Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Progetto finale

Fruit Recognition Project

Autore:
Andrea Guzzo - 761818 - a.guzzo1@campus.unimib.it

Luglio 2019

Abstract

Nel mondo del Deep Learning esistono molte tecniche per la creazione di reti neurali in grado di riconoscere oggetti a partire da immagini. La soluzione proposta vuole quindi fare una panoramica di possibili tecniche e differenti soluzioni volte alla realizzazione di una rete neurale in grado di riconoscere differenti tipi di frutta all'interno di immagini. L'utilizzo di diversi approcci ha permesso il confronto, l'identificazione e la sperimentazione di alcune pratiche illustrate all'interno del documento per fornire in ultima istanza alcuni suggerimenti e consigli sull'utilizzo delle tecniche e metodologie trattate.

1 Introduzione

Nel corso degli ultimi anni, il riconoscimento di oggetti all'interno di immagini è diventato un task comune e di più semplice risoluzione rispetto al passato. Grazie all'aumentare della potenza di calcolo disponibile, a nuovi tipi di architetture di reti neurali Deep, ad algoritmi sempre più efficienti implementati da framework di facile utilizzo e comprensione,è possibile ottenere delle ottime performance a partire da datasets disponibili gratuitamente online, adattando le soluzioni realizzate in modo sempre meno complesso ai differenti casi d'uso. A tal proposito per comprendere l'uso delle differenti tecniche e per sperimentare alcuni dei diversi approcci che è possibile adottare, per addestrare delle reti neurali in grado di riconoscere oggetti all'interno di immagini più o meno complesse, è stato realizzato uno studio contenente tre metodologie di lavoro differenti:

1. addestramento di reti neurali utilizzando tecniche di Transfer Learning [1]

2. utilizzo di tecniche di Layer Freezing per creare strutture personalizzate a partire da reti neurali esistenti

3. implementazione di un algoritmo convoluzionale progettato da zero.

Il confronto tra questi tre differenti approcci, sia in termini di performance, che in termini di velocità di realizzazione e di training ha permesso di delineare alcune pratiche di uso comune, alcune strategie che possono essere facilmente adattate in progetti di più larga complessità su tasks simili.

Come ultimo risultato del lavoro svolto, sono stati analizzati e discussi differenti approcci di rilascio e organizzazione delle architetture per consentire la messa in produzione di algoritmi previsionali in ambito Deep Learning.

2 Dataset

Per il progetto è stato utilizzato un dataset disponibile gratuitamente all'interno della piattaforma: Kaggle chiamato: moltean fruits 360 dataset [2] Questa collezione di dati viene spesso utilizzata per confrontare tecniche di Deep Learning, per studiare e confrontare approcci e metodi differenti. È composto da un numero di immagini totale pari a: 71125, con un traininset diviso in 53177 immagini contenti un frutto per ogni immagine con dimensione per ogni immagine di 100x100 pixels. Il validation set contiene al suo interno 17845 immagini sempre rappresentati un frutto per ogni immagine, mentre è presente all'interno dei dati disponibile anche un set di immagini contenente 103 immagini dove è possibile trovare più di un frutto all'interno di ogni singola immagine. Il numero di classi all'interno dei dati al momento dell'utilizzo e del progetto è di 103 classi differenti, ognuna appartenente ad un frutto differente. Il dataset non è comprensivo di tutti i tipi di frutta esistenti e viene costantemente aggiornato e arricchito.
Il progetto dunque assume che la numerosità delle classi all'interno del dataset sia sufficiente a coprire la maggior parte delle casistiche di frutti comunemente venduti in commercio e disponibili.
La particolarità di questo dataset è che sono disponibili una sequenza di immagini per ogni frutto ottenute con una serie di fotografie scattate in sequenza che consentono la visione del frutto da diversi tipi di angolazione.

Per favorire l'apprendimento delle reti neurali utilizzate, sono state applicate anche differenti trasformazioni al dataset:

- operazioni di ridimensionamento
- operazioni di data augmentation

Le operazioni di ridimensionamento implementate all'interno di funzioni generiche hanno consentito il facile ridimensionamento delle immagini in modo da adattare l'input necessario alle differenti architetture di reti neurali utilizzate. Per la data augmentation è stato implementato un metodo che consente la generazione e la manipolazione a partire da immagini esistente,

generando una sequenza di immagini con differenti attributi a partire da quelle di partenza: con differente orientamento, applicando zoom, rotazioni e spostamenti. La generazione di immagini a partire da quelle esistenti è una procedura comunemente nota che consente di manipolare il dataset di partenza per aggiungere nuovi record per il training degli algoritmi. [3] Grazie alla buona qualità del dataset disponibile non è stato necessario eseguire alcun tipo di data cleaning e organizzazione delle informazioni. È stato comunque realizzato un connettore che consente il download e il setup automatico del dataset utilizzato a partire da Kaggle, sfruttando le kaggle API. Le reti neurali convoluzionali pre-trainate sono state selezionate in quanto utilizzano come dataset di riferimento ImageNet che contiene al suo interno immagini e classi simili a quelle presenti nel dataset considerato.

Figure 1: Immagine del dataset appartenente alla classe: apple golden 1

Figure 2: Immagine del dataset appartenente alla classe: peach

Figure 3: Immagine del dataset appartenente alla classe: kiwi

3 Approccio metodologico

Per raggiungere l'obiettivo del progetto, ovvero la sperimentazione e la validazione di differenti tecniche e approcci per la classificazione delle immagini sono stati effettuati tre tipi di esperimenti. Per effettuare tali sperimentazioni è stata impiegata una singola e determinata architettura hardware che ha permesso di effettuare il training e la validazione degli algoritmi impiegati. Tutti i benchmark, analisi, prototipazione e training sono stati effettuati utilizzando python, keras come framework di riferimento basato su tensorflow GPU versione 1, una scheda video Nvidia GTX1070 con 8 gb di vram, 16 gb di ram del sistema e un processore i5-8300. Il codice realizzato è disponibile su github al seguente indirizzo: `https://github.com/JeyDi/FruitRecognition`

Il Transfer Learning è una tecnica largamente utilizzata nel mondo del Deep learning, diventando uno strumento chiave nel mondo delle applicazioni di intelligenza artificiale, in particolare nelle aree di Computer Vision, Natural Language Processing, Speech Recognition, consentendo di produrre risultati determinanti sia in termini di performance, sia in termini di adattamento che di velocità di messa in produzione. La chiave di questo tipo di modelli è appunto la facilità di realizzazione di modelli accurati in un tempo significativo. Elemento fondamentale di questo approccio è il riutilizzo di modelli pre-trainati, ovvero modelli di Deep Learning che sono stati addestrati su dataset simili per compiere tasks che si avvicinano a quelli da implementare. Un algoritmo in grado di compiere il riconoscimento di oggetti all'interno di immagini sfrutta come tipologia di architettura delle reti neurali convoluzionali solitamente dotate di uno schema architetturale comune: un input definito, una base convoluzionale che consente di eseguire una fase di feature extraction a partire dalle immagini e successivamente un classificatore in grado di classificare correttamente le immagini in base alle classi considerate per arrivare ad una predizione quanto più veritiera possibile.

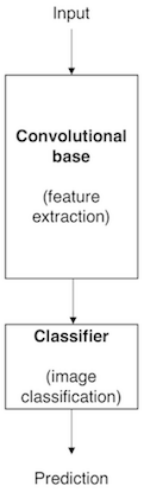


Figure 4: Architettura di una CNN

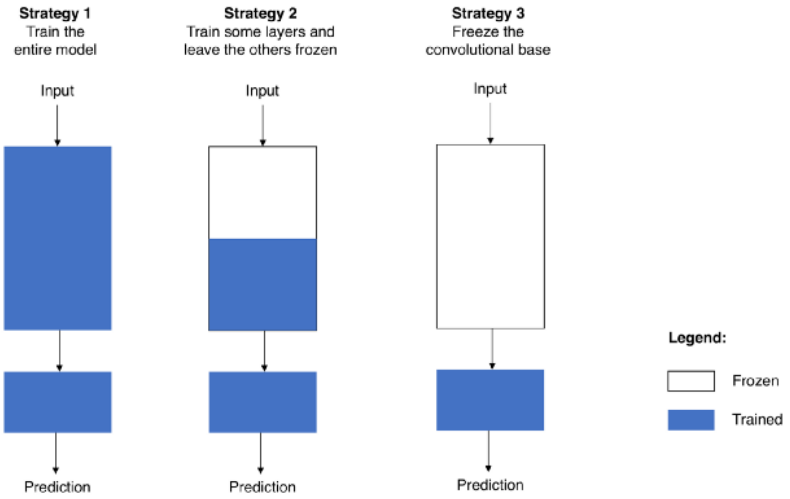


Figure 5: Strategie di transfer learning

Il numero di layers e di nodi all'interno della base convoluzionale può variare a seconda del tipo di architettura e ovviamente questo esempio vuol solamente essere una semplificazione rispetto all'implementazione reale di una

rete neurale convoluzionale. Una volta ottenuta una rete convoluzionale pre-trainata su un dataset e su un task simile si procede all'inserimento all'interno dell'architettura il proprio classificatore che consente di riconoscere le nuove immagini in input che verranno utilizzate rispetto alle nuove classi definite. Per fare questo si possono adottare tre tipi di approcci:

1. Fare il training dell'intero modello
2. Fare il training di alcuni layers, lasciando alcuni layer in stato frozen
3. Fare il freeze della base convoluzionale ritrainando solamente alcuni layer della parte convoluzionale

Ognuno di questi approcci ha alcuni vantaggi e svantaggi, negli esperimenti trattati è stato compiuto un esperimento per l'approccio 1 e un esperimento per l'approccio 2.

Per valutare l'efficacia di ogni esperimento realizzato e valutarne le performance al fine di selezionare l'approccio più conveniente in termini di sviluppo è stato adottato il seguente criterio di scelta:

- analisi delle performance del modello
- difficoltà di implementazione
- tempo di training

Grazie all'analisi di questi parametri è stato possibile determinare quale approccio fosse più conveniente per compiere task simili.

Successivamente all'implementazione e al training dei modelli si è anche scelto di salvare i modelli in formato ONNX in modo da esportare le reti addestrate anche su altri framework e per predisporre il rilascio della soluzione attraverso esperimenti dedicati di predizioni all'arrivo di nuove immagini.

3.1 Esperimento 1: Transfer Learning

Per il primo esperimento compiuto si è deciso di usare un approccio basato sul transfer learning con l'approccio numero 1, ovvero eliminando il classificatore della rete pre-trainata precedente, congelando la rete esistente e aggiungendo un nuovo classificatore basato sulle 103 classi delle immagini considerate nel dataset. La rete pre-trainata esistente utilizzata per l'esperimento

è ResNet50 [4] ottenuta dai modelli disponibili di Keras [5].
Resnet50 è una rete neurale convoluzionale addestrata su ImageNet ed è stata utilizzata in due modi: durante il primo test sono stati utilizzati i pesi della rete provenienti dal dataset originale di ImageNet, mentre per il secondo test non sono stati importati i pesi. Questo approccio è stato scelto in quanto il dataset a disposizione non era esageratamente grande, ma simile rispetto al dataset di partenza.
Le immagini del dataset considerato sono state dapprima rimodellate con una altezza e una larghezza di 224 pixel in modo da adattarsi all'input standard utilizzato per il training della rete neurale originale, con un batch size di 16 immagini prima di ogni ottimizzazione del gradiente [6].
A tal proposito è stato inserito in coda al modello un nuovo classificatore, ovvero è stato convertito l'ultimo layer di Resnet50 come un global average pooling 2D, aggiunto un nuovo layer denso fully-connected con 512 unità con funzione di attivazione Relu e infine un fully connected layer di output con 103 unità che rappresentano le classi all'interno del nostro dataset con funzione di attivazione softmax in quanto è necessario di essere in grado di discriminare o meno l'appartenenza ad una determinata classe.

A seguito di diversi tentativi con differenti combinazioni di loss functions e ottimizzatori, nella compilazione del modello si è scelto di utilizzare come loss function una categorical crossentropy [7] che viene largamente utilizzata e consigliata per problemi di classificazione dove è necessario avere un solo risultato corretto tra le classi disponibili. È stato utilizzando come ottimizzatore un: Stochastic gradient descent optimizer.

Come ulteriore esperimento finale di confronto tra modelli, è stato svolto anche un test utilizzando come modello InceptionV3 [8] sempre disponibile dai modelli pretrainati di Keras e addestrato su ImageNet.

L'architettura della rete a seguito delle operazioni di Transfer Learning è rimasta la stessa rispetto agli altri due modelli, si è reso necessario adattare l'input, trasformando le immagini con dimensione 299 pixels di altezza e 299 pixels di larghezza.

In questo modo è stato possibile verificare e validare il comportamento e le differenze utilizzando due modelli differenti con la stessa tecnica.

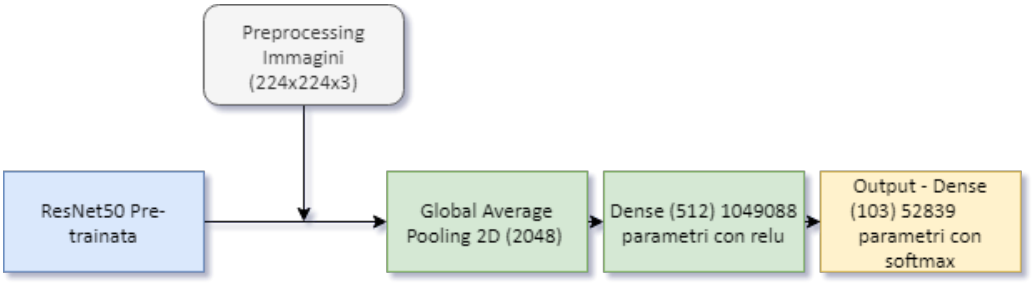


Figure 6: Disegno architetturale esperimento 1 Versione 1 ResNet50

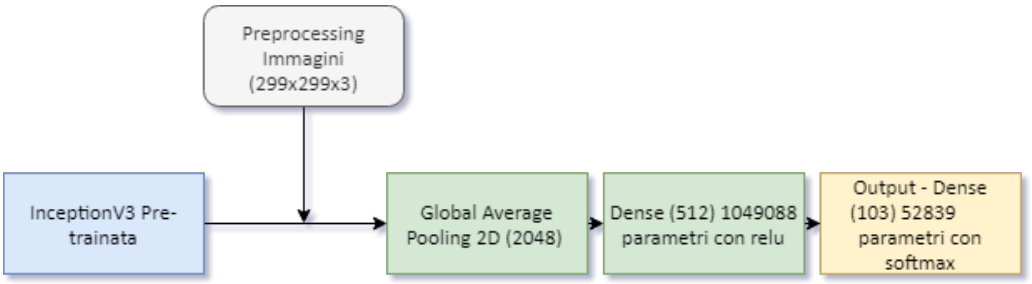


Figure 7: Disegno architetturale esperimento 1 Versione 1 InceptionV3

Le architetture implementate con le due tecniche illustrate si riassumono nelle figure 6 per quanto riguarda l'utilizzo di ResNet50, mentre 7 per quanto riguarda l'utilizzo di InceptionV3

3.2 Esperimento 2: Freeze Layers

Nel secondo esperimento si è deciso di utilizzare un approccio di Transfer Learning simile al primo, ma questa volta effettuando un congelamento parziale della rete, rimuovendo lo strato di classificazione finale che segue lo strato convoluzionale di feature extraction e attivando anche gli ultimi 4 layers convoluzionali. Differentemente rispetto all'esperimento 1 è stata però considerata un altro tipo di rete: VGG16 sempre ottenuta dai modelli pretrainati di Keras.
La rete utilizzata quindi ha attivi gli ultimi 4 layers convoluzionali appartenenti alla rete originale (tre layers convoluzionali e un max pooling finale), viene quindi inserito sopra ad essi una propria combinazione di layers così composti: un flattern layer a seguito della parte convoluzionale, un dense

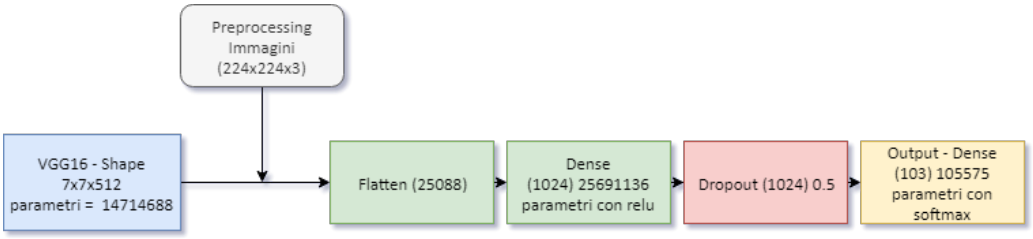


Figure 8: Disegno architetturale esperimento 2 Versione 1 con VGG16

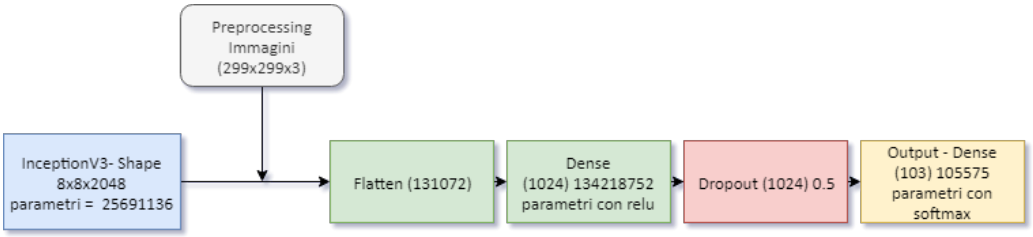


Figure 9: Disegno architetturale esperimento 2 Versione 2 con InceptionV3

layer di 1024 unità con funzione di attivazione: Relu, un layer di Dropout con tasso equivalente a 0.5 che corrisponde alla frazione di unità di input su cui effettuare il drop e infine l'ultimo strato denso formato da 103 unità equivalenti alle classi utilizzate sul nostro dataset, usando come funzione di attivazione una relu.
La loss function utilizzata è una categorical crossentropy come nell'esperimento 1, mentre l'ottimizzatore impiegato è sempre un Stochastic gradient descent optimizer.

Come ulteriore prova a sostegno di questo secondo esperimento, è stato eseguito un altro tentativo utilizzando come rete di riferimento pre-trainata non più VGG16, ma InceptionV3 sempre con i pesi relativi a ImageNet, con un input differente, in modo da adattare le immagini alla nuova rete, ma con la stessa struttura precedente.
L'architettura relativa all'esperimento 2 versione 1 con VGG16 è rappresentata nell'immagine 8 , mentre l'architettura per il secondo test sempre dell'esperimento 2 è rappresentata dall'immagine 9

Layer Type	**Dimension**	**Output**
Convolutional	5 x 5 x 4	16
Max Pooling	2 x 2 - Stride 2	-
Convolutional	5 x 5 x 16	32
Max Pooling	2 x 2 - Stride 2	-
Convolutional	5 x 5 x 32	64
Max Pooling	2 x 2 - Stride 2	-
Convolutional	5 x 5 x 64	128
Max Pooling	2 x 2 - Stride 2	-
Fully Connected	5 x 5 x 128	1024
Fully Connected	1024	256
Softmax	256	60

Table 1: Tabella di configurazione della rete neurale convoluzionale creata a mano

3.3 Esperimento 3: Implementazione rete neurale

Dopo aver effettuato diversi esperimenti utilizzando tecniche di transfer learning si è deciso di implementare anche una rete neurale partendo dal paper di riferimento associato al dataset [9] in modo da confrontare i risultati e verificare se un approccio completamente manuale fosse migliore rispetto all'utilizzo di reti neurali esistenti, sia in termini di facilità di realizzazione, tempo di training che di risultati effettivi.
Le immagini usate per il training e per la validazione sono state trattate sempre con gli stessi metodi precedenti con una risoluzione di 100x100 pixels di altezza e larghezza.
La configurazione della rete è illustrata nella seguente tabella:

Per il modello utilizzato sono state impiegate come funzione di loss: categorical crosse ntropy in quanto i nostri target da predirre sono 103 classi categoriche numeriche tramite softmax. Come ottimizzatore si è scelto di utilizzare: adam ovvero adaptive moment estimation che ha risultati migliori rispetto ad altri metodi stocastici di ottimizzazione [10].

Il disegno architetturale della rete implementata con le relative specifiche strutturali è rappresentato dall'immagine 10

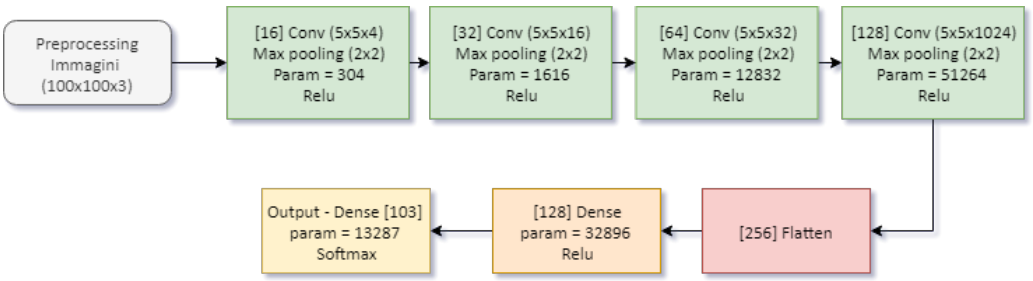


Figure 10: Disegno architetturale esperimento 3

4 Risultati e valutazione

Vengono quindi riportate le soluzioni e i risultati ottenuti conseguentemente all'apprendimento degli algoritmi.

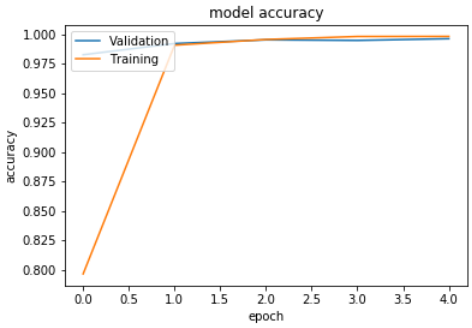
4.1 Esperimento 1

4.1.1 Versione 1

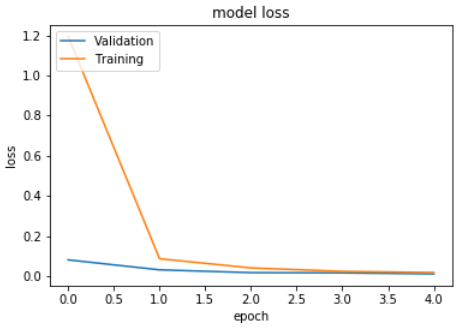
Per l'esperimento 1 sono stati ottenuti i seguenti risultati

Dataset	**Accuracy**	**Loss**
Trainingset	0.9983263440961317	0.01822905129225315
Validation	0.9963575231157187	0.011467085026564319

Table 2: Tabella dei risultati dell'esperimento 1 versione 1 con ResNet50



(a) Esperimento 1 v1, risultati accuratezza



(b) Esperimento 1 v1, risultati loss

Il tempo di training della rete sull'architettura hardware utilizzata è di:

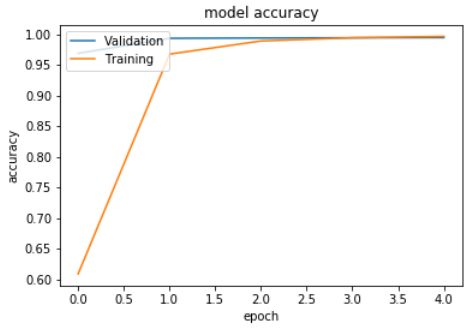
4.1.2 Versione 2

Per l'esperimento 1 versione 2, usando la rete pre-trainata InceptionV3 definita precedentemente sono stati ottenuti i seguenti risultati

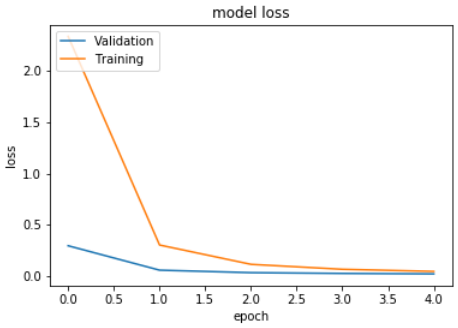
Dataset	**Accuracy**	**Loss**
Trainingset	0.9967091035598097	0.04332224593808697
Validation	0.9947884561501821	0.019240403047854925

Table 3: Tabella dei risultati dell'esperimento 1 versione 2 con InceptionV3

Mentre i grafici relativi all'andamento delle performance rispetto al numero di epoche utilizzate sono i seguenti



(a) Esperimento 1 v2, risultati accuratezza



(b) Esperimento 1 v2, risultati loss

Il tempo medio di training di una rete sull'architettura hardware utilizzata è di circa due ore e 30 minuti, mentre il modello in formato onnx su disco pesa: all'incirca 200 MB in entrambi i casi.

4.2 Esperimento 2

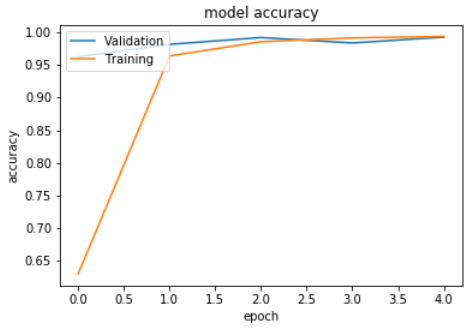
4.2.1 Versione 1

Per l'esperimento 2 versione 1, usando la rete pre-trainata VGG16 definita precedentemente sono stati ottenuti i seguenti risultati

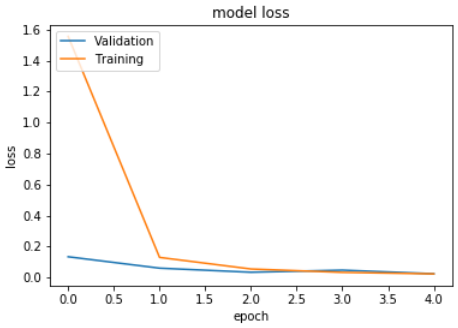
Dataset	**Accuracy**	**Loss**
Trainingset	0.9935310378547116	0.023791958318985264
Validation	0.9924908938077893	0.02455971877568624

Table 4: Tabella dei risultati dell'esperimento 2 versione 1 con VGG16

Mentre i grafici relativi all'andamento delle performance rispetto al numero di epoche utilizzate sono i seguenti



(a) Esperimento 2 v1, risultati accuratezza



(b) Esperimento 2 v1, risultati loss

Il tempo di training della rete sull'architettura hardware utilizzata è di:

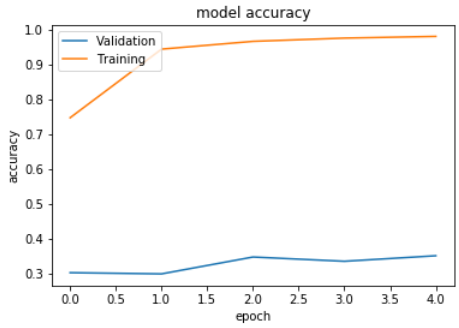
4.2.2 Versione 2

Per l'esperimento 2 versione 2, usando la rete pre-trainata InceptionV3 definita precedentemente sono stati ottenuti i seguenti risultati

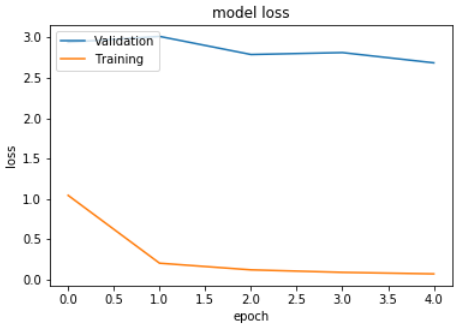
Dataset	**Accuracy**	**Loss**
Trainingset	0.9823796002030953	0.06881602044664113
Validation	0.3503502381636202	2.6863035547729965

Table 5: Tabella dei risultati dell'esperimento 2 versione 2 con InceptionV3

Mentre i grafici relativi all'andamento delle performance rispetto al numero di epoche utilizzate sono i seguenti



(a) Esperimento 2 v2, risultati accuratezza



(b) Esperimento 2 v2, risultati loss

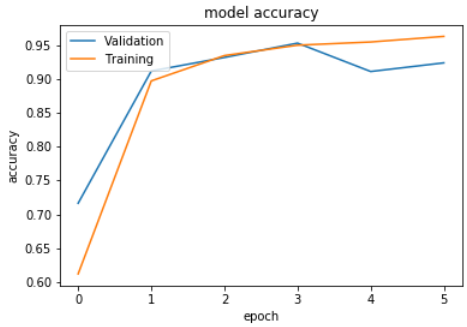
Il tempo medio di training di una rete sull'architettura hardware utilizzata è di circa quattro ore, mentre il modello in formato onnx su disco pesa: all'incirca 160 MB in entrambi i casi.

4.3 Esperimento 3

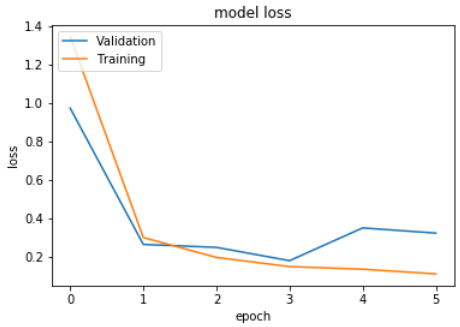
Per l'esperimento 3 realizzato con la rete costruita artigianalmente e definita precedentemente sono stati ottenuti i seguenti risultati

Dataset	**Accuracy**	**Loss**
Trainingset	0.964025800629212	0.11182551736345149
Validation	0.9247408237601569	0.323432678943311

Table 6: Tabella dei risultati dell'esperimento 3 con la rete costruita artigianalmente



(a) Esperimento 3, risultati accuratezza



(b) Esperimento 3, risultati loss

Il tempo medio di training della rete sull'architettura hardware utilizzata è di circa tre ore, mentre il modello in formato onnx su disco pesa: 443 KB

Nel secondo esperimento dire che ci sono problemi di: forgetting. All'epoca 1 e 2 ho un salto sulla loss, in questo momento quindi il modello sta overfittando. Per superare questo problema è necessario fare: gradual unfreezing. Altra tecnica: chain-thaw. I primi esempi hanno un errore molto alto, iniziano a fare back propagation e cambia tanto la rappresentazione astratta.

Terzo modello, risultati buoni e interessanti, non tanto quanto transfer learning normale.

5 Discussione dei risultati

I risultati ottenuti dalle reti sul dataset utilizzato sono molto soddisfacenti, sia in termini di accuratezza che in termini di loss. Si riportano le due tabelle comparative finali con i risultati ottenuti da tutte le reti impiegate negli esperimenti 7 8

Rete	**Training acc**	**Training loss**
Exp1, ResNet50	0.9967091035598097	0.04332224593808697
Exp1, InceptionV3	0.9967091035598097	0.04332224593808697
Exp2, VGG16	0.9935310378547116	0.023791958318985264
Exp2, InceptionV3	0.9823796002030953	0.06881602044664113
Exp3, Custom Model	0.964025800629212	0.11182551736345149

Table 7: Confronto dei risultati ottenuti sul trainingset tra tutti i modelli

Rete	**Validation acc**	**Validation loss**
Exp1, ResNet50	0.9947884561501821	0.019240403047854925
Exp1, InceptionV3	0.9947884561501821	0.019240403047854925
Exp2, VGG16	0.9924908938077893	0.02455971877568624
Exp2, InceptionV3	0.3503502381636202	2.6863035547729965
Exp3, Custom Model	0.9247408237601569	0.323432678943311

Table 8: Confronto dei risultati ottenuti sul validation set tra tutti i modelli

Dalle tabelle e dai valori di training e loss possiamo evidenziare come le reti ottenute grazie al primo esperimento di transfer learning, utilizzando le reti così come sono state salvate, sono sicuramente migliori rispetto agli altri approcci utilizzati, anche se con piccole differenze. Ci sono grandi differenze invece riguardo ai tempi di training e al peso dei modelli finali una volta esportati in formato onnx. Il modello più leggero in termini di spazio occupato su disco è sicuramente la rete ottenuta grazie all'implementazione custom che però presenta dei valori di loss più alti rispetto agli altri modelli. Questo parametro di confronto di spazio su disco è molto importante perchè ci permette di identificare un modello leggero e performante da poter utilizzare su numerose applicazioni, soprattutto in ambito mobile.

È comunque interessante contestualizzare anche i grafici di loss e accuracy presentati nel capitolo precedente per l'esperimento 2, rispetto all'esperimento 1. Nonostante il numero di epoche impiegate sia basso per l'esperimento 2 possiamo constatare come, nell'esperimento 2 versione 2 utilizzando InceptionV3, i risultati siano profondamente differenti in termini di performance sul validation set rispetto all'esperimento 2 versione 1 e rispetto all'esperimento 1. Questo comportamento è sicuramente causato dal basso numero di epoche impiegato su una rete molto più grande di VGG16 impiegata nella versione 1 dell'esperimento 2. Avendo a disposizione una rete più grande e complessa è necessario utilizzare un tempo di training superiore, su un numero di epoche superiore oltre che ad aumentare la profondità di layer non congelati in modo da favorire l'aggiornamento di ulteriori layers rispetto alle immagini impiegate.

È interessante notare come nell'esperimento 3 ci sia al crescere delle epoche un leggero overfitting sul validation test rispetto al training. Sicuramente un tuning degli iperparametri utilizzati dalla rete potrebbe essere di grande aiuto nell'ottimizzare le performance generali.

5.1 Valutazione degli errori

Considerando i modelli impiegati nell'esperimento 1 e nell'esperimento 3 è importante validare ed evidenziare quali sono gli errori all'interno del validation set utilizzato per verificare le performance del modello. In questo modo si è in grado di definire su quali immagini i modelli sbagliano più frequentemente e per quale tipo di input.

Il numero di immagini valutate non correttamente per l'esperimento 1 su una predizione del validation set è di 47 su 17845 immagini. Il rapporto di errori di classificazione sempre sullo stesso dataset per l'esperimento 3 è di 53 su 17845 immagini.

Alcuni esempi di immagini non correttamente classificate sono le seguenti (in entrambi i casi):

Figure 16: Errore nella predizione: ground truth: Apple Red 2, prediction: Apple Braeburn, confidenza: 0.759

Figure 17: Errore nella predizione: ground truth: Pear, prediction: Apple Golden 3, confidenza: 0.452

Figure 18: Errore nella predizione: ground truth: Pepino, prediction: Grape White 2, confidenza: 0.481

Figure 19: Errore nella predizione: ground truth: Strawberry Wedge, prediction: Avocado pipe , confidenza: 0.699

5.2 Sviluppi futuri

Gli esperimenti di transfer learning hanno dato degli ottimi risultati, c'è ancora ampio spazio di miglioramento per quanto riguarda i modelli trattati nell'esperimento 2. Per quanto riguarda l'esperimento 3 invece un progressivo fine-tuning dei parametri del modello potrebbe aiutare molto nel generare delle performance migliori rispetto a quanto ottenuto con i primi test e quindi generare un modello definitamente più ottimale e funzionale per qualsiasi tipo di applicazione. Il successivo passo dopo questa sperimentazione sarà l'implementazione di un'architettura end-to-end di rilascio dei modelli in modo da favorire il loro utilizzo su diversi dispositivi, risultato ottenibile grazie all'implementazione in fase di training del salvataggio dei modelli in formato onnx. L'implementazione di questa architettura potrebbe essere di grande vantaggio per la realizzazione di macchinari in grado di distinguere la frutta da raccogliere all'interno di frutteti e campi consentendo, attraverso l'uso di appositi macchinari, la raccolta automatizzata dei prodotti agroalimentari. Potrebbe essere infine utile una sperimentazione con altri modelli

pre trainati in modo da condurre un test più accurato e di larga scala anche su approcci differenti che potrebbero fornire risultati migliori rispetto a quelli impiegati per questo test.

6 Conclusioni

Gli esperimenti compiuti con le procedure di transfer learning hanno dato degli ottimi risultati. Le tecniche utilizzate sono sicuramente un utile e valido approccio per costruire ottimi modelli velocemente e con bassa capacità di calcolo, anche se il rischio di overfitting è sempre presente e bisogna prestare molta attenzione all'adattabilità con il proprio dataset, il proprio dominio e le proprie classi, in base al modello scelto.
L'implementazione manuale del modello basata sul paper di riferimento ha dato dei buoni risultati comparativamente ai modelli riutilizzati, il problema è che il tempo di training e le risorse richieste sono comunque più alte rispetto alle tecniche di Transfer Learning trattate con il primo esperimento. La soluzione artigianale comunque ha rivelato buone performance e una rete più piccola e snella in grado di adattarsi bene a contesti in cui è importante avere dei modelli funzionali impiegabili su determinati casi d'uso e pratici come la realizzazione di un'applicazione web o mobile che necessita di performance sufficienti ma poco spazio occupato oppure nell'utilizzo di sistemi embedded. Lo standard onnx adottato e sperimentato per salvare le reti si è rivelato un approccio vincente per l'operazionalizzazione dei processi di training e prediction, unico aspetto negativo è la poca adattabilità con il framework di Keras per quanto riguarda la fase di prediction. Nel complesso, le soluzioni adottate e le architetture impiegate si sono rivelate corrette per rispondere alla domanda iniziale, ovvero trovare delle tecniche in grado di classificare correttamente differenti tipi di frutta a partire da immagini.

References

[1] Y. B. H. L. Jason Yosinski, Jeff Clune, "How transferable are features in deep neural networks?" p. pp. 9, 2014.

[2] "Horea muresan, mihai oltean, fruit recognition from images using deep learning," *Informatica*, vol. 10, pp. pp. 26–42, 2018.

[3] J. W. Luis Perez, "The effectiveness of data augmentation in image classification using deep learning," p. pp. 8, 2017.

[4] S. R. J. S. Kaiming He, Xiangyu Zhang, "Deep residual learning for image recognition," 2015.

[5] "Resnet keras documentation," https://keras.io/applications/#resnet.

[6] "Batch size explanation," https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[7] "Categorical crossentropy explanation," https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

[8] S. I. J. S. Z. W. Christian Szegedy, Vincent Vanhoucke, "Rethinking the inception architecture for computer vision," p. pp. 10, 2015.

[9] M. O. Horea Muresan, "Fruit recognition from images using deep learning," p. pp. 53, 2018.

[10] "Adam optimizer explanation," https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.