CookingAlly, applicazione di object detection per il riconoscimento di ingredienti e il suggerimento di ricette

Corso di Sistemi Digitali M

Anno Accademico 2023/2024

Gruppo:
Maria Beatrice Bottari, Noemi Messori, Arianna Pierini

Indice

Introduzione

L'obiettivo di questo progetto è stato sviluppare un'applicazione mobile in grado di riconoscere una varietà di ingredienti utilizzando la fotocamera e fornire suggerimenti su possibili ricette da realizzare con essi. Questo documento illustra il processo seguito per addestrare un modello personalizzato di riconoscimento degli ingredienti utilizzando YOLOv5, oltre agli strumenti impiegati per la creazione dell'applicazione e la definizione della sua logica.

Il report è articolato come segue:

- nel **capitolo 1** viene fornita una breve panoramica sulla Computer Vision, con particolare attenzione alle principali differenze tra Image Classification e Object Detection;
- nel **capitolo 2** vengono dettagliati gli strumenti utilizzati e il processo di addestramento della rete;
- nel **capitolo 3** viene delineato lo sviluppo dell'applicazione Android, descrivendo le sue caratteristiche e i suoi componenti;
- nel **capitolo 4** vengono esaminati i problemi riscontrati e si discutono i possibili sviluppi futuri.

Il codice, il modello e la demo del progetto possono essere reperiti nella seguente repository git: https://github.com/noemimessori/CookingAlly.git.

Capitolo 1

Introduzione alla Computer Vision, Object Detection e Image Classification

Negli ultimi decenni, i rapidi progressi nell'intelligenza artificiale e nell'elaborazione delle immagini hanno portato alla creazione di sistemi in grado di analizzare e interpretare contenuti visivi con una precisione sempre maggiore. La Computer Vision è un campo interdisciplinare che si concentra sullo sviluppo di algoritmi e tecniche per l'interpretazione automatica di immagini e di video.
In questo contesto, due concetti chiave sono l'Object Detection e l'Image Classification. Queste tecniche sono fondamentali per comprendere e analizzare il contenuto visivo delle immagini, ma differiscono significativamente nel loro approccio e nei risultati ottenuti.

1.1 Introduzione alla Computer Vision

La *Computer Vision* o *visione artificiale* è un campo dell'informatica e dell'intelligenza artificiale che studia algoritmi e tecniche per permettere ai computer di riprodurre funzioni e processi dell'apparato visivo umano. Quest'ultima non si occupa solo del riconoscimento di oggetti, persone o animali all'interno di un'immagine singola o in sequenza, ma anche di estrarre informazioni utili per la loro elaborazione, a livelli sempre più alti di astrazione e comprensione. In altri termini, si tratta della capacità di ricostruire un contesto intorno all'immagine, dandole un vero e proprio significato. Essa, grazie alla sua ampia possibilità di applicazione, può essere considerata interdisciplinare: dalla robotica a settori differenti come sicurezza, medicina e automotive.

Nella pratica, il funzionamento della Computer Vision si manifesta attraverso un'analisi approfondita delle immagini al fine di riconoscere e interpretare gli oggetti rappresentati. Le tecniche di riconoscimento delle immagini e degli oggetti comprendono diversi processi che vanno dalla classificazione alla segmentazione, al riconoscimento facciale. Ad esempio, nel contesto del riconoscimento delle automobili, gli algoritmi di Computer Vision sono in grado di esaminare e identificare specifici veicoli all'interno delle immagini.
Gli algoritmi di Computer Vision sono in grado di condurre analisi più o meno approfondite a seconda delle tecniche adottate, del tipo di immagine e del compito specifico assegnato.
Tra i task più comuni si possono nominare:

- Image Classification: analisi del contenuto dell'immagine e attribuzione di un'etichetta. Esempio: classificazione di animali come cane e gatto.
- Object Detection: identificazione di una o più entità all'interno di un'immagine.
- Image Segmentation: suddivisione dell'immagine in sezioni.
- Face Recognition: riconoscimento di volti di persone.

Le soluzioni alle quali è rivolta più attenzione sono generalmente le prime due: Image Classification e Object Detection.

1.2 Image Classification

L'Image Classification è la capacità di un algoritmo di Intelligenza Artificiale di classificare un'immagine tramite un set di etichette predefinite, chiamate "classi". Questo task è quindi un esempio di classificazione, dove tale può essere binaria o multiclasse, a seconda del compito che è necessario svolgere. Inoltre, al fine della realizzazione possono essere utilizzati molti algoritmi differenti. Solitamente, però, quelli più utilizzati per l'Image Classification, sono quelli che poi risultano anche più performanti, ovvero reti neurali convolutive o convoluzionali, abbreviate a CNN.

1.3 Object Detection

A differenza dell'Image Classification, l'Object Detection va oltre e mira a identificare e localizzare gli oggetti specifici all'interno di un'immagine. Questo compito richiede non solo di riconoscere quali oggetti sono presenti, ma anche di determinare la loro posizione all'interno dell'immagine attraverso elementi di delimitazione (*bounding box*) che circondano quanto individuato. Questa tecnica risulta più complessa della precedente poichè convolge l'individuazione della localizzazione precisa

degli oggetti all'interno di un'immagine.
Le applicazioni di Object Detection sono ampie e diversificate. Esse includono la sorveglianza video, il conteggio e il tracciamento di oggetti in tempo reale, la guida autonoma, la classificazione di immagini mediche e molto altro. La precisione e l'efficienza dell'Object Detection sono essenziali per l'implementazione di sistemi intelligenti in vari contesti applicativi.

1.4 Differenze tra Object Detection e Image Classification

La differenza chiave tra Image Classification e Object Detection risiede nel livello di dettaglio e precisione richiesti nel riconoscimento degli oggetti. Mentre la prima determina semplicemente la classe di una fotografia intera, l'Object Detection individua gli oggetti specifici e fornisce informazioni dettagliate sulla loro posizione all'interno dell'immagine, permettendo un'analisi visiva più specifica. L'Image Classification assegna una sola classe a tutta l'immagine; l'Object Detection, al contrario, restituisce una lista di oggetti riconosciuti nella stessa figura.

Per questo motivo, all'interno della nostra applicazione è stato sfruttato quest'ultimo task: fondamentale per individuare più oggetti differenti in una sola fotografia. Più in particolare, andiamo a riconoscere classi differenti di ingredienti.

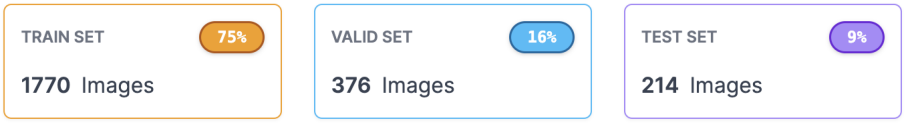
Capitolo 2

Svolgimento e strumenti utilizzati

Allo scopo di sviluppare questo sistema di riconoscimento di oggetti in un'immagine, abbiamo fatto largo uso di strumenti chiave come Jupyter Notebook per la configurazione dell'ambiente Python, Roboflow per la creazione del dataset, YOLO v5 per l'addestramento del modello e TensorFlow Lite per l'esportazione della rete. Infine, Android Studio è stato impiegato per lo sviluppo dell'applicazione Android.

2.1 Roboflow

RoboFlow è una piattaforma introdotta nel gennaio 2020 che permette di semplificare il complesso processo di preparazione e gestione dei dati necessari per il Machine Learning e la Computer Vision. Grazie a questa risorsa, abbiamo potuto creare un dataset fondamentale per addestrare il nostro modello di riconoscimento degli ingredienti. Per garantire un bilanciamento adeguato, abbiamo raccolto circa 400 immagini per ogni classe di ingredienti: Carota, Formaggio, Uovo, Patata e Zucchina. Le immagini sono state ottenute da diverse fonti, inclusi dataset pubblici e motori di ricerca come Google Images, per garantire varietà e completezza. Successivamente, abbiamo diviso il dataset in 1770 immagini per l'addestramento, 376 per la fase di validazione e 214 per il testing, con un totale di 2360 immagini. Utilizzando gli strumenti di annotazione forniti da RoboFlow, abbiamo potuto definire con estrema precisione le bounding box per ciascun ingrediente in ogni immagine.



2.2 YOLOv5

YOLOv5, acronimo di "You Only Look Once", è una rete implementata da Ultralytics per il rilevamento di oggetti, che consente di individuare bounding box in un'immagine utilizzando un dataset per l'addestramento. Al termine di questo processo, vengono determinati un insieme di pesi che consentono l'estrazione delle bounding box per ciascuna immagine analizzata. Questo modello è disponibile in quattro varianti: small (s), medium (m), large (l) ed extra large (x), ciascuna con differenti livelli di prestazioni e tempi di addestramento.

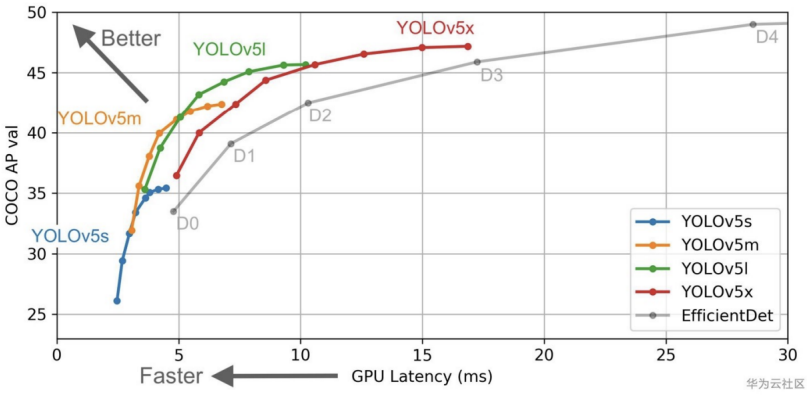


Figura 2.1: Confronto delle prestazioni delle 4 versioni di Yolov5

Abbiamo scelto di utilizzare la quinta versione di YOLO per la sua facilità d'uso e velocità grazie all'implementazione di nuove tecnologie. Rispetto alle versioni precedenti, infatti, presenta numerosi vantaggi, tra cui:

- Requisiti minimi di installazione, richiedendo solamente l'installazione di Torch e alcune librerie Python.
- Flessibilità nell'esecuzione di inferenze su singole immagini, immagini in batch, feed video o porte webcam.
- Tempi di addestramento estremamente rapidi, riducendo così i tempi morti durante la costruzione del modello.

Per addestrare YOLOv5 al fine di riconoscere un dataset di oggetti personalizzati, abbiamo effettuato i seguenti passaggi:

1. Preparazione dell'ambiente di lavoro e installazione dei pesi.
2. Preparazione ed elaborazione del dataset personalizzato.

3. Definizione del modello e dell'architettura di YOLOv5 in base alle esigenze specifiche.
4. Addestramento della rete YOLOv5 utilizzando il dataset personalizzato.
5. Valutazione delle prestazioni del modello mediante test e analisi dei risultati ottenuti.

Configurazione dell'ambiente di lavoro

Abbiamo optato per l'utilizzo di Jupyter Notebook come ambiente di lavoro, consentendoci di lavorare localmente da qualsiasi luogo durante il lungo processo di addestramento della rete. Innanzitutto, abbiamo scaricato le repository necessarie di YOLOv5 da https://github.com/ultralytics/yolov5, insieme alle dipendenze essenziali come scipy, numpy, pandas e torch, al fine di preparare l'ambiente di lavoro per l'esecuzione dei comandi di addestramento e inferenza per il rilevamento degli oggetti. Inoltre, abbiamo utilizzato l'API key di Roboflow nel notebook per scaricare il nostro dataset insieme al file data.yaml contenente le principali caratteristiche del dataset, come il numero di classi, i nomi e la suddivisione delle immagini.

Definizione del modello e sua architettura

Dunque, abbiamo realizzato un file di configurazione personalizzato per il rilevatore di oggetti. Poiché il nostro obiettivo era riconoscere poche classi di oggetti, abbiamo optato per il modello base più piccolo e veloce di YOLOv5, cioè YOLOv5s. Dopo aver apportato le modifiche necessarie, come l'impostazione del numero di classi e gli anchors, abbiamo ottenuto il nostro file custom_yolov5s.yaml, pronto per la fase di addestramento.

Addestramento della rete

Per avviare l'addestramento, è necessario eseguire il comando "training" nel notebook, specificando le seguenti opzioni:

- "img": definisce la dimensione delle immagini di ingresso.
- "batch": determina la dimensione del batch.
- "epochs": definisce il numero di epoche di addestramento.
- "data": imposta il percorso del file yaml.
- "cfg": specifica la configurazione del modello.
- "weights": specifica un percorso personalizzato per i pesi.

- "name": nomi dei risultati.
- "cache": caching delle immagini per addestramento ottimizzato.

Per il nostro allenamento, abbiamo impostato la dimensione delle immagini (‘img‘) a 320 pixel e il numero di epoche (‘epochs‘) a 200 al fine di ottenere prestazioni più elevate possibile dall’algoritmo. Il completamento dell’addestramento ha richiesto circa 4 giorni, eseguito su un MacBook Pro privo di GPU, con chip M2 e 8 GB di RAM.

```
python train.py --img 320 --batch 16 --epochs 150 --data {dataset.location}/data.yaml --
    cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache
```

Valutazioni sulle prestazioni del modello

Dopo aver concluso la fase di addestramento, abbiamo proceduto con un’attenta valutazione della sua efficacia, esaminando le metriche di valutazione su TensorBoard. Inoltre, abbiamo eseguito il comando "detect.py" per verificare se gli ingredienti presenti nelle immagini della cartella di test venissero rilevati correttamente. I risultati ottenuti sono stati più che soddisfacenti, confermando l’accuratezza del modello addestrato. In particolare, abbiamo ottenuto valori di Precision e Recall piuttosto positivi.

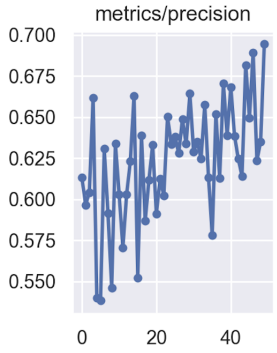


Figura 2.2: Precision

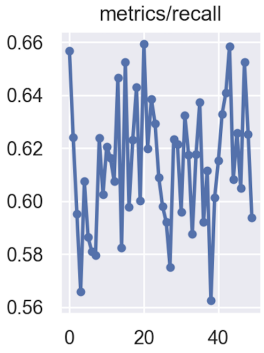


Figura 2.3: Recall

2.3 TensorFlow Lite

TensorFlow Lite è una suite di strumenti progettati per permettere agli sviluppatori di eseguire modelli TensorFlow su una vasta gamma di dispositivi, inclusi quelli

mobili, embedded e IoT.
Questa tecnologia è caratterizzata da due componenti principali:

- **L'interprete**: consente l'esecuzione di modelli appositamente ottimizzati su vari tipi di hardware, come telefoni cellulari, sistemi embedded Linux e microcontrollori.
- **Il convertitore**: permette la trasformazione dei modelli TensorFlow in un formato compatibile con l'interprete, ottimizzando le prestazioni. Durante il processo di conversione, possono essere introdotte ottimizzazioni specifiche per migliorare le prestazioni complessive.

Per ottenere un file TFLite, abbiamo utilizzato il comando "export.py", il quale ha generato il file "best-fp16.tflite". Dunque, abbiamo integrato questo file nella cartella di applicazione Android, importando poi l'intero progetto su Android Studio. In questo modo, l'app può utilizzare il nostro modello TensorFlow Lite per eseguire le operazioni di rilevamento degli ingredienti.

Capitolo 3

Sviluppo di CookingAlly

Come già anticipato, lo sviluppo dell'applicazione è stato condotto sulla piattaforma Android Studio, partendo dall'architettura e dalle componenti fondamentali per la gestione dell'Object Detection. Successivamente, sono stati importati e adattati tutti i moduli necessari per la realizzazione della logica di business.

3.1 AndroidStudio

AndroidStudio rappresenta l'IDE ufficiale per lo sviluppo di applicazioni Android, fornendo un ambiente integrato e completo per gli sviluppatori. Basato sull'IDE di IntelliJ IDEA, offre un'ampia gamma di funzionalità essenziali per la scrittura del codice, il debugging, la gestione delle risorse e altro ancora.
Tra le sue caratteristiche principali abbiamo trovato particolarmente utile il sistema di compilazione basato su Gradle, che ha facilitato il processo di sviluppo e la gestione delle dipendenze del progetto. Inoltre, la sua emulazione veloce su dispositivi virtuali è stata cruciale per una prototipazione rapida ed efficiente, consentendoci di testare facilmente l'applicazione su più dispositivi Android.

Struttura del progetto base

Il nostro punto di partenza è stata la struttura di un'applicazione Android che fa uso di un modello TFLite per il rilevamento degli oggetti in tempo reale utilizzando lo streaming proveniente dalla telecamera. In questa architettura, disponibile nel repository https://github.com/AarohiSingla/TFLite-Object-Detection-Android-App-Tutorial-Using-YOLOv5/tree/main, abbiamo importato il nostro modello TFLite.
Come in ogni applicazione Android, il modulo cardine dell'architettura è *app*, suddiviso principalmente in quattro parti:

- La directory **java**, contenente il codice Java che implementa le funzionalità e la logica dell'applicazione. La sotto-directory *tflite* ospita l'implementazione del sistema di rilevamento e classificazione degli oggetti.
- La directory **res**, che include risorse come layout XML, immagini e altri file, che definiscono l'aspetto e il comportamento dell'interfaccia utente.
- La directory **assets**,che contiene file aggiuntivi necessari, come file di configurazione e il modello TFLite per il rilevamento degli oggetti.
- Il file di configurazione *AndroidManifest.xml*, che definisce le caratteristiche, le autorizzazioni e le informazioni sulle varie componenti dell'applicazione.

Activity

Le Activity sono le entità protagoniste all'interno di un'applicazione Android, rappresentando le interfacce con cui gli utenti interagiscono direttamente. Formano il flusso principale per accedere alle funzionalità dell'applicazione, pertanto è cruciale progettare una navigazione coerente e fluida tra di esse.
Nella pratica, sono classi Java che estendono Activity e vengono registrate come nodi *<activity>* nell'AndroidManifest.xml. L'utente passerà da un'Activity all'altra in modo simile a come naviga tra le pagine dei siti web; ma qui tutto avviene attraverso il potente meccanismo degli Intent, ovvero una forma di messaggistica gestita dal sistema operativo con cui un componente può richiedere l'esecuzione di un'azione da parte di un altro componente.
La MainActivity è la figura centrale che supervisiona l'intera applicazione. Si occupa di configurare l'interfaccia utente iniziale e di gestire gli eventi di interazione con l'utente, come i clic sui pulsanti e gli input, oltre a mostrare il log di tutti gli ingredienti rilevati.
Per quanto riguarda l'Object Detection, le principali Activity sono:

- **CameraActivity**, dedicata all'acquisizione di immagini dalla fotocamera del dispositivo su cui verrà eseguita l'Object Detection.
- **DetectionActivity**, dove vengono visualizzati i risultati dell'Object Detection, le relative probabilità e altre informazioni pertinenti.

3.2 Struttura di CookingAlly

Oltre ad aver personalizzato la struttura di base per le nostre esigenze, in modo da garantire coerenza con i nostri obiettivi, abbiamo integrato due nuovi moduli funzionali, partendo proprio dalla MainActivity.

Processamento Excel e dipendenze Gradle

Abbiamo implementato una serie di funzioni, chiamate in cascata da *AddFoodByExcel*, in grado di elaborare il file Excel *recipes_total.xlsx*, che si comporta come una sorta di "database esterno". All'interno di questo file sono conservati tutti i dati delle ricette, i quali vengono successivamente memorizzati sotto forma di **Recipe**, rendendo tutto facilmente gestibile e conveniente. Questo approccio ha permesso poi di organizzare tutte le informazioni in un layout più ordinato e user-friendly.

Per manipolare il file Excel, abbiamo utilizzato le API di Apache POI, una delle librerie gratuite più diffuse e affidabili per la manipolazione dei file di Microsoft Office. Apache POI supporta una vasta gamma di versioni di Excel e fornisce un'API completa per la lettura e la scrittura, rendendola una scelta versatile e potente per lavorare con i file Excel. Nel contesto del nostro progetto gestito da Gradle, abbiamo integrato le funzionalità di Apache POI per la manipolazione dei file Excel includendo le dipendenze necessarie nel file build.gradle, all'interno del blocco dependencies, dove sono elencate anche le dipendenze per l'utilizzo di TensorFlow.

Il file Excel *recipes_total.xlsx* è conservato nella directory "assets", insieme al file TFLite e alle immagini destinate a illustrare le ricette selezionate. A differenza delle risorse presenti nella directory *res*, queste non vengono né compilate in formato binario né etichettate con un ID univoco. Per accedere a queste risorse all'interno dell'applicazione, verrà utilizzata una classe Java chiamata AssetManager. Questo consentirà di ottenere uno stream per ciascuna risorsa, facilitando il loro utilizzo durante l'esecuzione dell'applicazione.

ViewRecipeActivity

Per migliorare la visualizzazione dettagliata delle ricette, abbiamo introdotto un'Activity supplementare. Essa viene attivata dalla MainActivity al ricevimento di un nuovo intent, il quale contiene gli ingredienti rilevati e le ricette corrispondenti.
L'interfaccia grafica di questa Activity è stata costruita utilizzando il layout definito nel file *view_recipes_layout.xml* mediante l'utilizzo della funzione *setContentView(R.layout.view_recipes_layout)*, dove la classe R riflette la posizione delle risorse all'interno delle sottocartelle di res. Tutti i dettagli della ricetta associata vengono estratti dall'intent e mostrati nell'interfaccia utente.
Questa strategia ha notevolmente ampliato la flessibilità, la dinamicità e l'usabilità complessiva dell'interfaccia utente, consentendo di fornire informazioni dettagliate sulle ricette quali ingredienti, istruzioni e immagini, insieme a pulsanti per una navigazione agevole all'interno dell'applicazione.

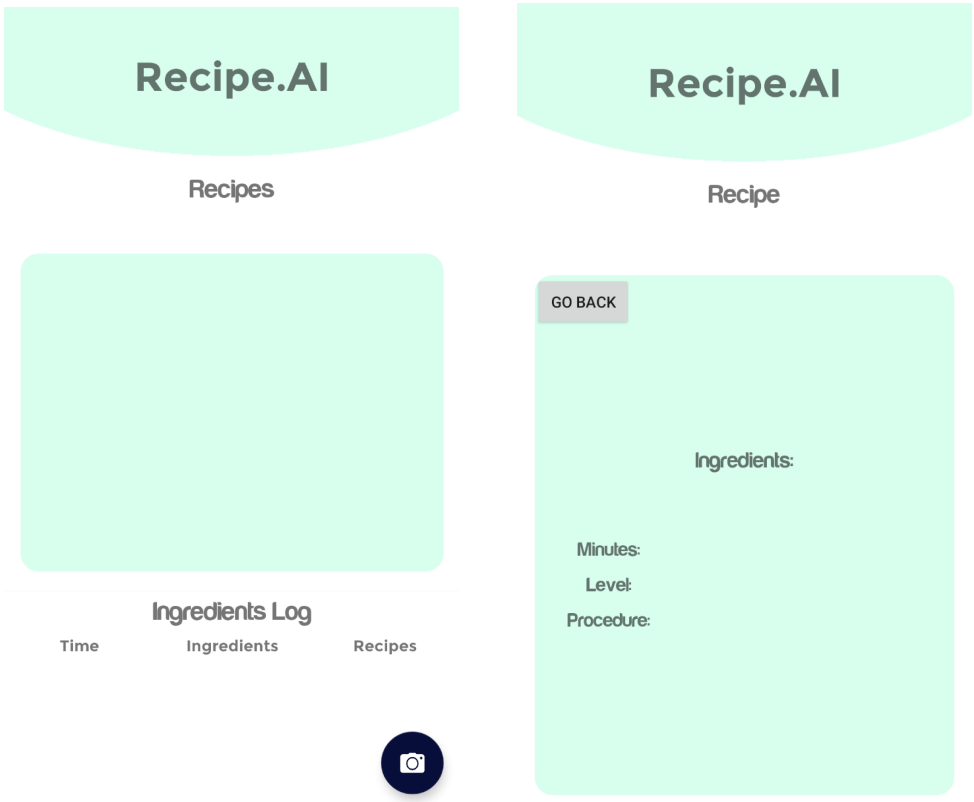


Figura 3.1: activity_main.xml e view_recipe_layout.xml

Capitolo 4

Conclusioni

Dopo aver testato l'applicazione su vari dispositivi Android, inclusi il Google Pixel e il Samsung Galaxy, abbiamo osservato una buona capacità di rilevamento degli ingredienti attraverso la fotocamera. L'app è in grado di distinguere con successo ingredienti molto simili, come la patata e l'uovo, anche quando sono molto vicini tra loro. Inoltre, abbiamo constatato che l'applicazione presenta prestazioni affidabili, mantenendo un utilizzo della CPU contenuto (circa il 20%) e garantendo una velocità computazionale costante e rapida.

4.1 Problemi riscontrati

Una delle prime sfide affrontate è stata la necessità di ottenere un dataset sufficientemente ampio e bilanciato. È stato necessario apportare diverse modifiche per garantire che ci fosse un numero adeguato di immagini per ciascuna classe e che queste fossero coerenti tra loro. La scelta della dimensione delle immagini durante l'addestramento è stata cruciale: un aumento eccessivo avrebbe comportato tempi di addestramento troppo lunghi, mentre una diminuzione avrebbe compromesso l'efficacia del rilevamento. Abbiamo quindi optato per una dimensione media di 320 pixel.
Per accelerare il processo di addestramento, abbiamo sperimentato l'utilizzo dei nuovi Metal Performance Shaders su Macbook. Questo insieme di librerie software, sviluppate da Apple, dovrebbe ottimizzare le prestazioni di calcolo con kernel ottimizzati per le caratteristiche uniche di ciascuna famiglia di GPU Metal. Tuttavia, i risultati non sono stati soddisfacenti, in quanto abbiamo appreso che non sono ancora stati pienamente testati per YOLOv5 sui chip M1 ed M2 di Apple, come confermato anche da diverse fonti nei forum.
Per quanto riguarda lo sviluppo dell'applicazione, una delle principali sfide è stata l'importazione delle giuste librerie per la lettura dei file Excel. È stato fondamentale garantire la compatibilità tra le librerie stesse e il livello di API dell'applicazione.

Inoltre, abbiamo dovuto gestire il passaggio dei dati tra le varie attività dell'app. Per farlo, abbiamo utilizzato il metodo putExtra per inviare dati e creato nuovi Intent. Il problema principale era dovuto al fatto che ogni volta che si passava dalla DetectorActivity alla MainActivity, anziché utilizzare l'istanza più recente di MainActivity già presente nello stack delle attività, ne veniva creata una nuova. Di conseguenza, quando si ritornava alla MainActivity per visualizzare le ricette, si finiva per non vedere le ricette aggiornate. Per risolvere, abbiamo utilizzato il flag intent.setFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT) nella Detector Activity, il quale garantisce che, se un'istanza dell'attività di destinazione è già presente nello stack, venga riportata in primo piano anziché crearne una ulteriore. Inoltre, abbiamo implementato il metodo onNewIntent() nella MainActivity, in modo da fare riferimento alla corretta istanza.

4.2 Sviluppi futuri

Il progetto si presta ad ulteriori sviluppi futuri, come la ricerca di un dataset più ampio e completo che potrebbe migliorare sensibilmente l'accuratezza del modello e l'espansione del numero di classi di ingredienti riconosciuti. Inoltre, è possibile considerare l'aggiunta di video esplicativi e immagini più dettagliate nel display delle ricette per arricchire ulteriormente l'esperienza dell'utente.

Bibliografia

[1] Redazione Osservatori Digital Innovation, *Computer Vision: definizione, funzionamento e applicazioni*, https://blog.osservatori.net/it_it/computer-vision-definizione-applicazioni, (visitato il 08/04/2024)

[2] IBM, *Cos'è la Computer Vision?*, https://www.ibm.com/it-it/topics/computer-vision, (visitato il 08/04/2024)

[3] Yuliia Kniazieva, *What's the Difference Between Image Classification & Object Detection?*, https://labelyourdata.com/articles/object-detection-vs-image-classification, (visitato il 08/04/2024)

[4] Glenn Jocher, *yolov5*, https://github.com/ultralytics/yolov5, (visitato il 20/03/2024)

[5] Ultralytics, *Dati personalizzati per la formazione*, https://docs.ultralytics.com/it/yolov5/tutorials/train_custom_data/, (visitato il 20/03/2024)