Attività progettuale di Sistemi Digitali

Karina Chichifoi - Michele Righi

Laurea Magistrale in Ingegneria Informatica - Università di Bologna

A.A. 2021-2022

Abstract

Per il progetto d'esame di sistemi digitali è stata utilizzata una rete convoluzionale in grado di riconoscere immagini dei primi 151 Pokémon. In questa attività progettuale confrontiamo ResNet152, MobileNetV2 e MobileNetV3 per il riconoscimento di immagini, allenati con un dataset da 11945 elementi.

1 Definizione del problema e dataset

Nel progetto Poké-Pi-Dex abbiamo realizzato un applicazione su Raspberry OS 32 bit in grado di riconoscere immagini tratte dal cartone animato, peluche, carte, modelli 3D e pixel art.

Avevamo a nostra disposizione un dataset da 7000 immagini trovato su Kaggle e abbiamo utilizzato una rete neurale con 3 livelli convoluzionali e due fully connected per questo task. Le performance del modello ottenuto erano molto soddisfacenti, tuttavia provandolo su Raspberry ci siamo resi conto che il riconoscimento non forniva molti risultati corretti.

Il vecchio dataset presentava diversi errori, pertanto abbiamo ritagliato 11945[1] nuove foto a mano, mantenendo lo stesso aspect ratio. Di queste, abbiamo considerato 8.238 foto (circa 55 foto per classe) e riallenato la rete. Abbiamo notato un netto miglioramento con dei test fatti dal vivo.

Lo scopo dell'estensione del nostro progetto è di allenare un nuovo modello con le 11945 immagini, utilizzando diverse architetture di reti neurali, ed effettuare un confronto di performance tra di esse.

Per lo sviluppo e l'allenamento del modello abbiamo utilizzato PyTorch 1.9. Il codice è disponibile in questo repository GitHub.

1.1 Preprocessing

Il nuovo dataset presenta circa 79 immagini per classe, che sono state opportunamente distribuite in training set, validation set e test set. Sono state effettuate operazioni di *data augmentation* come:

- **random resized crop**: ritaglio casuale di una porzione dell'immagine pari a `224x224`;
- **random horizontal flip**: specchiamento dell'immagine lungo l'asse y;
- **random perspective**: prospettiva casuale di un'immagine data una probabilità `p=0.5` e una scala di distosione `distortion_scale=0.6`.

[1] https://www.kaggle.com/unexpectedscepticism/11945-pokemon-from-first-gen

Figure 1: Operazioni di data augmentation effettuate

Infine, si è utilizzato `torchvision.transforms.Normalize()` per normalizzare il tensore che rappresenta l'immagine, in modo da ottenere media nulla e deviazione standard unitaria.

$$\tilde{x} = \frac{x - \mu_x}{\sigma_x}$$

2 Training

2.1 Transfer Learning

Il Transfer Learning è una tecnica che consiste nell'utilizzare reti preallenate su altri dataset (e.g. ImageNet), sostituendo l'ultimo livello fully connected con uno nuovo da riallenare. Risulta molto utile quando si ha un dataset con pochi esempi.

Esistono due tecniche di transfer learning:

- **Fixed feature extractor**: i pesi della rete vengono congelati, a eccezione dell'ultimo livello fully connected (classificatore).
- **Fine tuning**: vengono allenati sia il classificatore che il feature extractor.

In particolare, abbiamo effettuato un *fine tuning* di ResNet158, MobileNetV2 e MobileNetV3 con learning rate decay: partendo da un valore di $1 \cdot 10^{-3}$, tramite l'utilizzo dello scheduler `StepLR`, abbiamo abbassato progressivamente il learning rate di un fattore pari a 0.1 ogni 7 epoche.

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

2.2 Stochastic Gradient Descent

Come algoritmo di ottimizzazione abbiamo scelto Stochastic Gradient Descent (SGD) con Momentum, che stima l'errore del gradiente per lo stato corrente del modello, utilizzando esempi dal training set; successivamente vengono aggiornati i pesi del modello attraverso l'algoritmo di *backpropagation*. Viene utilizzato il `momentum=0.9`, che consente una maggiore velocità di convergenza della funzione di loss. Si consideri la seguente equazione che rappresenta l'SGD:

$$v^{t+1} = \beta v^t - lr \nabla L(\theta^t)$$
$$\theta^{t+1} = \theta + v^{t+1}$$

Essa può essere paragonata all'equazione del moto di una sfera che, rotolando su una superficie pendente verso il basso, acquisisce velocità. La velocità v^{t+1} non diminuisce appena la superficie cambia curvatura. Il momentum $\beta \in [0, 1)$ non è altro che il coefficiente che rappresenta

l'incremento della velocità di rotolamento v^t verso il fondo della superficie (punto di convergenza della funzione di loss).

```
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
```

2.3 ResNet152

ResNet152 è una rete con 152 livelli convoluzionali costituita da blocchi residuali: nelle reti tradizionali si è osservato che un numero di layer eccessivo porta alla saturazione delle performance. I blocchi residuali, strutturati come in figura 2, prevengono questo problema e consentono di allenare reti profonde.

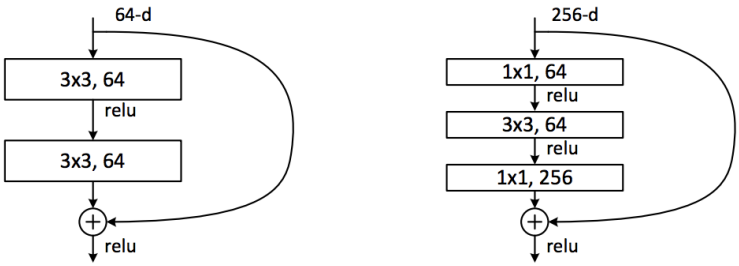

Figure 2: A sinistra il blocco residuale standard usato in ResNet 18 e 34, a destra il blocco a collo di bottiglia utilizzato in ResNet 50, 101 e 152.

2.4 MobileNetV2

MobileNet è una rete convoluzionale con 53 livelli basata su ResNet. È pensata per essere eseguita su dispositivi mobili: tramite una versione modificata dei blocchi residuali (chiamati *inverted bottleneck residual block*), costituiti da convoluzioni *depth-wise separable*, mantiene basso il numero di FLOP. Le convoluzioni *depth-wise separable* sono una variante delle convoluzioni standard in cui ciascun filtro processa un solo canale. A differenza dei blocchi residuali presenti in ResNet:

- viene aumentato il numero di canali del tensore in input mediante una convoluzione `1x1`;
- il tensore viene processato da una convoluzione `3x3` (*depth-wise separable*);
- il numero di canali viene nuovamente ridotto al numero originale con un'altra convoluzione `1x1`.

2.5 MobileNetV3

MobileNetV3 è una nuova generazione di MobileNet, sviluppata facendo uso di Neural Architecture Search (NAS), una tecnica per automatizzare il design di reti neurali, mediante una ricerca nello spazio delle possibili architetture. Esistono due versioni di MobileNetV3: small e large. In particolare, nei nostri test, abbiamo usato la variante large.

2.6 Risultati del training

L'allenamento è stato eseguito in locale su una GPU 2070 Super. I risultati del training sono riassunti nella tabella seguente:

Modello	Train loss	Val loss	Test loss	Train acc	Val acc	Test acc
ResNet152	0.2914	0.1288	0.107	93.54%	96.72%	97.171 %
MobileNetV2	0.5433	0.2815	0.224	87%	93.52%	94.509 %
MobileNetV3	0.6595	0.3366	0.299	82.83%	91.75%	91.015%

Modello	Tempo di Training	Dimensione
ResNet152	38' 38"	223.9 MB
MobileNetV2	7' 53"	9.45 MB
MobileNetV3	6' 55"	21.1 MB

Il migliore risultato è stato ottenuto da ResNet152, tuttavia il tempo di training e la dimensione finale del modello risulta nettamente superiore agli altri due modelli. MobileNetV2 risulta un ottimo compromesso tra accuratezza e dimensioni del modello. Sorprendentemente, MobileNetV3 ha avuto le peggiori performance, con l'eccezione del tempo di training, inferiore a MobileNetV2.

3 Conversione in TFLite

Dato che la nostra applicazione viene eseguita su sistemi a 32 bit, abbiamo convertito il modello in formato TFLite. Con applicazioni su sistemi a 64 bit, come Raspberry OS 64 bit, Android e iOS risulta più adatto PyTorch Mobile per un migliore deployment, che viene lasciato come sviluppo futuro.

4 Conclusioni

In questo progetto abbiamo effettuato *fine tuning* su una rete preallenata con ImageNet, in modo da ottenere performance migliori nonostante le dimensioni ridotte del dataset di Pokémon. In particolare abbiamo testato tre modelli, e il migliore in termini di performance è ResNet152, con un accuratezza pari a circa 97%. Infine, abbiamo esportato il modello in TensorFlow Lite per poterlo utilizzare su dispositivi embedded a 32 bit.