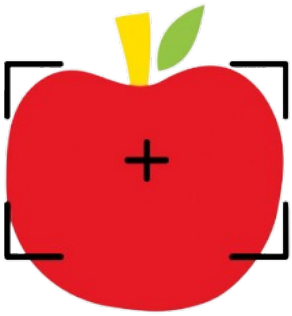
Progetto di Classificazione di Prodotti del Supermercato con MobileNetV2 e Deployment su Android



Andrea Berti

A.A. 2023-2024

Contents

Introduzione

Questo progetto si focalizza sulla classificazione di immagini utilizzando MobileNetV2, una rete neurale convoluzionale leggera progettata per garantire efficienza computazionale su dispositivi mobili. MobileNetV2 introduce innovazioni come gli inverted residuals e i linear bottlenecks, che permettono di mantenere un'alta precisione riducendo al contempo il numero di parametri e le operazioni computazionali. Queste caratteristiche rendono la rete particolarmente adatta per ambienti con risorse limitate, come dispositivi mobili e embedded.

L'obiettivo principale del progetto è lo sviluppo di un'applicazione Android capace di riconoscere prodotti da supermercato e di annunciarne il nome tramite sintesi vocale. Questo strumento è pensato per offrire un concreto supporto alle persone non vedenti, sia durante la spesa che nel riconoscimento dei prodotti una volta acquistati. L'app utilizzerà la fotocamera dello smartphone per identificare gli oggetti e un sistema di sintesi vocale per comunicarne il nome.

In particolare, l'app mira a facilitare la distinzione tra articoli con la stessa forma o confezione ma gusto e caratteristiche differenti. Questo aspetto rappresenta una delle maggiori sfide per le persone non vedenti, poiché, non avendo indicazioni in Braille, prodotti visivamente simili come ad esempio yogurt di diversi gusti o differenti tipologie di Coca-Cola risultano indistinguibili senza l'ausilio di una tecnologia di riconoscimento.

1 Preparazione del Dataset

1.1 Estrazione di Frame dai Video

Il dataset è stato creato estraendo sequenze di frame da un video per ottenere una varietà di immagini rappresentative delle categorie target.
E' da notare che dato il focus del progetto, sono state scelte per appartenere al dataset numerosi gruppi di classi riferite allo stesso prodotto ma di differente gusto o tipologia.

Classe	**Numero di Immagini**
Gocciole Classiche	400
Gocciole Cocco	400
Gocciole Conad	400
Gocciole Dark	400
Latte Azzurro	400
Latte Blu	400
Latte Rosso	400
Mela Gialla	400
Mela Rossa	400
Pink Lady	400
Acqua Naturale	400
Acqua Frizzante	400
Philadelphia Normale	400
Philadelphia Salmone	400
Philadelphia Erbette	400
Philadelphia Conad	400
Pane Bianco	400
Pane Integrale	400
Pane Grano Duro	400
Pane Cereali	400
The Pesca	400
The Limone	400
The Verde	400
Coca Cola Classica	400
Coca Cola Zero	400
Cola Conad	400
Totale Immagini	**10400**
Dimensione del Dataset	**936 MB**

1.2 Distribuzione dataset

- **Training Set:** 6552 immagini (63%)
- **Validation Set:** 2808 immagini (27%)
- **Test Set:** 1040 immagini (10%)

2 Keras

Keras è una libreria di alto livello per il deep learning, che funge da interfaccia front-end per librerie di backend come TensorFlow.
Progettata per essere semplice e intuitiva, Keras mi ha consentito di addestrare un modello di rete neurale senza dovermi preoccupare dei dettagli implementativi di basso livello. Offre infatti moduli per strati, attivazioni, ottimizzatori e funzioni di perdita. Inoltre, facilita il preprocessing dei dati e l'aumento dei dati, e permette l'uso di modelli pre-addestrati per il transfer learning.
La compatibilità con TensorFlow e la vasta documentazione rendono Keras uno strumento versatile e accessibile per il deep learning.

3 Dataset Pipeline

3.1 Caricamento e Preprocessamento del Dataset

Il dataset è stato caricato e le immagini sono state preprocessate attraverso le seguenti operazioni:

- **Ritaglio:** Le immagini sono state ritagliate in modo che siano di dimensioni quadrate, eliminando i bordi in eccesso.
- **Ridimensionamento:** Le immagini sono state ridimensionate a 224x224 pixel.
- **Standardizzazione:** Le immagini sono state normalizzate per avere valori di pixel compresi tra -1 e 1.

3.2 Data Augmentation

È stato applicato un livello di *data augmentation* per aumentare la varietà delle immagini di addestramento e migliorare la generalizzazione del modello. Le tecniche utilizzate includono:

- **Flip orizzontale e verticale**
- **Rotazione casuale**
- **Traslazione casuale**
- **Zoom casuale**

4 Configurazione della GPU

4.1 Gestione della Memoria GPU

Per garantire un uso efficiente delle risorse, TensorFlow è stato configurato per gestire dinamicamente la memoria GPU, evitando allocazioni eccessive che potrebbero causare problemi su GPU con risorse limitate, come il portatile su cui verrà addestrato il modello.

5 Definizione del Modello

5.1 Training per una Rete Neurale

Il processo di addestramento di una rete neurale prevede due fasi principali: *forward pass* e *backpropagation*. Durante il *forward pass*, l'input viene passato attraverso il modello per ottenere una previsione. Questo processo può essere descritto dalla seguente equazione:

$$\hat{y} = f(x; \theta) \tag{1}$$

dove:

- $\hat{y}$ è la previsione del modello,
- x è l'input,
- θ rappresenta i pesi e i bias del modello,
- f è la funzione di attivazione applicata nei vari strati del modello.

Durante la fase di *backpropagation*, i pesi del modello vengono aggiornati per minimizzare la funzione di perdita utilizzando algoritmi di ottimizzazione come l'Adam. La funzione di perdita, nel nostro caso **Categorical Crossentropy**, è definita come:

$$\mathcal{L}(\hat{y}, y) = -\sum_i y_i \log(\hat{y}_i) \tag{2}$$

dove:

- $\mathcal{L}$ è la funzione di perdita,
- y è il valore vero,
- $\hat{y}$ è la previsione del modello.

Il gradiente della funzione di perdita rispetto ai pesi del modello è calcolato tramite la *chain rule*, e il peso viene aggiornato utilizzando:

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta \mathcal{L} \tag{3}$$

dove:

- θ_{new} e θ_{old} sono rispettivamente i pesi aggiornati e quelli precedenti,
- η è il tasso di apprendimento,
- $\nabla_\theta \mathcal{L}$ è il gradiente della funzione di perdita rispetto ai pesi.

L'algoritmo di ottimizzazione **Adam**, ADAptive Moment estimation, aggiorna i pesi utilizzando una combinazione di gradiente e momenti, migliorando l'efficienza dell'addestramento. La formulazione dell'aggiornamento dei pesi con Adam è data da:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{4}$$

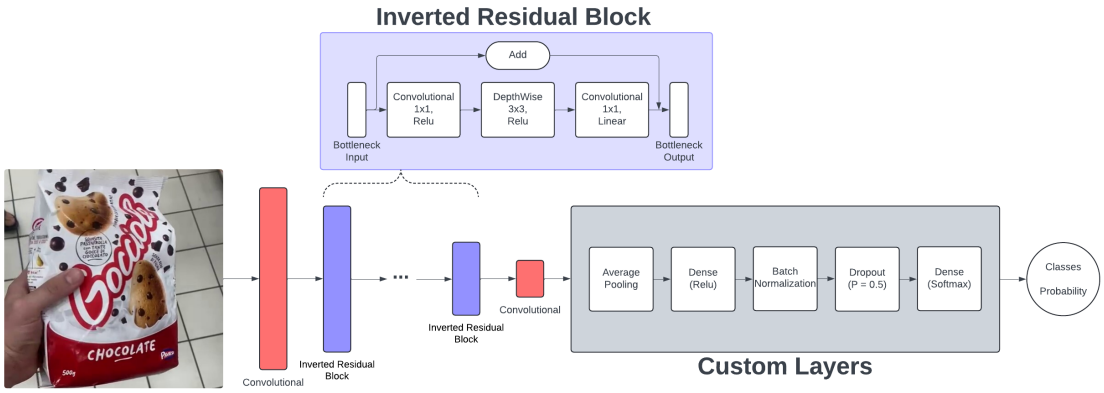
dove:

- $\hat{m}_t$ è la stima del primo momento (media),
- $\hat{v}_t$ è la stima del secondo momento (varianza),
- ϵ è un piccolo valore per evitare la divisione per zero.

5.2 MobileNetV2

MobileNetV2 è un modello di rete neurale progettato per essere efficiente e performante su dispositivi mobili e con risorse limitate. Le sue principali caratteristiche sono:

- **Inverted Residual Blocks:** Questo design usa una sequenza di operazioni per elaborare le immagini in modo più efficiente. Inizia con una convoluzione di espansione 1x1 per aumentare temporaneamente il numero di canali (profondità) dell'immagine, seguita da una convoluzione depthwise (che applica un filtro separato a ciascun canale dell'immagine). Infine, utilizza un'altra convoluzione 1x1 pointwise per ridurre nuovamente il numero di canali. Le connessioni residue (skip connections) aiutano a mantenere le informazioni importanti durante il processo.
- **Depthwise Separable Convolutions:** Questo approccio divide la convoluzione in due passaggi distinti: la convoluzione depthwise, che applica un filtro a ciascun canale dell'immagine separatamente, e la convoluzione pointwise, che utilizza un filtro 1x1 per combinare i risultati e di conseguenza il numero di canali. Questi sono rispettivamente il secondo e il terzo passo di ogni Inverted Residual Blocks. In questo modo vengono ridotti significativamente il numero di parametri e il costo computazionale rispetto alle convoluzioni tradizionali, rendendo il modello più leggero e veloce.
- **Linear Bottleneck Layers:** I blocchi bottleneck lineari sono progettati per essere semplici e diretti. Dopo una serie di operazioni di convoluzione il blocco finale utilizza una convoluzione 1x1 con attivazione lineare (come ReLU), mantenendo la rappresentazione dei dati più pulita e prevenendo la perdita di informazioni, questa tecnica viene utilizzata nel terzo passo di ogni Inverted Residual Block.
- **Residual Connections:** Le connessioni residue, o skip connections, permettono ai dati di saltare alcune fasi del processo di apprendimento e di essere aggiunti direttamente all'output successivo. Questo aiuta a prevenire il problema del vanishing gradient, in cui i gradienti diventano molto piccoli e rallentano l'apprendimento, migliorando così l'efficacia dell'allenamento del modello.
- **ReLU6 Activation Function:** La funzione di attivazione ReLU6 limita l'uscita della funzione di attivazione a un valore massimo di 6. Questo aiuta a controllare e normalizzare i valori di attivazione, il che è particolarmente utile durante la quantizzazione del modello per ridurre la precisione dei calcoli e migliorare l'efficienza sui dispositivi mobili.
- **Efficient Design for Mobile Devices:** MobileNetV2 è progettato per essere altamente efficiente in termini di calcolo e memoria. Utilizza tecniche come le convoluzioni separabili e i blocchi invertiti per ridurre il numero di operazioni e di parametri, rendendo il modello adatto per l'esecuzione su dispositivi con risorse limitate, come smartphone e tablet.



6 Addestramento del Modello

6.1 Transfer Learning

Il Transfer Learning è una tecnica che consiste nell'utilizzare reti preallenate su altri dataset, sostituendo l'ultimo livello fully connected con uno nuovo da ri-allenare. Esistono due tecniche di transfer learning:

- **Fixed feature extraction:** i pesi della rete vengono congelati, a eccezione dell'ultimo livello fully connected (classificatore) o degli ultimi layer personalizzati.
- **Fine tuning:** viene allenata sia la rete, o parte di essa, sia il feature extractor.

In particolare, ho effettuato prima una Fixed Feature Extraction allenando solo gli ultimi layer specifici per la classificazione e successivamente un Fine Tuning della Rete sbloccando alcuni degli ultimi layer.

6.2 Download Pesi del Modello Pre-addestrato

Caricare un modello pre-addestrato su ImageNet fornisce un punto di partenza solido, infatti questa rete ha già appreso rappresentazioni generali utili per la classificazione delle immagini, banalmente che i pixel al centro sono più importanti poichè vi si trova più frequentemente il soggetto, riducendo il tempo di addestramento e migliorando le prestazioni.

6.3 Costruzione e Compilazione del Modello

Una volta importato il modello viene ricostruito l'ultimo blocco personalizzato in questo modo:

- **Global Average Pooling Layer:** Sostituisce i fully connected layer tradizionali riducendo la dimensione spaziale del tensore a un singolo valore per feature map, sintetizzando l'informazione globale e riducendo il rischio di overfitting. Migliora la robustezza del modello riducendo il numero di parametri.

- **Fully Connected Layer:** Un layer denso che apprende dai dati. Ogni neurone è connesso a tutti i neuroni del layer precedente.
- **Batch Normalization:** Normalizza l'output di ogni batch per accelerare la convergenza, ridurre la dipendenza dall'inizializzazione dei pesi e mitigare il problema del vanishing/exploding gradient.
- **Dropout Layer:** Disabilita casualmente una frazione dei neuroni durante l'addestramento, prevenendo l'overfitting e migliorando la generalizzazione del modello.
- **Dense Layer con Softmax:** Layer di output con attivazione softmax per la classificazione multi-classe.

6.4 Scelta Optimizer, Loss Function e Metrica di Valutazione

La compilazione del modello include la scelta della funzione di perdita, dell'ottimizzatore e delle metriche di valutazione. In questo caso:

- **Categorical Crossentropy** come Loss Function. Questa funzione di perdita è comunemente utilizzata per problemi di classificazione multiclasse.

  L'uso di Categorical Crossentropy consente di penalizzare pesantemente le previsioni errate, poiché il logaritmo della probabilità predetta tende a ridurre il valore della funzione di perdita se la probabilità è bassa. Questa penalizzazione incoraggia il modello a migliorare la sua accuratezza di previsione per le classi più importanti, aiutando il modello a convergere verso una soluzione ottimale.
- **Adam** come Ottimizzatore, 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 ottimizzatore 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.
- **Accuracy Value** come metrica di valutazione.

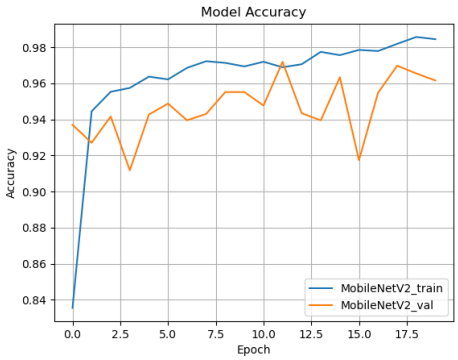
6.5 Scelta Callbacks per l'addestramento

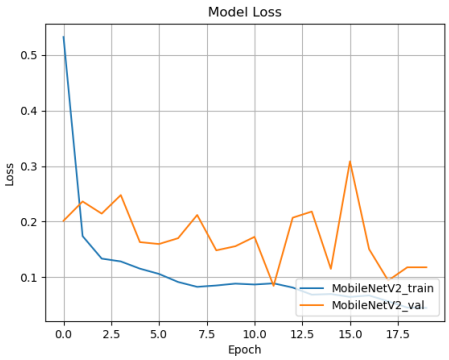
Il processo di addestramento prevede l'uso di callback per ottimizzare il training e prevenire l'overfitting. Nel progetto ne vengono utilizzate due:

- **EarlyStopping:** Monitora una metrica, in questo caso la *validation loss* e interrompe l'addestramento se non viene osservato un miglioramento della metrica per un numero prestabilito di epoche (patience). Questo aiuta a prevenire l'overfitting interrompendo l'addestramento quando il modello smette di migliorare.
- **ReduceLROnPlateau:** Riduce dinamicamente il tasso di apprendimento quando la metrica monitorata, in questo caso la *validation loss*, smette di migliorare. La riduzione del tasso di apprendimento avviene se la metrica rimane stagnante per un certo numero di epoche. Questo aiuta a evitare che il modello rimanga bloccato in un ottimo locale e migliora la convergenza.

6.6 Fixed Feature Extraction

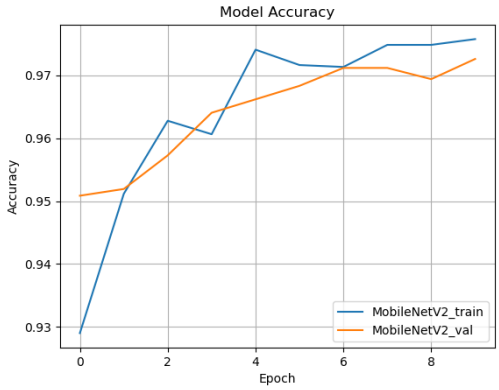
Congelare i pesi dei modelli pre-addestrati e addestrare solo i nuovi layer personalizzati permette al modello di adattarsi rapidamente al nuovo dataset, preservando le conoscenze generali acquisite. Qui di seguito vengono mostrati i grafici rispettivamente dell'*accuracy value* e della *loss value* dopo i primi 20 addestramenti utilizzando questa metodologia.

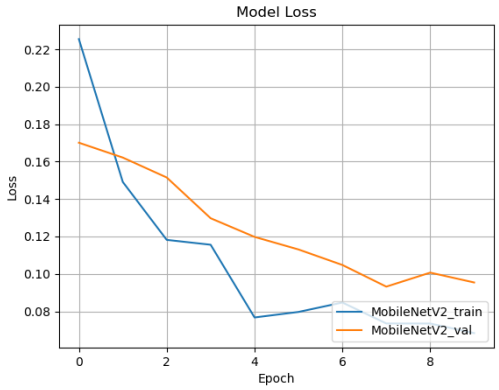




6.7 Fine Tuning

Il fine-tuning della rete viene fatto sbloccando e aggiornando alcuni degli ultimi blocchi di layer pre-addestrati.
Qui di seguito vengono mostrati i grafici rispettivamente dell'*accuracy value* e della *loss value* dopo aver "scongelato" i pesi degli ultimi blocchi e aver riallenato la rete per ulteriori 10 epoche.
E' da notare come si possa osservare una diminuzione dell'accuratezza sul dataset di addestramento rispetto al precendente training, mentre si registra un miglioramento sul dataset di validazione.
Questo fenomeno indica che, sebbene il modello possa perdere parte della sua performance sul dataset di addestramento, sta migliorando la sua capacità di generalizzazione e adattamento ai dati non visti.





7 Valutazione del Modello

7.1 Risultati Addestramento

MobileNetV2	Train Loss	Train Acc.	Val. Loss	Val. Acc.	Epoche	Tempo
Fixed Feature Extraction	0.0434	0.9845	0.1175	0.9615	20	57 min
Fine Tuning	0.0738	0.9743	0.0954	0.9726	10	38 min

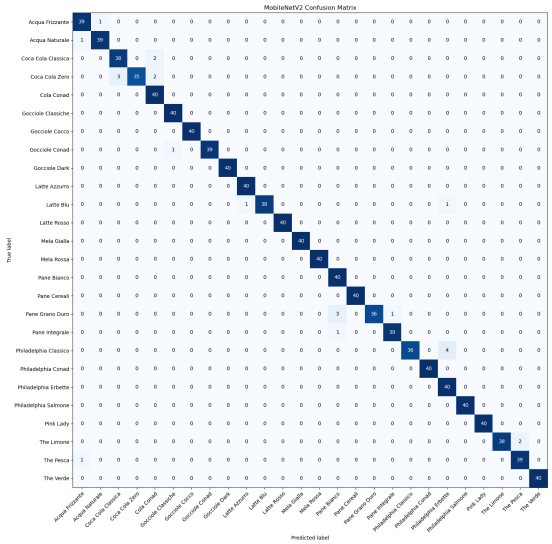
7.2 Metriche di Prestazione

Queste metriche, ottenute utilizzando il dataset di test, includono:

- **Accuratezza:** Percentuale di classificazioni corrette sul dataset di test.
  Il modello ha raggiunto un'accuratezza del **0.9769%**.
- **Precisione, Recall, F1-Score:** Forniscono una visione più dettagliata delle prestazioni del modello su ciascuna classe, calcolate sul dataset di test. Qui si riportano solo le medie:
  - **Media Precisione: 98%**
  - **Media Recall: 98%**
  - **Media F1-Score: 98%**

7.3 Matrice di Confusione

La matrice di confusione aiuta a identificare errori specifici di classificazione, anche questa è stata ottenuta utilizzando il dataset di test.



7.4 Grad-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM) è una tecnica usata per capire quali parti di un'immagine influenzano maggiormente la decisione di una rete neurale.
Questa tecnica genera una mappa di calore che evidenzia le aree più importanti dell'immagine per la predizione del modello. Funzionamento:

- **Predizione e selezione della classe target:** Dato un modello e un'immagine in input, otteniamo le predizioni predictions, che rappresentano le probabilità per ciascuna classe. La classe target c viene selezionata come:

$$c = \arg\max(\text{predictions})$$

- **Calcolo dei gradienti per i layer depthwise:** Per ciascun layer depthwise k, calcoliamo i gradienti della predizione della classe target y^c rispetto alle attivazioni del layer A^k. I gradienti si calcolano come:

$$\frac{\partial y^c}{\partial A^k_{ij}}$$

  dove: $\frac{\partial y^c}{\partial A^k_{ij}}$ è il gradiente della probabilità della classe target rispetto all'attivazione A^k_{ij} al pixel (i, j) nel layer depthwise k.

- **Ponderazione delle attivazioni per ogni layer:** I gradienti calcolati vengono mediati per ottenere un peso α^c_k, che rappresenta l'importanza complessiva di ciascun canale del layer depthwise. Il peso α^c_k si calcola come:

$$\alpha^c_k = \frac{1}{Z} \sum_{i=1}^{H} \sum_{j=1}^{W} \frac{\partial y^c}{\partial A^k_{ij}}$$

  dove:

  - H è l'altezza delle feature map,
  - W è la larghezza delle feature map,
  - $Z = H \times W$ è il numero totale di elementi spaziali nelle feature map A^k.

- **Generazione della mappa di attivazione (heatmap) per l'intero layer:** Utilizzando i pesi α^c_k, calcoliamo la mappa di attivazione combinata per il layer k. La mappa di attivazione $L^c_{\text{Grad-CAM}}$ è data dalla somma ponderata delle attivazioni del layer depthwise:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k\right)$$

  dove:

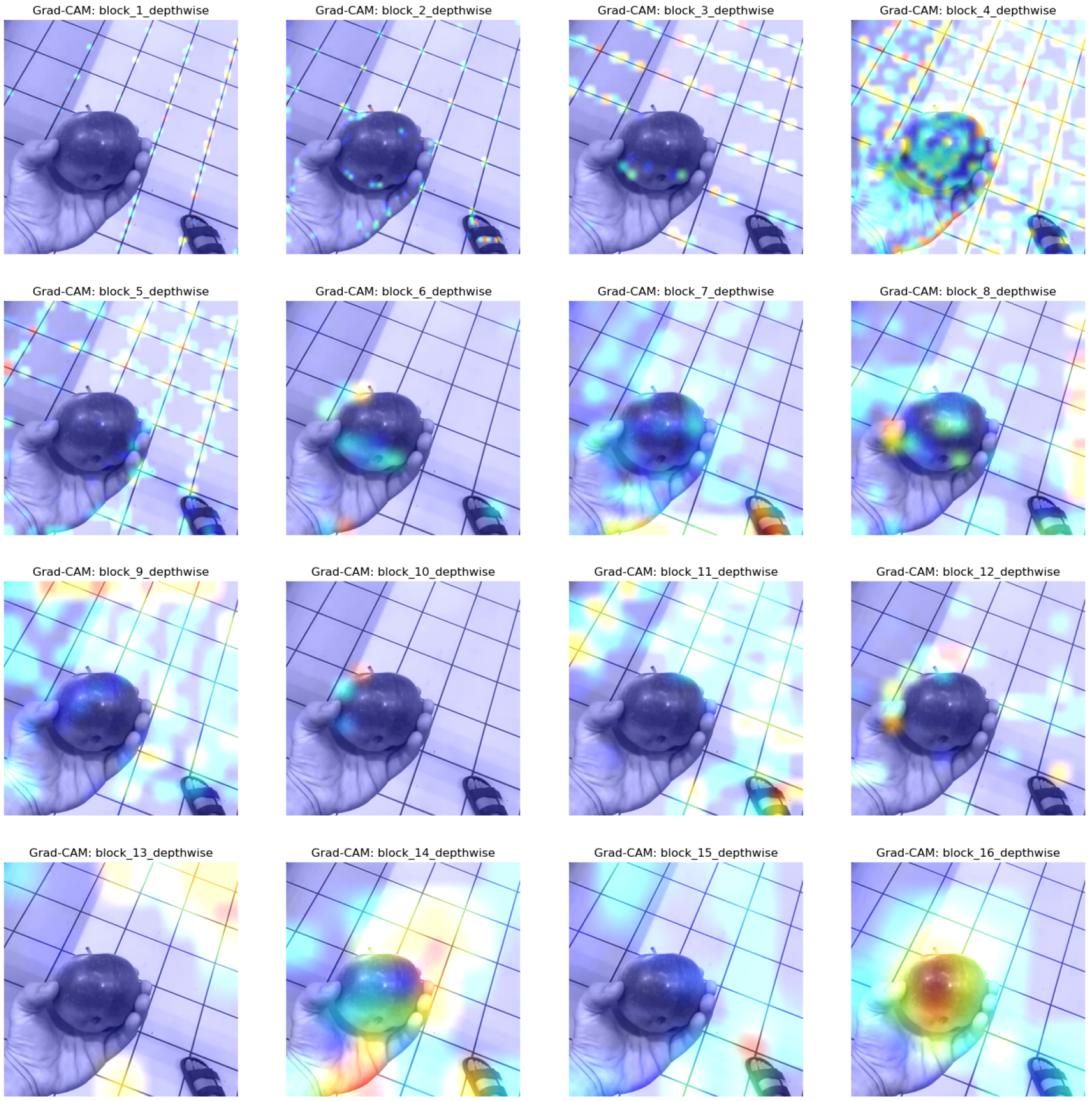
  - $\alpha^c_k A^k$ rappresenta il prodotto tra i pesi α^c_k e le attivazioni A^k del feature map k,

– $\text{ReLU}(x) = \max(0, x)$ è la funzione di attivazione ReLU che conserva solo i valori positivi.

- **Normalizzazione e visualizzazione:** Ogni mappa di attivazione $L^k_{\text{Grad-CAM}}$ viene normalizzata in modo che i valori siano compresi tra 0 e 1:

$$L^k_{\text{Grad-CAM}} = \frac{L^k_{\text{Grad-CAM}} - \min(L^k_{\text{Grad-CAM}})}{\max(L^k_{\text{Grad-CAM}}) - \min(L^k_{\text{Grad-CAM}})}$$

Infine, ogni mappa di attivazione viene sovrapposta all'immagine originale per visualizzare le aree di interesse specifiche per ciascun layer depthwise.



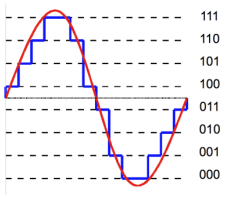
8 Conversione del Modello

8.1 Conversione in TensorFlow Lite

Il modello addestrato è stato convertito in TensorFlow Lite per l'implementazione su dispositivi mobili. La conversione è stata ottimizzata per ridurre la dimensione del modello mantenendo alte prestazioni.
Durante il processo di conversione, possono essere applicati ulteriori metodi di ottimizzazione, come la quantizzazione dei pesi, per rendere il modello ancora più leggero e adatto per ambienti con risorse limitate.

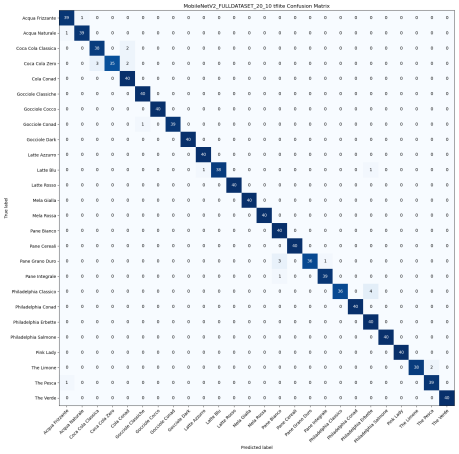
8.2 Quantizzazione

La quantizzazione riduce la precisione dei pesi e delle attivazioni del modello da floating-point a integer.
Questo approccio riduce la dimensione del modello e accelera i tempi di inferenza, e risulta particolarmente utile nei i dispositivi con risorse limitate.
In Keras, esistono diversi tipi di quantizzazione, tra cui: **Quantizzazione post-training**, **Quantizzazione dinamica** e **Quantizzazione full integer**.

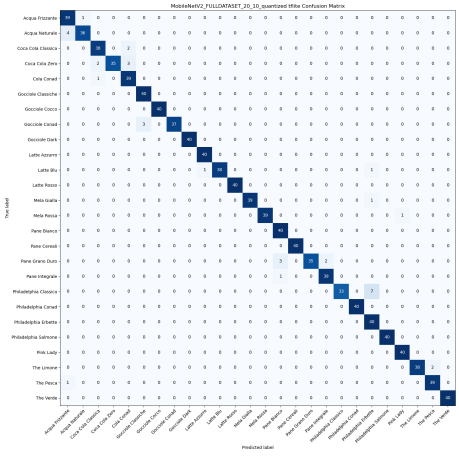


8.3 Test del Modello Lite

Il modello convertito è stato valutato nuovamente sul dataset di test mediante una matrice di confusione. Queste analisi hanno confermato che il modello mantiene un buon livello di accuratezza, riducendo significativamente i requisiti computazionali.
Sia il modello .tflite standard che la sua versione quantizzata hanno mostrato ottime performance in termini di precisione, rendendoli adatti per potenziali applicazioni in scenari real-world.



Lite Model Confusion Matrix



Quantized Lite Model Confusion Matrix

9 Deployment su Android Studio

9.1 Stack Android

L'implementazione in Android Studio integra TensorFlow Lite per la classificazione delle immagini in tempo reale. Lo stack include librerie per il machine learning e la gestione dell'interfaccia utente.

9.2 CameraX

CameraX è una libreria Android per la gestione del flusso video e l'acquisizione di immagini. È stata utilizzata per gestire l'input continuo della fotocamera posteriore, assicurando efficienza e semplicità d'integrazione con TensorFlow Lite.

9.3 Kotlin

L'applicazione è stata sviluppata utilizzando Kotlin, un linguaggio che garantisce una maggiore concisione rispetto a Java e una piena interoperabilità con le librerie Android.

9.4 Gestione delle Gesture

Le gesture permettono un'interazione intuitiva con l'applicazione:

- **Click:** Avvia il processo di inferenza sul frame catturato da CameraX. Il risultato viene comunicato tramite la sintesi vocale Text-To-Speech. Un ulteriore click permette di riavviare il processo di inferenza.
- **Swipe Destro:** Avvia la riproduzione vocale di una breve descrizione associata alla classe identificata.
- **Swipe Sinistro:** Avvia la riproduzione vocale del peso o dimensione relativa all'oggetto identificato.

9.5 Integrazione con Android Studio

L'integrazione del modello TensorFlow Lite nell'applicazione Android è stata realizzata seguendo i seguenti passaggi:

- **Importazione del Modello:** Il file `.tflite` è stato importato nella directory `assets`.
- **Configurazione delle Dipendenze:** Nel file `build.gradle` sono state inserite le dipendenze per TensorFlow Lite e CameraX.
- **Interfaccia Utente:** L'UI è stata progettata per visualizzare i risultati della classificazione e permettere un'interazione semplice tramite gesture.
- **Inferenza e Preprocessamento:** Le immagini catturate sono preprocessate (ridimensionamento e normalizzazione) prima di essere passate al modello. L'inferenza viene effettuata utilizzando la API TensorFlow Lite Interpreter, e i risultati mostrati all'utente tramite la sintesi vocale Text-To-Speech.

10 Conclusioni

Il progetto ha dimostrato come l'approccio di transfer learning con MobileNetV2 sia efficace per la classificazione di immagini su dispositivi mobili. L'integrazione di TensorFlow Lite in Android Studio ha poi permesso lo sviluppo di un'applicazione capace di eseguire inferenze rapide e accurate.

L'applicazione ha sfruttato l'uso delle Gesture e del Text-To-Speech per garantire un'interazione fluida, intuitiva e accessibile. Questo lavoro conferma come le tecnologie di machine learning possano essere adottate in contesti reali, aprendo la strada a soluzioni pratiche in ambito mobile, anche in condizioni di risorse computazionali limitate.

In sintesi, il progetto ha validato l'efficacia di MobileNetV2 e del transfer learning su piattaforme mobili, dimostrando l'impatto positivo delle reti neurali convoluzionali integrate in applicazioni quotidiane.

Repository

`https://github.com/bertiandrea/ProgettoSistemiDigitali`

Riferimenti

- TensorFlow Documentation. (2023).
  `https://www.tensorflow.org/guide`
- EfficientNetV2: Rethinking Model Scaling for Convolutional Neural Networks.
  `https://arxiv.org/abs/1801.04381`
- MobileNetV2: Inverted Residuals and Linear Bottlenecks.
  `https://arxiv.org/abs/1610.02391`
- Android Camera Samples.
  `https://github.com/android/camera-samples`
- Image Classification with Deep Neural Networks for Groceries.
  `https://github.com/nkanu17/image-classification-deep-neural-networks-groceries/tree/master`
- Grocery Store Dataset.
  `https://github.com/marcusklasson/GroceryStoreDataset`
- Keras Applications: Pre-trained Models.
  `https://keras.io/api/applications/`
- Image classification via fine-tuning with EfficientNet.
  `https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/`
- Tutorial - Transfer Learning.
  `https://www.tensorflow.org/tutorials/images/transfer_learning?hl=it`