Sneakers Classifier

Progetto Sistemi Digitali M – 2021/2022

Sviluppato: Cristaudo Giuseppe & Santandrea Pietro

Indice

1. Abstract

Da quando la rivoluzione industriale rese la gomma facilmente reperibile e la produzione di scarpe da ginnastica economica ed efficiente, le sneakers (tradotto scarpe da ginnastica) divennero sempre più popolari. In particolare, grazie all'introduzione di nuove tecnologie performanti per gli atleti, le sneakers cominciarono a essere approvate da celebrità dello sport, come Michael Jordan, creando un pretesto per la produzione di sneakers a tiratura limitata. Questo segnò la cultura della moda e le scarpe da ginnastica cominciarono a essere considerate speciali.

Fig 1.1 The Air Jordan I and Adidas Superstar on display.Image via Jonathan Dorado

Così nacque una prima cultura di nicchia delle sneakers basata non più solo nell'indossarle ma anche nel collezionarle. Anche senza includere questa piccola nicchia, le sneakers sono di gran lunga il modello al momento più popolare tra le calzature. Ci sono centinaia, se non migliaia, di modelli facilmente acquistabili, ricche di stile ma soprattutto sempre disponibili.
Tuttavia, negli ultimi anni, l'attenzione mediatica si è spostata verso tutte quelle sneakers che le maggiori imprese del settore producono in tiratura limitata, andando a creare un vero senso di rarità dietro i propri modelli.
A partire da queste solide basi, l'applicazione Sneakers Classifier si pone come obiettivo quello di andare incontro alla mole sempre maggiore di

consumatori in cerca del modello dei propri sogni e lo fa da un'immagine presa dal web o più semplicemente da una foto scattata in qualsiasi momento. Tutto questo è disponibile dalla propria tasca grazie allo sviluppo di un'applicazione per dispositivi Android, attualmente il 72,19% del mercato complessivo mondiale dei dispositivi mobile (https://www.affde.com/it/android-market-share.html).

2. Dataset

La realizzazione di un buon classificatore di immagini parte dalla selezione del giusto dataset. Sfortunatamente, al momento non sono disponibili (pubblicamente) dataset per il nostro campo di studio e per questo motivo abbiamo dovuto costruirne uno *ad hoc*. Il primo passo è stato quello di capire come e soprattutto dove reperire un numero sufficiente di immagini per raggiungere il nostro scopo. Per rispondere alla prima domanda, dopo una breve ricerca, siamo convogliati all'utilizzo di una estensione disponibile gratuitamente su chrome (https://download-all-images.mobilefirst.me), la quale permette di scaricare tutte le immagini presenti in una pagina web in un unico archivio.
Dopodiché, abbiamo cominciato a scaricare le immagini da Google Images usando le seguenti query per ottenere il maggior numero di risultati in diversi scenari per ogni modello:

- **modello** -ps -gs -td.
- **modello** for sale -ps -gs -td.
- **modello** on feet -ps -gs -td.
- **modello** outfit -ps -gs -td.

Utilizzando l'operatore "- " siamo riusciti ad effettuare una prima scrematura dei risultati in modo da eliminare determinate keywords (riferite ai modelli da bambini) dalla ricerca. Tuttavia, alla fine di questa operazione il numero di immagini di partenza non era sufficiente per garantire risultati ottimali. Per questo motivo, abbiamo ampliato il nostro dataset andando a prelevare nuovi campioni da siti di riferimento del mondo sneakers come StockX (https://stockx.com/) e GOAT (https://www.goat.com/), in particolare nella sua sezione styles (https://www.goat.com/styles - attualmente non disponibile). Infine, il seguente script Python da noi sviluppato, dato un file

di input contente tutti gli identificativi (SKU) delle sneakers attualmente in commercio, ha permesso di scaricare dal sito restocks (https://restocks.net/it) l'immagine della scarpa di lato.

```
for SKU in SKUs:
    url = 'https://restock.gg/api/products/search?query=' + SKU
    r = s.get(url)
    try:
        r_json = json.loads(r.text)
        if r_json['error'] == 0:
            restocks = r_json['response'][0]['product_data'][0]
            name = restocks['Name'].lower()
            image = restocks['Image']
            file_name = image.split('/')[-1]
```

Fig 2.1 Python script che permette di ottenere URL di immagini di restocks.net dato un array di stringhe (SKU)

```
def download_image(folder_name, name, url):
    # Open the url image, set stream to True, this will return the stream content.
    r = requests.get(url, stream = True)

    # Check if the image was retrieved successfully
    if r.status_code == 200:
        # Set decode_content value to True, otherwise the downloaded image file's size will be zero.
        r.raw.decode_content = True

    # Open a local file with wb ( write binary ) permission.
    filename = folder_name + '/' + name + '.jpg'
    with open(filename,'wb') as f:
        shutil.copyfileobj(r.raw, f)
```

Fig 2.2 Funzione Python che scarica un'immagine presente in un determinato URL

A questo punto, ci siamo limitati a campionare gli otto modelli di sneakers più influenti al momento al fine di garantire una classificazione migliore preferendo la qualità rispetto alla quantità. Il dataset era così costituito da otto classi, ognuna con circa mille immagini di riferimento. Tuttavia, dopo una pulizia manuale delle immagini che non rientravano nelle classi e di una pulizia tramite un apposito script Python (https://stackoverflow.com/questions/65438156/tensorflow-keras-error-unknown-image-file-format-one-of-jpeg-png-gif-bmp-re) al fine di avere solo immagini nel formato corretto per essere utilizzate da Tensorflow, ogni classe era costituita da circa ottocento immagini. Tra queste, abbiamo dato maggiore impronta ai modelli più richiesti in modo da garantire una classificazione migliore per quest'ultimi.
La versione finale del dataset utilizzato è consultabile al seguente link:

https://drive.google.com/drive/folders/1KE-AkUwC0Xpou8H7YYqzHbq-D6RzdyKI?usp=sharing

3. Implementazione della rete neurale

La rete neurale per il riconoscimento delle sneakers è stata sviluppata utilizzando le API ad alto livello messe a disposizione da Keras, il quale si appoggia sul framework Tensorflow. Per la scrittura e l'esecuzione del codice sono stati usati Jupyter Notebook e Google Colab per poter usufruire della loro flessibilità e semplicità d'uso nell'eseguire un numero elevato di prove su diverse tipologie di reti neurali.

3.1. Rete custom

Un primo approccio è stato quello di utilizzare una rete neurale custom, con un numero ridotto di livelli per constatare un livello base da cui partire e migliorarne i risultati.

3.2. ResNet

Dopo aver suddiviso il dataset in 80% per l'addestramento e 20% per i test tramite API Keras, abbiamo valutato l'utilizzo di una ResNet.

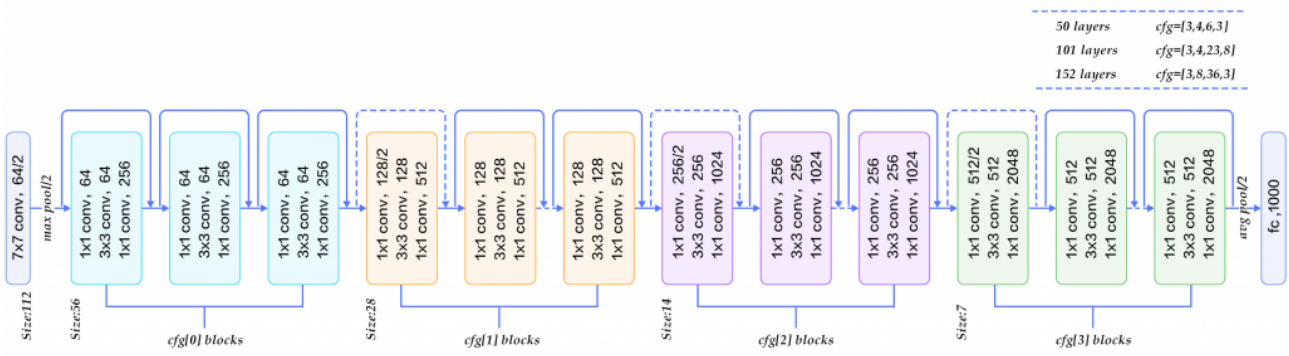


Fig 3.1 Struttura di una rete neurale basata su ResNet

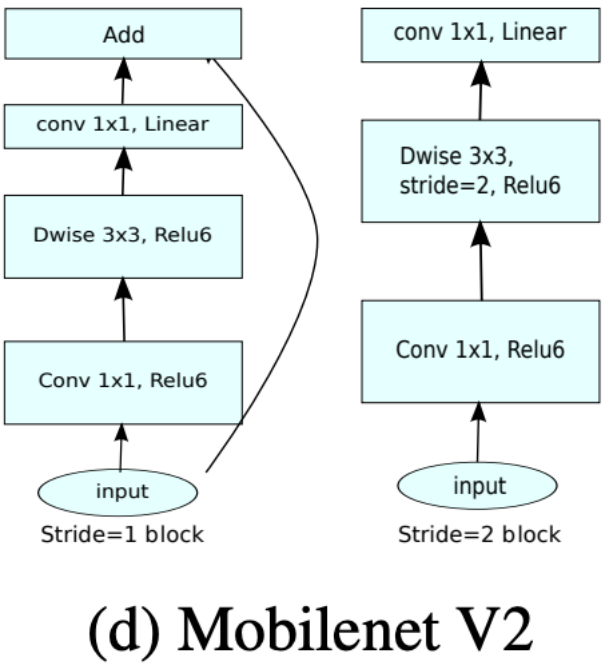
La scelta nasce dal fatto che una ResNet è una rete molto profonda, composta da 152 livelli che facendo uso delle skip connection risulta molto adatta per task legate alla computer vision. Successivamente le immagini

sono state processate, ridimensionandole a 125 x 125 pixel, poiché le ReseNet utilizzate sono state pre-addestrate con tali specifiche.
Partendo ad allenare la rete con 10 epoche per verificare il corretto apprendimento delle classi, abbiamo poi rifinito i livelli fino ad arrivare a 50 epoche ottenendo i seguenti risultati:

- ResNet18 – **93%** accuracy.
- ResNet50 – **96%** accuracy.
- ResNet101 – **97%** accuracy.

Come si può notare dai risultati, le ResNet forniscono un'elevata accuracy durante l'addestramento anche se durante la verifica sul dataset per il test la performance si abbassa ma comunque rimane accettabile per i nostri obiettivi. Tuttavia, una volta salvati i pesi relativi all'addestramento, il file risultante aveva un peso elevato di circa 100MB. Questo poteva far sorgere problemi durante il deploy del modello su un dispositivo con ridotte capacità di memoria o di calcolo e per questo motivo abbiamo valutato l'utilizzo di una MobileNet.

3.3. MobileNet



MobileNetV2 è un'architettura di rete neurale convoluzionale ottimizzata per dispositivi mobili. Si basa su una *inverted residual structure* in cui le connessioni residue si trovano tra gli strati del collo di bottiglia. Nel complesso, l'architettura di MobileNetV2 contiene il livello iniziale di convoluzione completo con 32 filtri, seguito da 19 livelli residui di colli di bottiglia.

Fig 3.2 Struttura di una rete neurale basata su MobileNetV2

Come è stato fatto per la ResNet, anche in questo caso il dataset è stato suddiviso in 80% dedicato all'addestramento e 20% ai test. Inoltre, poiché il dataset di partenza era relativamente piccolo è stato arricchito tramite tecniche di *data augmentation*: tecnica usata per creare versioni modificate delle immagini del dataset aumentando la capacità della rete di generalizzare anche in situazioni diverse e realistiche. Nel nostro caso abbiamo utilizzato operazioni di rotazione e ribaltamento in modo da permettere alla rete di essere ancora più flessibile per le immagini che verranno scattate direttamente dalla fotocamera, le quali potranno presentarsi in diverse angolazioni. Abbiamo dunque utilizzato pesi predefiniti ImageNet, aggiungendo due layer fully-connected al termine della rete pre-addestrata per elaborare l'output della rete stessa e garantire una input size conforme ai modelli selezionati. In questo modo abbiamo utilizzato MobileNetV2 essenzialmente come feature extractor sostituendo le classi riconoscibili da tale modello con quelle da noi designate.
Anche in questo caso abbiamo addestrato la rete su 50 epoche ottenendo risultati soddisfacenti anche se meno accurati (**91%** accuracy) rispetto alla ResNet.

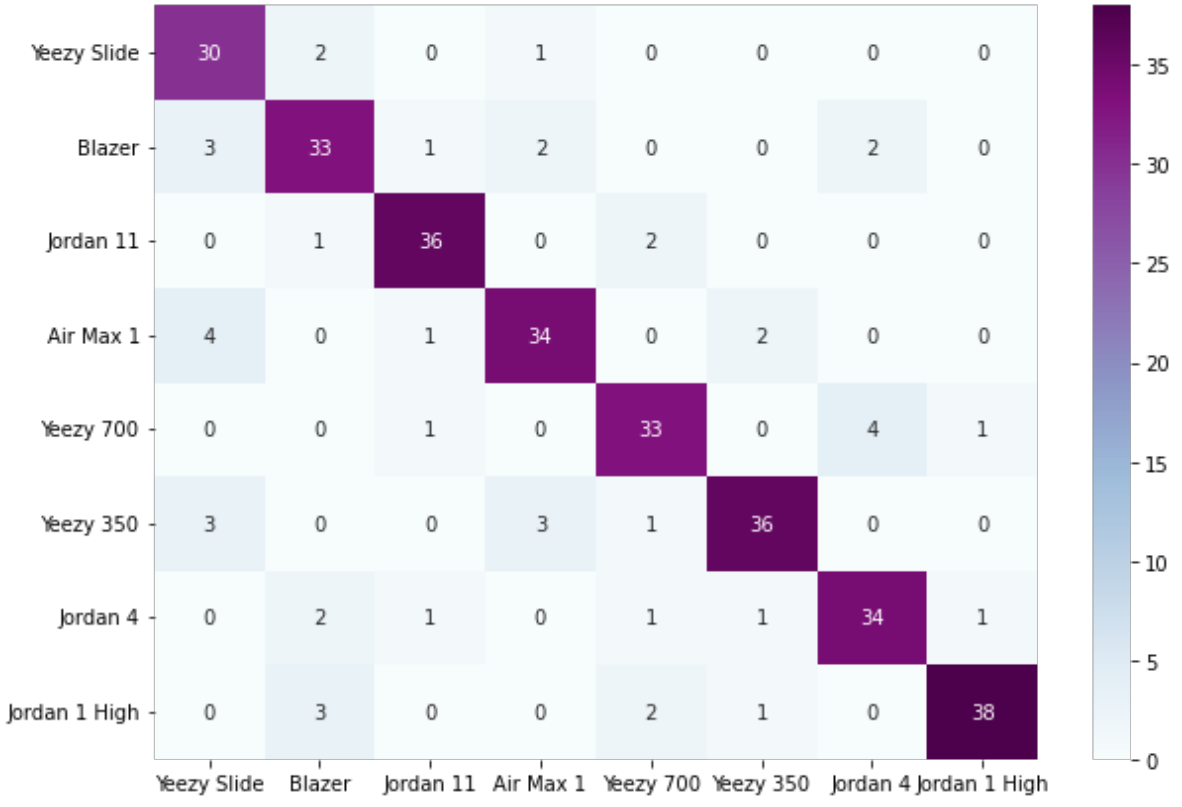


Fig 3.3 Confusion matrix ottenuta testando la rete addestrata con circa 40 test images per ogni modello

Utilizzando le ottimizzazioni fornite dalla MobileNetV2 abbiamo ottenuto un modello leggero. Quest'ultimo è stato successivamente quantizzato post-training utilizzando la configurazione di default di Keras, la quale quantizza i pesi da floating point a integer 8 bit rendendo la rete della dimensione di qualche megabyte sacrificando in piccola parte la precisione della rete.

4. Deploy su dispositivo Android

Con il modello ottimizzato per dispositivi mobile siamo passati al progettare l'applicazione Android. L'implementazione è stata fatta utilizzando il linguaggio Java e l'ambiente di sviluppo Android Studio. L'applicazione è stata sviluppata per funzionare su dispositivi che adottano un livello di API Android uguale o maggiore al livello 23, il quale corrisponde ad una versione Android Marshmellow o successiva.

4.1. Interfaccia grafica

Sneakers Classifier mostra una schermata principale molto semplice dove si distinguono:

- una *ImageView* utilizzata per caricare immagini dalla galleria o mostrare immagini catturate tramite la fotocamera.

- un *pulsante* "TAKE A PICTURE" per aprire la fotocamera del proprio dispositivo.

- un *pulsante* "CLASSIFY" per produrre il risultato della rete.

- una *TextView* in cui vengono mostrati i risultati prodotti dalla rete e un link diretto da cui poter acquistare la sneakers.

Fig 4.1 *Schermata principale Sneakers Classifier*

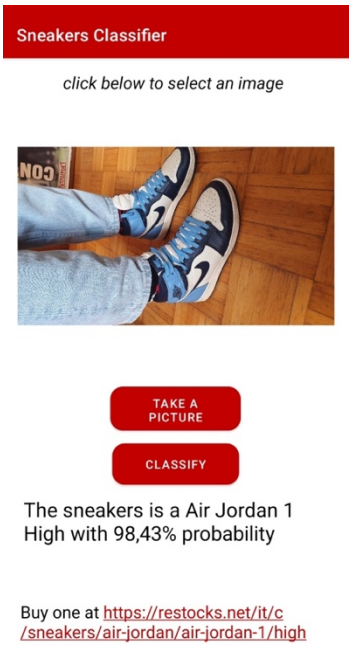
Nel caso in cui si clicchi sul pulsante "CLASSIFY" senza aver inserito un'immagine, l'applicazione restituirà un warning all'utente in cui invita a selezionare un'immagine. Questo è stato possibile utilizzando le funzionalità messe a disposizione dalla classe android.widget.Toast.

Fig 4.2 *Warning mostrato dall'applicazione quando viene premuto il pulsante classify senza aver inserito un'immagine*

4.2. Struttura e funzionamento

L'applicazione è strutturata su due classi:

- **MainActivity**: carica le risorse, verifica i permessi e mostra all'utente l'interfaccia precedentemente descritta.
- **ImageClassifier**: si occupa di convertire l'immagine caricata nella ImageView in un formato supportato da Tensorflow e produce una mappa di valori corrispondenti alla classe e la relativa percentuale di affinità.



Il calcolo riguardante il riconoscimento dell'immagine in questione è eseguito al momento del caricamento dell'immagine nella ImageView, questo per evitare eventuali attese all'utente per la generazione del risultato che sarà riportato nella TextView quando viene premuto il pulsante "CLASSIFY".

Fig 4.3 *Risultato finale mostrato dall'applicazione*

Abbiamo infine testato l'applicazione per verificarne il corretto funzionamento. Utilizzando il caricamento dell'immagine dalla memoria abbiamo osservato una precisione nel riconoscimento non degradata rispetto alla versione non quantizzata. Mentre per quanto riguarda l'inferenza mediante un'immagine scattata sul momento dal device, l'accuratezza dipende notevolmente da come viene eseguito lo scatto e dal sensore presente sul dispositivo mobile.

5. Conclusioni

In questo progetto è stata definita e sviluppata una rete neurale, utilizzando il modello MobileNetV2, per il riconoscimento di immagini raffiguranti diversi modelli di sneakers.
Nonostante le difficoltà riscontrate nella costruzione del dataset e il numero ridotto di immagini di riferimento recuperate, la rete ha mostrato risultati soddisfacenti anche per immagini catturate direttamente dalla fotocamera dello smartphone.
Il lavoro svolto mostra il potenziale delle reti neurali nel riconoscimento delle immagini in un ambito non ancora trattato (visto la mancanza di dataset pubblici) come quello delle sneakers. Questo panorama inesplorato potrebbe portare a forti guadagni economici essendo il mondo delle sneakers sempre più in espansione e carico di capitali.

5.1. Sviluppi Futuri

Infine, il progetto si predispone per eventuali sviluppi futuri come l'integrazione di un numero sempre maggiore di modelli oppure alle funzionalità di object tracking il quali permettono di tracciare la posizione di un oggetto all'interno di una scena, nel nostro caso di una sneakers.