

# Progetto ML e SII: Object recognition

**Riccardo D'Angelo Gargano**

**Alessandro Martinelli**

**Andrea Serrecchia**

## 1: Introduzione

Il progetto ha previsto la creazione, la sperimentazione e l'uso di reti neurali e convolutive per il riconoscimento di oggetti presenti nelle immagini. Il nostro lavoro si è evoluto in quattro direzioni:

- Creazione di una rete neurale per il riconoscimento di numeri scritti a mano facendo uso del noto dataset MNIST.
- Creazione di una nuova rete convolutiva per il riconoscimento di quattro (automobile, aereo, cane e cavallo) delle dieci categorie fornite dal dataset CIFAR-10.
- Creazione di una rete convolutiva partendo da un modello già addestrato, facendo uso del transfer learning.
- Utilizzo di Tensorflow a scatola nera per il riconoscimento di oggetti in tempo reale, inquadrandoli con un telefono o una webcam.

Nei prossimi capitoli saranno analizzati nel dettaglio il secondo e il terzo punto. Il riconoscimento di valori numerici è ormai un problema ampiamente superato, anche facendo uso di semplici reti neurali, motivo per cui non è fornita una descrizione dell'approccio che abbiamo perseguito. Con riferimento al quarto punto, nonostante non sia stato necessaria la scrittura del codice della rete, è stata proposta una modifica che permette il riconoscimento real time degli oggetti.

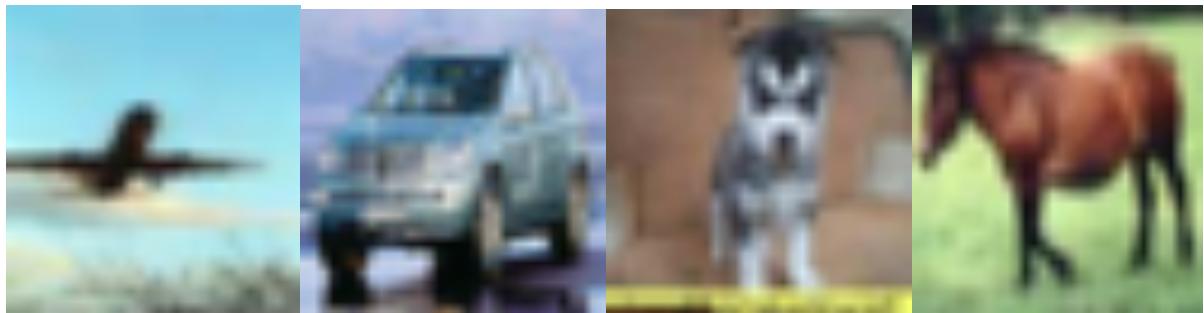
## 2: Creazione di una nuova rete convolutiva

Si è scelto di usare solamente 4 classi delle 10 disponibili poiché la rete è creata da zero e non è pre-addestrata, risultando l'addestramento risulterebbe troppo pesante. Il dataset usato è più grande rispetto a quello impiegato nella rete che usa il transfert learning ed è composto da:

- Training set: 5000 immagini per classe (opportunamente diviso in training e validation)
- Test set: 1000 immagini per classe

Le immagini, inoltre, sono pre-elaborate e portate ad una risoluzione molto bassa: 32x32 pixel.

La figura successiva mostra un esempio del dataset utilizzato.



Le etichette delle immagini sono rappresentate tramite la **codifica one-hot**.

La rete base usata su questo dataset è costituita da 2 strati convolutivi, intervallati da strati di **pooling** e **dropout** e seguiti da 2 strati **fully connected**, per poi finire con la **softmax**. La struttura è la seguente:

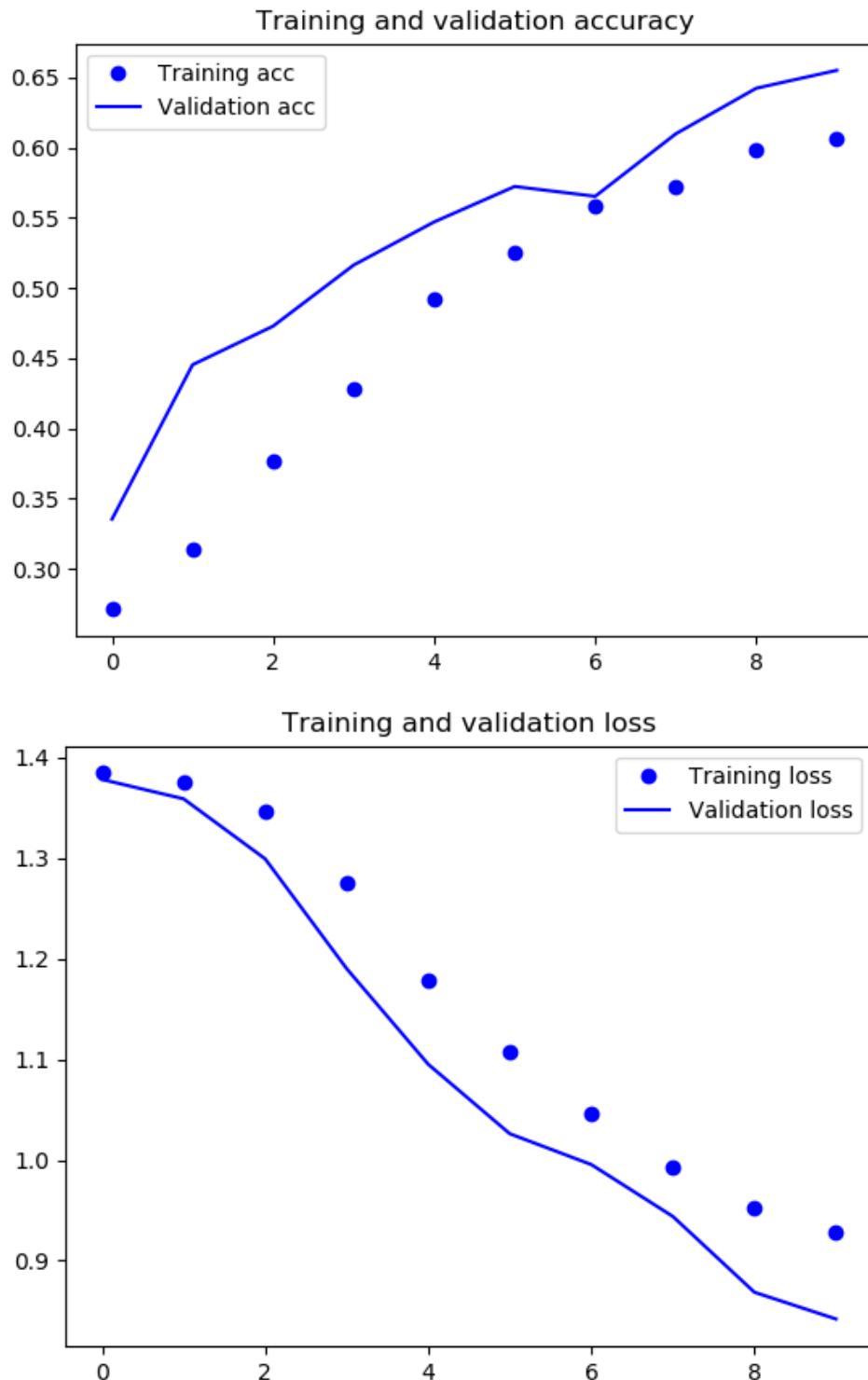
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 32, 32, 64)	320
max_pooling2d	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	8224
max_pooling2d_1	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
=====		

dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 4)	1028
=====		
Total params: 1,189,988		
Trainable params: 1,189,988		
Non-trainable params: 0		
<hr/> Train on 16000 samples, validate on 4000 samples		

- Il primo strato è un operatore di convoluzione con una finestra di dimensione 3x3, funzione di attivazione ‘relu’, stride di 2 pixel, padding di tipo ‘same’ e dà in output 64 filtri. Questo strato convolutivo è seguito da uno strato di max\_pooling 2x2 che dimezza la profondità dell’immagine e il numero dei pesi. Inoltre, è applicata la tecnica del dropout con parametro 0.2 che “spegne” casualmente il 20% dei nodi per evitare di incorrere in dipendenze indesiderate tra i pesi che causano overfitting.
- Il secondo strato è anch’esso un operatore di convoluzione con una finestra di dimensione 2x2, funzione di attivazione ‘relu’, stride di 1 pixel e, quindi, nessun padding. In questo caso dà in output 32 filtri. Tale strato è seguito da un max\_pooling dello stesso tipo del precedente ed è usata anche in questo caso la tecnica del dropout con parametro 0.2.
- Dopo gli strati convolutivi viene appiattita l’immagine: le informazioni contenute nei 32 filtri di dimensione 8x8 vengono riportate in un unico vettore 2048x1 per poter essere usate come input per gli strati fully-connected.
- Il primo strato fully-connected è composto da 512 neuroni che usano la funzione di attivazione ‘relu’.
- Il secondo strato fully-connected è composto da 256 neuroni che usano la funzione di attivazione ‘relu’.
- L’ultimo strato è rappresentato dalla softmax che dà in output la probabilità di appartenenza di un’immagine alle diverse classi.

Modifiche alla struttura della rete non hanno portato significativi miglioramenti dell’accuratezza o della loss. Inoltre, tutte le prove sono state fatte con 10 epoch poiché altri cicli di addestramento non portavano a miglioramenti nel modello. La funzione di loss usata è la categorical\_crossentropy ovvero la **cross-entropy** che è utilizzata quando si hanno più di 2 classi. Quello che si è cercato di variare per migliorare l’accuratezza sono stati gli iperparametri per l’ottimizzazione.

## 2.1: Addestramento del modello



Questi grafici sono relativi all'addestramento del modello, usando la funzione stochastic gradient descent senza specificare i diversi iperparametri, utilizzando una dimensione dei mini-batch pari a 32 sample. I risultati ottenuti sono i seguenti:

- Tempo di addestramento: 1'28"
- Training loss: 0.9281, training accuracy = 0.6061

- Validation loss: 0.8420, validation accuracy = 0.6552
- Test loss: 0.84252, test accuracy: 0.65575

## **2.2 Variazione parametri del modello:**

- Kernel size = 2 e strides 1 nel primo strato convolutivo. L'addestramento risulta più lento con un tempo totale di 5'15'' dovuto al maggior numero di operazioni da fare nel primo strato. D'altro canto, però, il modello risulta essere più efficace con una loss di 0.7652 e un'accuracy di 0.6925 sul validation set.
- Momentum (nesterov): introduzione del parametro momentum pari a 0.9. A parità di epoche ci si avvicina molto più all'ottimo ottenendo una loss di 0.4476 e un'accuracy 0.82825 sul validation set.
- Mini-batch size = 64: questo velocizza l'addestramento e otteniamo valori di efficacia del modello molto simili a quelli ottenuti in precedenza.
- La variazione di altri parametri della funzione SGD (learning rate, learning rate decay) non porta a ulteriori miglioramenti.
- Un ulteriore miglioramento, invece, avviene usando una diversa funzione di ottimizzazione. La funzione in questione è la 'adam' con la quale otteniamo una loss di 0.3665 e un'accuracy di 0.8953 sempre sul validation set.

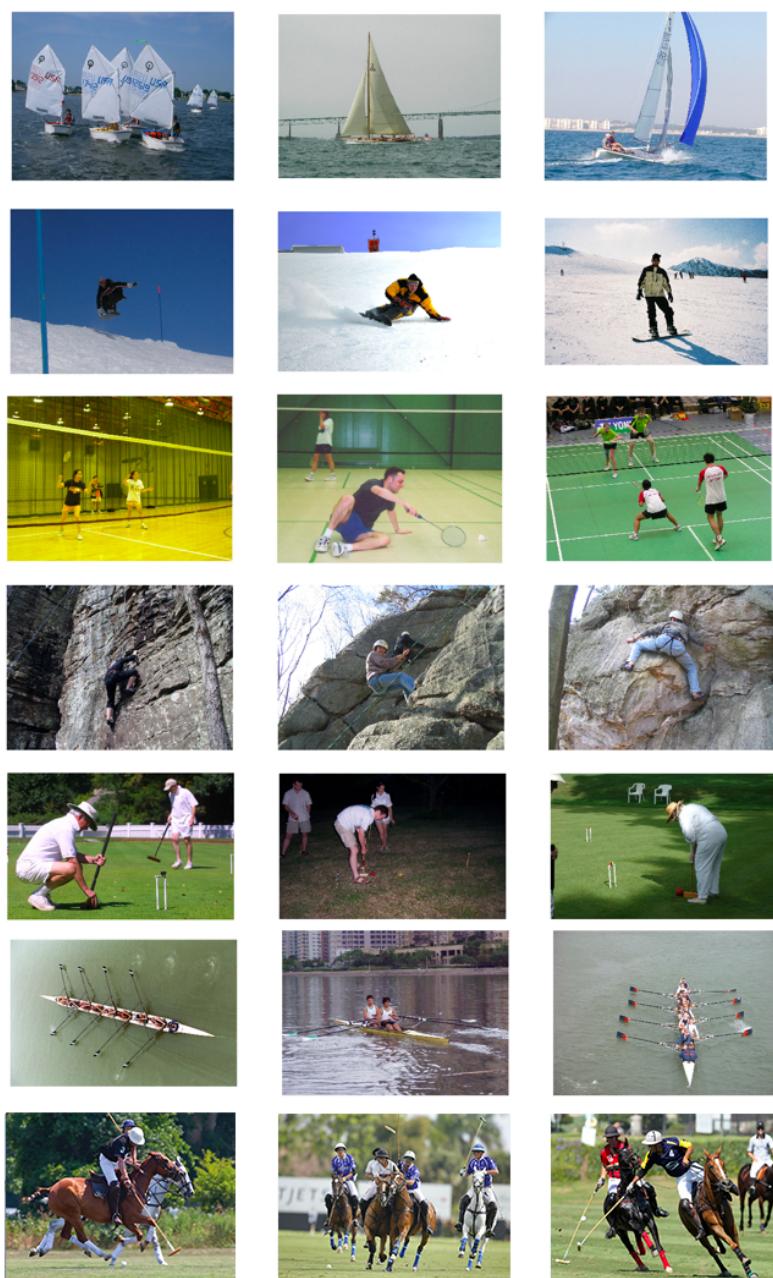
### **3: Uso del transfer leaning**

Utilizzando il transfer leaning si è cercato di creare una rete in grado di riconoscere lo sport praticato dai soggetti presenti nelle immagini. Gli sport considerati appartengono a 7 categorie: canottaggio, badminton, polo, snowboard, croquet, barca a vela e arrampicata. Il dataset è stato creato utilizzando il progetto “UIUC Sports Event Dataset” della Stanford University, aggiungendo altre immagini ottenute da Google con semplici ricerche. Le immagini sono quindi suddivise in:

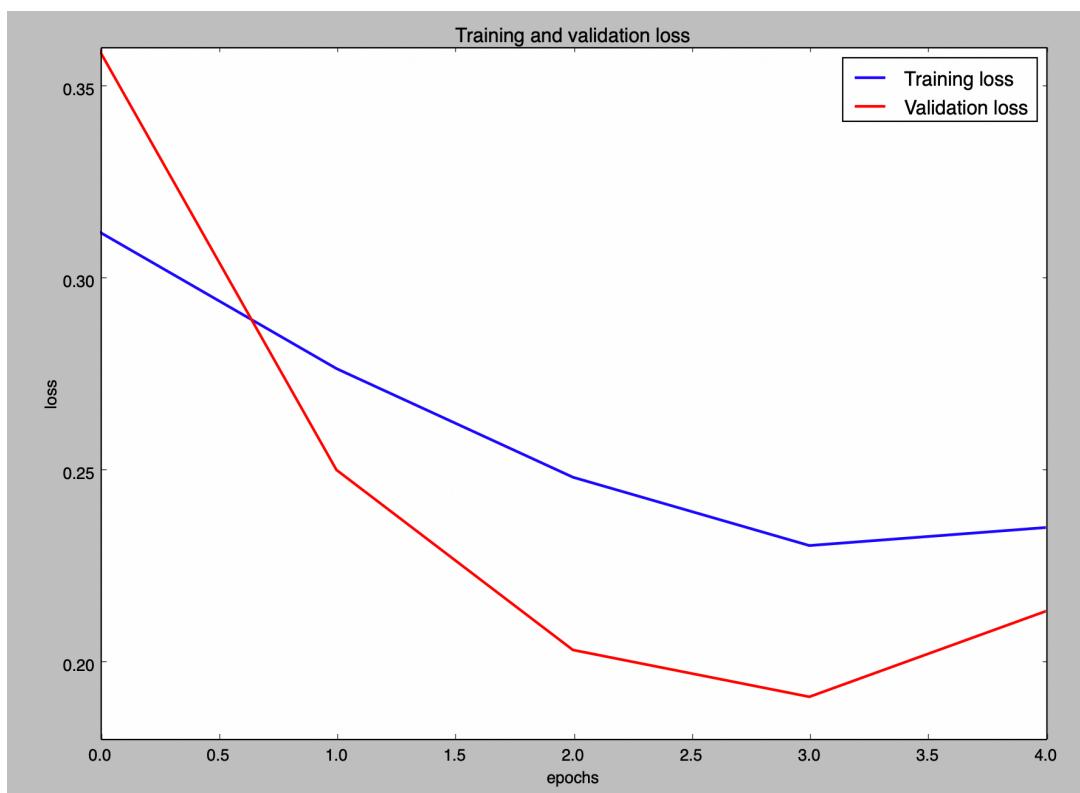
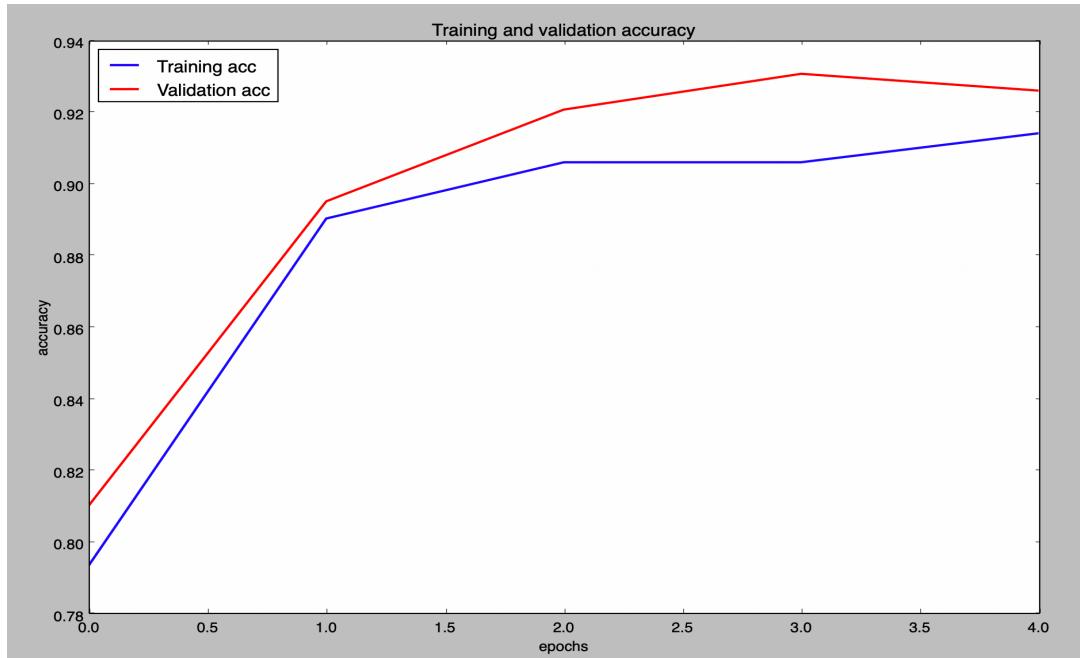
- Training set: 926 immagini
- Validation set: 211 immagini
- Test set: 515 immagini

Esse hanno dimensioni variabili e per tale motivo sono pre processate, trasformando la dimensione in 224x224 pixel.

L'immagine successiva mostra un esempio del dataset utilizzato



Il modello da cui siamo partiti è MobileNet, addestrato utilizzando ImageNet. Da questa rete è stato eliminato l'ultimo strato e sono stati introdotti tre livelli fully connected, contenenti, rispettivamente, 1024, 1024 e 512 neuroni, attivati da una relu. Poiché gli sport da classificare sono sette, è presente una softmax. L'addestramento degli strati aggiuntivi è stato effettuato solo con 5 epoch, utilizzando un batch size pari a 32 e, anche in questo caso, usando come funzione di loss la categorical\_crossentropy. Pur avendo utilizzato un numero limitato di epoch, la rete presenta delle buone prestazioni, ottenendo un'accuratezza di circa il 90% e una loss pari a 0,38 sul test set. Gli andamenti sul training e sul validation set sono mostrati dai grafici successivi:



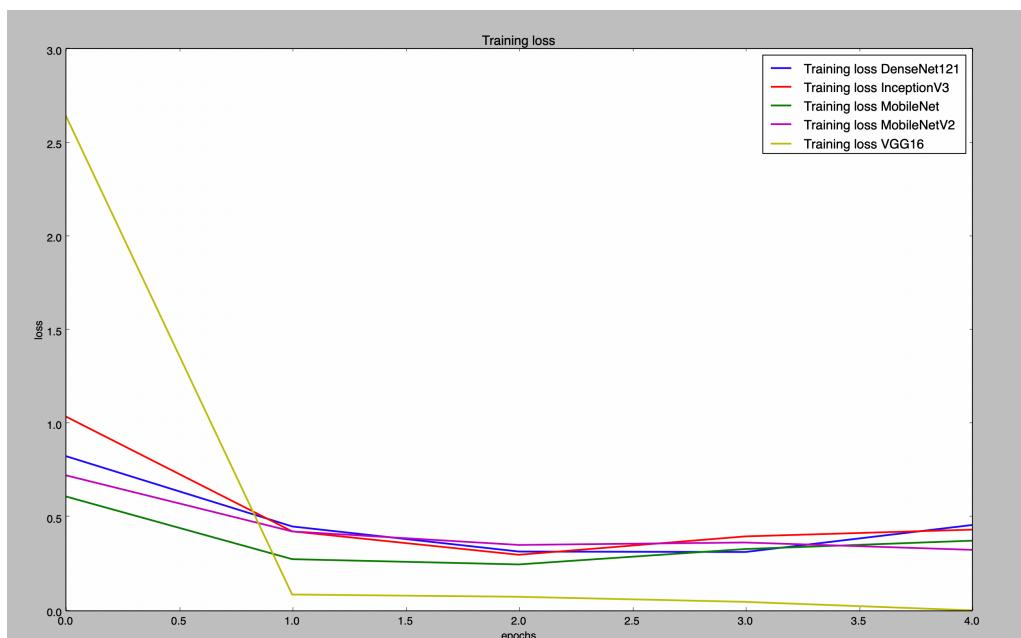
Nei prossimi paragrafi sono descritte alcune modifiche che sono state effettuate sulla rete, con il fine di osservare l'impatto causato da questi cambiamenti. Tali modifiche non sono finalizzate ad aumentare l'accuratezza della rete, ma a capire come si comporta il modello nelle varie situazioni analizzate.

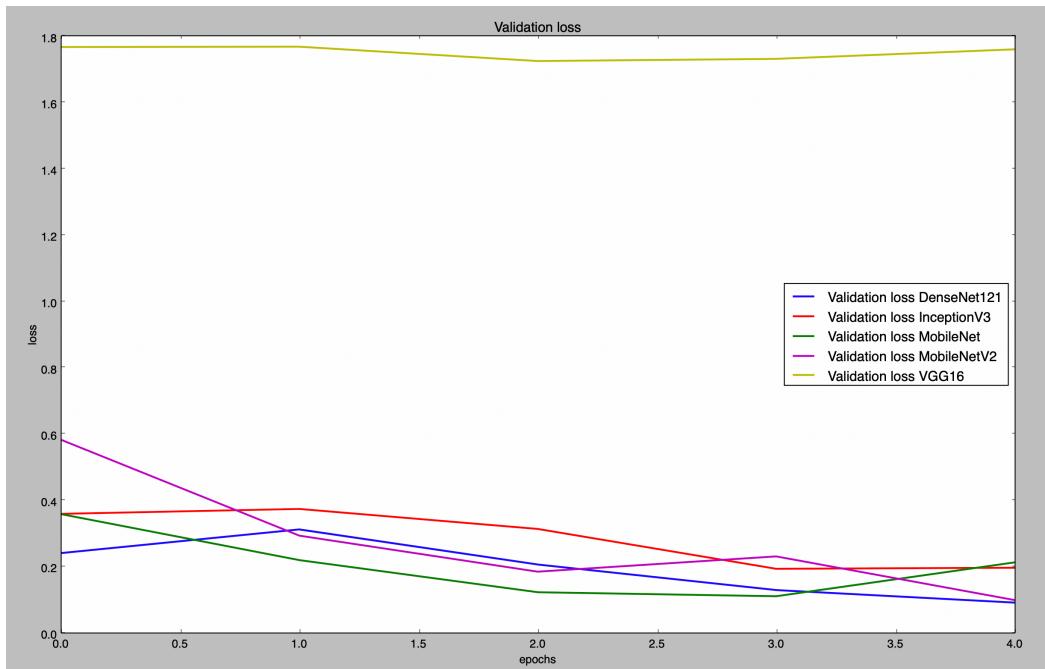
### **3.1: Modifica del modello di partenza**

Il modello che è stato utilizzato per effettuare il transfer learning, come anticipato nel paragrafo precedente, è MobileNet. Questo, ovviamente, non è l'unico modello presente in Keras per la classificazione delle immagini: abbiamo quindi usato altri modelli, senza modificare la struttura e le caratteristiche della rete aggiunta dopo il modello pre addestrato, per analizzare il comportamento complessivo. I modelli selezionati, oltre MobileNet, sono DenseNet121, InceptionV3, MobileNetV2 e VGG16, tutti pre addestrati sullo stesso dataset. Le caratteristiche dei modelli citati in precedenza sono riassunte nella tabella successiva.

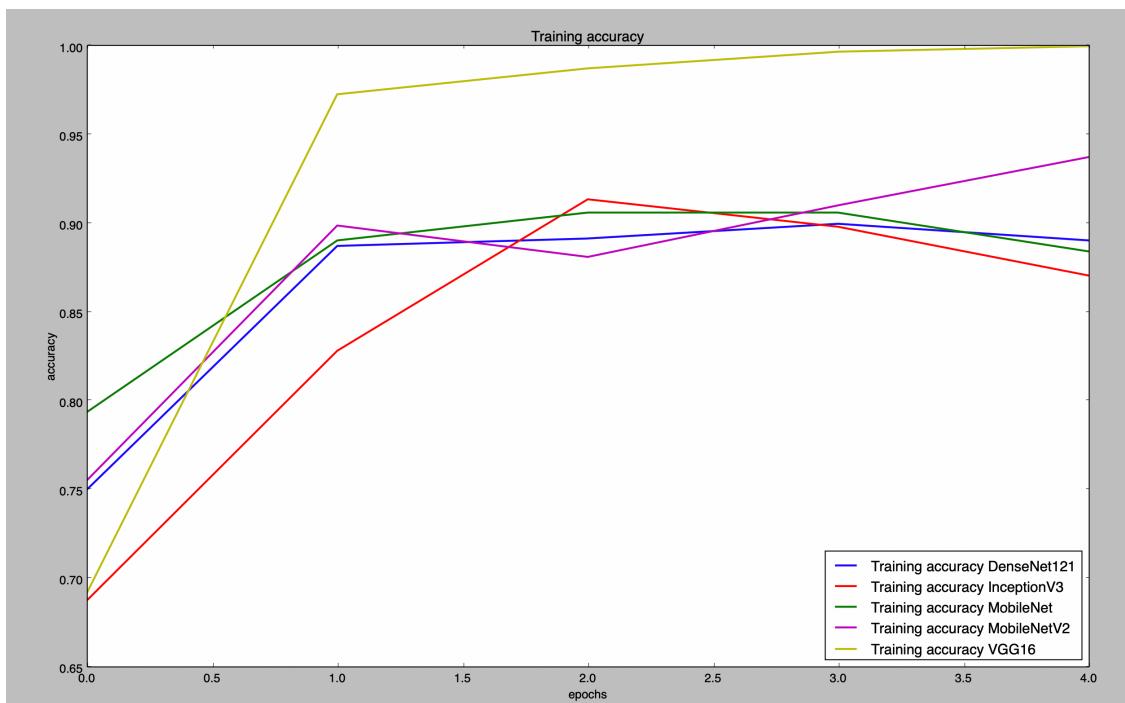
Modello	Accuratezza top 1	Accuratezza top 5	Parametri	Profondità
MobileNet	0,704	0,895	4253864	88
DenseNet121	0,750	0,923	8062504	121
InceptionV3	0,779	0,937	23851784	159
MobileNetV2	0,713	0,901	3538984	88
VGG16	0,713	0,901	138357544	23

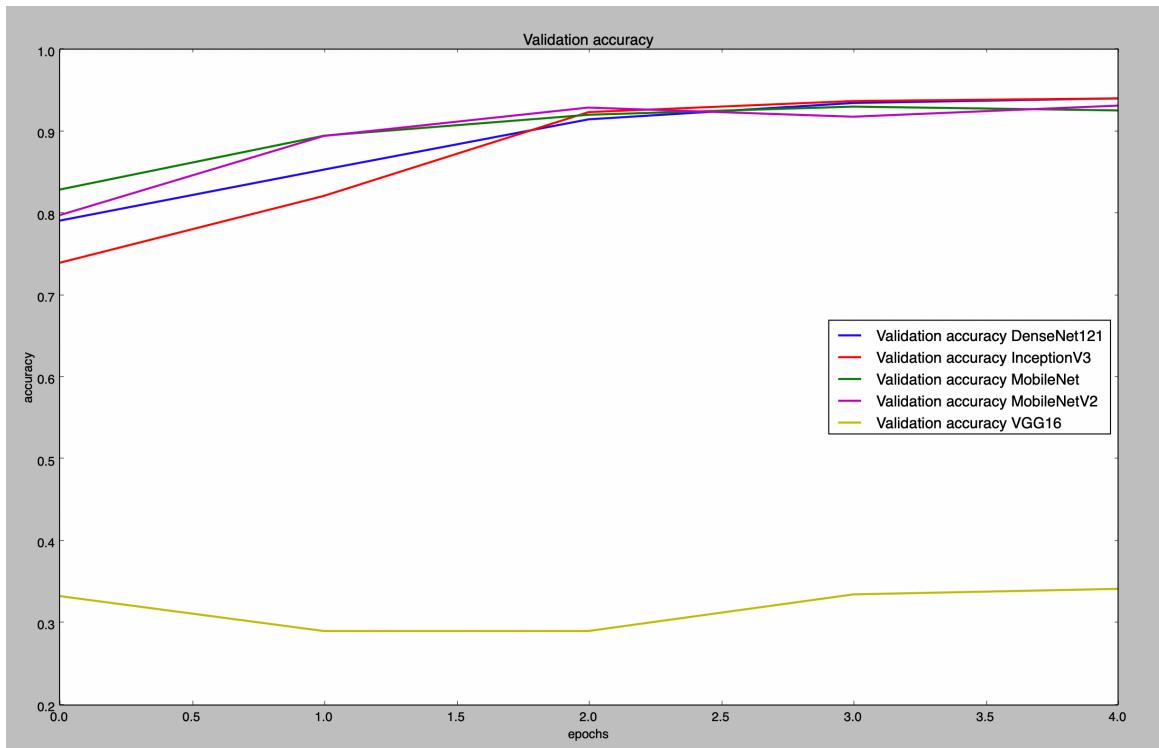
Tutti i modelli, ad eccezione di VGG16, portano a prestazioni che possono essere comparate, senza differenze significative sulla loss e sull'accuratezza. L'uso di VGG16 introduce invece l'overfitting: il modello si comporta bene solo sui dati del training set, non riuscendo a generalizzare sul validation e sul test set. Questo aspetto è messo in luce dai grafici successivi, in cui è possibile confrontare la loss e l'accuratezza di tutti i modelli. In effetti, già confrontando la loss ottenuta sul validation e sul training set emerge la presenza dell'overfitting: usando VGG16, la loss nel training set tende a zero, cosa che non succede con gli altri modelli, ma nel validation set è estremamente elevata.





Quanto asserito è confermato osservando l'accuratezza: anche in questo caso nel training set si hanno presentazioni molto buone, che non sono invece comprovate nel validation set.



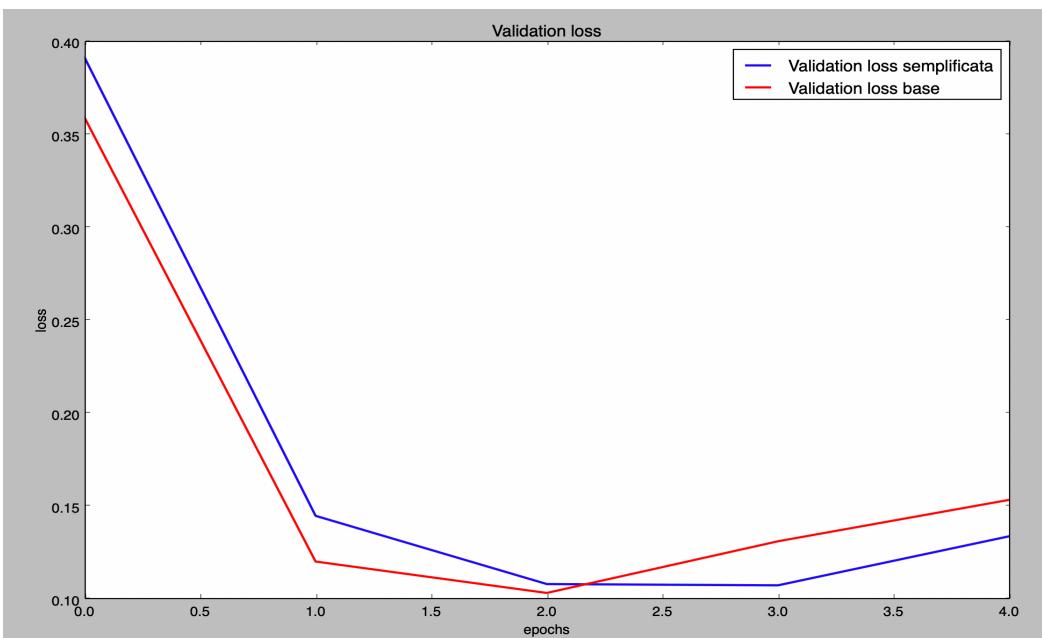
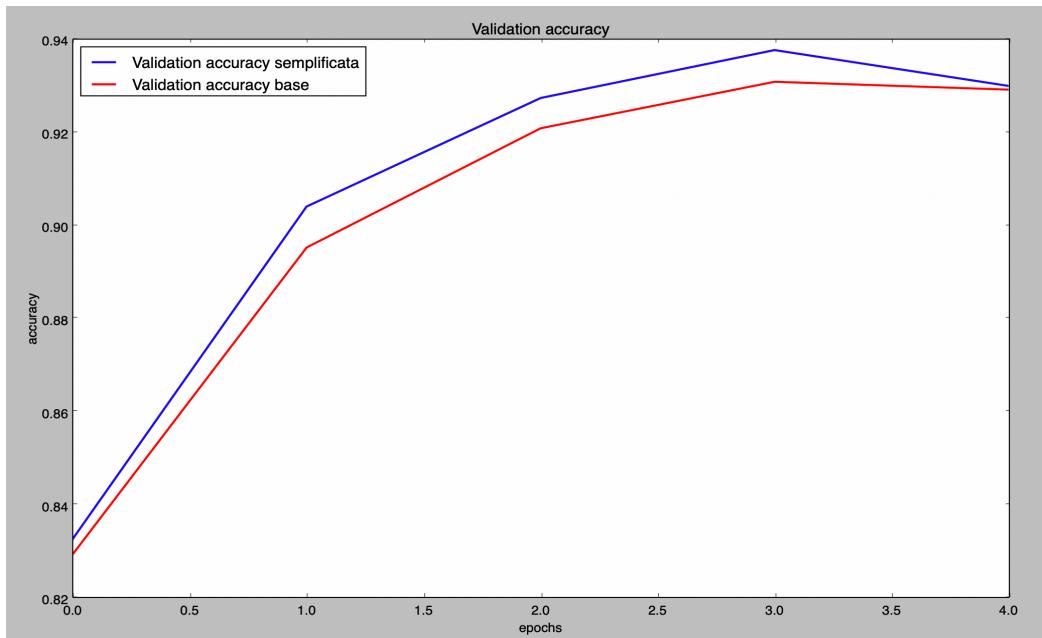


Le prestazioni sul test set sono riassunte nella tabella successiva emergendo ancora il problema dell'overfitting citato in precedenza, mentre gli altri modelli hanno prestazioni paragonabili.

Modello	Loss	Accuratezza
DenseNet121	0,29	92,4%
InceptionV3	0,39	91,4%
MobileNet	0,38	89,5%
MobileNetV2	0,46	89,9%
VGG16	1,8	37%

### 3.2: Semplificazione degli strati aggiuntivi

Dopo il modello pre addestrato devono essere introdotti degli strati aggiuntivi per “personalizzare” la classificazione sul task che si vuole intraprendere. La nostra rete prevede tre strati fully connected, contenenti, rispettivamente, 1024, 1024 e 512 neuroni, seguiti da una softmax. Una possibile semplificazione potrebbe essere utilizzare un solo strato fully connected con 256 neuroni, seguito, ovviamente, da una softmax come nel caso precedente. Questa semplificazione, dal punto di vista delle prestazioni, non introduce grossi miglioramenti: in effetti l’andamento del loss e dell’accuratezza sul validation set può essere sovrapposto.

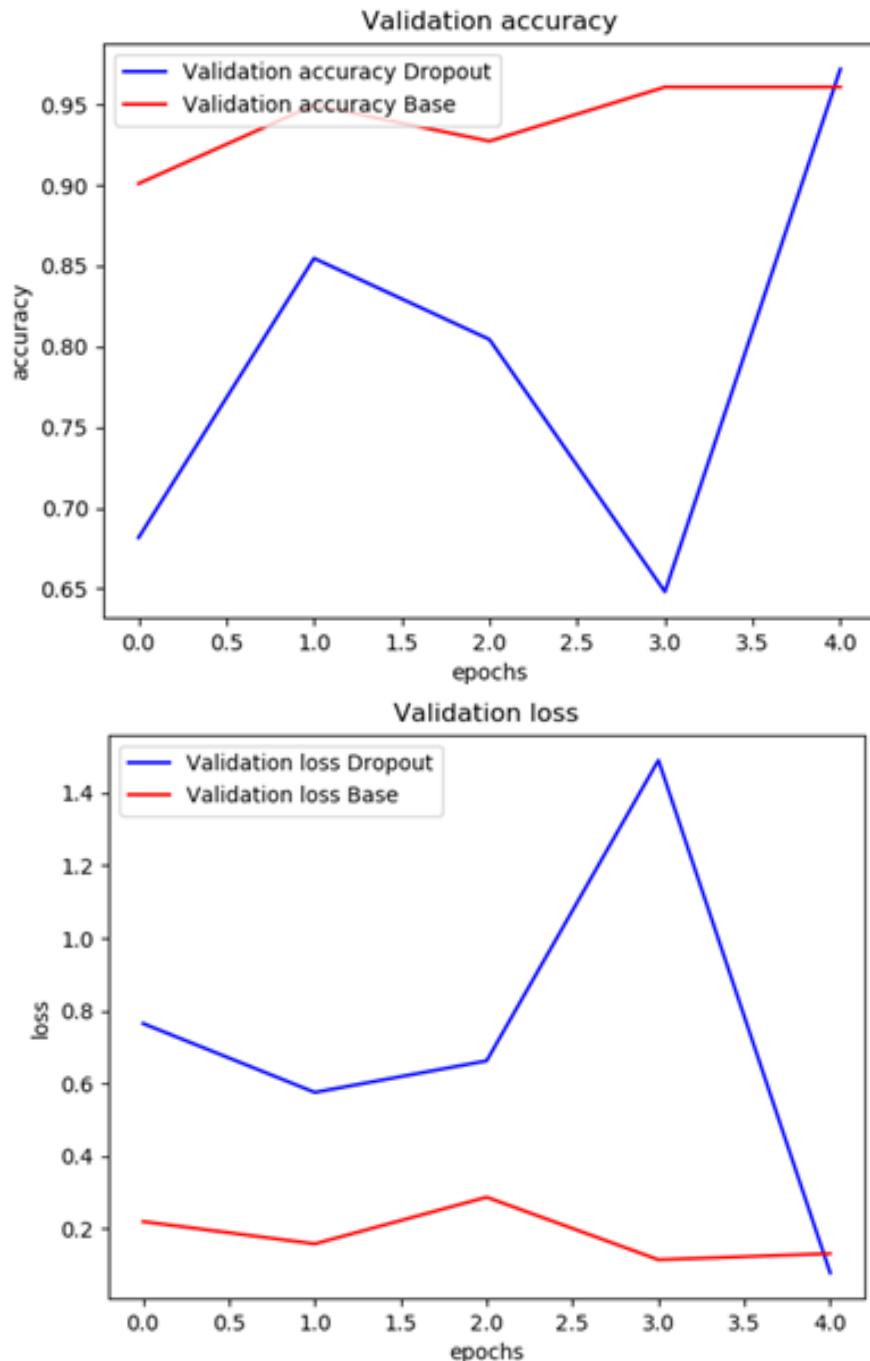


L’accuratezza sul test set sale da 89,5% a 90,5%, mentre la loss rimane invariata a meno di valori decimali bassi.

### 3.3: Utilizzo tecnica del Dropout

Nella rete base è stata applicata la tecnica del dropout con parametro 0.2 sullo strato fully connected con 512 neuroni.

L'andamento dell'accuratezza e della loss sul training set sono comparabili, mentre sul validation set si ha inizialmente un andamento peggiore che poi migliora linearmente nell'ultima epoca



L'accuratezza sul test set sale da 91% a 93% e la loss scende da 0.41 a 0.33.

Successivamente è stata utilizzata la tecnica del Dropout su ogni layer (tranne quello di output) con parametro 0.2, ma questo comporta un calo delle prestazioni sul validation set e sul test set.

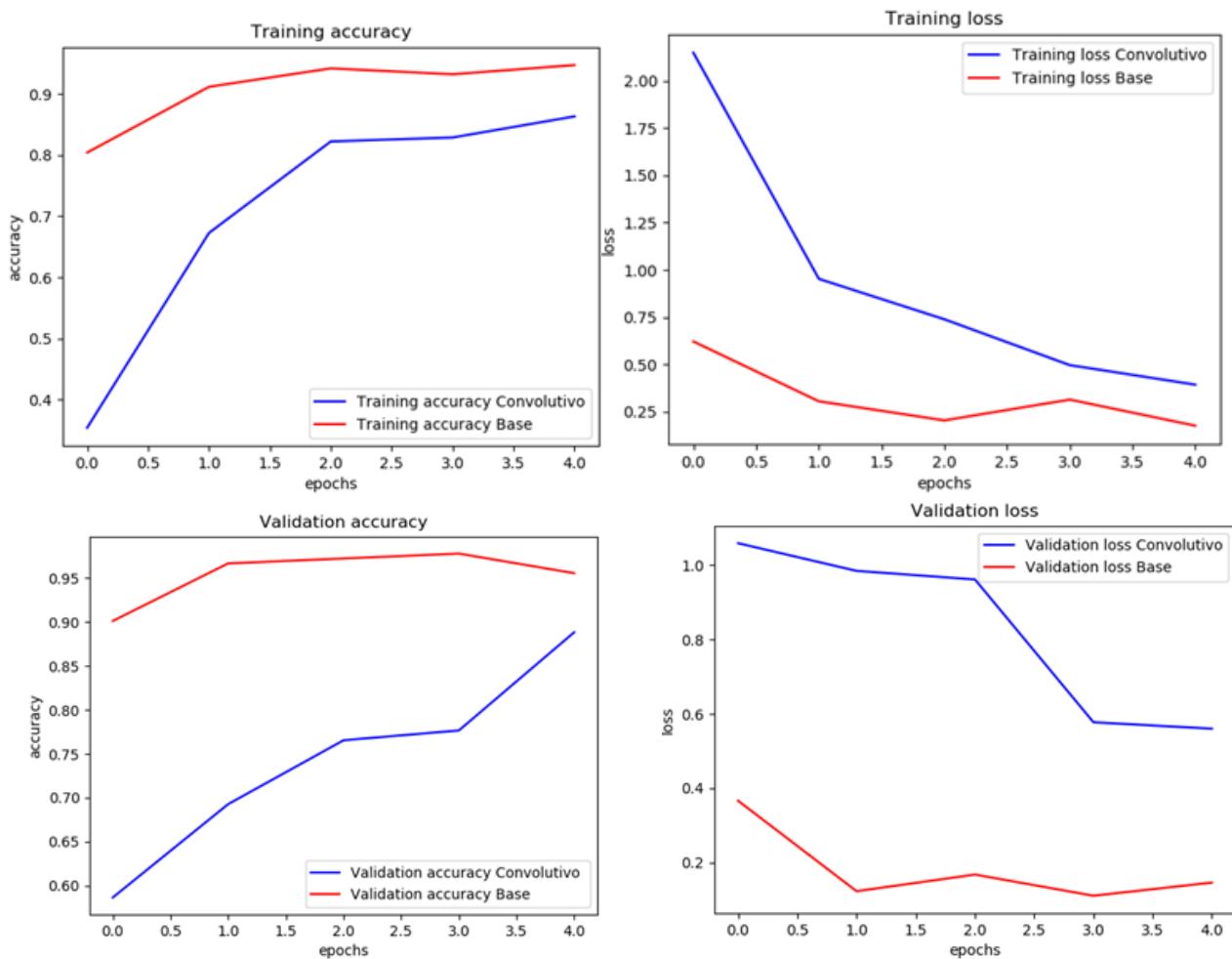
### 3.4: Aggiunta di uno strato convolutivo

Nella rete base è stato aggiunto uno strato convolutivo con kernel size 3x3, funzione di attivazione relu e 128 filtri.

Questo strato è seguito da un'operazione di Maxpooling con pooling size 2x2 e un'operazione di appiattimento. Seguono poi gli strati fully connected e softmax della nostra rete.

A tutti gli strati (tranne quello di output) è stata applicata la tecnica del Dropout con parametro 0.3 per ridurre l'overfitting.

Le immagini di seguito mostrano i risultati ottenuti:



Si nota che l'accuratezza e la loss sul training set peggiorano rispetto alla rete base e di conseguenza anche le prestazioni sul validation set.

Sul test set l'accuratezza diminuisce da 93% a 87% e la loss aumenta da 0.33 a 0.68.

Successivamente si è provato a variare il numero di filtri in output e il kernel size dell'operatore di convoluzione, ma non sono stati ottenuti miglioramenti.

Infine, si è provato a semplificare la rete diminuendo il numero di strati fully connected e il numero di neuroni, ma di nuovo i risultati sono stati negativi.

Si conclude che in questa situazione l'aggiunta di uno strato convolutivo non apporta miglioramenti.

#### **4: Uso di Tensorflow a scatola nera**

Tensorflow può essere utilizzato anche a scatola nera, senza la necessità di scrivere il codice per ciò che riguarda il machine learning. Anche in questo caso è possibile addestrare la rete per task personalizzati, partendo da un modello pre addestrato. Come spiegato in precedenza, però, a differenza del transfert leaning analizzato nel capitolo 3, non è necessaria la scrittura del codice per rete, ma basta una “configurazione” di Tensorflow. Questo è il motivo per cui ci siamo concentrati maggiormente sulla creazione delle reti, piuttosto che una loro configurazione. Le API offerte da Tensorflow permettono di riconoscere gli oggetti presenti all'interno di un'immagine. Abbiamo modificato il comportamento tradizionale, riuscendo ad effettuare un riconoscimento real time a partire da ciò che è inquadrato da un telefono, mostrando i risultati nel computer. Poiché il riconoscimento deve avvenire in tempo reale, non è possibile utilizzare modelli con accuratezza estremamente elevata, poiché questi richiedono necessariamente più tempo per l'elaborazione. Il modello che abbiamo selezionato è “ssd\_mobilenet\_v1\_coco” proprio per il motivo citato in precedenza e, presumibilmente, non sarebbe stato utilizzato per un riconoscimento statico delle immagini.

L'immagine successiva mostra un esempio d'uso di quanto descritto.

