Heritage Detector

Utilizzo delle reti neurali in un'applicazione Android per classificare architetture di interesse culturale e valutarne la vulnerabilità sismica

Nicholas Antonio Carroll, Corinna Marchili

8 ottobre 2021

Abstract

L'obiettivo del progetto consiste nella realizzazione di un modello di classificazione d'immagini mediante l'addestramento di reti neurali per il riconoscimento di strutture architettoniche d'interesse culturale.
Il workflow prevede la creazione di un dataset, l'addestramento di un modello di Machine Learning mediante il framework Tensorflow, la conversione del modello per l'uso di TensorFlow Lite e lo sviluppo di un'applicazione Android con Android Studio.

1 Il dataset

Per gli scopi del progetto è stato considerato un sottoinsieme dei beni culturali di interesse architettonico, identificati dall'agenzia UNESCO nella pubblicazione "Evaluation of Seismic Risk on UNESCO Cultural Heritage sites in Europe".
Il dataset originato comprende 2500 immagini classificabili come archi trionfali, castelli, chiese, ponti ad arco o torri. L'automatizzazione del processo di reperimento delle immagini dal web è stata possibile grazie all'utilizzo del tool Google Image Downloader, sviluppato in Python.
La scelta delle categorie sopracitate è stata guidata dal carattere rappresentativo di quest'ultime rispetto all'ambito dell'architettura di interesse culturale, nonché dalla loro ampia diffusione sul territorio italiano ed europeo.

Le immagini scaricate sono state filtrate manualmente e selezionate sulla base della pertinenza alla categoria di riferimento, dando priorità alle foto in cui il soggetto fosse chiaramente riconoscibile. Particolare attenzione è stata prestata al "filtraggio" delle immagini che potessero causare confusione, come le foto di chiese che, presentando in primo piano il campanile, rischiavano di essere ambiguamente classificate come torri.
Un'ulteriore accorgimento che si è reso necessario è consistito nella rimozione dei duplicati eventualmente scaricati automaticamente. A tal proposito è stato utilizzato uno script in Python, seguito da un'ulteriore scrematura manuale per rimuovere casi particolari (ad esempio, immagini diverse dello stesso monumento ma scattate da angolazioni molto simili).
Le immagini rimanenti sono state ridimensionate e rinominate. Quest'ultima operazione è stata resa possibile dall'utilizzo di un ulteriore script in Python che assegnasse a ciascuna immagine il nome della cartella della rispettiva classe seguito da un ordinale, facilitando il labeling del dataset.

L'utilizzo del dataset Per l'addestramento della rete il dataset è stato suddiviso come segue:

- 80% training set
- 10% validation set
- 10% testing set

Date le dimensioni esigue del dataset, queste proporzioni sono risultate le più coerenti: testando altre configurazioni, si è reso evidente come un numero inferiore di campioni nel training set peggiorasse la performance nella fase di addestramento del modello.

1.1 Data augmentation

La dimensione ridotta del dataset ha reso particolarmente interessante l'adozione di tecniche di data augmentation.
A seguito di valutazioni ed esperimenti, la scelta è ricaduta su alcune tecniche di base:

- Shearing: distorsione dell'immagine lungo un asse, generalmente adottata per imitare o correggere angoli di percezione;
- Zooming: distorsione dell'immagine, operando uno zoom sulla parte centrale;
- Horizontal flip: specchiamento dell'immagine lungo l'asse y.

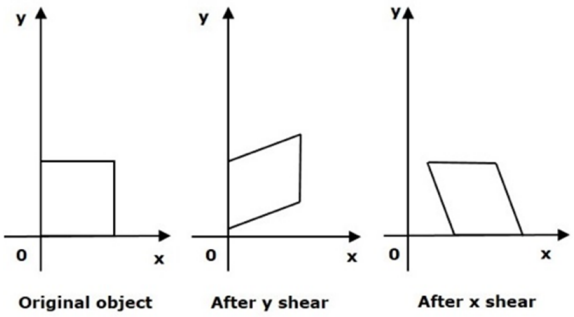


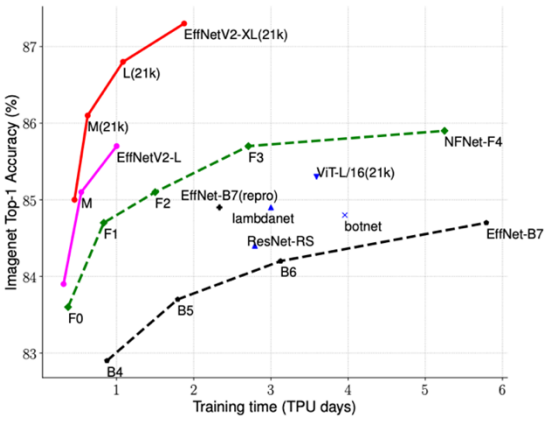
Figura 1 – Shearing lungo gli assi

2 L'addestramento della rete

Per lo sviluppo e l'addestramento del modello è stato utilizzato Tensorflow 2.X con Keras. Un primo tentativo è stato portato a termine utilizzando la rete VGG16, impiegando dei pesi pre-addestrati per velocizzare la fase di training. I risultati sono stati buoni ma non pienamente soddisfacenti, raggiungendo approssimativamente un'accuratezza del 75%. In conseguenza di ciò si è effettuato un secondo tentativo utilizzando la rete EfficientNetV2-s.

2.1 EfficientnetV2

EfficientnetV2 è una famiglia di modelli di classificatori di immagini che ottiene una migliore efficienza parametrica e accuratezza, richiedendo inoltre minori tempi di addestramento delle altre architetture allo stato dell'arte (la velocità è da 5 a 11 volte inferiore rispetto ad altre reti)??



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

L'architettura I layers 1, 2 e 3 sono composti dall'operatore Fused-MBConv, mentre 4, 5 e 6 da MBConv. MBConv utilizza le convoluzioni Depthwise, lente nei primi layer ma molto efficaci in quelli finali. Queste ultime hanno meno parametri e richiedono meno risorse delle convoluzioni normali, tuttavia non sono in grado di utilizzare a pieno gli acceleratori moderni.Fused-MBConv sostituisce la convoluzione 3x3 e l'espansione convoluzionale 1x1 con un singolo layer convoluzionale 3x3, ed è in grado di sfruttare al meglio gli acceleratori mobile.
L'applicazione di Fused-MBConv agli stadi 1-3 migliora la velocità di addestramento, pur aumentando di poco la dimensione del modello. Di conseguenza, qualora si volessero sostituire tutti gli strati MBConv con dei Fused-MBConv, si otterrebbero un modello molto più grande e un training notevolmente più lento.

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

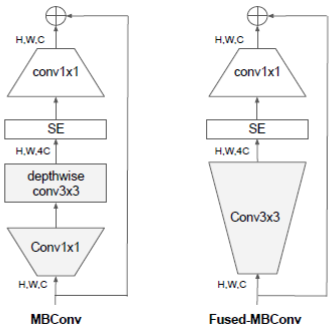


Figura 2 – I layers previsti in Efficientnet

L'addestramento Tra gli elementi più interessanti di EfficientnetV2 è necessario annoverare l'utilizzo del progressive learning per migliorare l'accuratezza e la rapidità dell'addestramento [x0x].
Il progressive learning consiste nell'uso di immagini più piccole con una regolarizzazione debole nelle prime epoche, affinché la rete possa imparare rapidamente le rappresentazioni più semplici, per poi ingrandire progressivamente le immagini nelle epoche successive e aumentare la regolarizzazione.
I tipi di regolarizzazione adottati sono i seguenti:

- Dropout: regolarizzazione a livello di network, si rimuovono in modo casuale dei canali dell'immagine;
- RandAugment: tipologia di data augmentation che consiste nella modifica dell'immagine;
- Mixup: tipologia di data augmentation che consiste nel "mescolare" due immagini sulla base di un mixup ratio variabile.

3 Training

Anche nel caso di EfficientNetV2 è stato adottato un modello pre-addestrato, per ridurre l'impiego di tempo e risorse computazionali. La rete richiede in input immagini di dimensioni 384x384 a tre canali. Le scelte operate per il training sono riassunte come segue:

Figura 3 – Regolarizzazione dell'immagine

- SGD con learning rate di 0.005 e momentum di 0.9 come optimizer;
- Categorical CrossEntropy con label smoothing di 0.1 per il calcolo della loss;
- Batch di dimensione 16.

La rete è stata addestrata su Google Colab per 100 epoche, con risultati che tendono a crescere molto velocemente all'inizio dell'addestramento per poi stabilizzarsi entro circa 15 epoche.
I risultati sono più soddisfacenti rispetto a quanto ottenuto dal modello precedente, raggiungendo un'accuratezza finale del 96%.

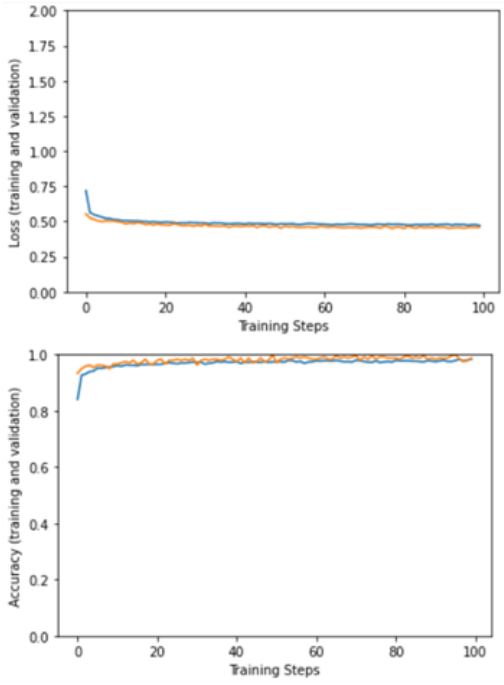


Figura 4 – Loss ed accuracy del modello, in relazione all'avanzamento del training

4 Trasformazione in TFlite e quantizzazione

Il modello ottenuto è stato convertito in un formato FlatBuffer (estensione *.tflite*) che garantisce una dimensione ridotta e un'inferenza più rapida, essendo in grado di accedere ai dati senza ulteriori passaggi di parsing. Il modello è stato infine quantizzato con l'obiettivo di ottimizzarlo e renderlo il più possibile rispondente alle necessità dello sviluppo mobile.
L'ottimizzazione adottata è la dynamic range quantization, che consente di ridurre del 75% la dimensione del modello e raddoppiare o triplicare la velocità di inferenza.
Questo tipo di quantizzazione consiste nel trasformare i pesi del modello da float (32 bit) ad interi (8 bit). Durante l'inferenza i pesi vengono nuovamente trasformati in float, tuttavia le conversioni sono salvate in memoria per velocizzare la procedura. La dimensione finale del modello TFLite quantizzato è di 22 MB laddove il modello originario occupava 88 MB.
Inevitabilmente questa procedura riduce l'accuratezza, tuttavia, operando un'analisi approfondita delle performance, è stato possibile appurare che nel caso in esame le differenze sono minime. Il modello quantizzato concorda con il corrispettivo originale nel 99.2% delle classificazioni, raggiungendo pertanto un'accuratezza finaledel 95.6%.

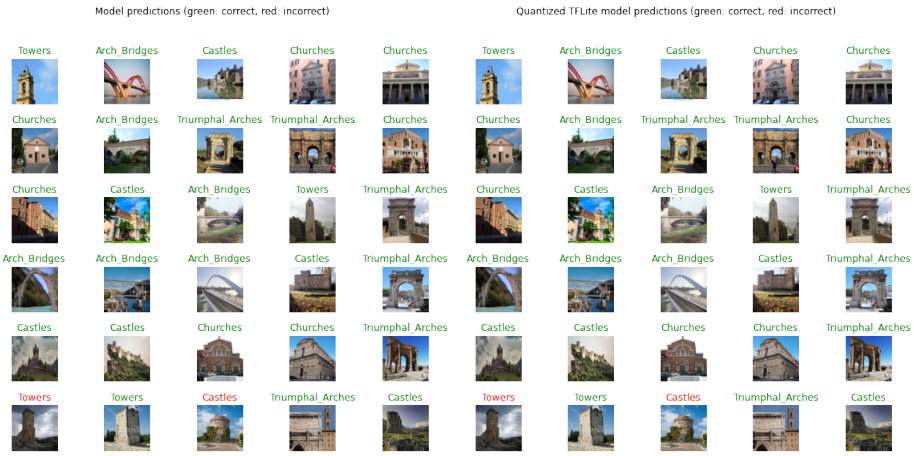


Figura 5 – Confronto tra le classificazioni operate rispettivamente da modello originale e quantizzato

5 L'applicazione Android

Il modello ottenuto dall'addestramento, trasformato in formato .tflite e quantizzato, è stato successivamente inserito mel contesto di un'applicazione Android realizzata in Kotlin su Android Studio che consente -data un'immagine- di predirne la classificazione e simulare un evento sismico per valutarne l'impatto qualitativo sull'architettura.
L'utente può selezionare un'immagine dalla galleria del dispositivo o scattare una foto; la classificazione inferita dal modello precedentemente addestrato consente di attribuire dei valori approssimativi di vulnerabilità sismica e duttilità (fig.6). Questi parametri sono in seguito utilizzati nel calcolo dell'intensità macrosismica e del relativo danno, applicando la formula

$$I_{MMI} = 1.0157 + 1.2566 M_w - 0.6547 \ln \sqrt{R^2 + 4} \tag{1}$$

TYPE	V_0	Q
Arch bridges	0.296	2.30
Castles	0.456	2.30
Churches	0.890	3.00
Columns	0.456	1.95
Monasteries	0.736	2.30
Mosques	0.730	2.65
Obelisks	0.456	1.95
Palaces	0.616	2.30
Temples	0.500	1.95
Towers	0.776	2.30
Trilithes	0.456	1.95
Triumphal arches	0.456	2.30

Figura 6 – Valori di vulnerabilità sismica e duttilità delle strutture architettoniche di interesse culturale

e riportando una breve descrizione del rischio associato al livello di danno ottenuto, calcolato secondo la formula

$$MeanDamage = 2.5 * (1.0 + \tanh \frac{I - 6.25V - 13.1}{q}) \tag{2}$$

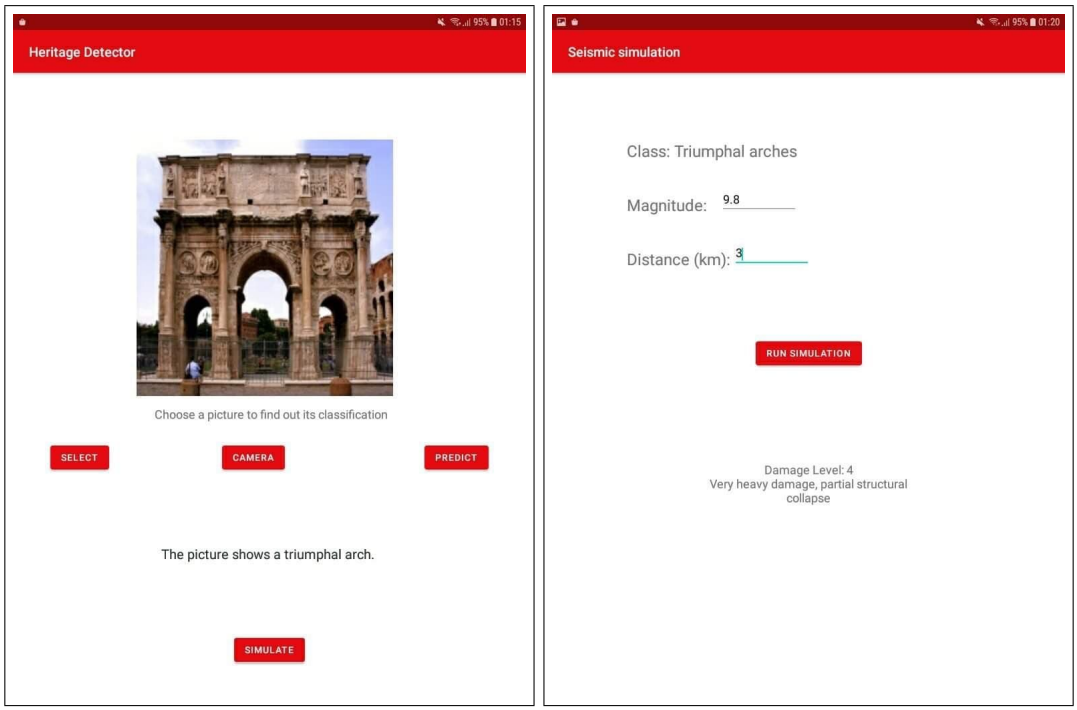


Figura 7 – Vista dell'applicazione su tablet