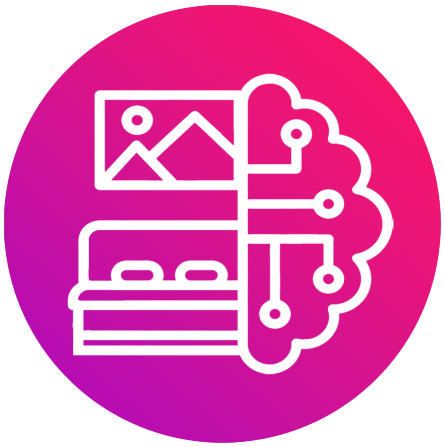
UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Interior Room Classifier



Progetto Machine Learning

Studenti:
Pietro Vassallo 1000008684
Gioele Cageggi 1000008660

Anno Accademico 2019/2020

Indice

Introduzione

La classificazione automatica della scena (talvolta denominata "riconoscimento della scena", o analisi della scena) è un problema di ricerca ricorrente nella Computer Vision, che consiste nell'assegnare un'etichetta come «spiaggia», «camera da letto» o semplicemente «interno» o «esterno» a un'immagine presentata come input, basata sul contenuto complessivo dell'immagine.

Consideriamo il seguente problema affrontato da noi umani: il riconoscimento e la classificazione di un ambiente reale che contiene superfici e oggetti multipli, organizzati in modo significativo avviene in un decimo di secondo o meno, grazie alla nostra capacità di catturare il senso della scena.

Quanto dovrebbe imparare un sistema artificiale prima di raggiungere le capacità di riconoscimento della scena di un essere umano?

Tralasciando la densità delle possibili scene che potrebbero essere classificate, il problema viene affrontato con le Convolutional Neural Network. Queste architetture ci permettono di identificare oggetti o scene utilizzando dataset di grandi dimensioni, ma nella seguente relazione ci occuperemo di «allenare» le CNN per adattarle al nostro problema: il riconoscimento di scene interne o dette anche indoor scenes. Pertanto, l'obiettivo è quello di identificare se la scena in input è una camera da letto, una cucina, un bagno o un salone.

È importante questa tipologia di task perché sono sempre di più gli scenari in cui è necessario per un dispositivo riconoscere l'ambiente interno ad una casa in cui si trova. Consideriamo il caso di un robot che si occupa di pulire le zone di casa che gli vengono assegnate: dovrà necessariamente costruire una mappa della casa per poter capire come muoversi ed identificare le aree di interesse.
Altri esempi potrebbero le applicazioni che si occupano di aggiungere oggetti in realtà virtuale, in modo tale da consigliare in base alla scena inquadrata i corrispondenti elementi da inserire.

Figure 1- Idea applicazione; Source: internet

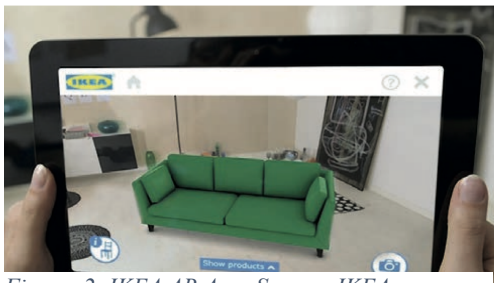


Figure 2- IKEA AR App; Source: IKEA

Acquisizione dati

Per la realizzazione del dataset sono stati acquisiti 86 video complessivamente di durata media di 25s. La cattura è avvenuta mediante due smartphones: iPhone 11 pro e iPhone XR.

VIDEO	
Video Rec.	4K (2160p)
Video Auto Focus	✓
Video Fps	60 fps
Video Stabilizer	✓
Slow Motion	✓ 240 fps
Video HDR	✓
Dual Rec	✓
Stereo Sound Rec	✓
Foto in Video	✓
Videocamera Frontale	4K (2160p), 60fps
Opzioni Fotocamera Frontale	HDR/Face Detection/EIS

Figure 3 - Scheda Tecnica video Iphone 11 Pro

VIDEO	
Video Rec.	4K (2160p)
Video Auto Focus	✓
Video Fps	60 fps
Video Stabilizer	✓
Slow Motion	✓ 240 fps
Dual Rec	✓
Foto in Video	✓
Videocamera Frontale	Full HD, 30fps
Opzioni Fotocamera Frontale	HDR/Face Detection/Modo Selfie

Figure 4 - Scheda Tecnica video Iphone XR

Le acquisizioni sono state effettuate attraverso la registrazione di video a 360 gradi in punti specifici delle stanze. Tali punti sono stati ottenuti mappando l'ambiente con punti di cattura a distanza di un metro circa tra loro:

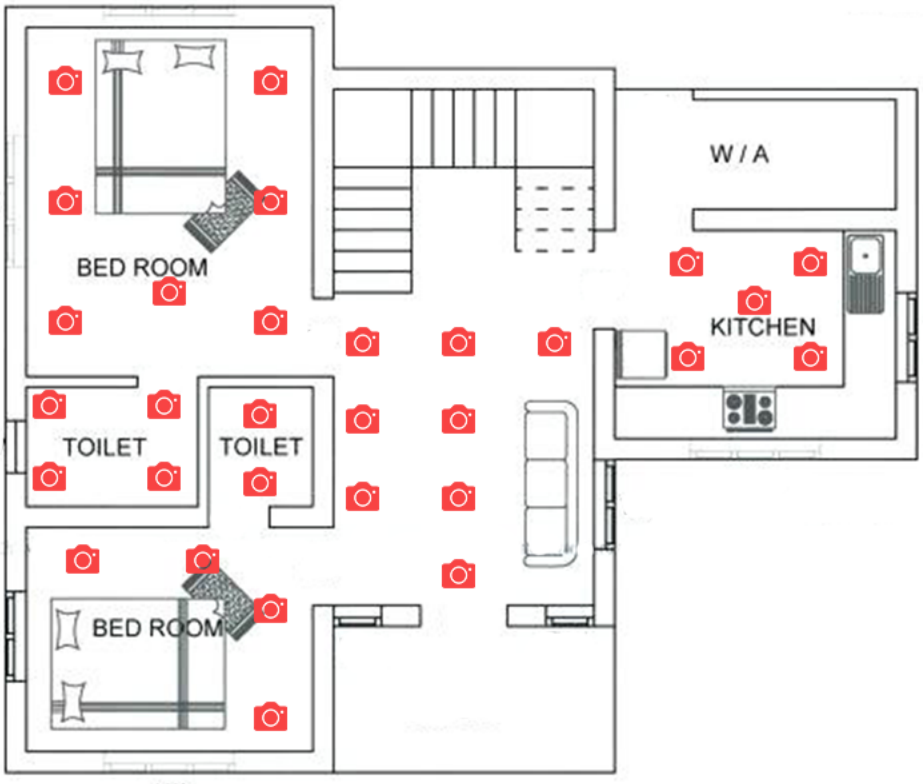


Figure 5 - Esempio di mappatura delle stanze per l'acquisizione dei video

Tutte le riprese relative ad una stanza (classe) sono state inserite nella corrispondente cartella, questo ci ha permesso di applicare una prima etichettatura. Successivamente sono stati estratti i frame, da tali riprese, che popoleranno il dataset. Per fare questo è stata utilizzata la funzione Filtro Scena di VLC, prelevando 10 frame per secondo. Dagli 86 video sono stati estratti 5821 immagini.
L'organizzazione finale delle immagini per il progetto è la seguente:

Figure 6 - Struttura classi

Le 5821 immagini contengono rispettivamente:

- 902 frame per il bagno
- 288 frame per la camera letto matrimoniale (cameraL)
- 1213 frame per cameretta
- 864 frame per la cucina
- 2553 frame per il salone

N.B.
- La mole di immagini per stanza è proporzionale alla grandezza della stanza, pertanto le stanze più grandi hanno più frame.
- Per la camera da letto matrimoniale è stata fatta l'acquisizione solo da un appartamento, questo giustifica il ridotto numero di frame.

Creazione Dataset

Per la creazione del dataset le immagini sono state caricate su Google Drive in modo da poter essere facilmente accessibili alla piattaforma Google Colaboratory.

Identificati i nomi delle classi usando la funzione glob della libreria os, è stato creato un dizionario che mappa le classi a valori numerici:

'bagno': 0, 'cucina': 1, 'cameretta': 2, 'cameraL': 3, 'salone': 4

Utilizzando la funzione DataFrame di Pandas abbiamo creato il dataset contenente due colonne, una relativa al path dell'immagine, l'altra alla corrispondente label:

	path	label
0	bagno/bagno100001.png	0
1	bagno/bagno100011.png	0
2	bagno/bagno100021.png	0
3	bagno/bagno100031.png	0
4	bagno/bagno100041.png	0

Figure 7 - Dataset con path delle immagini e corrispondente label

In seguito, è stato effettuato lo split del dataset in training set, test set e valutation test con rapporto [60, 10, 30]:

- 3489 frame per training set
- 581 frame per valutation set
- 1745 frame per test set

Sono stati effettuati allo stesso modo ulteriori split per esperimenti su VGG16 utilizzando le seguenti percentuali:

Split 2 [70, 15, 15] con seed 1755

- 4071 frame per training set
- 873 frame per valutation set
- 873 frame per test set

Split 3 [80, 10, 10] con seed 1932

- 4653 frame per training set
- 582 frame per valutation set
- 582 frame per test set

Infine, i vari set ottenuti sono stati salvati in formato csv e txt nelle corrispondenti cartelle del progetto.

Metodo

Per la procedura di training si è utilizzata la piattaforma Colaboratory di Google che, oltre all'editor, mette a disposizione anche risorse computazionali. I training hanno necessitato di risorse computazionali superiori a quelle offerte dal piano colaboratory standard, si è dunque acquistato un abbonamento di tipo pro.

Di seguito sono elencati tutti i modelli su cui si sono basate le analisi, allo scopo di valutarne e confrontarne le accuracy e loss ottenute.

VGG16

La VGG16 è composta da 16 livelli convoluzionali distribuiti in modo uniforme.

La rete contiene molti livelli ed è dunque sia più pesante (in termini di tempi di training e memoria richiesta) che più performante.

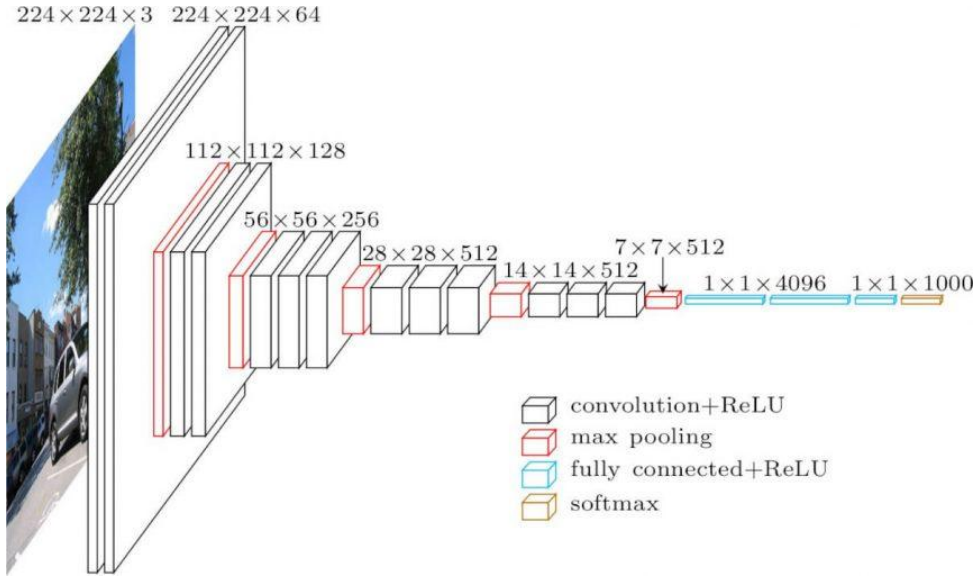


Figure 8 - Layers VGG16

ResNet18

Esistono diverse varianti di Resnet con diversi numeri di livelli, quello utilizzato è Resnet18.

I modelli con un numero minore di livelli raggiungono accuracy minori ma sono più veloci e occupano meno spazio in memoria. Questo ci consente un confronto con una rete molto più pesante come VGG

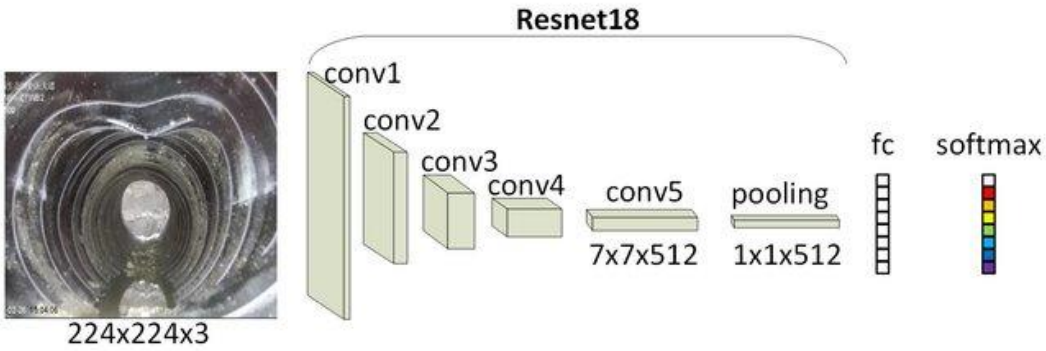


Figure 9 - Layers ResNet18

SqueezeNet1

È un modello di classificazione delle immagini con training basato sul set di dati ImageNet pubblicato nel 2016. È un modello molto compatto e veloce: occupa pochi megabyte in memoria e può essere utilizzato su CPU non molto performanti.

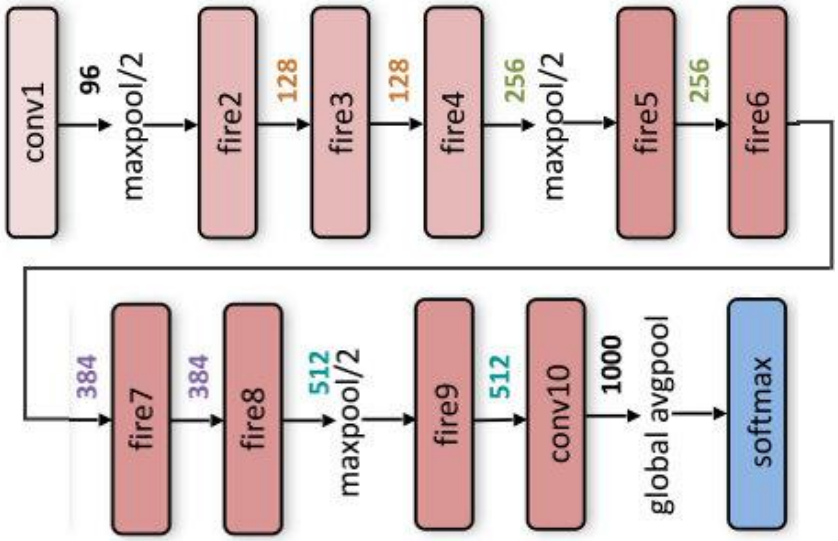


Figure 10 - Layers SqueezeNet1_0

Esperimenti e Risultati

Le immagini per ogni training sono state mescolate in modo casuale e sottoposte alle seguenti trasformazioni:

1. Ridimensionamento in 256x256 pixel
2. Un ritaglio casuale di 224x224 pixel
3. Un capovolgimento orizzontale casuale
4. Normalizzazione secondo media e deviazione standard, calcolate precedentemente

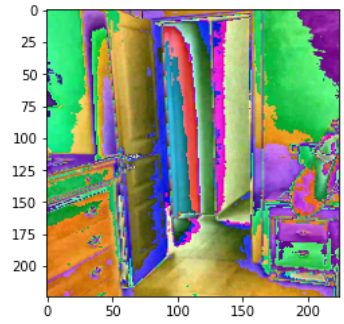


Figure 11-Immagine in seguito alle trasformazioni

VGG16

La prima rete che è stata allenata è VGG16 con i seguenti parametri:

- Learning rate settato a 0.01
- Momentum 0.9
- Epoche 10

In seguito ad un allenamento durato 27 minuti i risultati ottenuti sono i seguenti:

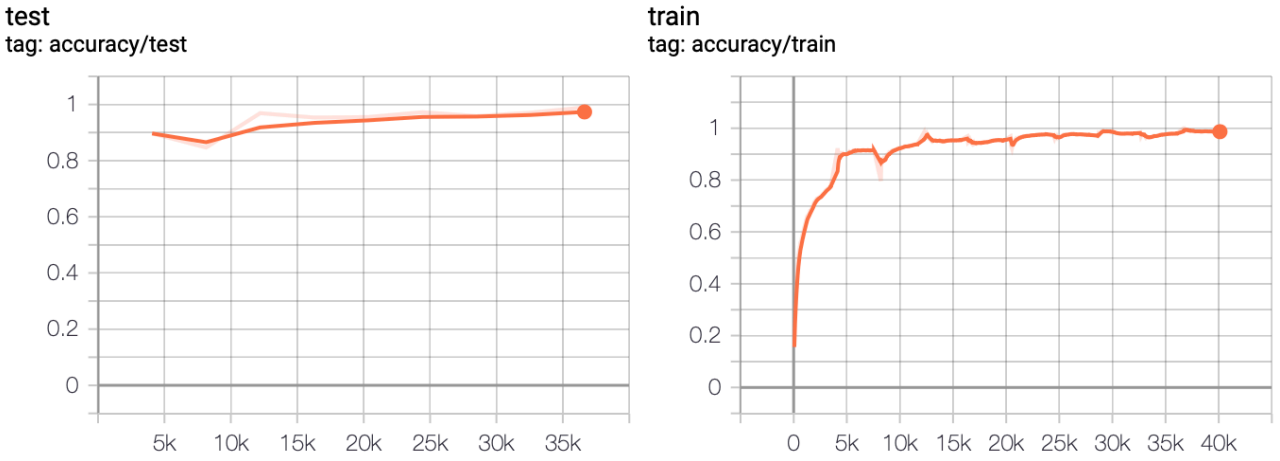


Figure 12 - Accuracy su Test e Train VGG16

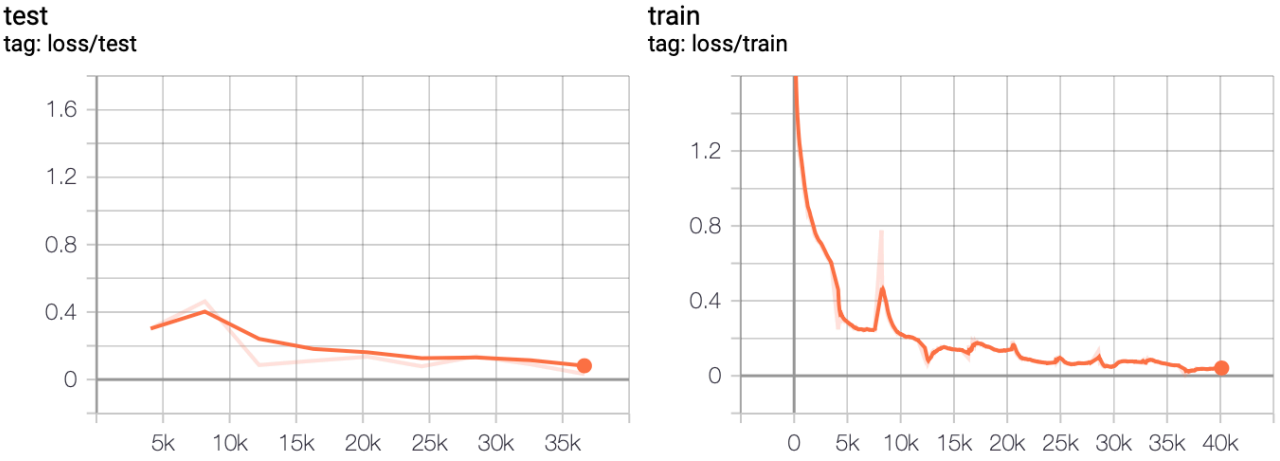


Figure 13 - Loss su Test e Train VGG16

	Accuracy	Loss
Training	0,9868	0,04141
Test	0,9897	0,03284

ResNet18

La seconda rete che è stata allenata è ResNet18 con i seguenti parametri:

- Learning rate settato a 0.01
- Momentum 0.9
- Epoche 15

In seguito ad un allenamento durato 42 minuti i risultati ottenuti sono i seguenti:

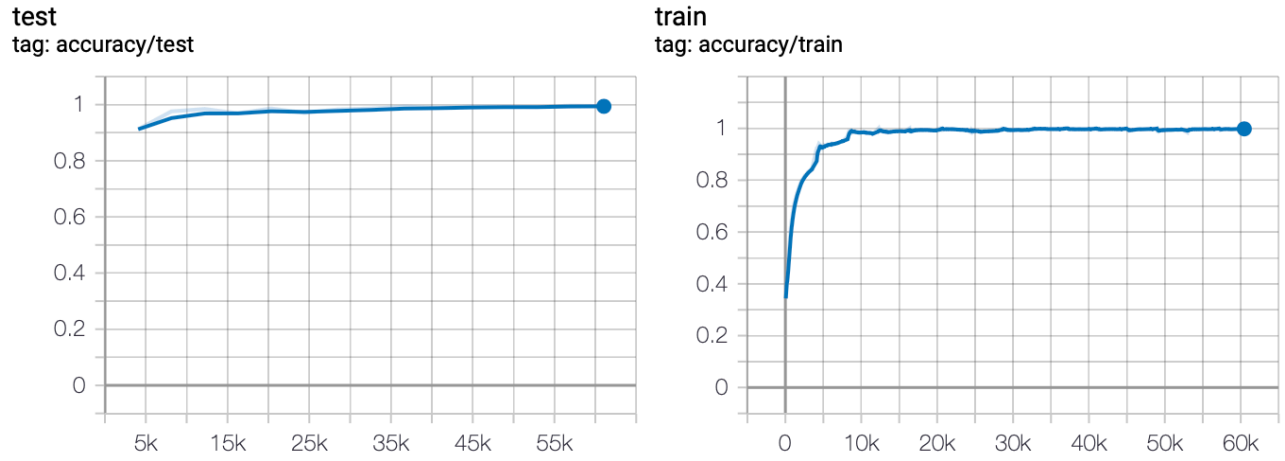


Figure 14 - Accuracy su Train e Test ResNet18

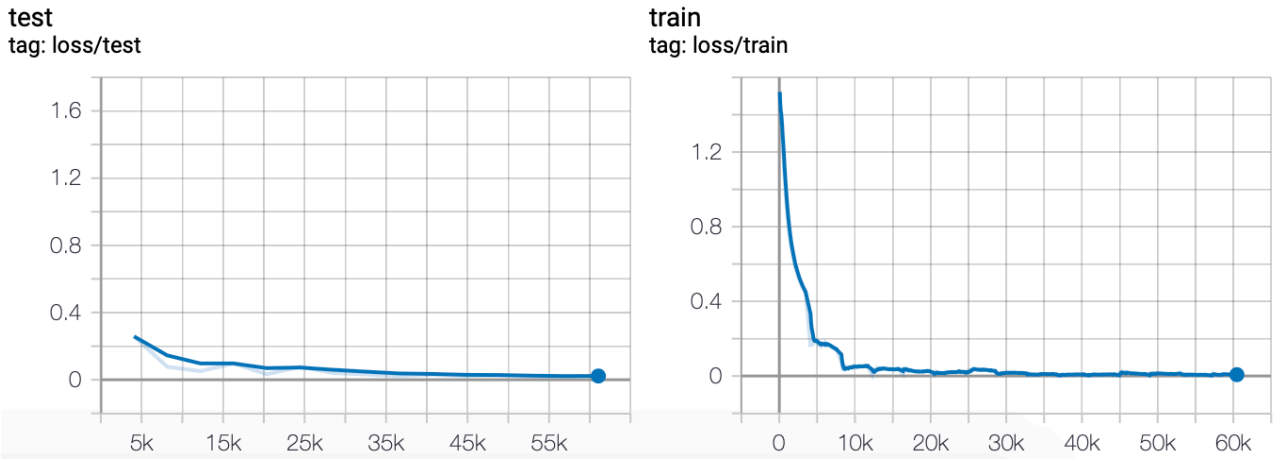


Figure 15 - Loss su Train e Test ResNet18

	Accuracy	Loss
Training	0,9983	6,8948e-3
Test	0,9948	0,02297

SqueezNet1_0

La seconda rete che è stata allenata è SqueezNet1_0 con i seguenti parametri:

- Learning rate settato a 0.01
- Momentum 0.9
- Epoche 30

In seguito ad un allenamento durato 1.3h i risultati ottenuti sono i seguenti:

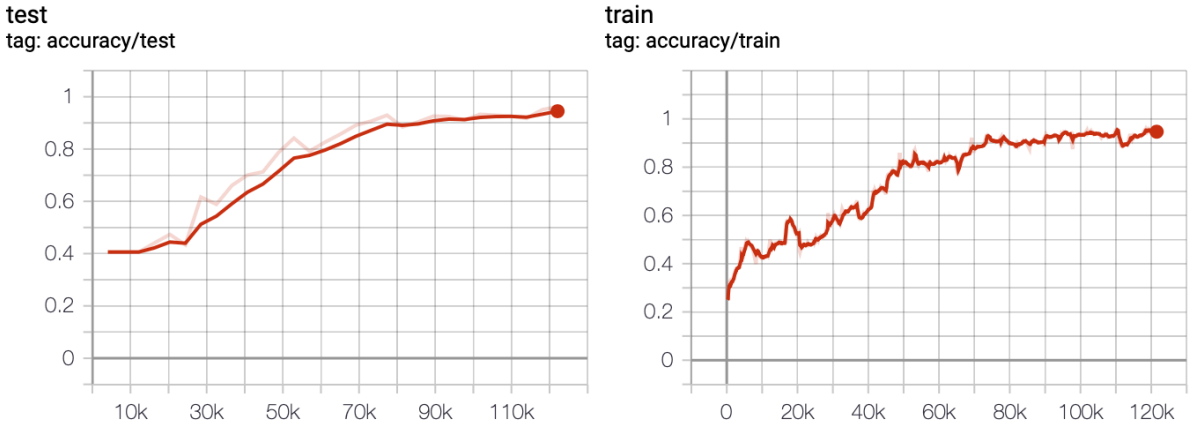


Figure 16 - Accuracy Train e Test SqueezNet

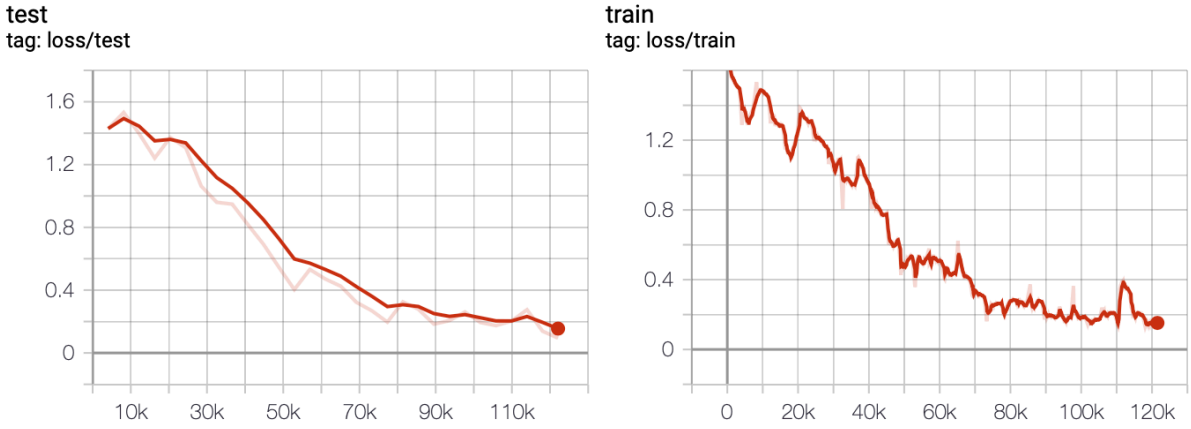


Figure 17 - Loss su Train e Test SqueezNet

	Accuracy	Loss
Training	0.9705	0.0860
Test	0.9513	0.1630

Confronto VGG16, ResNet e SqueezNet

	Accuracy su Train	Accuracy su Test
VGG16	0,9868	0,9897
ResNet	0,9983	0,9948
SqueezNet	0.9705	0.9513

Come si può notare i risultati migliori li forniscono tutte le reti testate ma ResNet si distingue come più prossima ad una Accuracy pari ad 1 su Test

Confronto VGG16

Anche se i dati relativi ai valori di Accuracy e Loss sono favorevoli per ResNet18 rispetto le altre due reti analizzate, utilizzando la demo realizzata per il seguente elaborato ci siamo resi conto che i risultati "più costanti" sono stati forniti da VGG16.

Incuriositi dal caso, ci siamo chiesti se i risultati di Accuracy e Loss ottenuti fossero stati casuali o meno, pertanto è stato condotto un ulteriore esperimento in cui si utilizzano split diversi dei dati in training, validation e test set con seed diverso. I risultati ottenuti sono mostrati nelle immagini seguenti:

- VGG16 con split 60 10 30
- VGG16 con split 80 10 10
- VGG16 con split 70 15 15

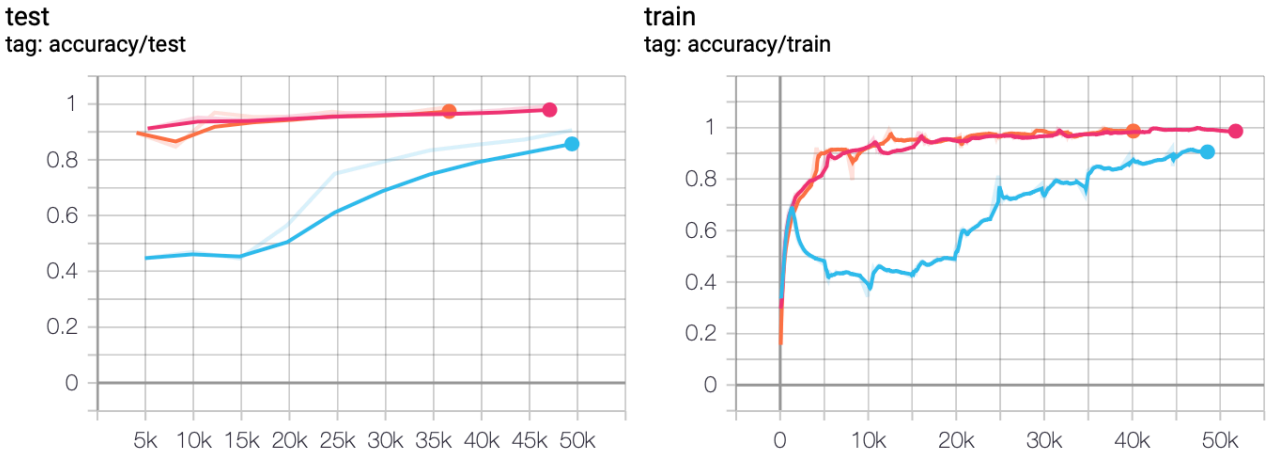


Figure 18 - Confronto Accuracy VGG16 vari split e seed

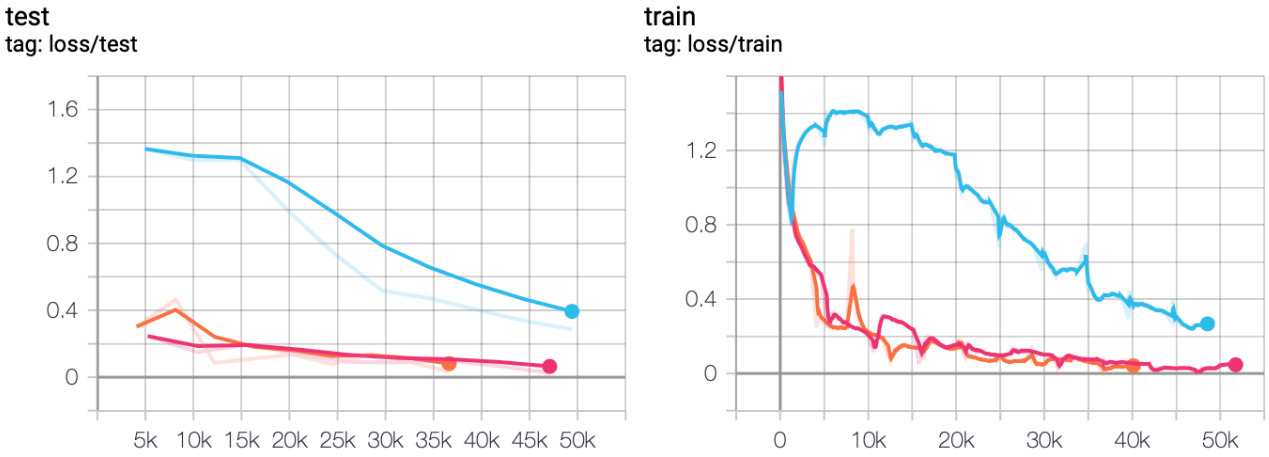


Figure 19 - Confronto Loss VGG16 vari split

	Accuracy Train	Accuracy Test	Loss Train	Loss Test
VGG16 [60 10 30]	0,9868	0,9897	0,04141	0,03284
VGG16 [70 15 15]	0,9054	0,9060	0,2680	0,2856
VGG16 [80 10 10]	0,9865	0,9931	0,04653	0,02604

Dai dati ottenuti si evince che due su tre degli esperimenti condotti hanno risultati pressoché simili, dunque possiamo affermare che i risultati ottenuti del primo esperimento non sono influenzati dalle percentuali scelte per lo splitting né dal seed fornito.

Demo

Abbiamo deciso di sviluppare una demo per sistema operativo Android che permette di testare in real time i modelli allenati in locale.

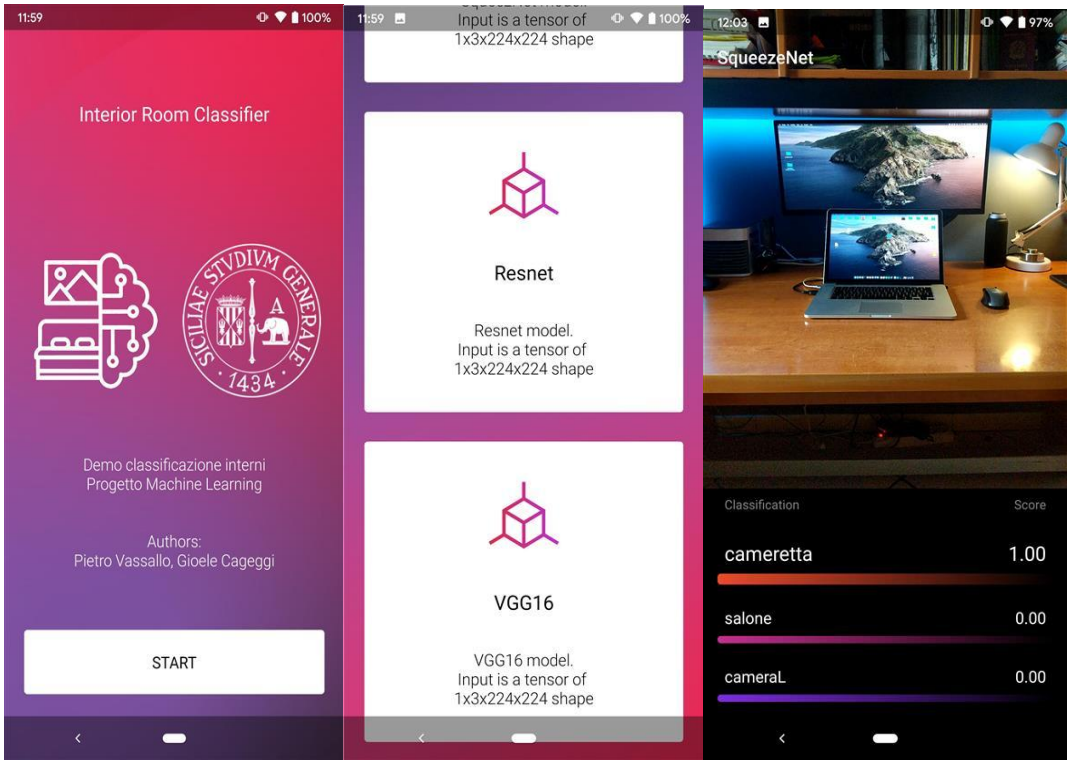


Figure 20 - Demo applicazione sviluppata per il progetto

Un video della demo in esecuzione può essere visionato al seguente link.

Implementazione modulo Pytorch

1. Dopo aver allenato i modelli essi sono stati serializzati utilizzando la funzione torch.jit.trace() che combinata alla save() ci permette di salvare il modello in formato model.pt. Questo è il modello che verrà inserito come asset nell'applicazione.
2. Sono state inserite le dipendenze di Pytorch Android nel Gradle:
   a. *org.pytorch:pytorch_android* : ci fornisce l'accesso alle API di Pytorch per Android
   b. *org.pytorch:pytorch_android_torchvision* : per convertire le immagini e bitmap in tensori
3. In seguito, viene estratta l'immagine dal live stream e convertita in bitmap
4. Viene caricato il modello Pytorch mediante le API fornite dalla libreria:
5. Viene convertita l'immagine bitmap in tensore e viene normalizzata per essere input
6. Viene avviata l'inferenza :

```
Tensor outputTensor = module.forward(IValue.from(inputTensor)).toTensor();
float[] scores = outputTensor.getDataAsFloatArray();
```

che restituisce array di float che dovranno essere dati in pasto ad una funzione softmax che concluderà la classificazione.

Maggiori dettagli implementativi possono essere recuperati al seguente link: https://pytorch.org/mobile/android/

Codice e contenuto consegnato

La demo è accessibile nel seguente repository GitHub:

https://github.com/mrjoelc/Interior-Room-Classifier

L'intero progetto è disponibile al seguente link di Google Drive:

https://drive.google.com/open?id=1dM4nvuWlHyTltzd4S2J0n3fSnkCMfa-V

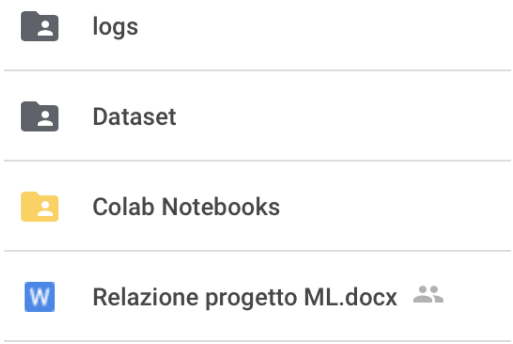


Figure 21 - Struttura progetto consegnato

logs

La cartella "logs" contiene tre cartelle corrispondenti alle tre reti allenate in cui sono contenuti per ogni rete:

- Il file di log che può essere visualizzato utilizzando TensorBoard
- Il modello esportato .pt necessario per la demo
- I finetuned per ogni epoca fatta eccezione per VGG16.

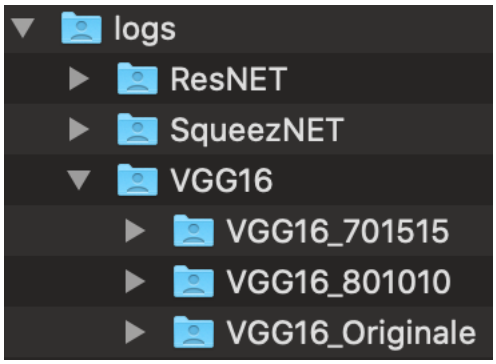


Figure 22 - Struttura cartella log

Per VGG16 ogni file finetuned.pth è di circa 512mb e avendo allenato la rete tre volte per 10 epoche, per via degli esperimenti effettuati, la cartella generale di VGG16 avrebbe una dimensione di oltre 15GB (utilizzando drive si hanno a disposizione soltanto 15gb di archiviazione gratuita) pertanto per ogni esperimento di VGG16 sono stati caricati i file .pth corrispondenti alla decima epoca.

Dataset

La cartella Dataset contiene tre cartelle: una relativa alle immagini che sono ulteriormente suddivise nelle cartelle delle classi di appartenenza (cucina, salone, bagno etc...), due relative al dataset in formato csv e txt.

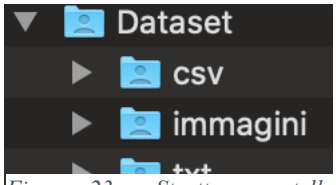


Figure 23 - Struttura cartella Dataset

Colab Notebooks

In questa cartella sono contenuti i vari notebook python nel particolare:

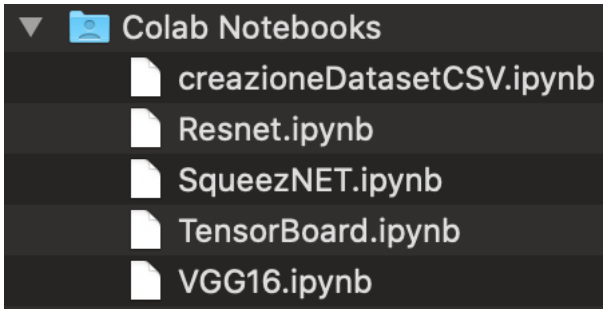


Figure 24 - Contenuto cartella Notebooks

- creazioneDatasetCSV: vengono creati i file che sono contenuti nella cartella Dataset/csv e Dataset/txt
- Resnet: training e test del modello Resnet18
- SqueezNET: training e test del modello Resnet18
- TensorBoard: visualizzazione del log presenti nella cartela logs mediante TensorBoard
- VGG16: training e test del modello VGG16 con esperimenti aggiuntivi annessi

Conclusioni

Il task che ci è stato assegnato per il progetto consiste nell'allenare delle reti convoluzionali per poter effettuare la classificazione di stanze di una casa. Nel particolare le classi che abbiamo preso in considerazione sono il bagno, la cucina, il salone e le camere da letto.

Per risolvere il task abbiamo girato video a 360 gradi in punti prefissati delle stanze dei due appartamenti, che ci sono serviti per ottenere un dataset di 5821 immagini. Questa fase, che di norma tende ad essere la più difficile e costosa, è stata agevolata poiché le acquisizioni sono state fatte negli appartamenti in cui viviamo. Tali immagini sono state inserite nelle corrispondenti cartelle relative alle stanze di appartenenza.

Successivamente ci siamo resi conto che per fare l'allenamento delle reti avevamo bisogno di macchine performanti e dopo un approccio fallimentare su AWS abbiamo deciso di utilizzare la piattaforma Google Colaboratory, sottoscrivendo un abbonamento Pro.

Le reti che abbiamo scelto per il task sono VGG16, SqueezNET e ResNET. Nel complesso esse si sono comportate piuttosto bene nel particolare ResNET ha raggiunto livelli di Accuracy e Loss ottimi. Tuttavia, in seguito alla modellazione di una demo che ci permettesse di girare il modello su smartphone, ci siamo accorti che nella realtà VGG16 da risultati più precisi, "senza confondersi" muovendosi nelle stanze per un live test. Questo risultato ci ha indotto ad effettuare un ulteriore esperimento: verificare che VGG16 dia gli stessi risultati modificando le percentuali e i seed degli split del dataset. L'esperimento si è concluso constatando che in seguito ai training, due su tre modelli allenati hanno fornito risultati soddisfacenti, concludendo che il primo risultato ottenuto con VGG16 non è stato casuale.

Sviluppi futuri potrebbero essere l'uso del deep learning metric per classificare stanze "vicine" tra loro e quindi provenienti da case diverse, oppure implementare l'applicazione per fornire predizioni su mobili da poter inserire nella scena catturata.