Progetto di Computational Intelligence

Reti Neurali Convolutive per la classificazione di immagini mediche

Fabio D'Onofrio n°556505
f.donofrio6@studenti.unipi.it

20 Settembre 2019

Indice

Elenco delle figure

1 Introduzione

Le reti neurali convolutive sono un particolare tipo di reti neurali ottimali per la classificazione delle immagini. A differenza delle reti neurali standard, in cui ciascun neurone di ogni layer è connesso a tutti i neuroni dello strato successivo, i layers delle RNC hanno una struttura "a reticolo", che riduce il numero di connessioni : in questo modo non solo diminuisce il numero di parametri che la rete dovrà imparare, ma la rende particolarmente adatta a processare dati con struttura a griglia, come ad esempio le immagini in 2D, che consistono appunto in matrici di pixels.

Grazie anche all'utilizzo di GPU, negli ultimi anni le RNC hanno avuto ampio successo in numerose applicazioni, tra cui i sistemi di diagnosi assistita dal computer, o CAD(Computer-Aided-Detection/Diagnosis), che assistono il medico radiologo nell'analisi di immagini biomedicali.

Nell'articolo proposto[1] vengono utilizzati quattro dei modelli che hanno ottenuto i migliori risultati sull'ImageNet, per la classificazione di masse e calcificazioni nelle mammografie. Lo scopo di questo lavoro è quello di progettare ed implementare una rete neurale convolutiva ad hoc in grado di risolvere lo stesso problema, disponendo di un training set di 2864 immagini e di un test set di 352. Essendo l'insieme di dati disponibili decisamente ridotto rispetto ai milioni di immagini dell'ImageNet su cui sono state preallenate le reti utilizzate nell'articolo di cui sopra, ci si aspetta di riuscire difficilmente ad ottenere accuratezze migliori di quelle ivi ottenute(che sono dell'ordine del 93%)

L'ambiente utilizzato è Google Colab, e la libreria Python Keras, con interfaccia TensorFlow.

2 Progetto ed implementazione della RNC allenata "from scratch"

In questa sede sono state valutate varie architetture di reti neurali convolutive, tutte consistenti nella ripetizione dello stesso pattern : ogni layer convolutivo è costituito da un'operazione di convoluzione(Conv) con un *kernel*(o *filter*) , l'attivazione(Act), ed eventualmente un'operazione di *pooling*(Pool).

Dopo un blocco costituito da un certo numero di ripetizioni di tale struttura, si ha la parte "densa" ovvero uno o più layers di neuroni completamente connessi, come in una rete neurale "standard", ed infine un solo neurone nel layer di uscita, essendo un problema di classificazione in due classi : 0 per le masse, 1 per le calcificazioni.

2.1 Scelta degli iperparametri

Simbolicamente, si può quindi descrivere il tipo di reti studiate nel seguente modo :

$$INPUT \rightarrow [[Conv \rightarrow Act]*N \rightarrow Pool?]*M \rightarrow [FC \rightarrow Act]*K \rightarrow FC\_output$$

In cui, in generale, N ed M sono vettori di numeri interi, mentre K è scalare. Durante la prima fase, sono state testate varie reti con profondità e caratteristiche diverse. Via via si sono stabiliti i parametri più influenti e, provando diverse combinazioni, oltre ad algoritmi di ottimizzazione diversi, si è giunti al modello finale. Si è proceduto testando prima reti neurali con soli 2 layers convolutivi e 1-2 layers completamente connessi. Come ci si poteva aspettare queste architetture non erano abbastanza profonde per essere in grado di imparare a classificare le immagini. Inoltre, dopo altre prove, si sono scartate le architetture con più convoluzioni consecutive, ovvero non intervallate da operazioni di pooling. Queste infatti non davano risultati accettabili, poichè senza "contrarre" il numero di neuroni tramite pooling, il numero di parametri diventava troppo alto.

2.1.1 Parte convolutiva

Si è quindi optato per testare varie architetture con 3 layers convolutivi, cercando di analizzare e descrivere l'influenza di ogni iperparametro sulle prestazioni delle reti.

- *Numero di feature maps* :

  Questo parametro indica quante volte il kernel di convoluzione viene applicato all'input, e viene anche chiamata "profondità" del layer convolutivo. Maggiore è questa profondità, e maggiore sarà il numero di features che il layer sarà in grado di rappresentare. Si è settato tale valore a 32 per ogni strato : in realtà solitamente è conveniente che tale parametro aumenti di strato in strato, tuttavia ci si è prefissato l'obbiettivo di tenere basso il numero complessivo dei parametri.

- *Dimensione del kernel* :

  Valore che determina, insieme alla profondità e alle dimensioni dell'input, il numero di parametri che definisce l'operazione di convoluzione. Sono state provate 4 combinazioni diverse, con dimensione di ogni kernel di 5 o 3, ma sempre non-crescente di layer in layer.

  Di seguito si può osservare l'istogramma che riporta le accuratezze medie ottenute con varie combinazioni delle dimensioni dei filtri, al variare anche dell'algoritmo di ottimizzazione utilizzato per il training :

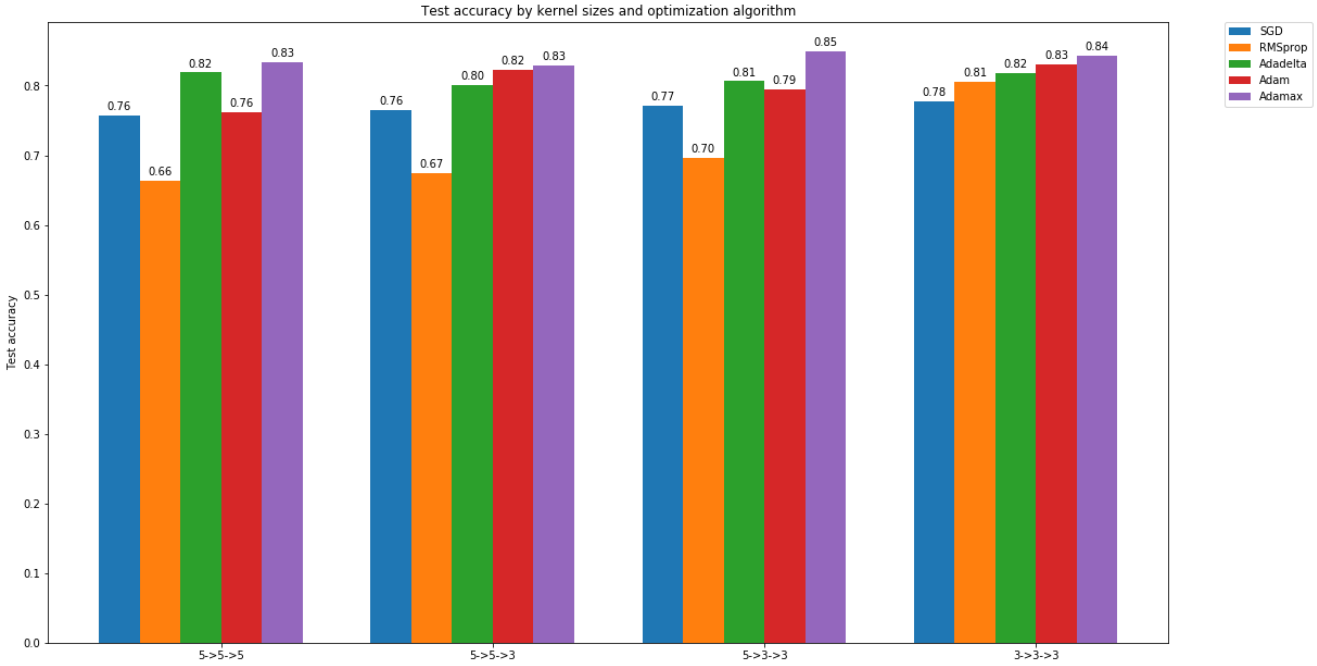


Figura 1: Test accuracy vs kernel sizes and optimization algorithm

Si può notare come adoperando dimensioni dei kernel maggiori, come nel primo gruppo di barre(5->5->5), si abbia una discreta variabilità dell'accuratezza ottenuta, e con 3 dei 5 algoritmi di ottimizzazione utilizzati, le rispettive reti(20 per ogni barra,che differiscono tra loro per tipo di padding e per numero di neuroni e di layers completamente connessi), non raggiungano precisioni in media maggiori del 76%

Utilizzando convoluzioni con kernel di dimensione 3x3 in tutti e 3 i layers convolutivi(ultimo gruppo di barre) si può osservare un miglioramento delle precisioni medie ottenute sul test set, con una media complessiva superiore all'81%

- *Padding* :

  Indica quanto "zero-padding" viene adoperato, ovvero quanti neuroni con valore di attivazione nullo vengono aggiunti nelle due dimensioni dell'input. Sono state provate per i 3 layers convolutivi varie combinazioni con padding di tipo 'same', che consiste nell'utilizzare lo zero-padding necessario a non alterare con l'operazione di convoluzione le dimensioni dell'input, e di tipo 'valid', che consiste nel non applicare alcuno zero-padding, effettuando la convoluzione quindi solo se il kernel entra interamente nell'input.

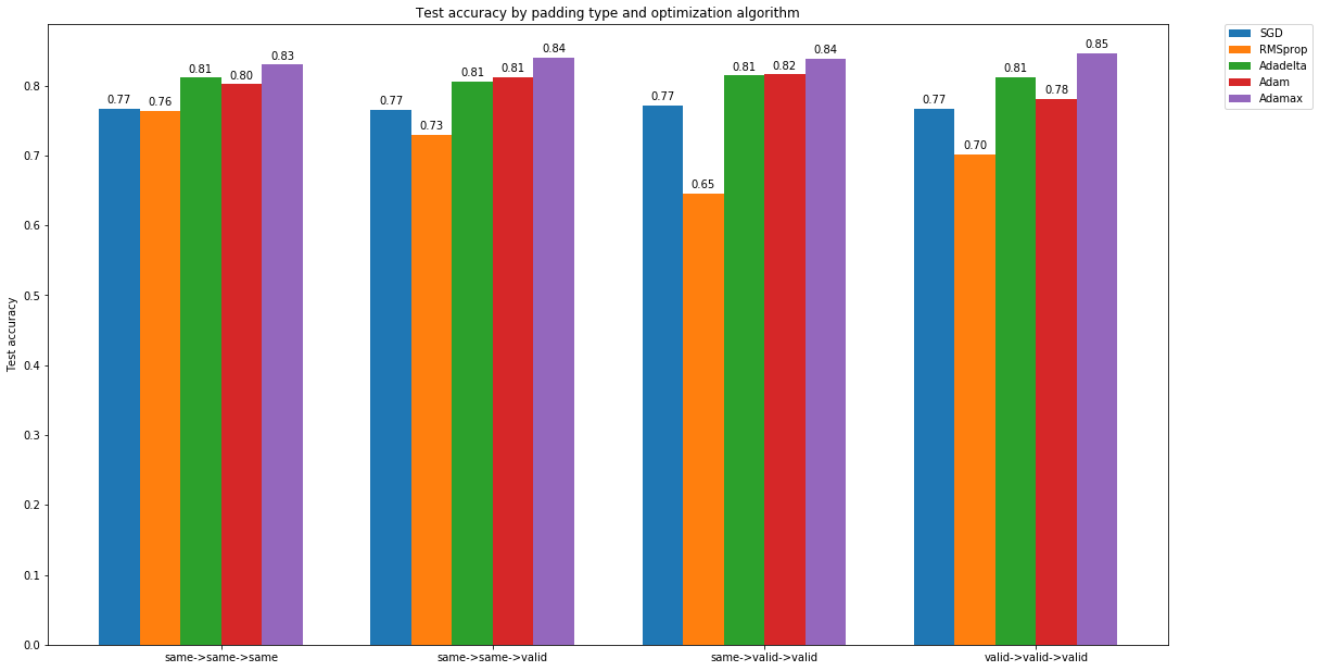


Figura 2: Test accuracy vs padding types and optimization algorithm

In questo caso l'influenza di questo iperparametro sembra essere minore, tuttavia si può apprezzare un leggero miglior comportamento delle reti che utilizzano padding di tipo 'same' in tutti e 3 i layers convolutivi. In questo modo infatti si preservano le dimensioni spaziali in ogni convoluzione, evitando di perdere le informazioni ottenibili dai bordi delle immagini.

- *Pooling* :

  Operazione di "contrazione" del numero di neuroni, permette di abbassare notevolmente il numero di parametri preservando però allo stesso tempo le informazioni, condensandole prendendo la media o il massimo di aree contigue di neuroni. In questa sede si è utilizzato Max pooling 2x2 con stride di 2 , ottenendo un'uscita dell'ultimo layer convolutivo di dimensioni ridotte(tra 15x15x32 a 18x18x32 a seconda dei kernel e del padding) e riducendo quindi il numero di parametri associati alle connessioni con il primo layer completamente connesso della parte "densa" : ogni operazione di pooling elimina il 75% di unità.

- *Stride* :

  Numero di pixels che vengono saltati nell'applicazione del kernel. Utilizzando un valore di 2 o 3, si riducono le dimensioni degli outputs di ogni layer, ma avendo usato già il Max pooling, si è fissato tale iperparametro a 1 per ogni convoluzione.

- *Funzione di attivazione* :

  Le funzioni di attivazione più usate nelle RNC sono la ReLU(Rectified Linear Unit) a la TanH(Tangente iperbolica). In questo lavoro è stata utilizzata la prima per tutti e 3 i layers convolutivi:

$$ReLU(x) = \max[0, x]$$

2.1.2 Parte densa

La parte densa, o MLP(MultiLayer Perceptrons) della RNC, consiste in un primo layer che riarrangia l'uscita dell'ultimo layer della parte convolutiva in un vettore unidimensionale(avente quindi da 15x15x32=7200 a 18x18x32=10368 unità, a seconda dei kernel e padding usati), e uno o più layers completamente connessi : il maggior numero dei parametri della rete convolutiva risiede in questa parte.

- *Numero di layers* :

  Sono stati testati 1-2 layers con 256-512 neuroni ciascuno, senza ottenere particolari differenze nelle prestazioni in questo caso.

- *Funzione di attivazione* :

  Anche per i layers completamente connessi è stata utilizzata la ReLU.

2.2 Algoritmo di ottimizzazione

Il processo di training della rete neurale consiste nell'aggiornare via via i parametri(θ) sulla base del gradiente della funzione di costo($\nabla_\theta J$).

Il metodo del "Gradient Descent" consiste nel seguire la direzione opposta a quella del gradiente, con un certo "learning rate" η che rappresenta la dimensione del passo adottato. La variante principale di tale metodo è il mini-batch Gradient Descent, anche chiamato SGD(Stochastic Gradient Descent), con il quale l'aggiornamento dei parametri viene effettuato, all'interno di una singola epoch, ogni n campioni dell'insieme di training(x,y) :

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

In questo modo si riduce la varianza degli aggiornamenti dei parametri ed è più computazionalmente efficiente. Dopo alcuni tentativi si è optato per un batch size di n=100. La difficoltà principale nell'utilizzare questo algoritmo risiede nella scelta del learning rate : un valore troppo basso può rallentare enormemente la convergenza o stallarla del tutto, mentre un valore troppo alto potrebbe causare un'oscillazione della funzione di costo intorno al minimo, senza raggiungerlo, o addirittura divergere.

Un modo per accelerare l'SGD nella direzione giusta e smorzare le oscillazioni è quello di utilizzare il cosiddetto "momento" : si aggiunge una frazione(di solito intorno allo 0.9) del vettore di aggiornamento usato al passo precedente, al vettore di aggiornamento del passo corrente.

Tuttavia, lo stesso learning rate viene comunque applicato a tutti i parametri, mentre può essere vantaggioso aggiornare questi ultimi in modo più selettivo a seconda dell'andamento del processo di ottimizzazione.

Quest'idea è alla base di metodi quali Adadelta , RMSprop , Adam e Adamax.

Nell'Adadelta ad esempio, il vettore di update dei parametri è una frazione del vettore del passo precedente, che dipende dai valori dei gradienti degli step passati :

$$\triangle\theta_t = \frac{RMS[\triangle\theta]_{t-1}}{RMS[g]_t} g_t$$

dove g_t è il gradiente della funzione di costo allo step t, mentre $RMS[g]_t$ è l'errore quadratico medio del gradiente e $RMS[\triangle\theta]_{t-1}$ è il root mean squared error del vettore di aggiornamento dei parametri al passo precedente $[\triangle\theta]_{t-1}$

Sono stati testati tutti e 7 gli algoritmi di ottimizzazione implementati in Keras, tuttavia i metodi Adagrad e Nadam la maggior parte delle volte davano problemi nella convergenza, pertanto i risultati ottenuti con questi due metodi non sono riportati.

Si riportano invece le prestazioni ottenute sulle diverse architetture, con i metodi SGD, RMSprop, Adadelta, Adam e Adamax.

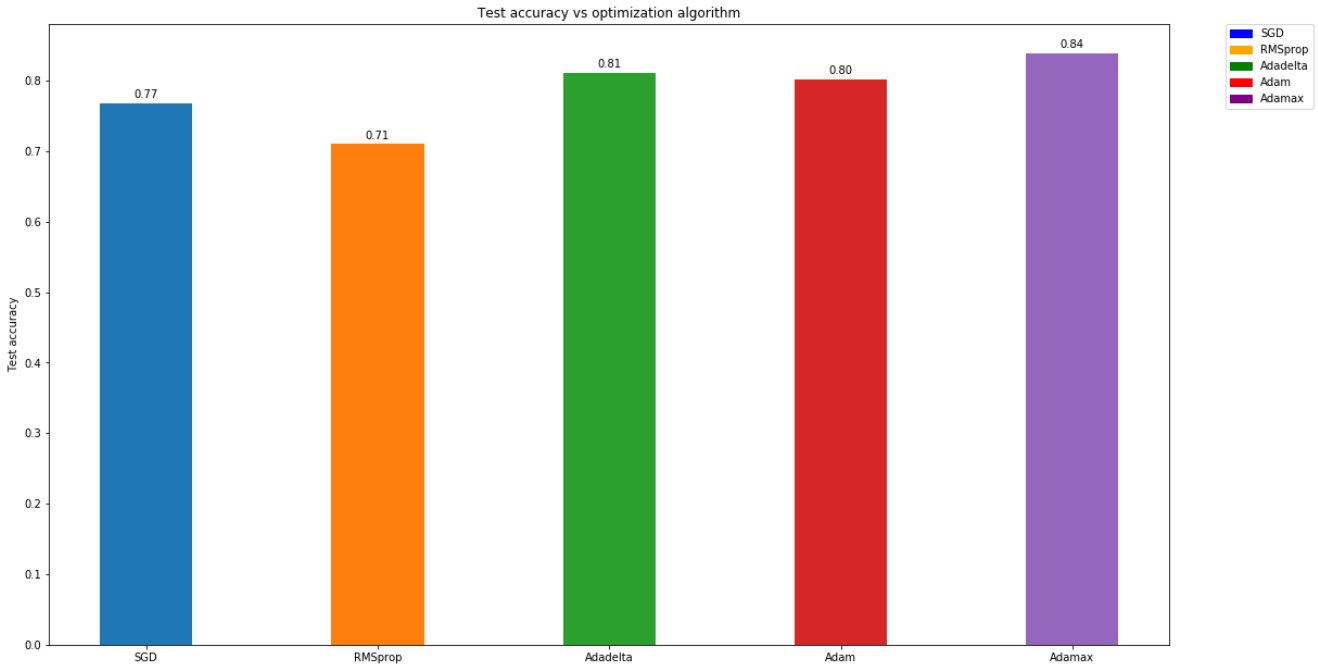


Figura 3: Test accuracy vs optimization algorithm

Ogni barra rappresenta la media delle accuratezze sul test set ottenute da 80 architetture diverse.

Si può notare, com'era già osservabile dai grafici precedenti, che l'algoritmo che ha dato i migliori risultati è l'Adamax, la cui precisione media risulta dell'84%

2.3 Data augmentation

Per provare a migliorare , si è poi aumentato ulteriormente la profondità della rete, aggiungendo un quarto layer convolutivo.

Sono state quindi testate varie architetture, prediligendo quelle con iperparametri scelti in base anche alle considerazioni precedenti. Per limitare l'overfitting, che praticamente tutti i modelli presentavano, a partire da un'accuratezza sul validation set dell'80-85% , si è poi utilizzato il "data augmentation", con il quale si aumenta la capacità della rete di generalizzare e quindi di classificare con miglior precisione insiemi di dati sui quali non è stata allenata.

Infatti, tramite la classe Keras *ImageDataGenerator* vengono creati a partire da ciascun batch di campioni, varie immagini trasformate casualmente con operazioni di rotazione,shift dei pixels, e via dicendo.

In questo modo durante il processo di training vengono presentati alla rete sempre "nuovi" ingressi, elevandone la capacità di generalizzazione.

2.4 Soluzione proposta

Bisogna considerare che non solo il training set, ma anche il test set a disposizione erano di scarse dimensioni, pertanto una certa variabilità è inevitabilmente presente anche nella "test_acc".

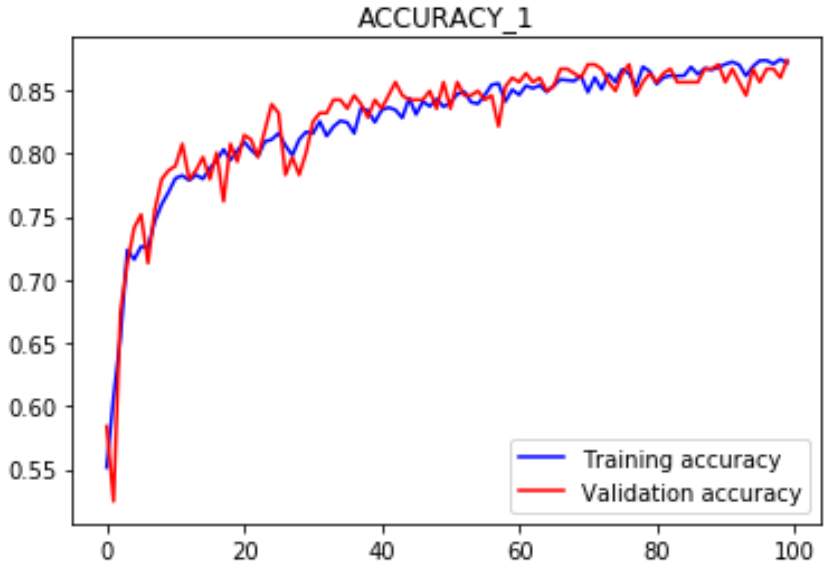
Si è comunque verificato in maniera sperimentale che le migliori architetture di RNC per questo tipo di task sono costituite da 3-4 layers convolutivi, con dimensioni dei filtri di 3x3 o 5x5, e 2 layers completamente connessi : performando data augmentation su questi modelli, con algoritmo di ottimizzazione Adamax, molte reti raggiungevano accuratezze sul test set dell'87-88 % .

Si presenta infine i risultati nel dettaglio della rete che ha dato la migliore accuratezza sul test set, pari al 90% .

Layer (type)	Output Shape	Param #
First_conv_3x3_kernel_same (	(None, 150, 150, 32)	320
First_2x2_MaxPooling (MaxPoo	(None, 75, 75, 32)	0
Second_conv_3x3_kernel_valid	(None, 73, 73, 32)	9248
Second_2x2_MaxPooling (MaxPo	(None, 36, 36, 32)	0
Third_conv_3x3_kernel_valid	(None, 34, 34, 32)	9248
Third_2x2_MaxPooling (MaxPoo	(None, 17, 17, 32)	0
Fourth_conv_3x3_kernel_valid	(None, 15, 15, 32)	9248
Fourth_2x2_MaxPooling (MaxPo	(None, 7, 7, 32)	0
Flatten (Flatten)	(None, 1568)	0
First_dense (Dense)	(None, 256)	401664
Second_dense (Dense)	(None, 256)	65792
Output_layer (Dense)	(None, 1)	257

Total params: 495,777
Trainable params: 495,777
Non-trainable params: 0

Figura 4: CNN architecture




```
352/352 [==============================] - 0s 271us/step
test loss is :  0.29906194453889673
test accuracy is :  0.9005681818181818
```

Figura 5: accuracy and loss

3 Utilizzo di una rete pre-allenata

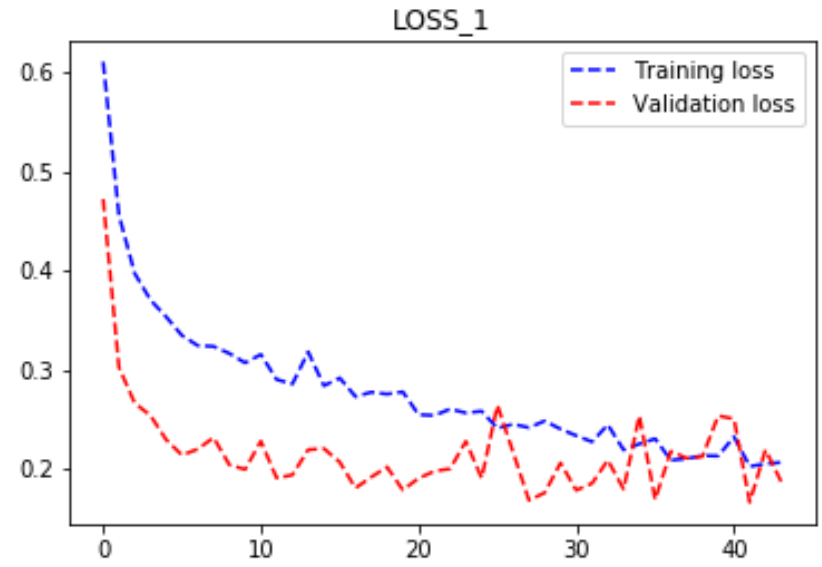
Dopo aver riadattato il set di dati in modo che potesse essere accettato in ingresso dalle reti dell' ImageNet, le quali sono implementate in Keras per elaborare ingressi a 3 canali, si è provato varie architetture.

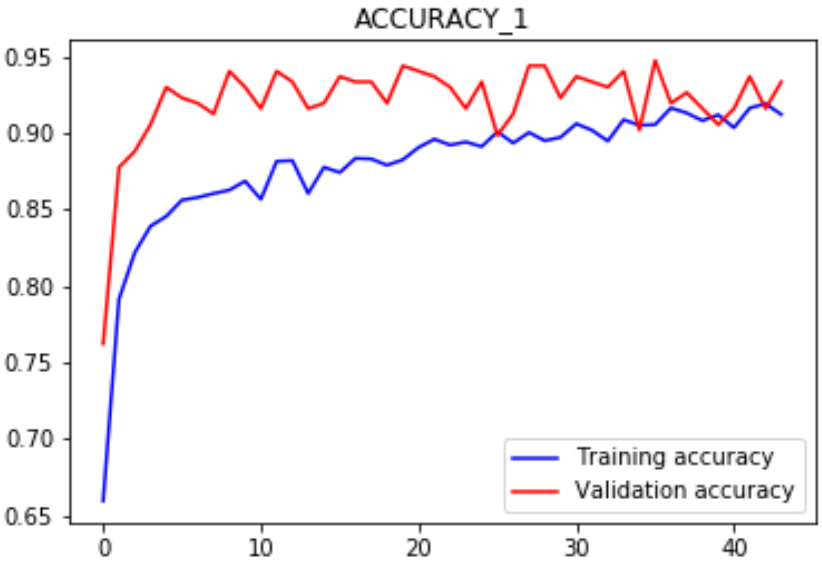
Sono state testate varie reti, tra cui Xception , VGG16 e VGG19, variando il numero di layers completamente connessi posti successivamente alle basi convolutive di queste ultime. La tecnica adottata per il training è stata quella classicamente utilizzata per il "transfer learning" : la parte densa di ogni rete è stata sostituita e riallenata da zero, lasciando invece inalterati i pesi della parte convolutiva. Poi si è sbloccato i pesi dell'ultimo blocco di ogni rete, effettuandone cioè il "fine tuning" con l'obbiettivo di adattare i parametri degli ultimi strati convolutivi al particolare task del problema. Ancora una volta, per combattere l'over fitting, si è fatto uso del data augmentation. I risultati migliori sono stati ottenuti usando la base convolutiva della VGG16, che ha una struttura effettivamente non molto diversa dalle reti utilizzate nella sezione precedente : le uniche differenze sono che l'operazione di Max pooling viene effettuata dopo ogni 2 o 3 convoluzioni e non dopo ogni convoluzione, e che la profondità dei layers convolutivi aumenta via via da 64 fino addirittura a 512. Quando si erano testate questo tipo di architetture per le reti allenate da zero i risultati erano insoddisfacenti, poichè aumentando il numero di convoluzioni consecutive e il numero di features, il numero di parametri diventava troppo alto debilitando le capacità generalizzatrici delle RNC allenate su un training set molto ridotto. Essendo invece la rete VGG16 già allenata su un set di immagini molto più profondo, nonostante i suoi quasi 15 milioni di parametri, riesce a classificare ancora le immagini del test set con un'accuratezza prossima al 90% . Di seguito una delle architetture testate, costituita dalla base convolutiva di VGG16 e un solo layer completamente connesso con 256 unità, allenata tramite fine tuning dell'ultimo blocco della rete preallenata(costituito da 3 layers convolutivi con kernel di dimensioni 3x3 e un Max pooling di 2x2) :

Layer (type)	Output Shape	Param #
VGG16_ConvBase (Model)	(None, 4, 4, 512)	14714688
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 256)	2097408
dense_2 (Dense)	(None, 1)	257

Total params: 16,812,353
Trainable params: 9,177,089
Non-trainable params: 7,635,264

Figura 6: pretrained CNN architecture






```
352/352 [==============================] - 7s 19ms/step
test loss is :  0.3655068142847581
test accuracy is :  0.8863636363636364
```

Figura 7: accuracy and loss of p.t. CNN

4 Ensamble di Reti Neurali

Si presenta infine un'Ensamble di reti neurali in cui vengono combinate le predizioni di 3 delle migliori reti allenate da zero, e della rete VGG16 precedente. Si riportano il loss(Binary Cross-Entropy) e l'accuratezza, calcolata sia come media delle predizioni di ciascuna rete sia come media pesata delle accuratezze sul validation set di ciascuna di esse :

```
Binary Cross Entropy loss of the average predictions is :  0.26416203314593906
Average accuracy is :  0.9005681818181818


Binary Cross Entropy loss of the weighted average predictions is :  0.2636623165245427
Weighted average accuracy is :  0.9034090909090909
```

Nell'ultimo caso si può nuotare un'accuratezza del 90.3% che pare un risultato tutto sommato accettabile considerando la ridotta dimensione del training set.

5 Organizzazione del codice

- *CNN.ipynb* con il quale è stata effettuata la ricerca degli iperparametri per la rete allenata da zero con la soluzione infine riportata sopra.
- *CNN_PT.ipynb* con cui sono state testate diverse basi convolutive di reti preallenate fino alla scelta della rete più performante.
- *Ensamble.ipynb* in cui sono state combinate le CNN migliori ottenute.

Riferimenti bibliografici

[1] **"Xi, P. Shu, C., Goubran, R., Abnormality Detection in Mammography using Deep Convolutional Neural Networks, 13th IEEE International Symposium on Medical Measurements and Applications, MeMeA 2018; Universita La SapienzaRome; Italy; 11 June 2018 through 13 June 2018; Category numberCFP18MEA-USB; Code 138764"**