UNIVERSITÀ
degli STUDI
di CATANIA

Progetto di Machine learning

LEAF - CLASSIFIER

Salvatore Borgesi (100009060)

Indice

Introduzione

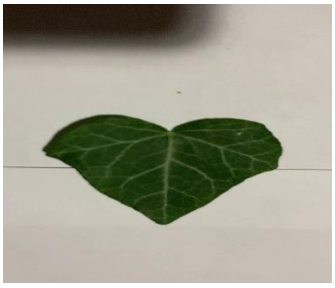
Il task che è stato assegnato dal docente, consiste nella realizzazione di un classificatore di immagini, più nel dettaglio l'obiettivo è quello di classificare delle foglie. Per questo progetto sono state utilizzate tre "classi" (dunque tre tipi di foglie differenti): Foglie di un albero di nespole, le foglie di un ulivo ed infine delle foglie di edera.

La classificazione delle piante, è sempre stata una grande sfida , infatti un classificatore di questo tipo può risultare utile per diversi motivi quali riconoscere la salute della pianta o banalmente per le persone che vogliono scoprire qual è il tipo di pianta che stanno osservando.

Raccolta del dataset

Il dataset è stato costruito considerando un caso ben specifico. Le foglie sono state adagiate su un foglio bianco e sono stati realizzati dei video a 30FPS (1008p) ruotando lo smartphone lungo i tre assi. Le registrazioni sono state eseguite con due device diversi : Xiaomi MI 9t ed un Iphone SE . Questo per rendere il dataset più eterogeneo possibile dato che le fotocamere dei due smartphone hanno caratteristiche diverse.

Le riprese, sono state effettuate sotto tre tipi di luce differente: sotto la luce del Sole , all'ombra ed infine con l'ausilio di una luce (bianca) artificiale. I video sono stati inseriti nella folder *Dataset* sotto i nomi di : *alloro, edera, nespole.* E' possibile trovare i video realizzati in questo link.


Partendo dalla raccolta del dataset sono stati gestiti due macro-esperimenti diversi. In questa fase introduttiva al dataset non verranno approfondite le motivazioni che hanno portato a tale scelta. Si entrerà più nel dettaglio nella sezione *Esperimenti*.

Suddivisione Dataset - Prova I

La prima prova portata avanti prevede che non si faccia distinzione tra i frames (estratti dal video) che finiranno nel *training set* con quelli che invece finiranno nel *validation/test* set.

Suddivisione Dataset – Prova II

Nel secondo esperimento, viene fatta una differenziazione tra i frames che vengono estrapolati da ciascun video. Per fare un esempio per il video *1.MOV* l'algoritmo di estrazione definirà che **tutte** le immagini estrapolate faranno parte dell'insieme di training, mentre per il video *2.MOV* tutti i frames saranno utilizzati per il validation o il test set.

Procedura di estrazione dei frames

All'interno del progetto si trova una folder *extractions* (sotto *src)* che contiene al proprio interno i script che consentono di estrapolare dai video le immagini.

1. *01_frames.py*: ha il compito di processare i video registrati e partendo da questi ricavare i frames. In questo script è presenta la logica descritta nel precedente paragrafo . Quest'ultima infatti, viene gestita mediante il valore di configurazione DISTINCT_TRAIN_SET . Se questo Booleano è definito uguale a True , allora sarà eseguito lo split del dataset come descritto *qui*.

2. *02_csv.py*: prende in input il file di testo (o più file di testo, questo dipende dalla configurazione) creato nello script citato sopra, e definisce i CSV di *training, validation e test* utilizzati poi, per il training dei modelli.

Installazione ed Esecuzione

Per installare il progetto , bisogna eseguire il clone dal seguente link GitHub. Per la realizzazione sono state usate diverse librerie (pytorch, numpy…) che dovranno essere installate. Per tale motivo va lanciato il comando

pip3 install-r requirements.txt

Per motivi di spazio la cartella che contiene il dataset non è stata caricata su github, ma è possibile scaricarla dal seguente link. Nella cartella sono presenti due sottocartelle : *Datasets e Frames*. La prima contiene i video dal quale sono state estrapolate le immagini presenti in *Frames.*

Se si vuole partire da zero, dovrà essere inserire la cartella *datasets* all'interno della folder di progetto e lanciare lo script

python -m src.extractions.01_frames

Questo script è stato descritto in una sezione precedente ed alcune sue caratteristiche sono configurabili dal file *config.py* presente sotto la folder *src*. Qualora si vogliano utilizzare i frames già estratti , dovrà essere copiata la cartella *Frames* presente su Drive all'interno del progetto e lanciare:

python -m src.extractions.02_csv

La cartella *frames* contiene le cartelle "all" ove sono stati inseriti tutti i frames estratti per il parametro di configurazione DISTINCT_TRAINTEST_SET = FALSE, mentre le cartelle *test e train* sono state popolate lanciando lo script di estrazione (*01_frames.py*) con DISTINCT_TRAINTEST_SET = TRUE

Dopo che sono stati generate le immagini (o copiate) , potrà essere lanciato il comando:

python -m src.Main

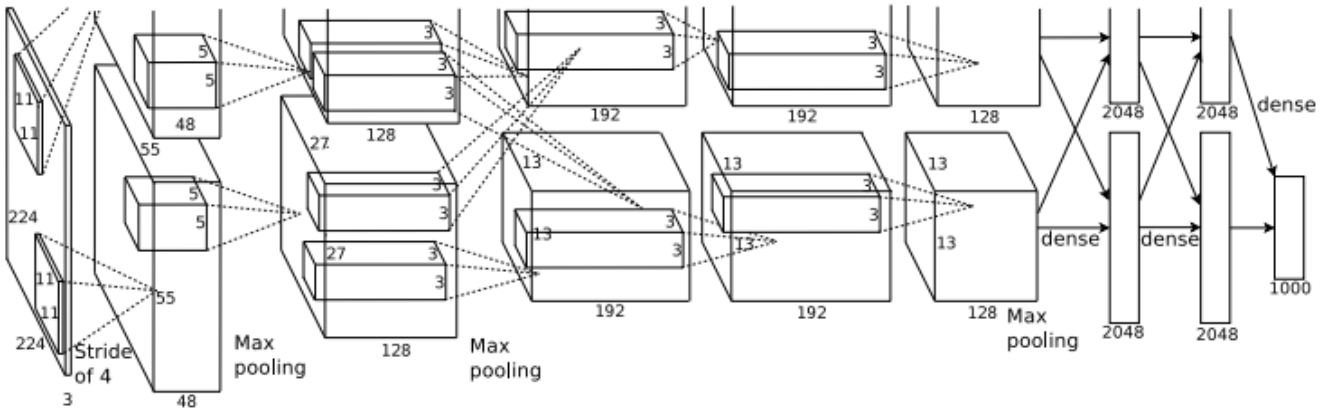
Questo consentirà l'avvio della fase di training. Per ciascuno dei file citati, risulta possibile associare diversi parametri mediante il file *config.py*.

Architetture utilizzate

Per la risoluzione del task sono stati utilizzati dei modelli trattati nel corso di Machine Learning : *Resnet, AlexNet, e SqueezeNet.*

AlexNet

E' una architettura neurale che nel 2012 vinse il ILSVRC. L'architettura consiste di otto livelli , di cui 5 convoluzionali .Gli ultimi tre livelli sono *fully-connected.*



Come si evince dal paper originale di AlexNet, a rendere importante questa rete sono alcuni aspetti :

1) Utilizzo della funzione di attivazione ReLU invece che tanh. Probabilmente oggi, la funzione ReLU (nelle nuove reti costruite) è ampiamente utilizzata, ma nel 2012 il suo utilizzo fu una novità.
2) Utilizzo di più GPU contemporaneamente per l'allenamento della rete. AlexNet fu allenata utilizzando una GTX 580 con 3GB di memoria , questo chiaramente poneva un limite al training della rete. Per questo motivo si scelse di parallelizzare il training su due GPU differenti che scambiano informazioni solo in alcuni livelli.
3) Overlapping pooling.
4) Data augmentation: In AlexNet sono state applicate due forme diverse di augmentation. La prima consiste nell'estrare una porzione dell'immagine 224x224 (dall'input 256x256). La seconda forma invece, consiste nell'alterare l'intensità dei canali RGB.

SqueezeNet

La motivazione per la quale è stata progettata questa rete, consiste nella costruzione di una rete con una accuracy alta, ma con un numero di parametri basso. Quest'ultimo punto fa sì che la rete, possa essere facilmente distribuita (ad esempio, risulta molto più facile e veloce trasferire un nuovo modello da un server ad una macchina autonoma). La strategia adottata da questa rete è la seguente:

1) I filtri 3x3 sono stati rimpiazzati da filtri 1x1, in modo tale da ottenere un numero di parametri fino a nove volte inferiore (rispetto all'utilizzo di filtri 3x3).
2) Viene decrementato il numero di input per i canali 3x3.
3) Le operazioni di *downsampling* vengono eseguite in fondo alla rete.

ResNet

In una rete neurale classica, ciascun livello ha il compito di *alimentare* il livello successivo. Invece, quello che accade in ResNet (è più in generale in una rete Residuale) è che ciascun livello comunica sia con il livello seguente, che con livelli che si trovano più in là nella rete. Come anche discusso nel paper di ResNet, la profondità

di una rete è di cruciale importanza, poiché una rete troppo profonda può portare al *Vanishing gradients.* Ancora, in una rete profonda, l'accuracy tende a saturarsi e di conseguenza a degradarsi rapidamente.

La soluzione è data dal blocco residuale descritto dall'immagine sottostante.

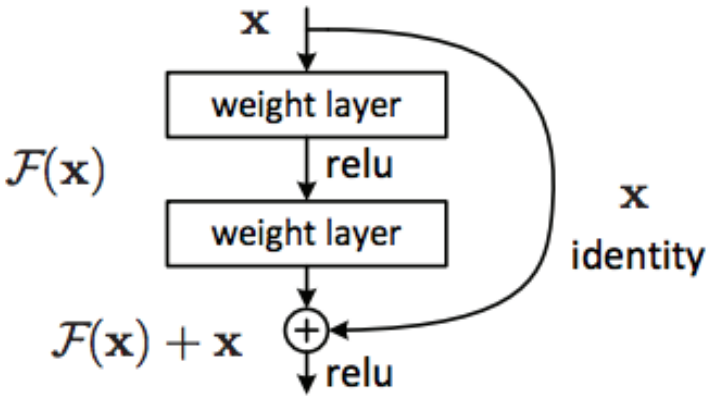
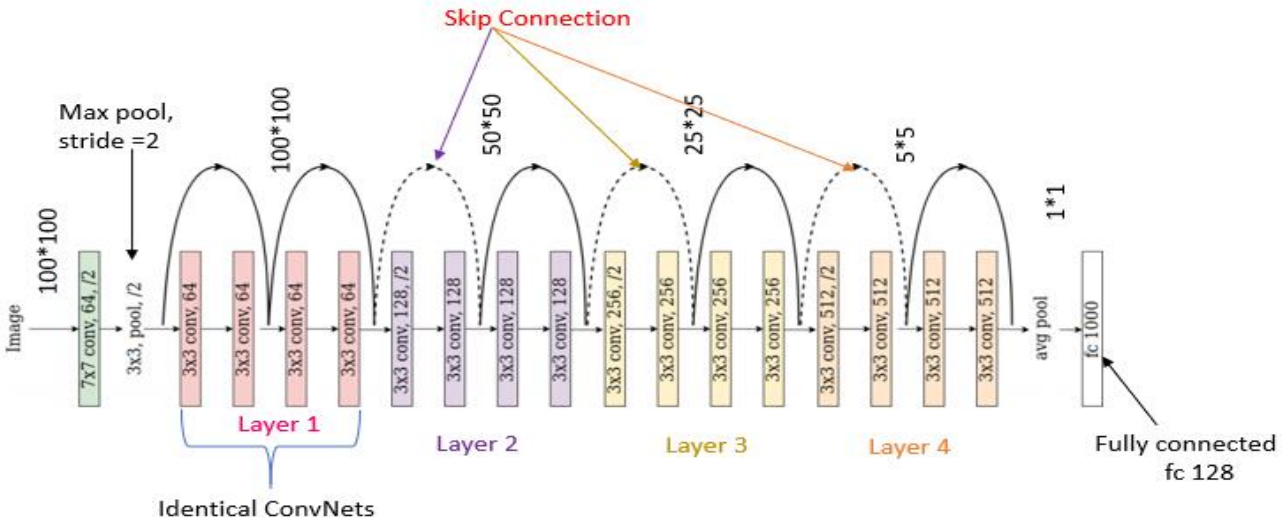


Figure 2. Residual learning: a building block.

Come si evince dall'immagine è possibile *saltare* alcuni livelli. Questo consente di connettere i livelli, saltandone alcuni. Esistono diverse varianti di ResNet (ResNet18, ResNet34 ecc…) che dipendono dal numero di livelli utilizzato.



ResNet-18 Architecture

Fruit 360 Input Image size= 100*100 px

Esperimenti

Sono state svolte diverse prove, sia suddividendo l'insieme dei frames come descritto nell'introduzione a questa relazione, che attivando/disattivando la variabile *feature extraction (*presente nel file config.py). In dettaglio, quando questa variabile booleana è impostata a *True*, saranno aggiornati i pesi dei soli livelli finali. Maggiori dettagli sull'implementazione sono consultabili nella documentazione di Pytorch.

Esperimento (Prova I)

Durante questa fase i frames sono stati suddivisi in due sottocartelle *train e test*. Il totale delle immagini estratte è pari a 10.813 . Di cui 15% test, 25% validation e 60% training set. Le trasformazioni applicate sono state le seguenti :

```
if data_augmentation:
    train_transforms = transforms.Compose([
                        transforms.Resize(224),
                        transforms.RandomCrop(224),
                        transforms.RandomVerticalFlip(),
                    #    transforms.RandomRotation(180),
                        transforms.ColorJitter(),
                        transforms.ToTensor(),
                        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

else:
    train_transforms = transforms.Compose([
                         transforms.Resize(224),
                         transforms.RandomCrop(224),
                         transforms.ToTensor(),
                         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])



validation_transforms = transforms.Compose([
                         transforms.Resize(256),
                         transforms.CenterCrop(224),
                         transforms.ToTensor(),
                         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

I Risultati Sono visibili in questa tabella (Epoche di allenamento pari a 30). I risultati sono da considerarsi con il campo Data augmentation uguale a True.

Modello	Epoche	Feature Extraction	LR	Tempo	Accuracy
AlexNet	50	True	0.005	1h20m	36%
AlexNet	30	False	0.005	1.44h	61.48%
SqueezeNet	50	True	0.001	2h43m	82.5%
SqueezeNet	30	False	0.005	3h30m	79.40%
ResNet	50	True	0.001	4h27m	85.89%
ResNet	30	False	0.005	1.5h	83%

Esperimento (Prova II)

In questo secondo test, i frames estrapolati sono stati inseriti nella cartella *all*, per un totale di 8100 immagini così suddivise : 20% test, 20% validation e 60% training set.

I risultati sono visibili nella tabella riportata sotto con il campo data augmentation uguale a True (Le trasformazioni applicate sono identiche a quelle viste sopra)

Modello	Epoche	Feature Extraction	LR	Tempo	Accuracy
AlexNet	50	True	0.01	51m	40.06%
AlexNet	50	False	0.001	57m	95.37%
SqueezeNet	50	True	0.001	51m	98.20%
SqueezeNet	50	False	0.001	59m	97.59%
ResNet	50	True	0.001	58m	97.77%
ResNet	50	False	0.001	1h06m	98.88%

Risultati

Se mettiamo a confronto i risultati della prima prova con il secondo esperimento, che i valori di accuracy ottenuti sono un po' più bassi. Di seguito viene mostrato l'andamento rispettivamente dei grafici della loss e train validation della prima prova:

- Arancione: ResNet
- Azzurro: SqueezeNet
- Grigio: AlexNet

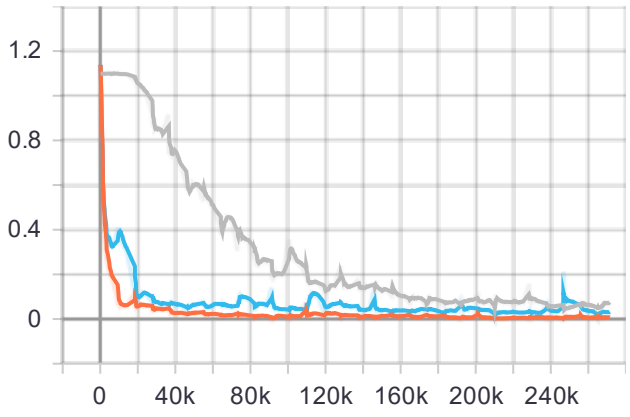


Figura 1.1 (Train loss)

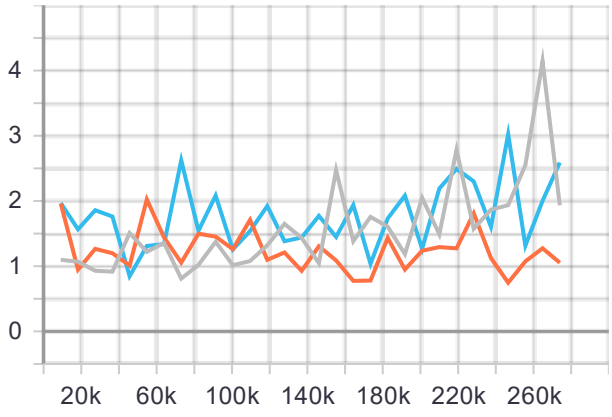


Figura 1.2 (Validation loss)

Come si può notare dai grafici mostrati sopra, mentre la loss di *training* converge, Le funzioni di *validation* oscillano e si distanziano dai valori della train loss. Questo evidenzia la presenza di overfitting.

Nel secondo esperimento i risultati che sono stati ottenuti sono totalmente diversi. In questo caso (come possiamo vedere dalle immagini sottostanti) , Sia validation che la train loss convergono e i livelli di accuracy (inseriti in tabella) sono molto alti. Tuttavia questo risultato ci trae "leggermente" in inganno per come sono stati suddivisi i dati in questo test . Infatti, nella fase di split dei dati alcuni frame facenti parte dello stesso video (e quindi molto simili tra loro!), sono stati suddivisi tra i vari set.

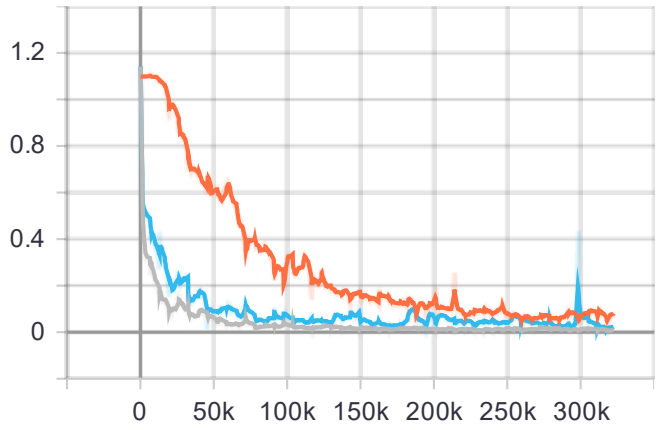


Figura 2.1 (train loss)

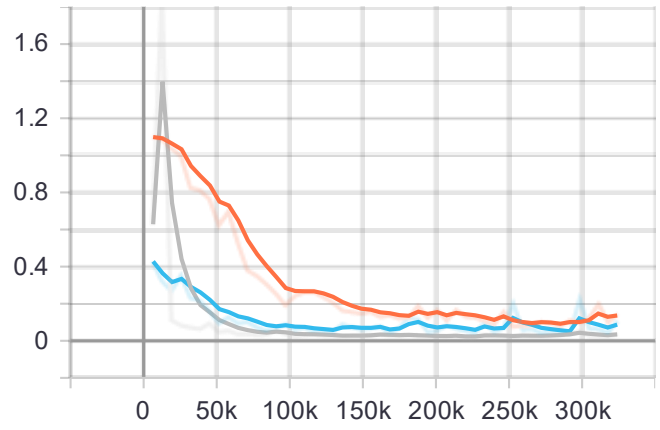


Figura 2.2 (validation loss)

- Arancione AlexNet
- Azzurro : SqueezeNet
- Grigio: ResNet

Sperimentalmente infatti, utilizzando l'interfaccia grafica, i modelli della seconda prova si comportano bene sulle immagini che hanno già visto ma fanno fatica a classificare correttamente altre immagini.

Ulteriori dettagli sull'andamento dei grafici (compresa l'accuracy) sono disponibili lanciando il comando :

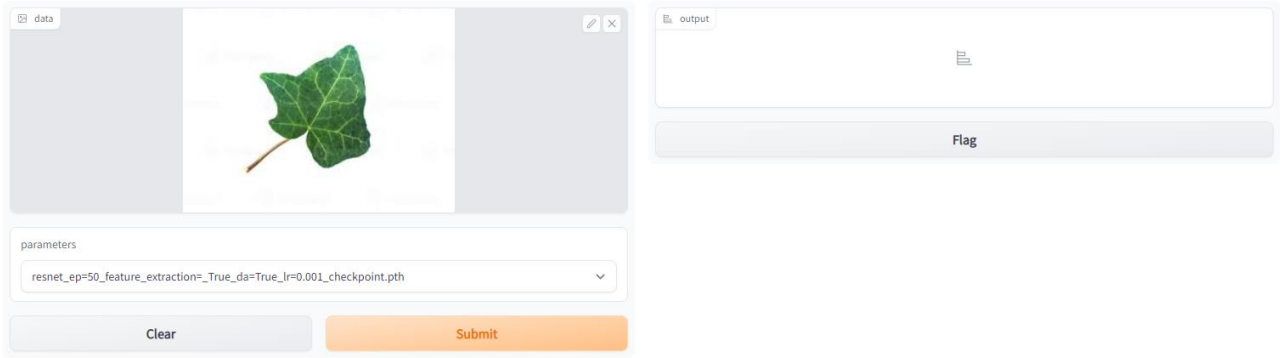
tensorboard –-logidr logs

"Logs" infatti è la cartella che racchiude tutti i logs generati durante la fase di training (quindi accuracy di validation/train e loss validation/train)

Interfaccia Grafica

E' stata implementata un'interfaccia mediante la libreria *Gradio.* Questa interfaccia può essere lanciata mediante il comando

Python -m src.gradio.gui



Dalla colonna sulla sinistra, è possibile scegliere un'immagine e , sotto, selezionare il modello che si vuole utilizzare per predire il tipo di foglia.

All'interno del file *config.py* è presente una variabile GRADIO_LOAD_CHECKPOINT_PATH che dovrà essere impostata per definire da quale cartella caricare i modelli allenati (ad esempio i parametri presenti già su git si trovano sotto la cartella *checkpoint/)*

Conclusioni

Dalle diverse prove svolte, si può dedurre innanzitutto che risulta di fondamentale importanza la corretta raccolta del dataset e la suddivisione successiva. In base al task assegnato infatti, si deve costruire un dataset che deve essere il più eterogeneo possibile. Fondamentale è anche giocare con i "parametri" di configurazione e sperimentare nuove configurazioni. Per costruire un progetto più ampio partendo da questo, sicuramente andrebbe ampliato l'insieme di dati, non solo da un punto di vista quantitativo, ma anche in ottica qualitativa (Raccogliendo quindi foglie magari dello stesso albero ma con forme e stati di salute diversa tra loro).

Bibliografia

1. Alexnet (https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf)
2. Resnet (https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf)
3. Squeezenet (https://arxiv.org/pdf/1602.07360.pdf)
4. FineTuning : (https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html)
5. Gradio (https://gradio.app/docs/)