.2.2 Grafica e Interfacce Utente

Nonostante esistano numerose librerie alternative per lo sviluppo di applicazioni grafiche, abbiamo scelto di utilizzare `Tkinter`, il package standard per l'interfaccia grafica in

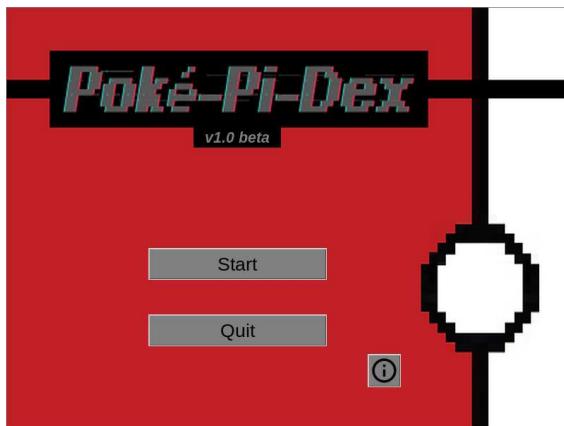


Python, data la sua semplicità e leggerezza. Tramite la classe `Tk`, il nostro applicativo crea una finestra all'avvio, che viene impostata di default a schermo intero. Dopodiché vengono costruiti tutti i componenti della GUI, fra cui le diverse viste che popola con i vari widget, e mostra il menu principale. Terminata questa fase di inizializzazione, tramite il metodo `mainloop()`, viene avviato il ciclo degli eventi che permette all'utente di interagire con l'interfaccia grafica.

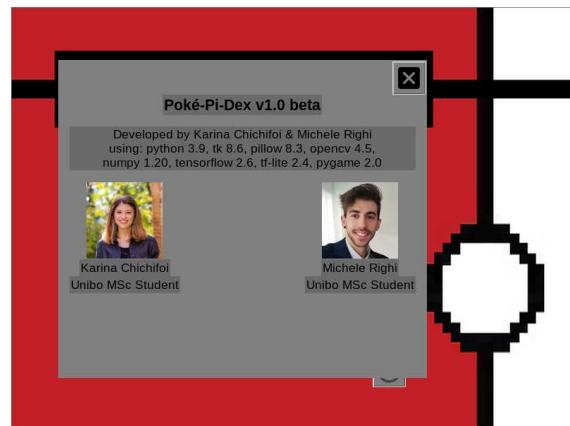
```
1 def start(self):
2     # Show the App Menu
3     self.show_menu()
4     self.video = vc.VideoCapture()
5     # After it is called once, the update method will be automatically called every
6     # delay milliseconds
7     self.delay = 1
8     self.update()
9     self.window.mainloop()
```

**Menu** All'avvio viene mostrato all'utente la vista del menu, in cui sono presenti tre bottoni:

- **Start**: consente di utilizzare la funzione di riconoscimento dei Pokémons.
- **Quit**: permette di uscire dall'app.
- **Info**: ottenere una schermata con ulteriori dettagli come versione del software, librerie utilizzate, crediti.



(a) Vista menu



(b) Vista info

Figura 28: Schermata principale



**Pokédex** Questa schermata consente di effettuare il riconoscimento del Pokémon e la visualizzazione dei relativi dettagli.

- A sinistra vi è una vista sempre presente che mostra l'output video della fotocamera e il bottone **Search**. Tale pulsante permette di effettuare la predizione del Pokémon, come mostrato in Figura 30a.
- La vista di destra, invece, è dinamica e mostra inizialmente i dettagli del Pokémon (Figura 30b). Nel caso in cui l'utente selezioni il menù delle impostazioni o di debug, soltanto nella vista di destra compariranno le relative opzioni (Figura 30c e 30d).

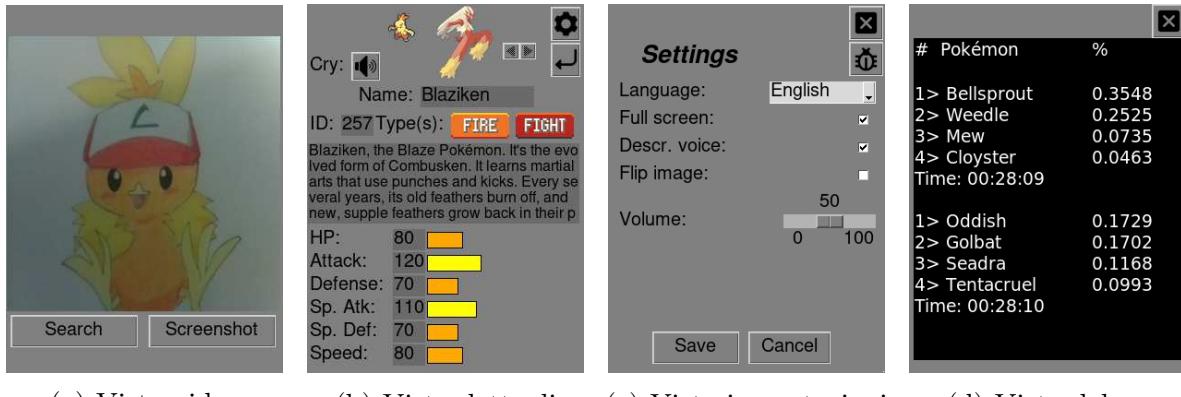
Quando l'utente seleziona il pulsante **Search**, il frame corrente viene passato al classificatore, che consente di ottenere predizioni. Queste ultime sono costituite dal nome del Pokémon e la percentuale di riconoscimento.



Figura 29: Interfaccia del Pokédex completa

**Debug Mode** Per rendere più semplice il testing dell'applicazione e l'analisi dei risultati ottenuti, abbiamo aggiunto all'interfaccia del Pokédex una vista di debug. Tale vista si presenta come un terminale di output, in cui ad ogni predizione vengono stampati in append la lista dei Pokémon riconosciuti, in numero compreso fra 2 e 10 inclusi, seguito dal timestamp (vedi Figura 30d).

Per attivarla possiamo eseguire l'applicazione specificando il parametro **-d**, facoltativamente seguito dal numero delle predizioni che vogliamo mostrare (se non specificato, il valore di default è 5). Un altro modo per entrare in debug mode è tramite la vista delle impostazioni, dove possiamo cliccare sul pulsante con l'icona dell'insetto un numero di



(a) Vista video      (b) Vista dettagli      (c) Vista impostazioni      (d) Vista debug

Figura 30: Tutte le viste della schermata del Pokédex

volte pari alle predizioni che vogliamo mostrare; in questo caso, premendo sul pulsante di salvataggio, verrà mostrata la vista di debug.

### 3.3 Video

Per ottenere l'input video dalla PiCamera abbiamo utilizzato [OpenCV](#).

Il file `video_capture.py` espone una classe omonima (Figura 31), che permette di scattare foto e restituirle in modo continuo, sfruttando il metodo `read()` chiamato sull'oggetto della sorgente video. Lo stream di frame ottenuti viene preso dal controller `poke-pi-dex.py`, mostrato a video, e aggiornato a ogni iterazione del ciclo eventi di Tkinter.

```

1 def update(self):
2     if self.update_video:
3         ret, frame = self.video.get_frame()
4         if ret:
5             self.photo =
6                 ImageTk.PhotoImage(image=Image.fromarray(frame).resize(image_size,
7                                         Image.ANTIALIAS))
7             self.canvas_video.create_image(res_width/4, res_width/4,
8                 image=self.photo, anchor=tk.CENTER)
8             self.window.after(self.delay, self.update)

```

### 3.4 Dati Pokémon

La vista dei dettagli mostra tutte le informazioni utili riguardo al Pokémon riconosciuto. In particolare:

- il nome;

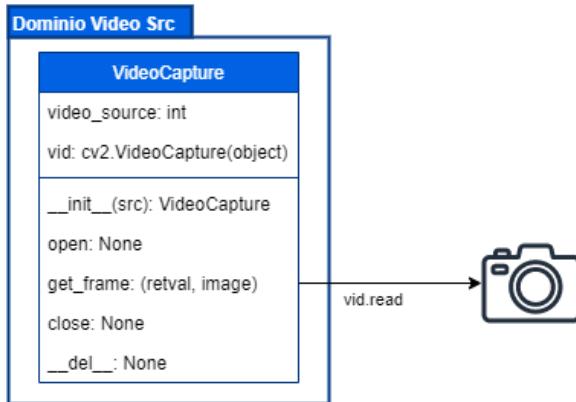


Figura 31: Classe della sorgente video

- l'immagine di anteprima;
- se il Pokémon le prevede, le sprite dell'evoluzione precedente e/o quella successiva, che possiamo scorrere grazie a dei bottoni;
- l'ID del Pokémon;
- il tipo o i tipi;
- la descrizione;
- le statistiche base.

Inoltre, è presente anche un pulsante per riprodurre il verso del Pokémon caricato. Dato che il nostro obiettivo era realizzare un dispositivo a sé stante, abbiamo deciso di creare una soluzione priva di connessione ad Internet. Abbiamo quindi trovato un [repository GitHub](#) contenente diversi file utili, tra cui un JSON con varie informazioni sui Pokémons di tutte le generazioni. Abbiamo effettuato delle modifiche, lasciando solo i Pokémons di prima generazione, correggendo degli errori, e aggiungendo le descrizioni in inglese e quelle in italiano. Inoltre, abbiamo raccolto una collezione di tracce audio, ridimensionato in scala gli sprite, e disegnato i pixel art delle etichette dei tipi, per renderle identiche alle sprite presenti sui giochi per GameBoy e dare all'applicazione un tocco retrò. Per quanto riguarda le descrizioni, nella versione inglese abbiamo utilizzato un [bot per la lettura del testo](#), in modo da renderlo simile al Pokédex presente nell'anime; per la versione in italiano abbiamo contattato un ragazzo per il doppiaggio.

Il file JSON viene caricato all'avvio dell'app e i dati dei Pokémons vengono immagazzinati in una struttura dati di tipo dizionario, indicizzati dal nome del Pokémon.

## 3.5 Impostazioni

La vista relativa alle impostazioni permette di modificare alcuni parametri di esecuzione dell'applicazione, tra cui:

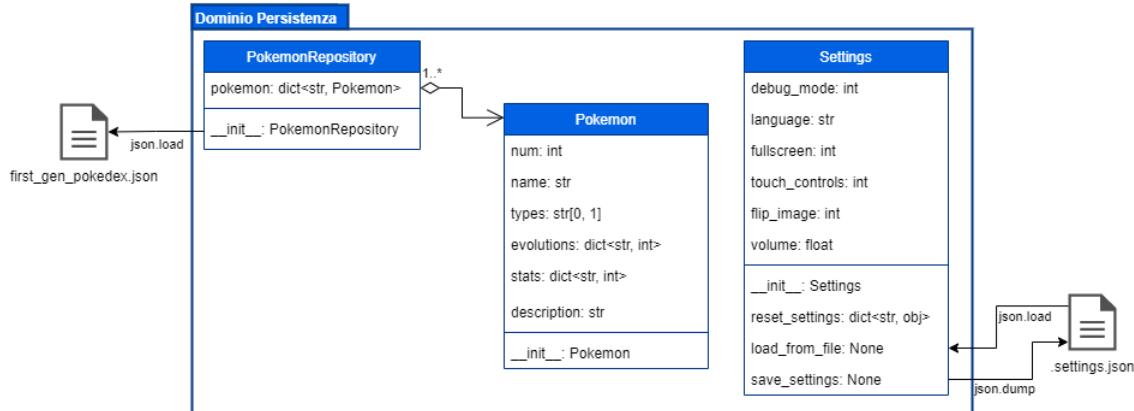


Figura 32: Diagramma di classe della persistenza

- la lingua (italiano o inglese);
- la modalità a schermo intero;
- la lettura automatica della descrizione dei Pokémon, appena ne vengono caricati i dettagli;
- la visualizzazione dell’immagine del video a specchio;
- il volume dell’applicazione.

Il funzionamento di questa parte dell’applicazione si basa sulla classe **Settings** (Figura 32), che permette di caricare, modificare e salvare le impostazioni tramite un file JSON. All’avvio il file viene caricato e, se non esiste oppure presenta dei parametri non corretti, viene ricreato utilizzando le impostazioni di default.

L’utente può aprire le impostazioni cliccando l’icona presente in alto a destra nella vista dei dettagli. Dopodiché, può modificare i parametri, i quali vengono salvati solo se seleziona il pulsante **Salva**, altrimenti vengono ignorati.

### 3.5.1 Lingua

Per implementare la modifica della lingua, abbiamo utilizzato un dizionario contenente per ciascuna voce la versione in inglese e la rispettiva traduzione in italiano. Questo dizionario viene caricato da un file `labels.json`, in modo tale che se in un momento successivo decidessimo di aggiungere ulteriori traduzioni, non sarà necessario modificare il codice dell’applicazione.

## 3.6 Audio

Il modulo audio viene inizializzato nel main, dopo il caricamento delle impostazioni, tramite la chiamata `pygame.mixer.pre_init(44100, -16, 1, 4096)`. Abbiamo dovuto



utilizzare anche questa funzione in quanto `pygame.init()` avrebbe inizializzato tutti i moduli importati con i valori predefiniti. Avendo un solo slot per lo speaker, abbiamo deciso di utilizzare un unico canale per la riproduzione degli audio, sia per il verso che per la descrizione.

Quando nell'applicazione avviene il riconoscimento, assieme ai dettagli del Pokémon identificato, viene caricato anche l'audio del verso e, se l'opzione è attivata, viene riprodotto il vocale della descrizione, nella lingua specificata. Poiché il canale utilizzato è uno unico, la riproduzione del verso del Pokémon interrompe quella della descrizione.

```
1 def load_cry(self):
2     self.cry = pygame.mixer.Sound(cries_path + str(self.loaded_pokemon.num) + ".ogg")
3
4 def play_cry(self):
5     if self.loaded_pokemon:
6         self.mono_channel.play(self.cry)
7     else:
8         print("No Pokémon has been loaded")
9
10 def play_description(self):
11     if self.loaded_pokemon:
12         self.mono_channel.play(pygame.mixer.Sound("resources/descriptions_" +
13             self.settings.language + "/" + str(self.loaded_pokemon.num) + ".mp3"))
14     else:
15         print("No Pokémon has been loaded")
```

## 3.7 Input

Il metodo principale di comunicazione con l'applicazione è tramite click o trascinamento del mouse sui vari controlli dell'interfaccia utente, come button, checkbox, slider. Poiché il raspberry è collegato ad un display touch, ciò si traduce ad un tocco o trascinamento del pennino.

Per rendere il risultato finale più simile ad una console, abbiamo deciso di aggiungere un'alternativa: l'utilizzo di una levetta analogica e due bottoni SPST. Abbiamo infatti inserito nell'applicazione un'opzione per abilitare questo metodo di comunicazione, che permette di muovere il cursore con il joystick, ed emulare il click sinistro e destro con i bottoni.

La libreria `gpiozero` permette di effettuare il mapping una funzione di callback per gli input derivanti da ciascun GPIO. Tuttavia, il display occupa 26 pin, tra cui quelli che costituiscono l'interfaccia I2C, di cui la levetta avrebbe bisogno. Per il momento abbiamo deciso di abilitare l'utilizzo dei bottoni: uno per la classificazione del Pokémon, e uno per tornare indietro (mentre per accedere alle impostazioni è necessario premerli entrambi contemporaneamente).



Per l'aggiunta del joystick sarà necessario emulare l'interfaccia I2C via software, operazione che è stata inclusa nei progetti futuri.

### 3.8 Classificatore

Per utilizzare il classificatore ed effettuare predizioni sulle immagini, abbiamo eseguito i seguenti step:

1. Abbiamo richiamato l'interprete di Tensorflow Lite, per fare inferenza e ottenere le nuove predizioni. L'interprete ha il vantaggio che può essere installato separatamente, senza scaricare l'intero framework di Tensorflow, in modo da risparmiare risorse e migliorare l'efficienza dell'app [13].

```
1 import tensorflow.lite.interpreter as tflite
```

2. Sono state utilizzate le funzioni `get_input_details()` e `get_output_details()`, che restituiscono una lista in cui ciascun elemento è un dizionario con i dettagli su rispettivamente tensore in input e output.

```
1 TFLITE_MODEL="./resources/classifier_model/model.tflite"
2 interpreter = tflite.Interpreter(TFLITE_MODEL)
3 interpreter.allocate_tensors()
4 # Get input and output tensors
5 input_details = interpreter.get_input_details()
6 output_details = interpreter.get_output_details()
```

3. L'immagine è stata caricata e convertita a risoluzione 224 x 224, mediante la libreria `Pillow`, convertita in un array numpy.
4. Viene effettuata una divisione per 255 sull'immagine per convertire i pixel RGB dall'intervallo di valori [0, 255] all'intervallo [0.0, 1.0]. Questo è dovuto al fatto che la rete, in fase di allenamento, ha ricevuto in input valori compresi tra [0.0, 1.0]. Abbiamo aggiunto una dimensione in più con `np.expand_dims`, dato che il modello si aspetta questo formato.
5. Si invoca l'Interpreter che legge i dati input dal 0-esimo array specificato.

```
1 img = Image.open(image_filename)
2 img = img.resize((224, 224), Image.ANTIALIAS)
3 img = np.asarray(img, dtype=np.float32)
4 img /= 255
```



```
5 img = np.expand_dims(img, axis=0)
6 input_tensor = np.array(img, dtype=np.float32)
7
8 # Load the TFLite model and allocate tensors.
9 interpreter.set_tensor(input_details[0]['index'], input_tensor)
10 interpreter.invoke()
11 output_data = interpreter.get_tensor(output_details[0]['index'])
```

6. Si importa il LabelEncoder, utilizzato per identificare le classi dei Pokémon.
7. Si restituiscono i nomi e le percentuali dei primi num\_top\_pokemon stimati dal modello.

```
1 label_encoder = get_label_encoder()
2 # Get best num_top_pokemon
3 results = np.squeeze(output_data, axis=0)
4 # find index of first num_top_pokemon predicted
5 top_k_idx = np.argsort(results)[-num_top_pokemon:][::-1]
6 top_k_values = results[top_k_idx]
7 top_k_labels = label_encoder.inverse_transform(top_k_idx)
8 return top_k_labels, top_k_values
```

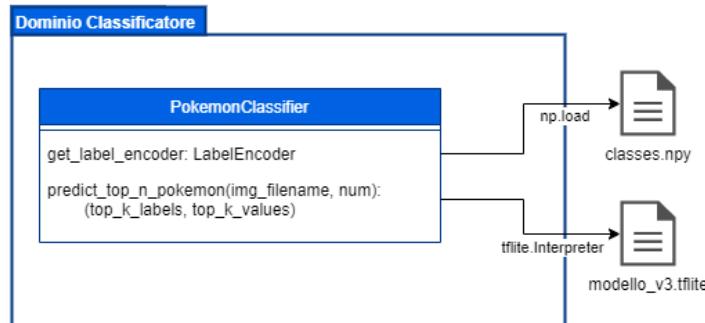


Figura 33: Classe del classificatore

### 3.9 Distorsione nelle Fotocamere

La PiCamera, come qualsiasi dispositivo di acquisizione di immagini, scatta fotografie che presentano una distorsione significativa dovuta alla lente; i due tipi principali di distorsione sono la distorsione *radiale* e quella *tangenziale*. La distorsione radiale mostra



le linee rette come se fossero curve ed è proporzionale alla lontananza dai punti rispetto al centro dell'immagine. La distorsione radiale può essere rappresentata come segue:

$$\begin{aligned}x_{distorted} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\y_{distorted} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6)\end{aligned}$$

Allo stesso modo, la distorsione tangenziale si verifica a causa del mancato allineamento tra la lente di acquisizione dell'immagine e il piano dell'immagine. Per questo motivo, alcune aree dell'immagine potrebbero sembrare più vicine del previsto. La quantità di distorsione tangenziale può essere rappresentata come segue:

$$\begin{aligned}x_{distorted} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\y_{distorted} &= y + [p_1(r^2 + 2y^2) + 2p_2xy]\end{aligned}$$

In breve, da queste equazioni possiamo ricavare cinque parametri, che prendono il nome di *coefficienti di distorsione*:

$$Coefficients of distortion = (k1 \ k2 \ p1 \ p2 \ k3)$$

Oltre a questi parametri, abbiamo bisogno di altre informazioni, come i parametri *intrinseci* ed *estrinseci* della fotocamera.

I parametri intrinseci sono specifici della fotocamera e includono informazioni come la lunghezza focale ( $f_x, f_y$ ) e i centri ottici ( $c_x, c_y$ ); si possono utilizzare per creare una *matrice della fotocamera*, in modo da rimuovere la distorsione dovuta agli obiettivi di una fotocamera specifica. Questa matrice è specifica e, una volta calcolata, può essere utilizzata su altre immagini scattate dalla stessa fotocamera. Si esprime nel seguente modo:

$$Matrix of the camera = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

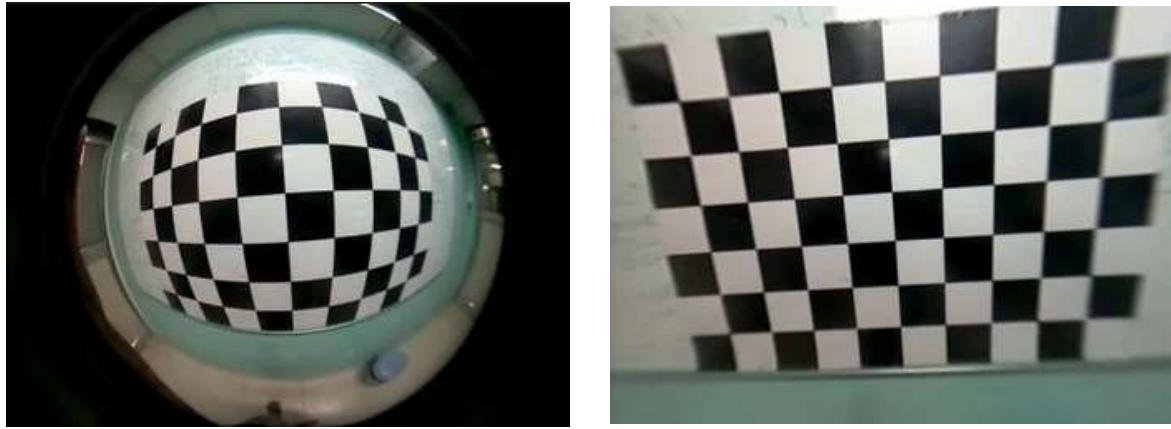
I parametri estrinseci corrispondono ai vettori di rotazione e traslazione che traducono le coordinate di un punto 3D in un sistema di coordinate.

Per trovare questi parametri, dobbiamo fornire alcune immagini campione di uno schema ben definito, ad esempio una scacchiera. Troviamo alcuni punti specifici di cui conosciamo già le posizioni relative, ad esempio gli angoli retti nella scacchiera. Poiché conosciamo le coordinate di questi punti nello spazio del mondo reale, e le coordinate nell'immagine, possiamo ricavare i coefficienti di distorsione. Per risultati migliori, abbiamo bisogno di almeno 10 modelli di test.

Una volta ottenuti tali parametri, possiamo utilizzarli per correggere nuove immagini ottenute dalla stessa fotocamera [14].

### 3.9.1 Calibrazione PiCamera

Come spiegato precedentemente, occorreva ricavare i coefficienti di distorsione e la matrice della fotocamera per la PiCamera. Una volta ottenuti, avremmo potuto utilizzarli per effettuare la calibrazione. A tale scopo abbiamo creato due script, uno per la raccolta delle



(a) Scacchiera distorta

(b) Scacchiera rettificata

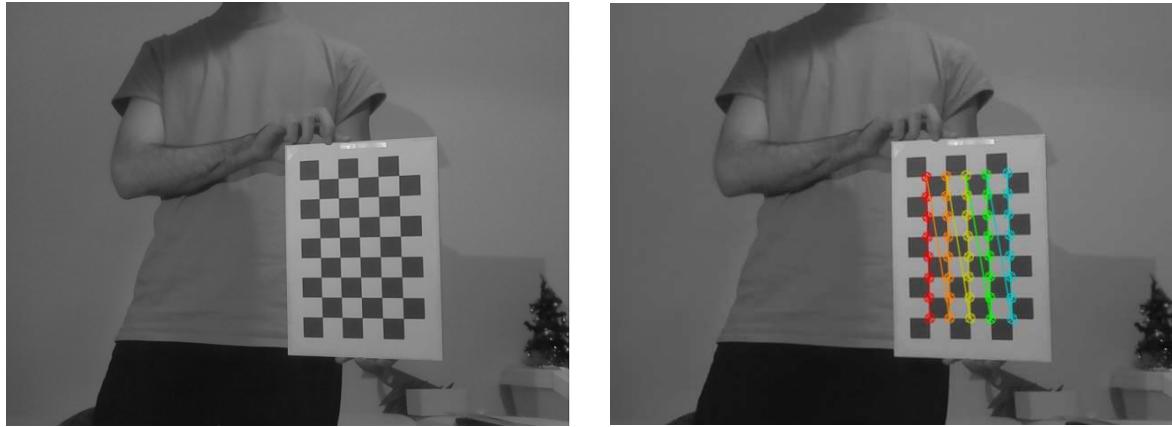
Figura 34: Esempio di correzione della distorsione

foto e uno per la calibrazione della fotocamera, sfruttando le funzioni messe a disposizione da OpenCV. Nel nostro caso abbiamo utilizzato una scacchiera  $9 \times 6$ , con ciascun quadrato di  $2.82\text{cm}$ , stampata su un foglio A4 ed incollata a un cartoncino rigido; inoltre, abbiamo scattato 30 foto di test, anziché 10. Infine, abbiamo creato un ultimo script, da inserire nella nostra applicazione, per effettuare la rettificazione dell'immagine del Pokémon.

**Raccolta Immagini** Lo script `capture_30_frames.py`, permette di scattare 30 foto di test, ciascuna con un ritardo di qualche secondo per consentirci di cambiare la posizione della scacchiera.

**Calibrazione** Lo script `calibrate_camera.py` permette di aprire le foto catturate con `capture_30_frames.py`, e per ciascuna foto, trovare il pattern della scacchiera, effettuare dei calcoli per correggerlo, e raccogliere dati sulle correzioni effettuate:

- utilizzando la funzione `cv2.findChessboardCorners()` (che accetta il numero di intersezioni tra i quadrati, nel nostro caso  $8 \times 5$ ), abbiamo ottenuto la posizione degli angoli della scacchiera;
- con la funzione `cv2.cornerSubPix()` abbiamo incrementato l'accuratezza di tali punti;
- sfruttando la funzione `cv2.drawChessboardCorners()` abbiamo disegnato il pattern sulla scacchiera (questo passaggio non era indispensabile, ma può essere interessante per capire come vengono rettificate le immagini, vedi ad esempio Figura 35b).



(a) Esempio foto distorta

(b) Esempio distorsione corretta

Figura 35: Calibrazione

In questo modo abbiamo ottenuto dei valori che prendono il nome di *object point* (punti a 3 dimensioni) e *image point* (punti a 2 dimensioni). Dopodiché, passandoli alla funzione `cv2.calibrateCamera()`, abbiamo ricavato la matrice della fotocamera e i coefficienti di distorsione ( $C_{dist}$ ):

$$\text{Matrice della Fotocamera} = \begin{bmatrix} 630.72150645 & 0 & 312.61449649 \\ 0 & 630.54052646 & 226.2012589 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C_{dist} = (0.28761658 \ -1.71752262 \ -0.01023438 \ 0.00380982 \ 3.0660223)$$

Per poterli riutilizzare in seguito, li abbiamo salvati in due file binari (`camera_matrix.npy` e `_coefficients.npy`).

**Rettificazione Immagini** Per effettuare la rettificazione, abbiamo creato lo script `image_rectifier.py` che carica i parametri ottenuti in precedenza dai file binari e li passa alla funzione `cv2.undistort()` che permette di rettificare un'immagine. In questo modo, quando l'utente scatta una foto dall'app, questa viene rettificata prima di essere passata al classificatore per la predizione.

```
1 def rectify_image(img):
2     camera_matrix = np.load("./resources/rectification_parameters/camera_matrix.npy")
3     dist_coefs =
4         np.load("./resources/rectification_parameters/distortion_coefficients.npy")
5     h, w = img.shape[:2]
6     # Undistort the image
7     new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(camera_matrix, dist_coefs,
8         (w, h), 1, (w, h))
9     dst = cv2.undistort(img, camera_matrix, dist_coefs, None, new_camera_matrix)
```



```
8     # Crop and Return the image
9     x, y, w, h = roi
10    dst = dst[y:y + h, x:x + w]
11    return dst
```



## 4 Demo

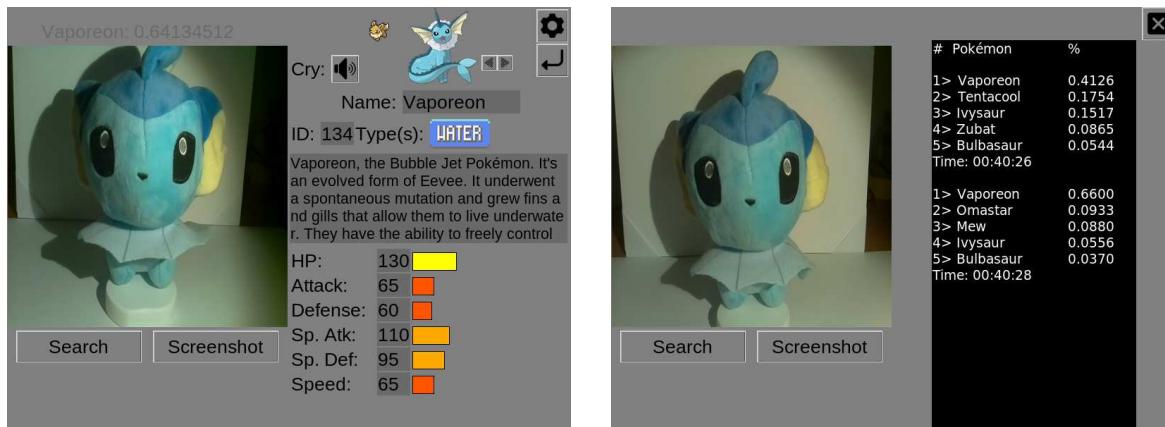
Per verificare il corretto funzionamento dell'applicazione, abbiamo testato Poké-Pi-Dex con diverse tipologie di oggetti rappresentanti diverse specie di Pokémon. Di seguito riporteremo i risultati di alcune delle predizioni più significative che abbiamo ottenuto, assieme agli oggetti utilizzati.

### 4.1 Peluche

**Vaporeon (n°134)** Nelle immagini seguenti, possiamo notare come il peluche di Vaporeon (Figura 36) venga riconosciuto correttamente, anche effettuando predizioni differenti. In particolare, in Figura 37b, possiamo notare come il primo Pokémon restituito dal classificatore sia Vaporeon, mentre quelli immediatamente successivi (Tentacool e Omastar) siano comunque Pokémon che presentano colorazioni tendenti all'azzurro.



Figura 36: Peluche di Vaporeon



(a) Vaporeon dettagli

(b) Vaporeon debug

Figura 37: Predizione corretta di Vaporeon



**Gengar (n°94)** Nelle immagini seguenti, osserviamo il riconoscimento di un peluche di Gengar, con predizioni differenti. In Figura 39b possiamo notare come tutti i Pokémon presenti nei risultati delle predizioni presentino una colorazione tendente al viola. Nel caso della predizione a tempo 00:41:37, possiamo osservare come il secondo Pokémon sia Venonat che, oltre ad avere colori molto simili a Gengar (manto viola, occhi tondeggianti e rossi), possiede anche una forma sferica.



Figura 38: Peluche di Gengar

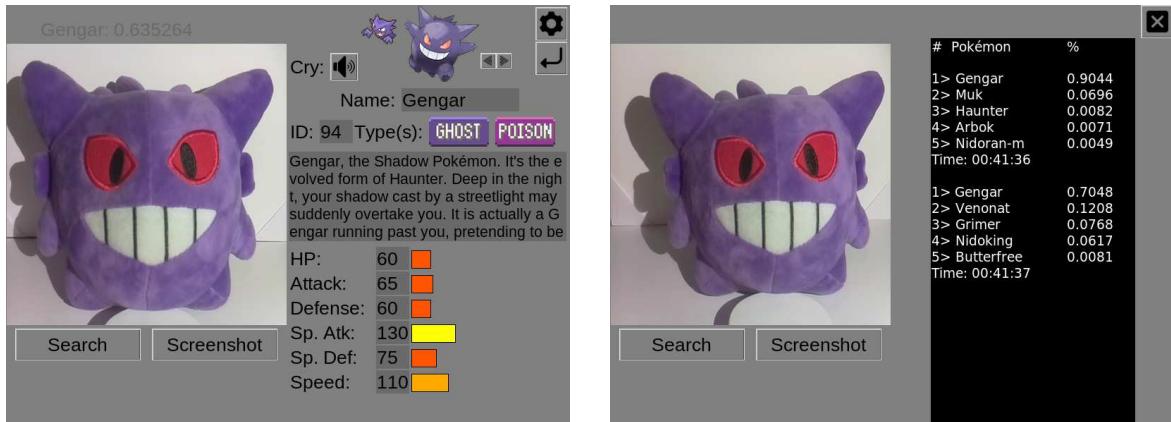
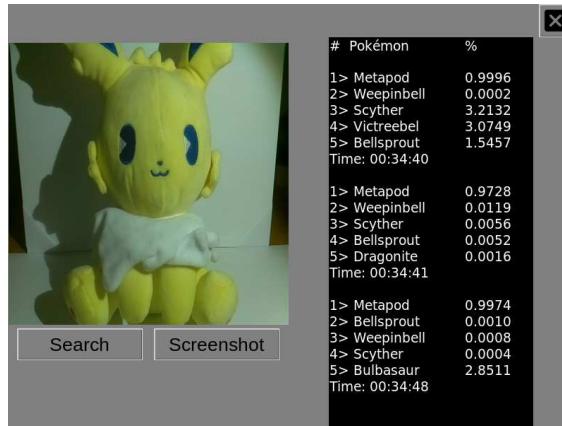


Figura 39: Predizione corretta di Gengar

**Jolteon (n°135)** Nelle immagini seguenti mostriamo una predizione errata, facendo qualche considerazione. Il peluche a disposizione non è una riproduzione fedele del Pokémon originale, in quanto presenta diversi elementi differenti, non solo nella forma, ma anche nei colori: orecchie, occhi, bocca (vedi Figura 40a). Inoltre, osserviamo come l'immagine visualizzata dall'applicazione, probabilmente a causa della luminosità scarsa, presenta una colorazione tendente al verde. Per questo motivo, come possiamo notare in Figura 40b, il classificatore ha restituito Metapod come primo risultato, dato che presenta tali colori.



(a) Peluche di Jolteon



(b) Jolteon debug

Figura 40: Predizione errata di Jolteon

## 4.2 Action Figure

**Psyduck (n°54)** Nelle seguenti immagini possiamo notare due predizioni corrette ed una parzialmente errata. Come nel caso precedente, osserviamo che la luminosità delle foto effettuate con la PiCamera non è ottimale, e i colori sulle tonalità del giallo tendono leggermente al verde. Di conseguenza, facendo riferimento alla Figura 42b, notiamo che nella prima predizione vi è al primo posto Metapod che, come spiegato prima, ha colori verdastri, mentre al secondo posto Psyduck. Nella seconda predizione, invece, Psyduck si trova al primo posto, ancora con una percentuale non altissima. È interessante notare come la maggior parte dei Pokémon presenti nella predizione possiedano una colorazione preferibilmente gialla, con qualche parte verde.

Il motivo di una differenza di confidenza così bassa (solo 8%) nella prima predizione della seconda immagine è probabilmente dovuto al fatto che la distorsione dei colori non è così accentuata come lo era invece nel caso di Jolteon.

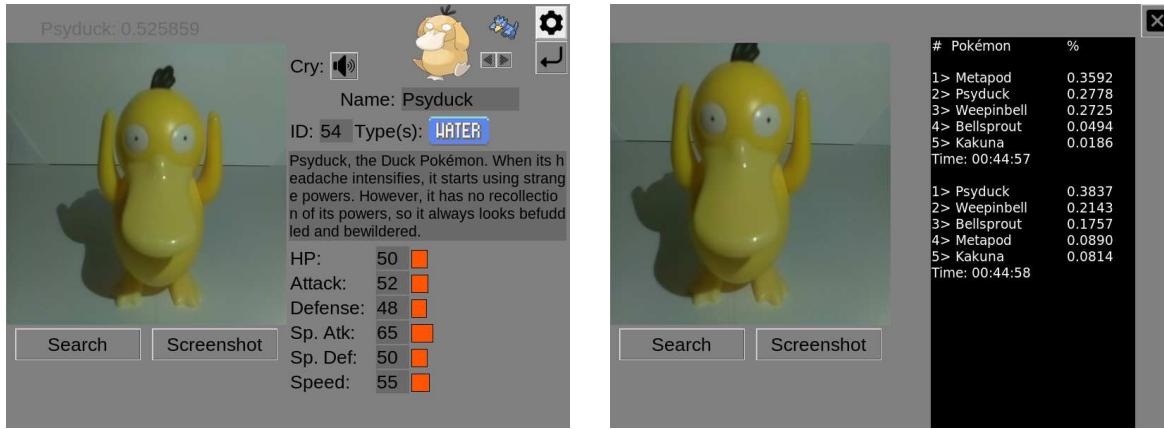


Figura 41: Figure di Psyduck



## 4.2 Action Figure

42



(a) Psyduck dettagli

(b) Psyduck debug

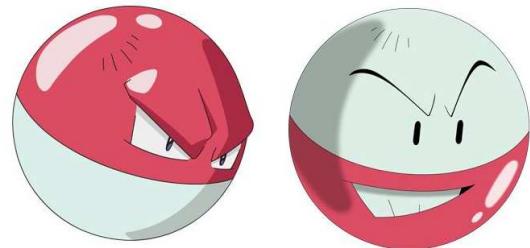
Figura 42: Predizione corretta di Psyduck

**Voltorb (n°100)** Il test che riportiamo di seguito è probabilmente uno dei più interessanti. Voltorb e la sua fase evolutiva Electrode sono ispirati entrambi a una Pokéball<sup>8</sup> (vedi Figura 43b).

Quando abbiamo provato a passare la figura di una Pokéball al classificatore, il risultato è stato sorprendente: facendo riferimento alla Figura 44a, possiamo notare come su tre predizioni, tutte e tre abbiano dato come primo risultato Voltorb. Nella seconda predizione abbiamo ottenuto Electrode come secondo risultato e nella terza quasi il 100% di confidenza. Inoltre, girando sotto sopra la Pokéball ed effettuando una predizione, abbiamo ottenuto Electrode come primo risultato, e Voltorb come secondo.



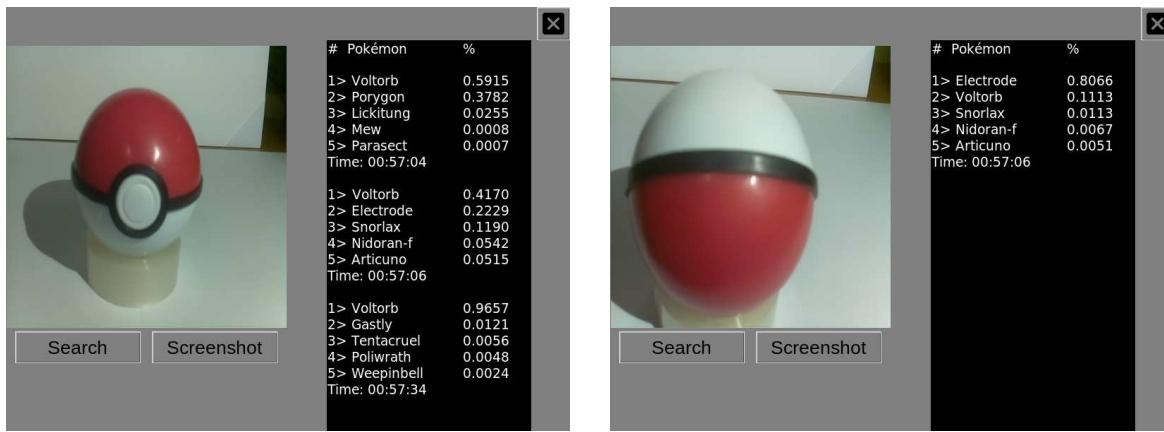
(a) Figure di una Pokéball



(b) Voltorb ed Electrode

Figura 43: I Pokémon Ball

<sup>8</sup>La Pokéball è l'oggetto tramì cui è possibile catturare i Pokémon. È anche uno dei simboli principali del brand.



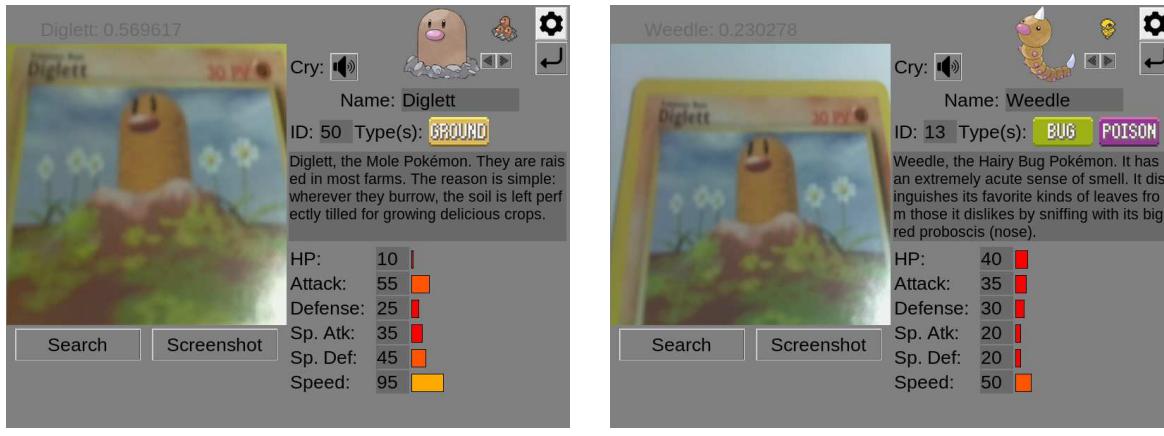
(a) Voltorb Debug

(b) Electrode debug

Figura 44: Predizione corretta di Voltorb ed Electrode

### 4.3 Carte da Gioco

**Diglett (n°50)** Nelle immagini seguenti mostriamo una predizione corretta di Diglett (Figura 45a), ed una errata (Figura 45b). In particolare, nella predizione errata possiamo notare che la percentuale di confidenza è piuttosto bassa, e il Pokémon risultante (Weedle) assomiglia per colori e dettagli a Diglett.



(a) Riconoscimento corretto di Diglett

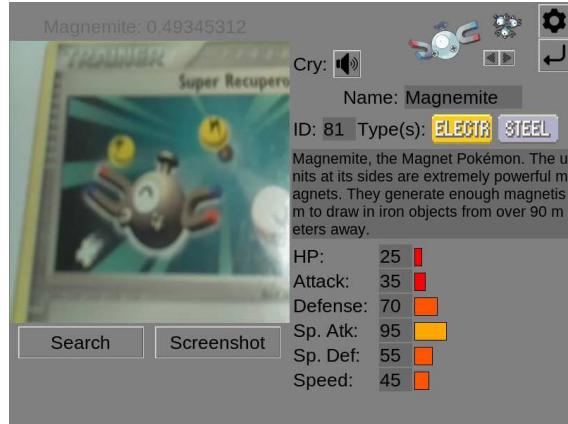
(b) Riconoscimento corretto di Diglett

Figura 45: Predizione di Diglett

**Magnemite (n°81)** Caso interessante: osserviamo che la carta fotografata non è esattamente una carta del Pokémon, ma una carta *Trainer*, ovvero una sorta di carta abilità, i cui disegni variano molto e non necessariamente devono includere un Pokémon. In questo caso Magnemite compare in piccolo e assieme ad altri elementi, ma il nostro classificatore è riuscito a riconoscerlo comunque.



(a) Carta di Magnemite



(b) Magnemite debug

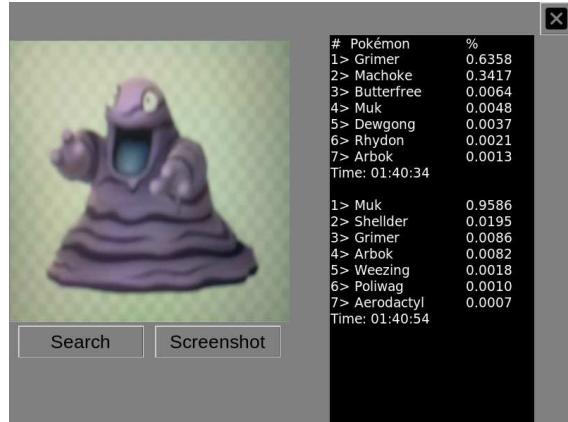
Figura 46: Predizione corretta di Magnemite

## 4.4 Immagini dal Telefono

**Grimer (n°88)** Un altro caso particolare è quello in cui si hanno Pokémon molto simili fra loro. Ad esempio, in Figura 47b, possiamo osservare come su due predizioni, nella prima Grimer sia stato riconosciuto correttamente, mentre nella seconda sia stato riconosciuto Muk, con una confidenza elevatissima. Notiamo che in entrambe le predizioni è presente il rispettivo (in quella di Grimer Muk, e viceversa), in quanto sono due Pokémon praticamente identici, che cambiano solitamente per la posa e qualche piccolo dettaglio su occhi e bocca. Osserviamo infine che la maggior parte dei Pokémon seguenti hanno colori tendenti al viola.



(a) Grimer e Muk



(b) Grimer debug

Figura 47: Pokémon invertiti



## 5 Conclusioni

In questo progetto abbiamo realizzato un dispositivo portatile di riconoscimento di Pokémon sotto forma di disegni, peluche e giocattoli, noto come *Pokédex*. A tale scopo abbiamo creato una rete neurale convoluzionale con tre livelli convoluzionali e due fully connected.

Abbiamo eseguito numerosi test e ci siamo accorti che il dataset utilizzato presentava diversi problemi. Pertanto, abbiamo selezionato e tagliato manualmente 11943 foto e riallenato la rete con 55 foto per classe.

Abbiamo convertito il modello in Tensorflow Lite, in modo da diminuire le dimensioni da 1.2 GB a 393 MB, e successivamente quantizzato, per comprimerlo ulteriormente fino a 98 MB.

Infine, abbiamo importato quest'ultimo nell'applicazione, che ha lo scopo di scattare fotografie e mostrare a video tutte le informazioni riguardanti il Pokémon riconosciuto.

Abbiamo creato l'app appositamente per eseguirla su Raspberry Pi 4 model B, a cui abbiamo collegato un display LCD da 3.5 pollici, una PiCamera, un powerbank, uno speaker, una levetta analogica e due bottoni; abbiamo quindi racchiuso tutte le componenti in una custodia di cartoncino riciclato realizzata ad hoc, e fedele alla versione originale presente nel cartone animato.

In seguito a diverse prove con il Pokédex realizzato da noi, abbiamo constatato che:

- il riconoscimento del Pokémon avviene in circa 200 ms e la predizione risulta corretta in circa l'85% dei casi nella vita reale;
- il `test_accuracy` risulta essere al 99%, ma dato che stiamo considerando il dominio delle immagini del dataset con cui è stato allenato, abbiamo osservato performance diverse nella vita reale;
- la predizione è ogni tanto errata, in particolare quando le foto vengono scattate con sfondi che ricordano colori di altri Pokémon, oppure con luminosità particolari<sup>9</sup>, specialmente quando la foto assumeva colori tendenti al grigio e al marrone. Qualche volta la rete può invertire Pokémon che si assomigliano molto fra loro, ad esempio Grimer e Muk (vedi Paragrafo 4.4);
- viste le diverse tipologie di oggetti da riconoscere, sarebbe servito un dataset più ampio (circa 1000 foto per classe) per un riconoscimento dei sample catturati dalla Picamera con errore trascurabile. In tal caso l'architettura del classificatore sarebbe dovuta essere decisamente più complessa rispetto a quella attualmente utilizzata.

Il classificatore finale, sebbene con qualche imprecisione, ha ottenuto dei risultati molto soddisfacenti, in quanto riconosce la maggior parte di carte, peluche e disegni. La semplicità dell'applicazione e dell'architettura del Raspberry ha consentito un facile utilizzo da chiunque, ovunque si trovi.

<sup>9</sup>Una possibile soluzione al problema della luminosità potrebbe essere l'inserimento di qualche Led bianco vicino alla lente della PiCamera, in modo tale da poter illuminare il Pokémon inquadrato



## 6 Sviluppi Futuri

Come possibili sviluppi futuri prevediamo di:

- utilizzare una rete neurale più complessa con il dataset da 11943 immagini, in modo da riconoscere più accuratamente i sample catturati con la PiCamera;
- aggiungere nuove forme di data augmentation sulle immagini. Un esempio potrebbe essere introdurre tecniche di cut-off, che consistono nell'inserire disturbi casuali sulle immagini;
- aggiungere un amplificatore per incrementare l'output audio dello speaker;
- inserire uno o più led bianchi sul retro, al fine di illuminare l'area inquadrata dalla PiCamera, e renderne l'accensione configurabile a tempo di esecuzione dall'applicazione;
- implementare la funzione della levetta analogica;
- inserire un interruttore switch per l'accensione del Raspberry;
- aggiungere un'opzione nell'applicazione per poter abilitare o disabilitare il congelamento dell'immagine dopo lo scatto (aggiungendo un bottone per salvare il frame ed uno per riprendere lo streaming video);
- realizzare un modello 3D del case e stamparlo;
- estendere il Pokédex alle generazioni successive;
- effettuare il porting dell'applicazione su sistemi mobile (Android e iOS).



## Riferimenti bibliografici

- [1] *Pokémon*. 2015. URL: <https://web.archive.org/web/20150527200032/https://www.britannica.com/EBchecked/topic/1474435/Pokemon>.
- [2] *Stat*. URL: <https://bulbapedia.bulbagarden.net/wiki/Stat>.
- [3] Agnieszka Mikołajczyk e Michał Grochowski. «Data augmentation for improving deep learning in image classification problem». In: mag. 2018, pp. 117–122. DOI: [10.1109/IIPHDW.2018.8388338](https://doi.org/10.1109/IIPHDW.2018.8388338).
- [4] Luis Perez e Jason Wang. «The Effectiveness of Data Augmentation in Image Classification using Deep Learning». In: *CoRR* abs/1712.04621 (2017). arXiv: [1712.04621](https://arxiv.org/abs/1712.04621). URL: <http://arxiv.org/abs/1712.04621>.
- [5] Connor Shorten e Taghi M. Khoshgoftaar. «A survey on Image Data Augmentation for Deep Learning». In: *Journal of Big Data* 6.1 (2019). ISSN: 2196-1115. DOI: [10.1186/s40537-019-0197-0](https://doi.org/10.1186/s40537-019-0197-0). URL: <https://doi.org/10.1186/s40537-019-0197-0>.
- [6] Jiuxiang Gu et al. «Recent Advances in Convolutional Neural Networks». In: *CoRR* abs/1512.07108 (2015). arXiv: [1512.07108](https://arxiv.org/abs/1512.07108). URL: <http://arxiv.org/abs/1512.07108>.
- [7] Sergey Ioffe e Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». In: *CoRR* abs/1502.03167 (2015). arXiv: [1502.03167](https://arxiv.org/abs/1502.03167). URL: <http://arxiv.org/abs/1502.03167>.
- [8] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: [1805.11604 \[stat.ML\]](https://arxiv.org/abs/1805.11604).
- [9] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: [1412.6980 \[cs.LG\]](https://arxiv.org/abs/1412.6980).
- [10] *tf.keras.callbacks.EarlyStopping*. URL: [https://www.tensorflow.org/api\\_docs/python/tf/keras/callbacks/EarlyStopping](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping).
- [11] *TensorFlow Lite converter*. URL: <https://www.tensorflow.org/lite/convert>.
- [12] *Post-training quantization*. URL: [https://www.tensorflow.org/lite/performance/post\\_training\\_quantization](https://www.tensorflow.org/lite/performance/post_training_quantization).
- [13] *TensorFlow Lite inference*. URL: <https://www.tensorflow.org/lite/guide/inference>.
- [14] *Camera Calibration*. URL: [https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html).