

# **Relazione Finale - Progetto Laboratorio di multimedialità**

**Membri gruppo:** Gianluca Visentin, Luca Di Simone, Giulio Raponi

## **Indice**

---

### **Introduzione**

- 1. Stato dell'arte**
- 2. Specifiche del progetto**
  - Dataset
  - Specifiche tecniche
- 3. Addestramento rete neurale dataset originale**
  - AlexNet
  - Implementazione della rete sul dataset "Monkey Species"
- 4. Elaborazione e analisi delle immagini**
  - Obiettivi
  - Aggiunta e riduzione di rumore gaussiano attraverso il filtro di Wiener su immagini in scala di grigi
  - Aggiunta e riduzione di rumore sale e pepe attraverso il filtro di Wiener su immagini RGB
  - Aggiunta e rimozione di rumore gaussiano attraverso una rete pre-addestrata su immagini RGB ridimensionate
  - Analisi del colore
- 5. Valutazione Risultati e conclusioni**

## **Introduzione**

# Obiettivo del progetto

---

Lo scopo del progetto riguarda l'implementazione di una rete neurale in merito ad un problema di classificazione delle immagini sul Dataset originale. In seguito è stato introdotto un processo di elaborazione delle immagini con il fine di studiare casi d'uso diversi legati allo stesso Dataset, analizzandone il comportamento della rete neurale a seconda delle varie applicazioni.

## Struttura della relazione

---

Nel primo capitolo è stata introdotta una panoramica generale sullo stato dell'arte, focalizzando l'aspetto architettonicale delle reti neurali convoluzionali. Nel secondo capitolo sono descritte le specifiche del progetto, riguardanti il Dataset e le configurazioni della macchina su cui è stata addestrata la rete. Nel terzo capitolo viene raffigurata la composizione della rete con le tecniche adottate e le motivazioni alla base di ciò. Nel quarto capitolo viene introdotto il processo di elaborazione delle immagini, cercando di ottenere risultati diversi in termini di performance e temporali nella fase di addestramento, descrivendo le varianti al Dataset originale. Infine, nell'ultimo capitolo si ha una valutazione dei risultati per ogni caso descritto nei suddetti capitoli con interpretazioni in merito.

## 1. Stato dell'arte

Nell'apprendimento automatico, una rete neurale convoluzionale (CNN o ConvNet dall'inglese Convolutional Neural Network) è un tipo di rete neurale artificiale feed-forward in cui il pattern di connettività tra i neuroni è ispirato dall'organizzazione della corteccia visiva animale, i cui neuroni individuali sono disposti in maniera tale da rispondere alle regioni di sovrapposizione che tassellano il campo visivo. Le reti neurali offrono un insieme di strumenti molto potente che permette di risolvere problemi nell'ambito della classificazione, della regressione e del controllo non-lineare. Un semplice esempio è l'analisi di un'immagine per verificare la presenza di determinati oggetti oppure di testi, ed eventualmente riconoscerli. Le reti neurali artificiali (abbreviato ANN o NN) sono modelli matematici composti da neuroni artificiali e vengono utilizzate per risolvere problemi ingegneristici di Intelligenza Artificiale legati a diversi ambiti tecnologici come l'informatica, l'elettronica o altre discipline. Il funzionamento di tali sistemi si ispira dichiaratamente a quello dei sistemi nervosi biologici, come l'insieme dei neuroni che fanno parte del cervello. Una rete neurale di fatto si presenta come un sistema "adattivo" in grado di modificare la sua struttura (i nodi e le interconnessioni) basandosi sia su dati esterni sia su informazioni interne che si connettono e passano attraverso la rete neurale durante la fase di apprendimento e ragionamento. In linea di massima questi sistemi sono composti da tre strati principali:

- Strato di **input**: ha il compito di ricevere segnali esterni ed elaborare i dati in ingresso;

- **Strato nascosto:** è quello che ha in carica il processo di elaborazione vero e proprio (e può anche essere strutturato con più colonne-livelli di neuroni);
- **Strato di output:** qui vengono raccolti i risultati dell'elaborazione dello strato H e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

Oltre ad avere un'elevata velocità di elaborazione le reti neurali hanno la capacità di imparare la soluzione da un determinato insieme di esempi. In molte applicazioni questo permette di raggiungere il bisogno di sviluppare un modello dei processi fisici alla base del problema, che spesso può essere difficile, se non impossibile, da trovare.

I risultati ottenuti mediante le reti neurali sono buoni, ma non sono spiegabili in modo chiaro, si usa perciò considerare la rete neurale come una “black box”. Tuttavia si possono riscontrare alcune difficoltà, soprattutto nell'ambito di applicazioni complesse con un'ingente mole di dati, in quanto il tempo di elaborazione risulta essere troppo elevato per un computer moderno. È bene specificare che i valori di output ottenuti dall'addestramento della rete non sempre sono ottimali, anzi spesso contengono margini di errore al loro interno, che vengono poi ridotti con un addestramento di tipo backpropagation.

## 2. Specifiche del progetto

### Dataset

---

Il Dataset, disponibile presso il link <https://www.kaggle.com/slothkong/10-monkey-species>, consiste di due cartelle, training e validation. Ogni cartella contiene 10 sottocartelle, le quali corrispondono alle diverse specie di scimmia. È formato da circa 1400 immagini, tutte con dimensioni di almeno 400x300 ed in formato JPG. È possibile ricollegare le sottocartelle (n0-n9) con le rispettive specie.

### Caratteristiche Tecniche

---

L'esecuzione e l'addestramento della rete neurale è stata effettuata su una macchina con le seguenti caratteristiche.

MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports)

Processore: 2,3 GHz Intel Core i5 quad-core

Ram: 8 GB 2133 MHz LPDDR3

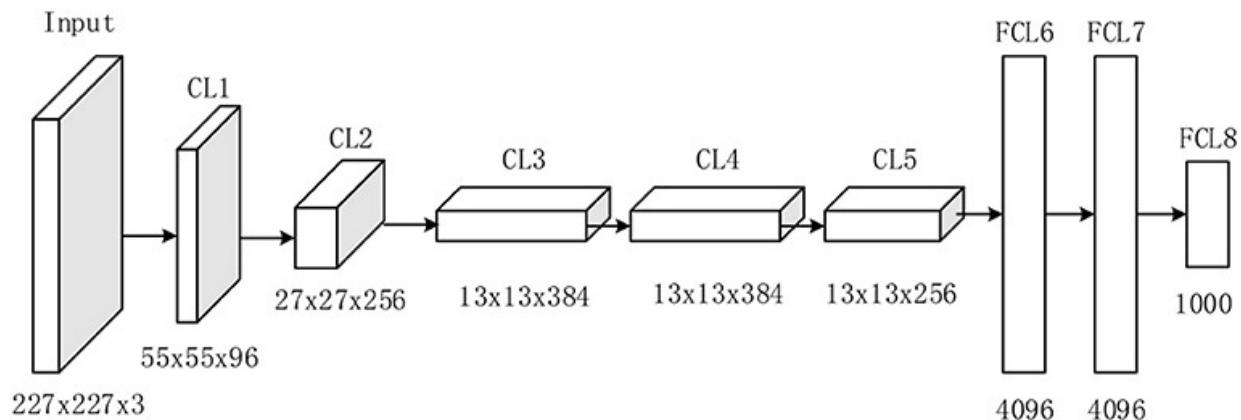
Scheda grafica: Intel Iris Plus Graphics 655 1536 MB

## 3. Addestramento rete neurale con Dataset originale

# AlexNet

---

AlexNet è una rete neurale di tipo convoluzionale, adatta al task riguardante la classificazione delle immagini. Essa è divisa in 8 layer (5 layer convoluzionali e 3 layer fully connected). La figura seguente mostra una rappresentazione standard della struttura della rete convoluzionale sopra citata.



L'AlexNet porta le dimensioni iniziali a 227x227x3: tre canali, permettendoci di gestire immagini a colori (schema RGB).

## Implementazione della rete sul Dataset "Monkey Species"

---

### Prima convoluzione

Iniziamo applicando 96 filtri convoluzionali (che tecnicamente chiamiamo kernel, quindi 96 kernel) di dimensioni 11x11x3 con uno stride (lo spostamento della finestra di lettura) di 4 (pixel) e padding 0. L'output prodotto ha un volume di 55x55x96. A questo aggiungiamo un layer di max pooling avente dimensioni di 3x3 con stride 2 per ridurre le dimensioni a 27x27x96 e infine applichiamo una funzione di attivazione di tipo ReLU.

### Seconda convoluzione

Questa volta applichiamo 256 kernel di dimensioni 5x5x48 con stride 1 e padding 2. A questo sovrapponiamo un Max Pooling 3x3 con stride 2, per ottenere un volume di 13x13x256 e nuovamente il LRN che applica una trasformazione ma mantiene inalterate le dimensioni.

### Terza convoluzione

384 kernel con dimensioni 3x3x256, stride 1 e padding 1 producono un output di 13x13x384.

## Quarta convoluzione

384 kernel con dimensioni 3x3x192, stride 1 e padding 1 producono un output di 13x13x384.

## Quinta Convoluzione

256 kernel con dimensioni 3x3x192, stride 1 e padding 1 producono un output di 13x13x256.

A questo sovrapponiamo un Max Pooling 3x3 con stride 2 per ottenere 6x6x256 feature maps.

Infine la nostra rete neurale profonda è composta da 3 Fully Connected (Dense) Layer:

- 2 Fully Connected (Dense) Layer sequenziali di 4096 neuroni;
- 1 Fully Connected (Dense) Layer di 10 neurons (numero di classi per la predizione del nostro Dataset).
- ~~~ matlab layers = [ layersTransfer  
fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20, ...  
'BiasLearnRateFactor',20) softmaxLayer classificationLayer]; pixelRange = [-30 30]; ~~~

Poichè, come già affermato precedentemente, AlexNet supporta come input solo immagini 227x227x3, abbiamo adottato una funzione  `imageDataAugmenter` per ridimensionare in modo automatico tutte le immagini alla dimensione standard. Nella stessa, abbiamo aggiunto alcune funzionalità, elencate di seguito, per prevenire un possibile overfitting della rete, riducendo il numero di parametri rispetto al numero di esempi.

```
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),trainingImages,
...
    'DataAugmentation',imageAugmenter);

augimdsValidation = augmentedImageDatastore(inputSize(1:2),testImages);
```

Successivamente vengono descritte alcune configurazioni messe in atto per addestrare il modello, rispettando i vincoli computazionali e di performance.

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',10, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',1e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',3, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

Dove:

- MiniBatch viene utilizzato per dividere il Dataset di training in piccoli batch, usati per calcolare i valori di errore per gli output.
- Un'epoca è un intero ciclo di addestramento della rete.

## 4. Elaborazione e analisi delle immagini

### Obiettivi

---

Dopo aver addestrato la rete neurale passando come input il Dataset originale, ci siamo preposti l'obiettivo di analizzare il comportamento della rete neurale a seconda delle varie applicazioni, salvando per ognuna le relative cartelle ottenute dalle modifiche al Dataset originale.

### Aggiunta e rimozione di rumore gaussiano attraverso il filtro di wiener su immagine grayscale

---

Nel primo caso d'uso analizzato ci siamo preoccupati di convertire ogni immagine appartenente al Dataset originale in un'immagine su scala di grigi, in quanto il filtro di Wiener non consente la riduzione del rumore su un input tridimensionale.

Conversione da RGB a grayscale.

```
I = rgb2gray(R);
```

dove R rappresenta l'immagine in input.

Aggiunta del rumore gaussiano all'immagine, con una varianza di 0.025

```
J = imnoise(I,'gaussian',0,0.025);
```

Una volta aggiunto il rumore Gaussiano ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla separatamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/NoiseGrayScale/n0/' ;  
imwrite(J,[destination,num2str(k),'.jpg']);
```

Riduzione del rumore attraverso una funzione wiener2 che filtra l'immagine grayscale usando un filtro Wiener passa-basso, dove [5 5] specifica la grandezza della sottomatrice del vicinato utilizzata per stimare il significato dell'immagine locale e la deviazione standard

```
K = wiener2(J,[5 5]);
```

Una volta eseguita la riduzione del rumore Gaussiano tramite il filtro di Wiener ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla successivamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/WienerGrayScale/n0/' ;  
imwrite(K,[destination,num2str(k),'.jpg']);
```

## Aggiunta e rimozione di rumore 'sale e pepe' attraverso il filtro di wiener su immagine RGB

In questo secondo caso, al contrario del precedente, le immagini non sono state convertite in grayscale bensì sono state analizzate in RGB. Il primo passo ha riguardato l'aggiunta di rumore sale e pepe ad ogni immagine del Dataset originale, con una varianza di 0.005

```
J = imnoise(I,'salt & pepper', 0.005);
```

Una volta aggiunto il rumore Sale e Pepe ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla separatamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/SaltPepperRGB/n0/' ;
```

Poichè il filtro di Wiener si applica ad immagini dotate di colore unidimensionale, abbiamo deciso dapprima di scompattare ogni immagine originale RGB nelle tre componenti, le quali verrano concatenate in un'immagine RGB alla fine del processo.

```
[noisyR,noisyG,noisyB] = imsplit(J);
```

Ottenute le singole scale di colore, abbiamo applicato il filtro di Wiener su ogni componente, con una matrice bidimensionale maggiore rispetto al caso precedente consentendoci di rimuovere in maniera accurata il rumore aggiunto.

```
x = wiener2(J(:,:,1), [8 8]);
y = wiener2(J(:,:,2), [8 8]);
z = wiener2(J(:,:,3), [8 8]);
```

Ricombiniamo questi tre canali per formare l'immagine RGB con riduzione del rumore.

```
denoisedRGB = cat(3,x,y,z);
```

Una volta eseguita la riduzione del rumore Sale e Pepe tramite il filtro di Wiener ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla successivamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/DenoisedSaltPepperRGB/n0/';
imwrite(denoisedRGB,[destination,num2str(k),'.jpg']);
```

## Aggiunta e rimozione di rumore gaussiano attraverso una rete pre-addestrata su immagine RGB ridimensionata

Nel terzo ed ultimo caso abbiamo effettuato un ridimensionamento delle immagini appartenenti al Dataset originale, successivamente è stato aggiunto un rumore gaussiano ad ogni immagine ridimensionata e infine abbiamo sfruttato una rete neurale pre-addestrata per la riduzione del rumore. L'esigenza di ridimensionare le foto è dovuta al fatto che la rete neurale DnCNN ha richiesto una potenza di calcolo considerevole. Come primo passo, ogni immagine viene letta e convertita in double.

```
pristineRGB = imread(filename);
pristineRGB = im2double(pristineRGB);
```

Ridimensioniamo l'immagine, specificando il fattore di scala e utilizzando il metodo di interpolazione predefinito e l'antialiasing.

```
J = imresize(pristineRGB, 0.5);
```

Aggiungiamo all'immagine un rumore bianco gaussiano con una varianza di 0,01. imnoise

aggiunge rumore ad ogni canale di colore in modo indipendente.

```
noisyRGB = imnoise(J,'gaussian',0,0.01);
```

Una volta aggiunto il rumore Gaussiano ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla separatamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/ResizeRGB/n0/';  
imwrite(J,[destination,num2str(k),'.jpg']);
```

Dividiamo l'immagine RGB con rumore nei singoli canali di colore.

```
[noisyR,noisyG,noisyB] = imsplit(noisyRGB);
```

Definiamo la rete DnCNN pre-addestrata.

```
net = denoisingNetwork('dncnn');
```

Sfruttiamo la rete DnCNN per rimuovere il rumore da ogni canale di colore.

```
denoisedR = denoiseImage(noisyR,net);  
denoisedG = denoiseImage(noisyG,net);  
denoisedB = denoiseImage(noisyB,net);
```

Ricombiniamo questi tre canali per formare l'immagine RGB con riduzione del rumore.

```
denoisedRGB = cat(3,denoisedR,denoisedG,denoisedG,denoisedB);
```

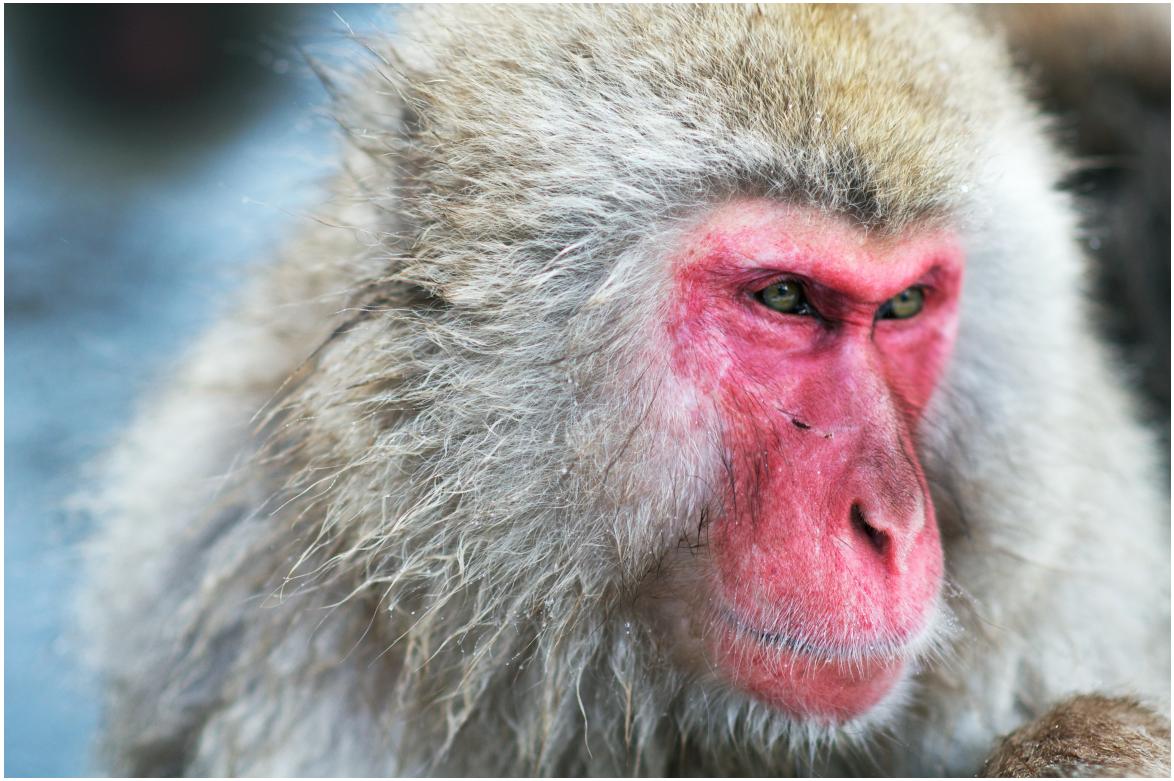
Una volta eseguita la riduzione del rumore Gaussiano tramite la rete DnCNN ci siamo preoccupati di andare salvare ogni immagine del Dataset in una cartella specifica, con il fine di analizzarla successivamente con la rete neurale Alexnet.

```
destination='/Users/nomeUtente/Desktop/ResizeDenoisedRGB/n0/';  
imwrite(denoisedRGB,[destination,num2str(k),'.jpg']);
```

## Esempio di immagine per i diversi casi d'uso

Di seguito viene rappresentata l'immagine appartenente alla sottocartella n3 (Macaco Giapponese) nei vari casi d'uso:

- Immagine Originale



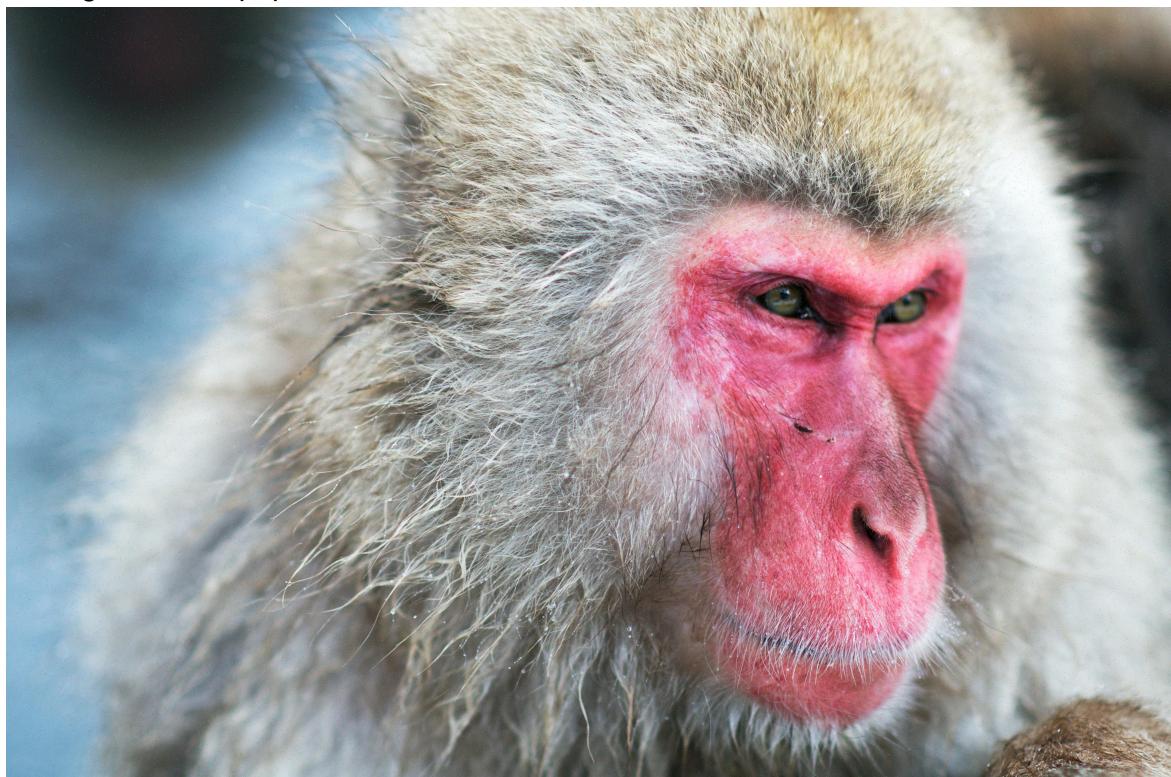
- Immagine Grayscale con rumore



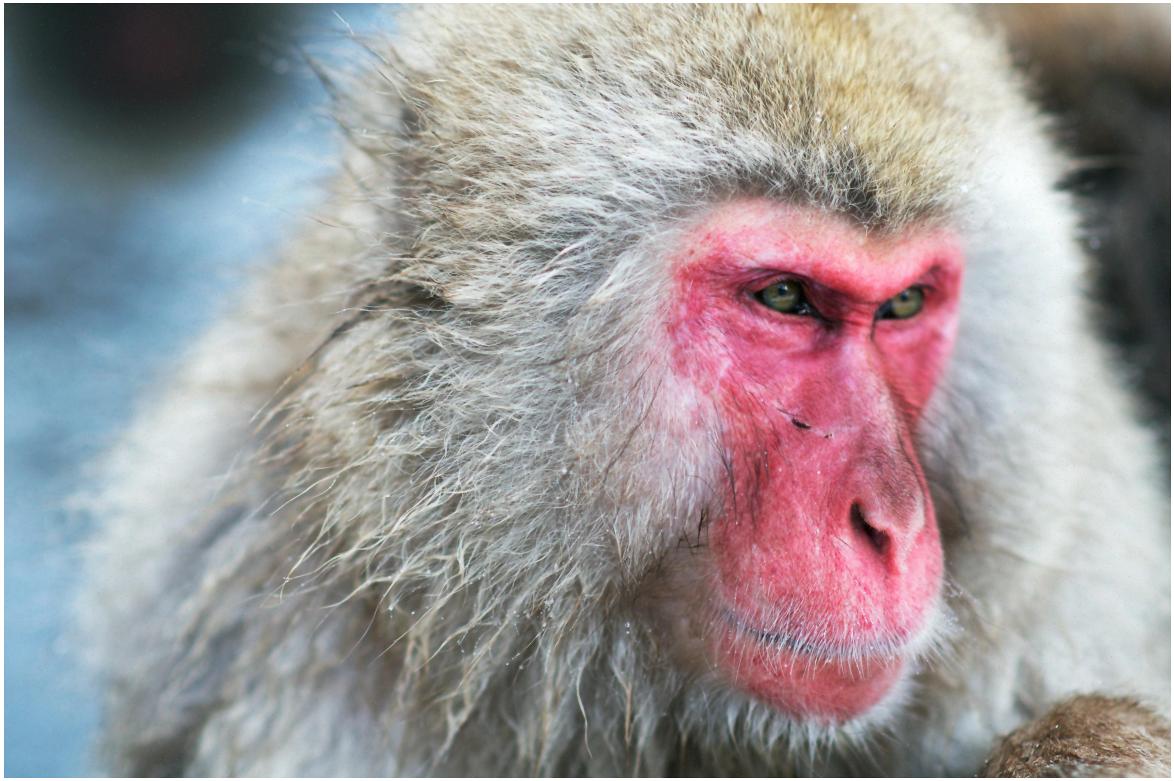
- Immagine Grayscale con riduzione



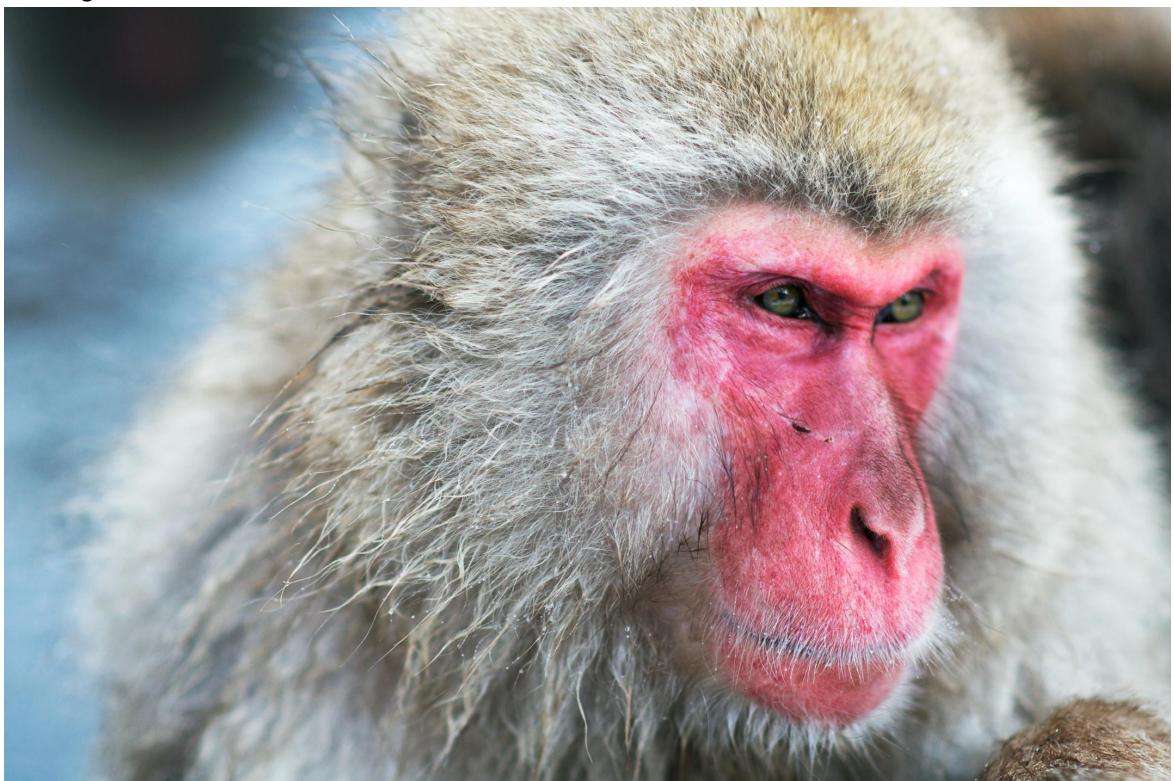
- Immagine Sale e pepe con rumore



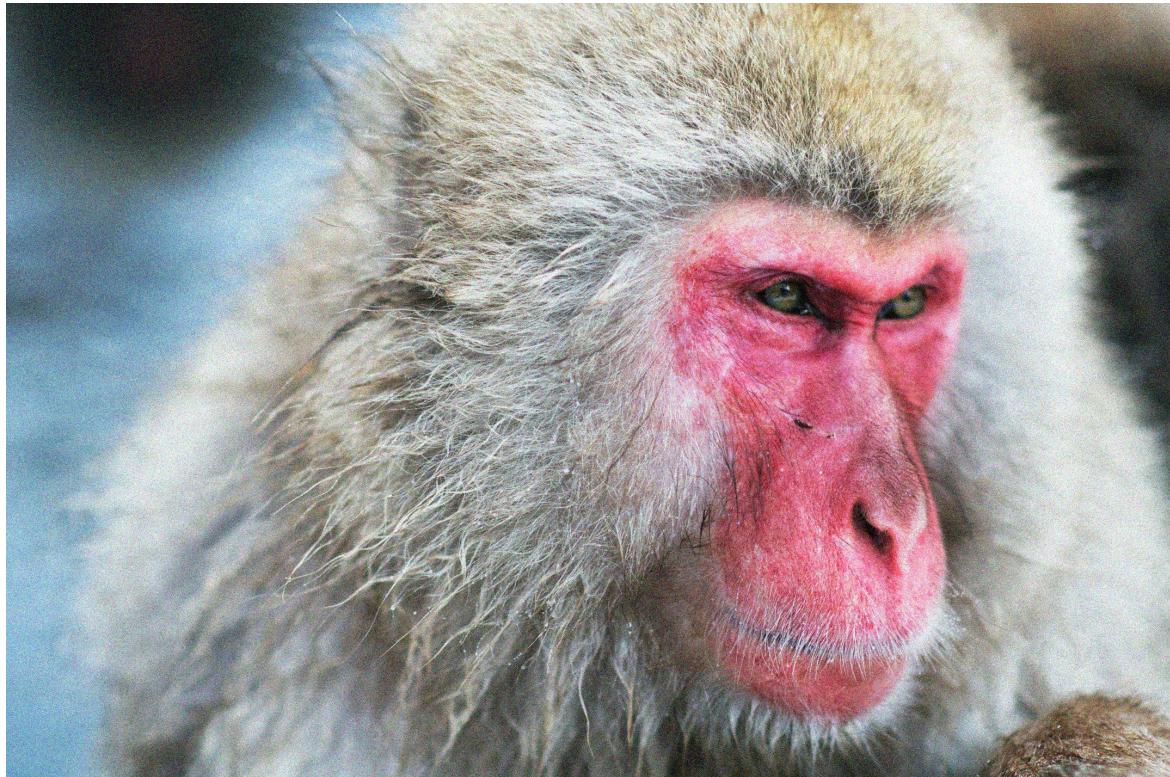
- Immagine Sale e pepe con riduzione di rumore



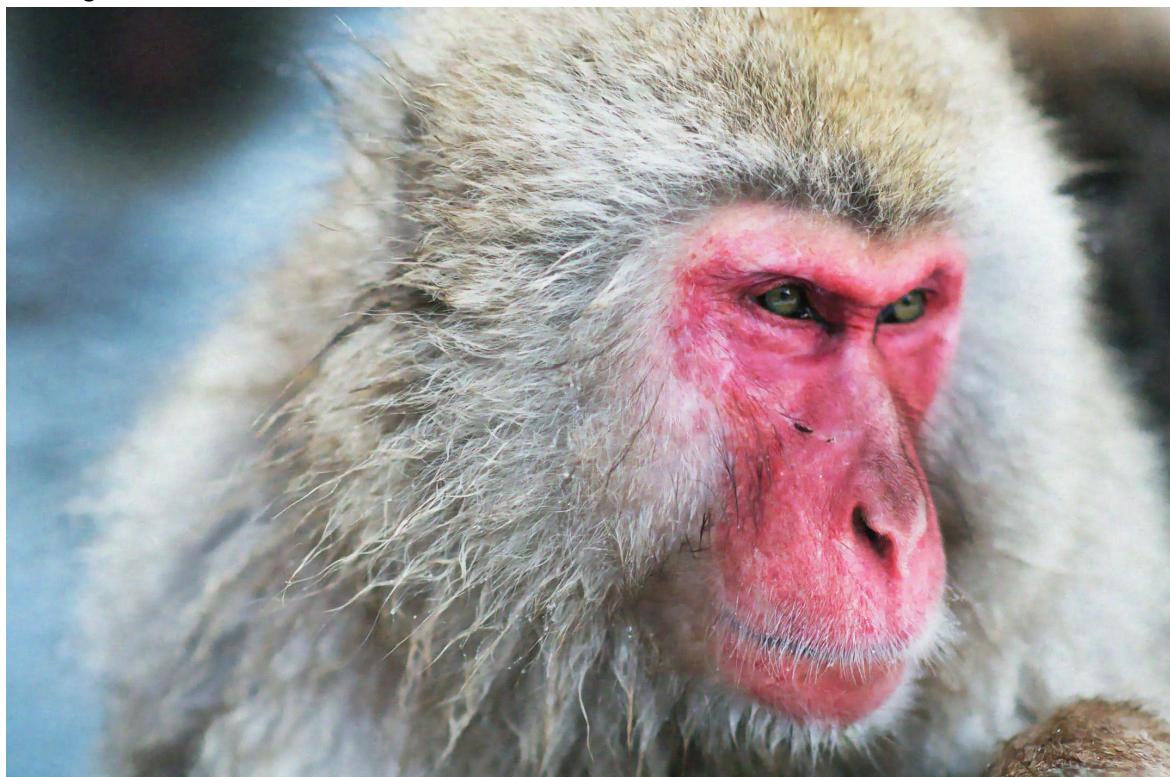
- Immagine ridimensionata



- Immagine ridimensionata con rumore



- Immagine ridimensionata con riduzione di rumore



## Analisi del colore

---

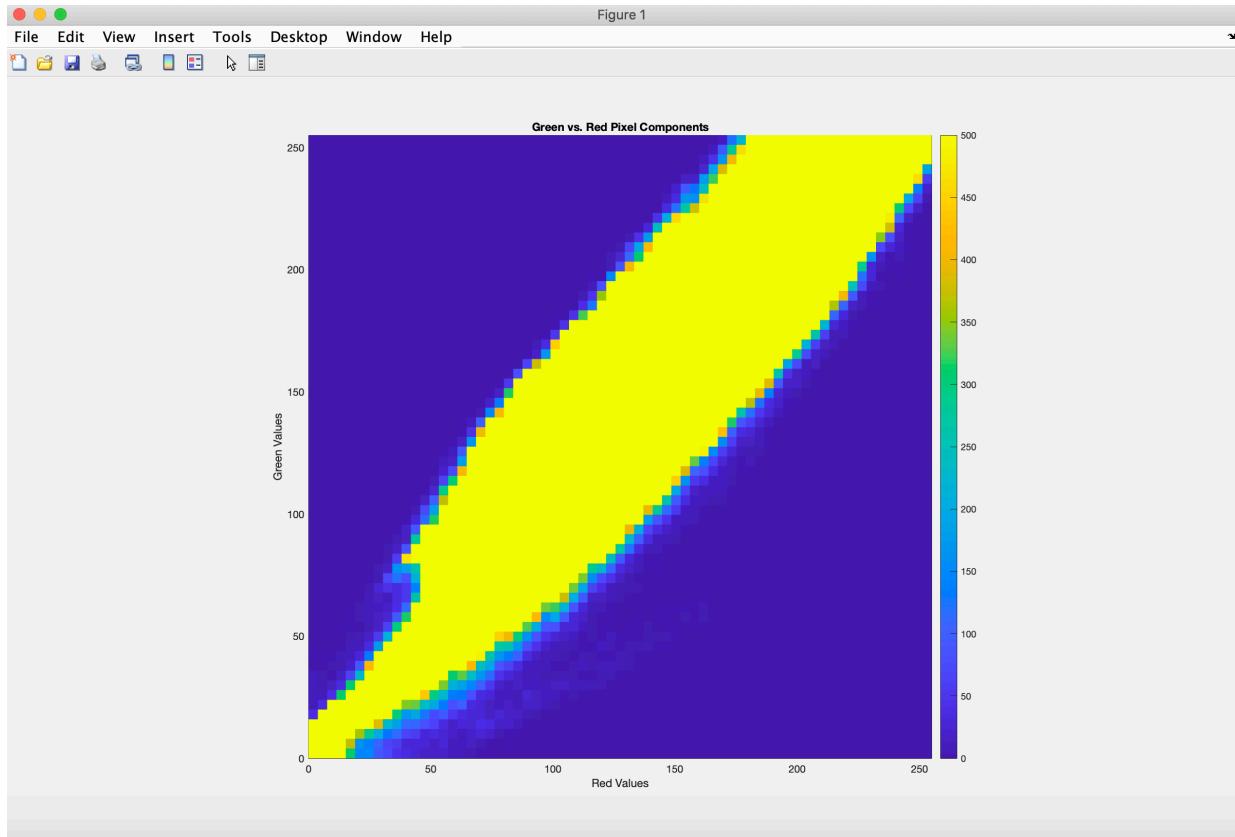
Il colore rappresenta un fattore chiave per la rappresentazione di oggetti attraverso immagini digitali. Un errato uso del colore potrebbe rendere l'immagine confusa, evidenziando in maniera ridotta le possibili caratteristiche di un oggetto, rendendo il task della classificazione più complesso.

Abbiamo perciò deciso di analizzare un'immagine del Dataset (no/0018) in termini di scala e distribuzione di colore, rappresentata di seguito:

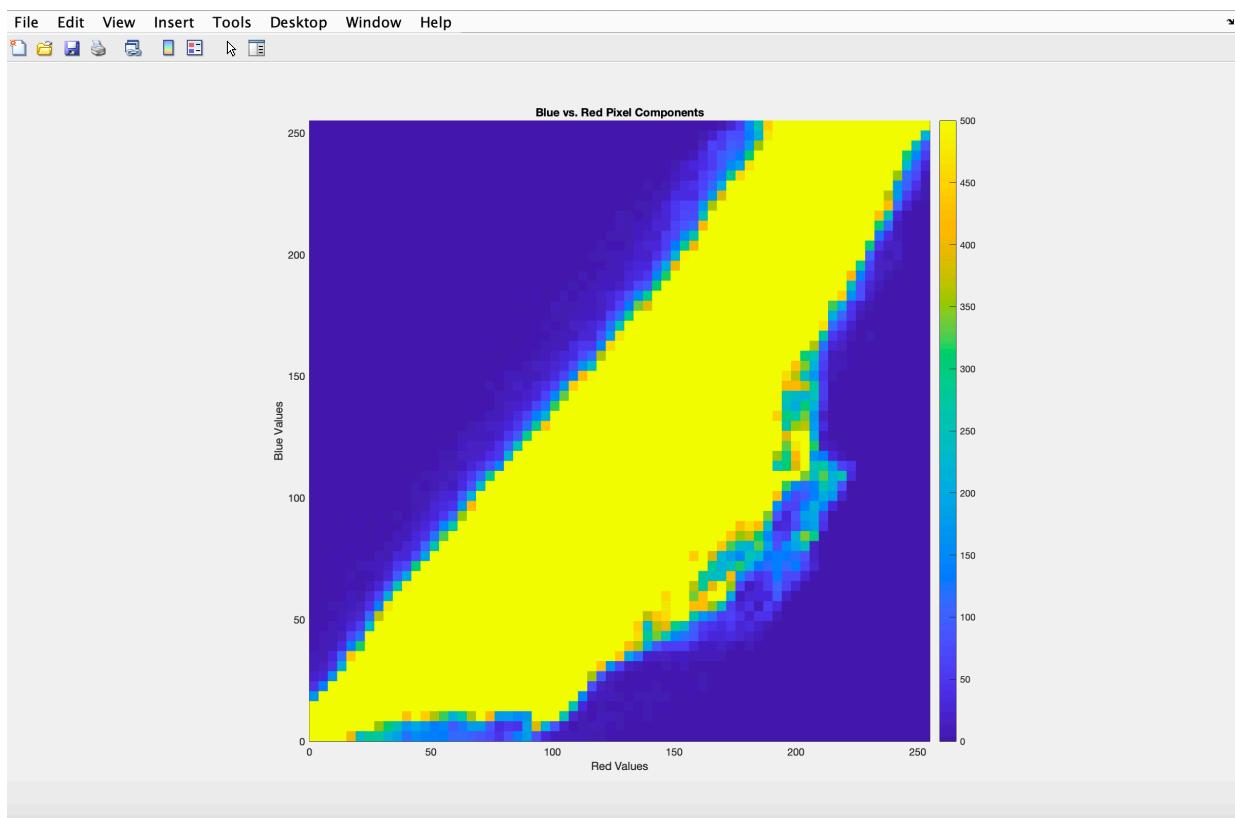


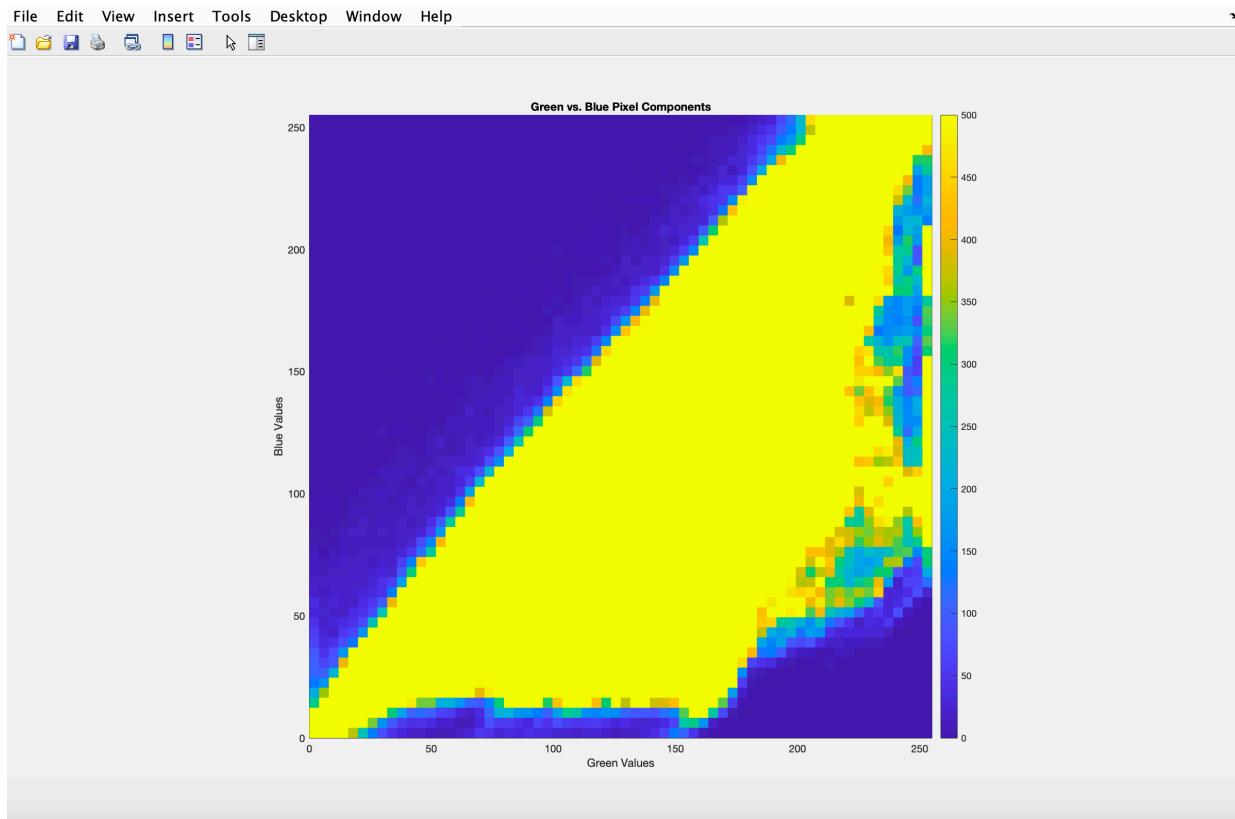
```
r = rgb(:,:,1);
g = rgb(:,:,2);
b = rgb(:,:,3);
histogram2(r,g,'DisplayStyle','tile','ShowEmptyBins','on', ...
    'XBinLimits',[0 255],'YBinLimits',[0 255]);
axis equal
colorbar
xlabel('Valori Rosso')
ylabel('Valori Blu')
title('Componenti a pixel Verde vs. Rosso')
ax = gca;
ax.CLim = [0 500];
```

Con l'utilizzo dello script sopra citato viene stampato un'istogramma bivariato, di seguito rappresentato, sulla distribuzione dei colori primari Rosso e Verde all'interno di ogni pixel.

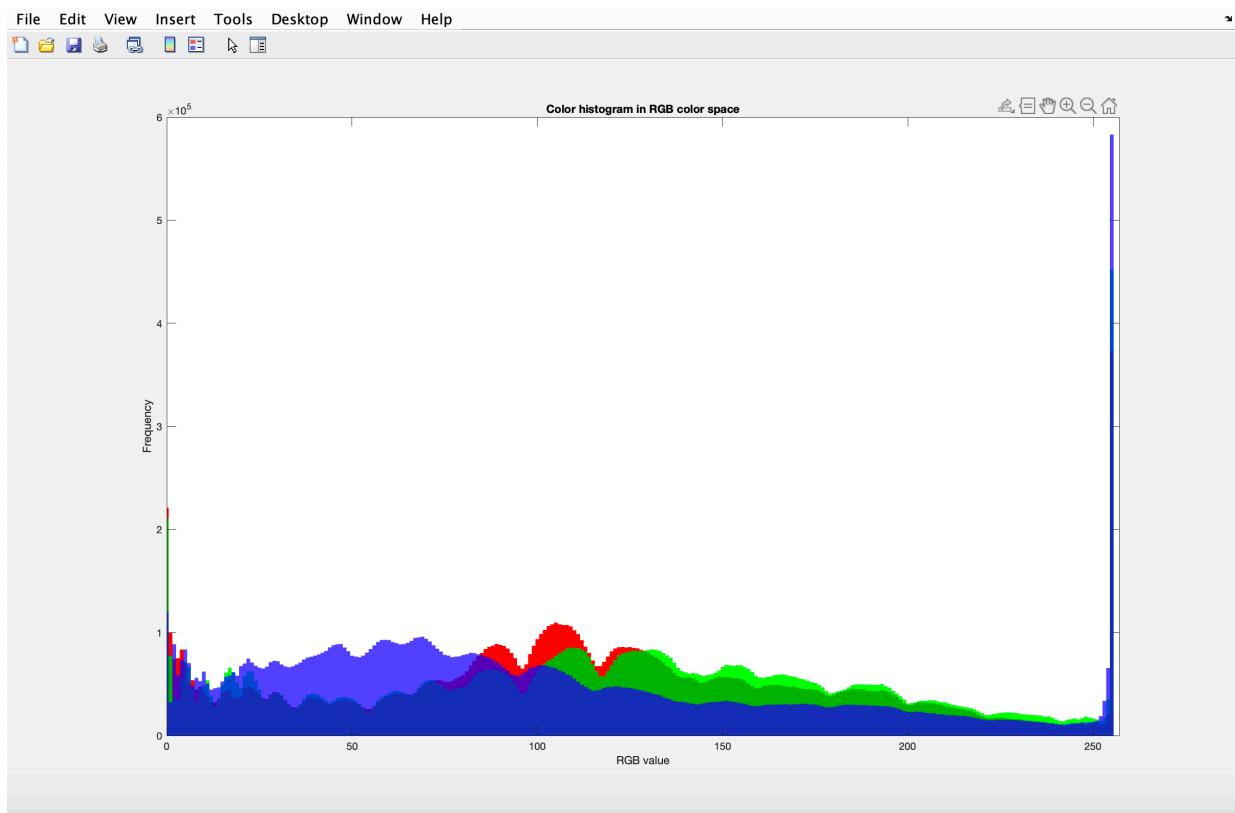


Con lo stesso metodo viene stampata una figura rappresentante la dominanza tra Rosso e Blu, come si evince nella prima, e Blu e Verde nella seconda.





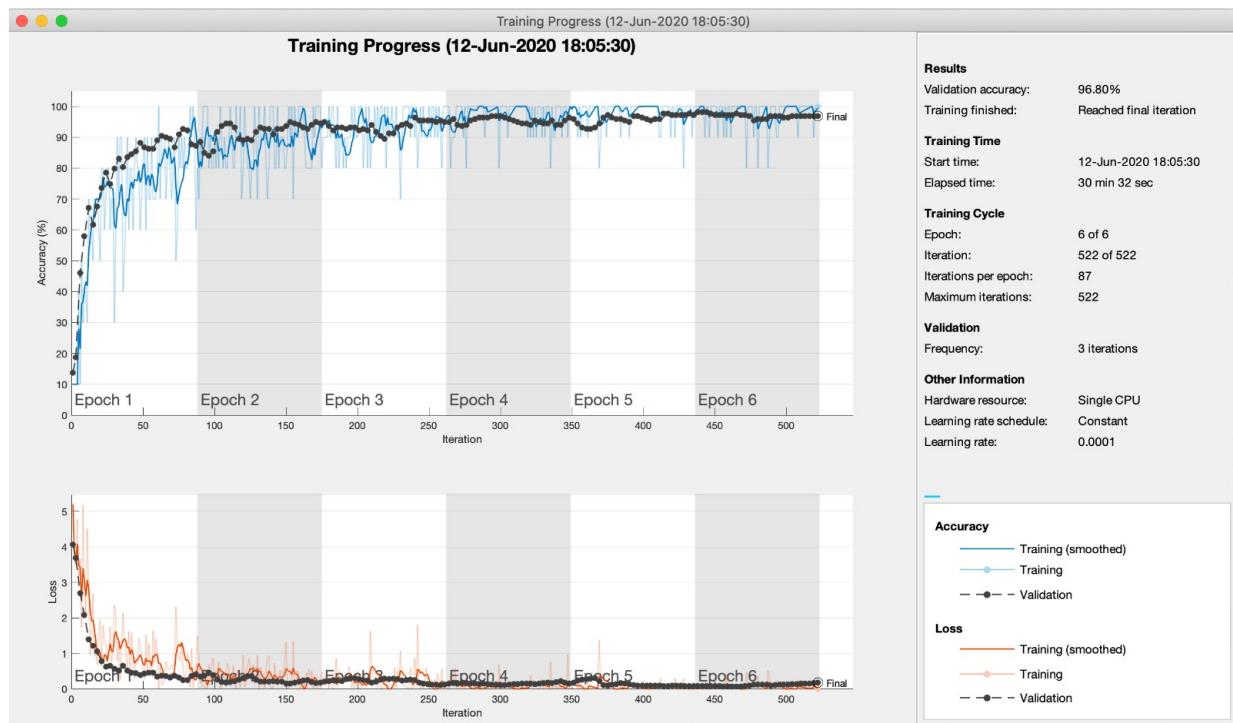
Per confermare i risultati, abbiamo deciso di stampare un istogramma a colori nello spazio RGB. Da esso viene evidenziata la predominanza del colore Blu rispetto alle altre componenti.



## 4. Valutazione risultati e conclusioni

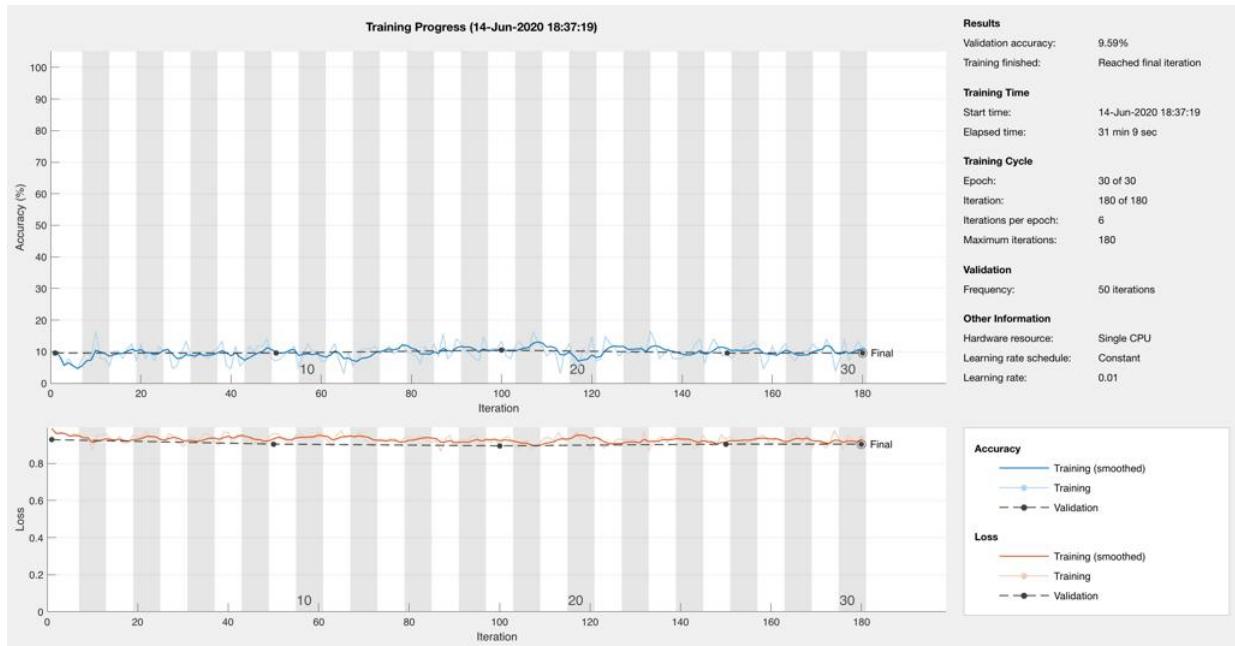
Questo elaborato si è focalizzato sull'analisi del comportamento della rete neurale AlexNet al variare degli input descritti nel capitolo 4. Di seguito vengono riportati i risultati ottenuti mediante il tool package di AlexNet:

### Addestramento rete neurale con dataset originale



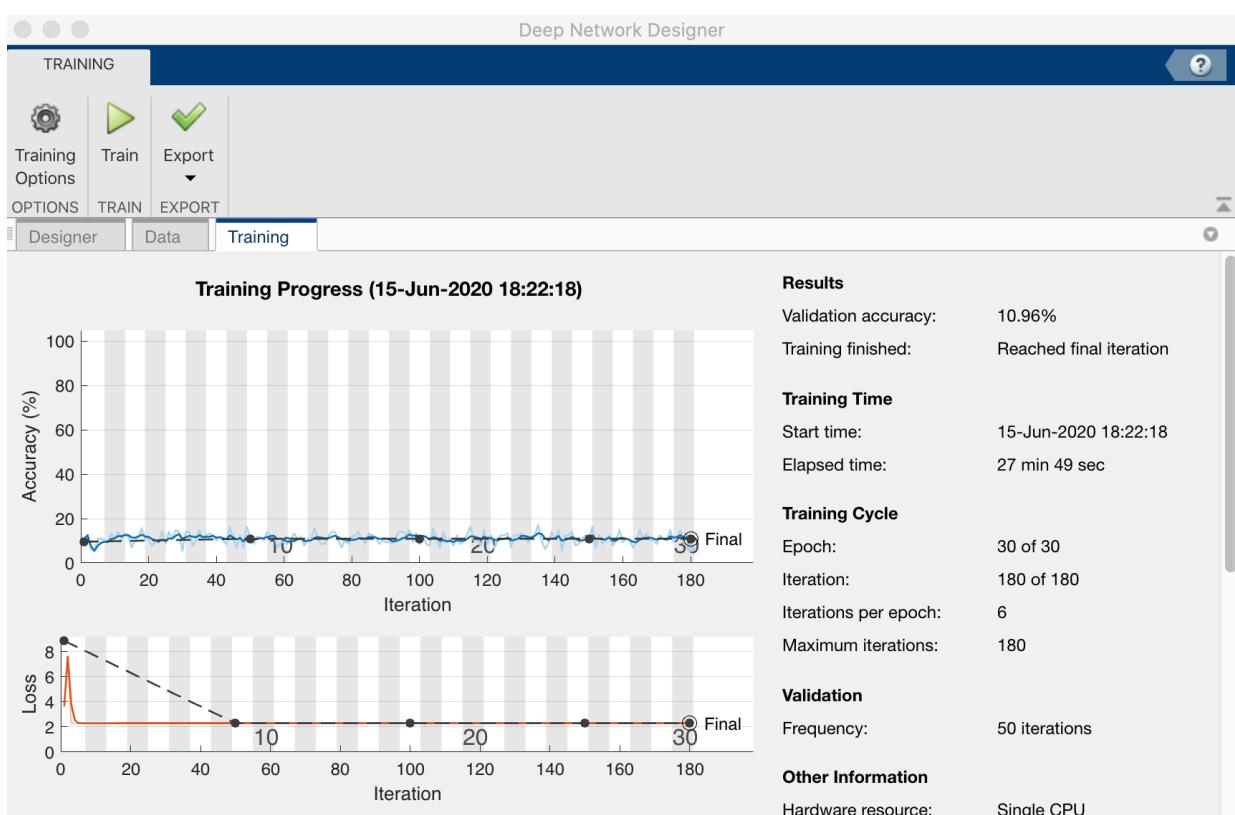
L'addestramento, effettuato su 6 epoche, presenta una precisione del 96.80% impiegando un tempo iterativo di 30 minuti.

### Addestramento rete neurale con aggiunta di rumore in scala di grigi



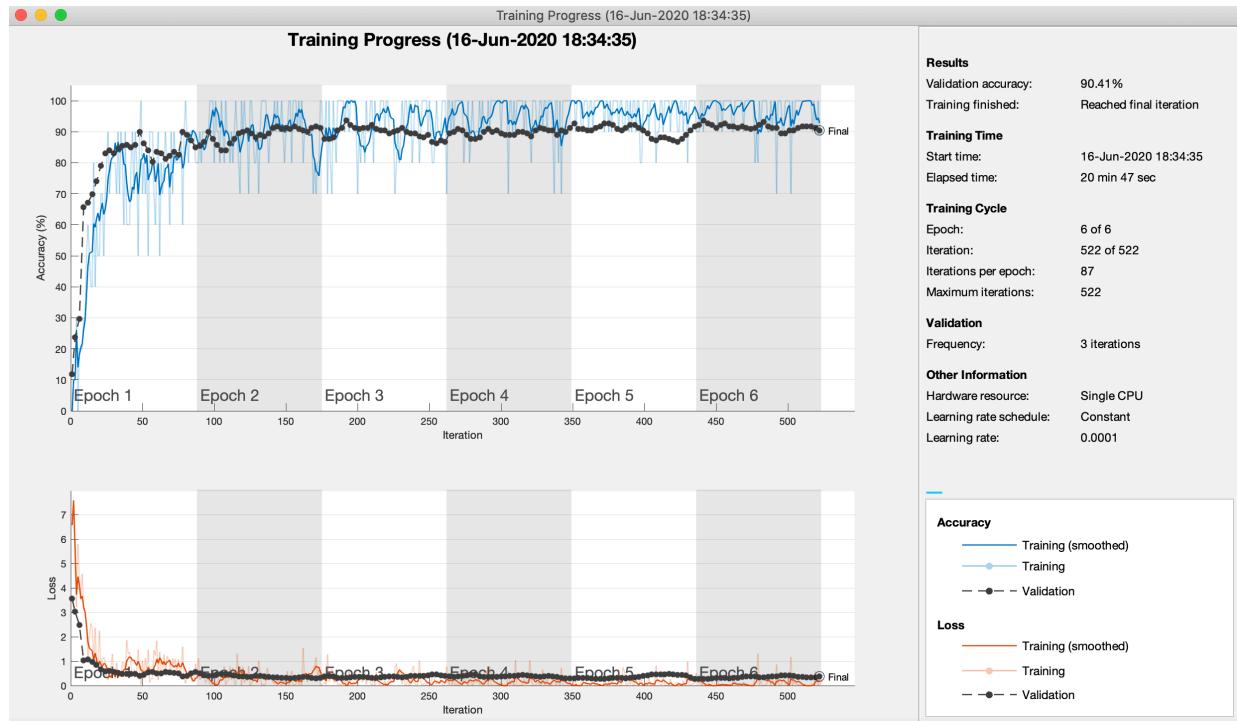
L'addestramento, effettuato su 6 epoch, mostra una precisione del 9.59% impiegando un tempo iterativo di 31 minuti.

## Addestramento rete neurale con riduzione di rumore in scala di grigi



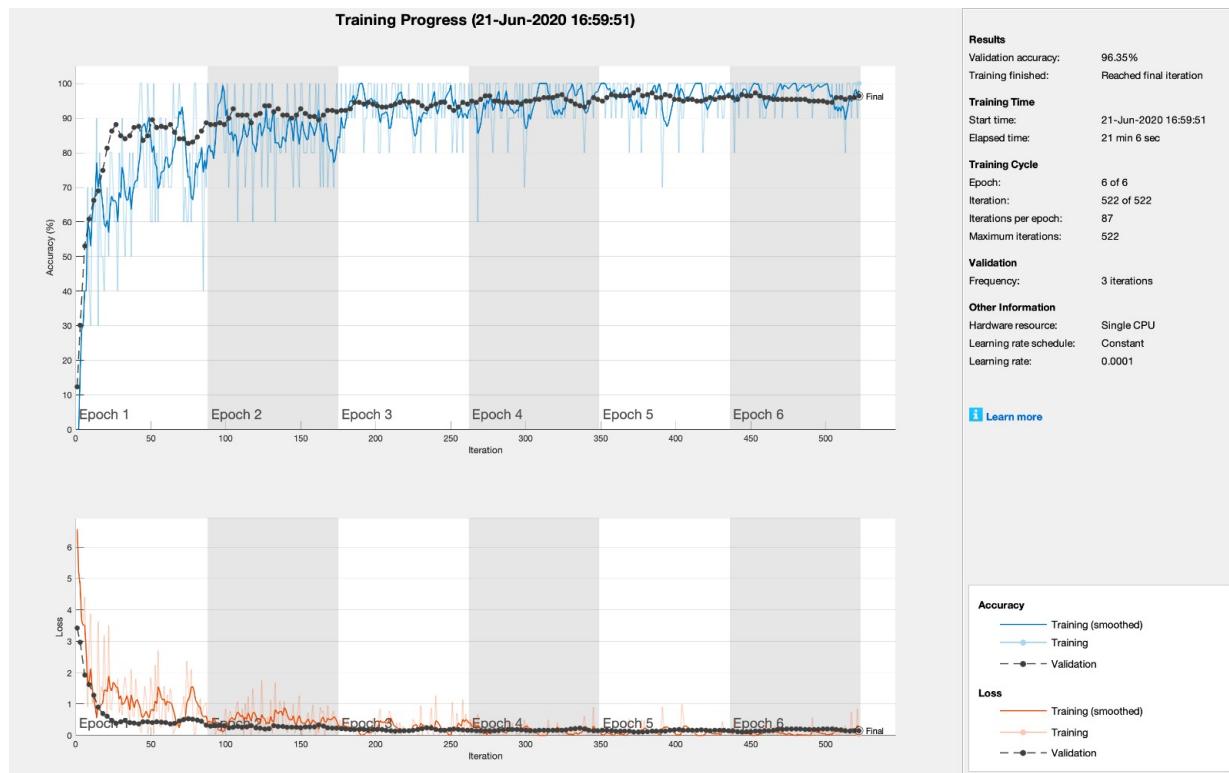
L'addestramento, effettuato su 6 epoch, mostra una precisione del 10.96% impiegando un tempo iterativo di 27 minuti.

# Addestramento rete neurale con aggiunta di rumore 'sale e pepe'



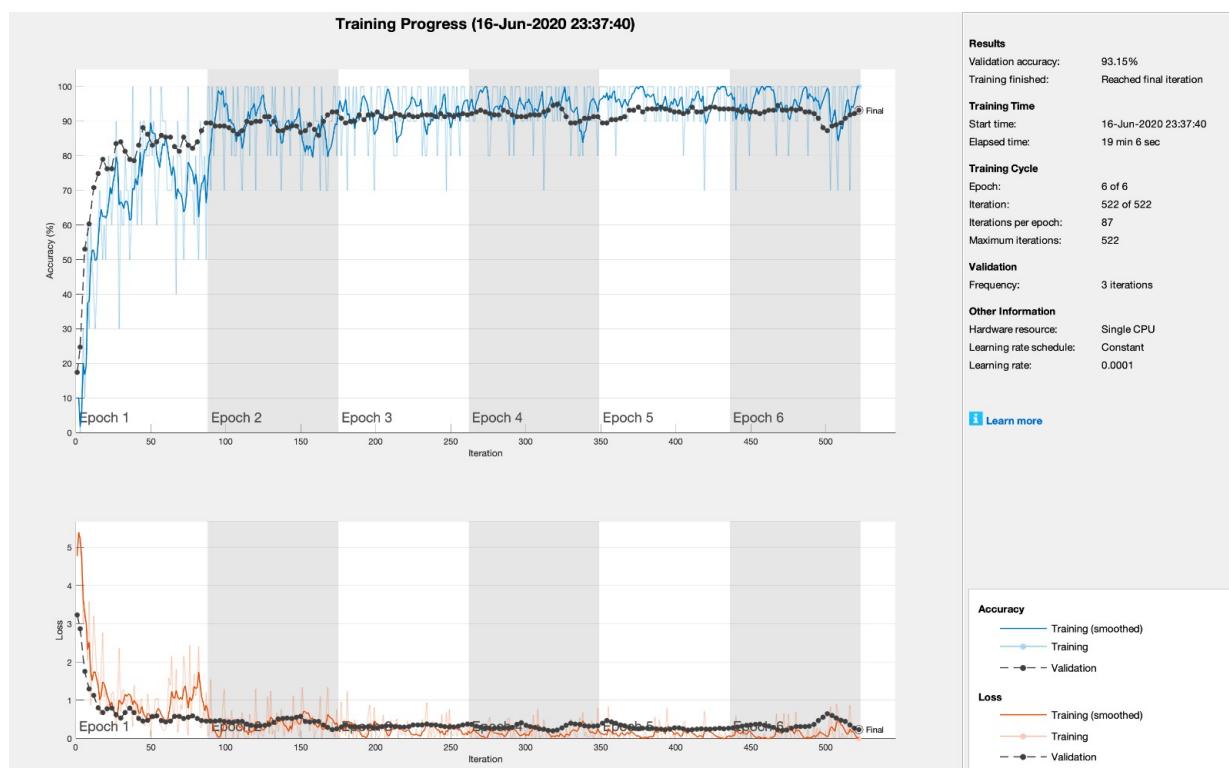
L'addestramento, effettuato su 6 epoche, mostra una precisione del 90.41% impiegando un tempo iterativo di circa 20 minuti.

# Addestramento rete neurale con riduzione di rumore 'sale e pepe'



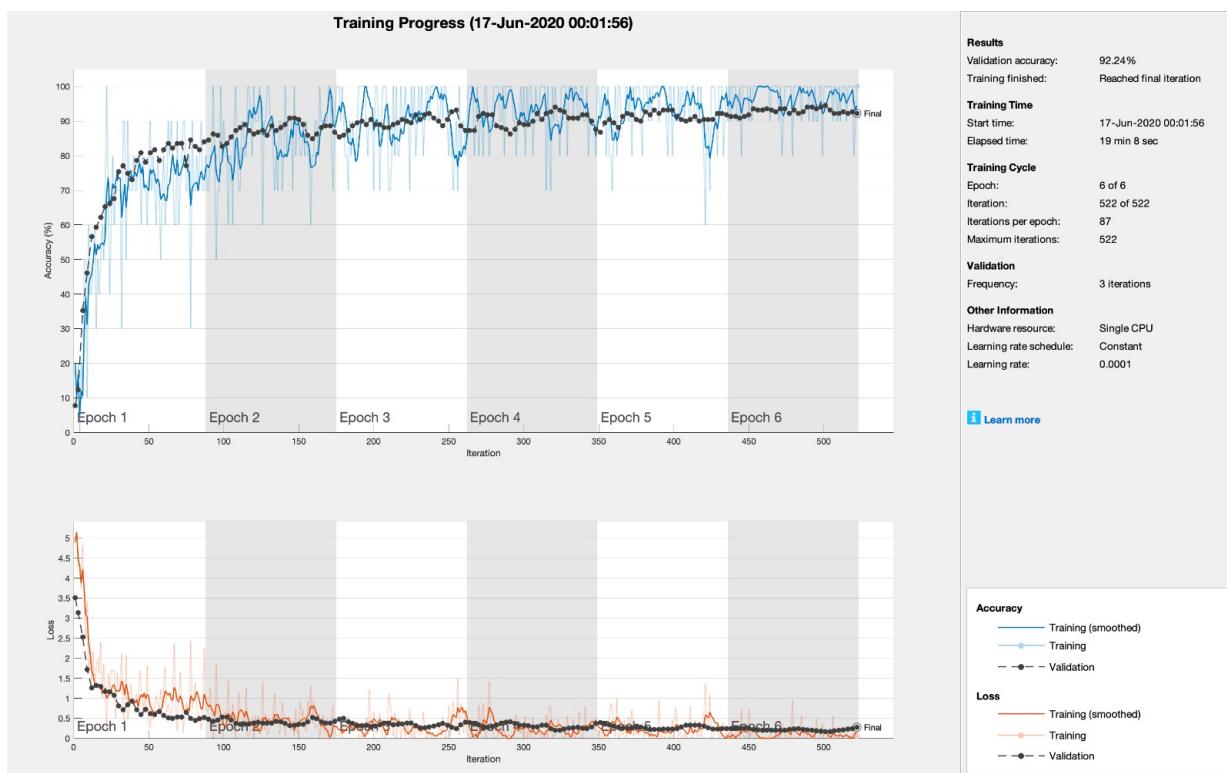
L'addestramento, effettuato su 6 epoche, mostra una precisione del 96.35% impiegando un tempo iterativo di circa 21 minuti.

## Addestramento rete neurale con dataset ridimensionato



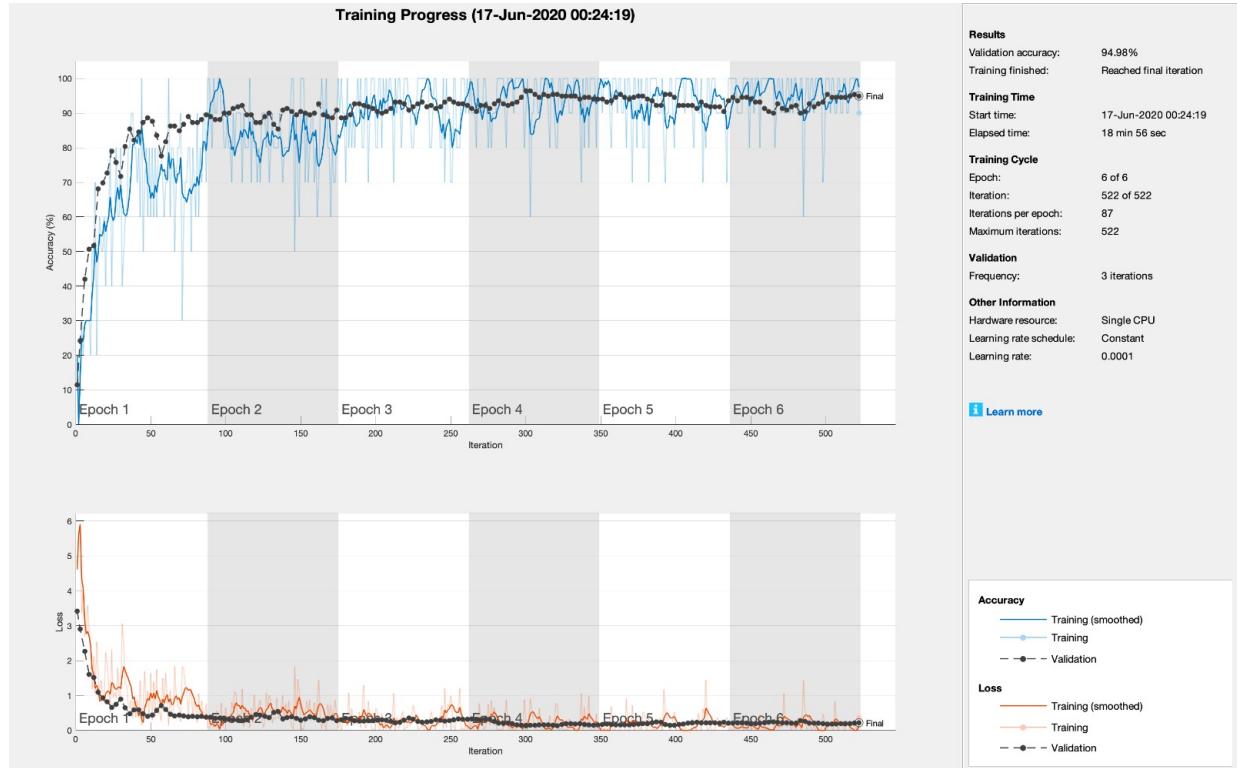
L'addestramento, effettuato su 6 epoch, mostra una precisione del 93.15% impiegando un tempo iterativo di circa 19 minuti.

## Addestramento rete neurale con dataset ridimensionato con aggiunta di rumore



L'addestramento, effettuato su 6 epoch, mostra una precisione del 92.24% impiegando un tempo iterativo di circa 19 minuti.

## Addestramento rete neurale con dataset ridimensionato con riduzione di rumore



L'addestramento, effettuato su 6 epoche, mostra una precisione del 94.98% impiegando un tempo iterativo di circa 19 minuti.

## Conclusioni

Come si evidenzia dai grafici precedenti, i risultati ottenuti dall'applicazione della rete neurale convoluzionale AlexNet hanno mostrato come, se da un lato l'accuracy calcolata sul dataset originale mostri un valore assoluto, dall'altro aumenta notevolmente il tempo di esecuzione. Quest'ultimo subisce una netta diminuzione in presenza del dataset ridimensionato, ottenendo prestazioni in termini di accuracy leggermente minori ma performance computazionali notevolmente migliori. Per quanto concerne la scelta del rumore additivo, gaussiano o sale e pepe, non genera variazioni significative tra i due casi. Piccola ma non indifferente variazione di accuracy si riscontra nei casi di aggiunta e rimozione del rumore mediante il filtro, sia nel caso di rumore gaussiano che sale e pepe. Infine possiamo notare come la conversione da immagine RGB a grayscale abbia reso molto difficile l'estrazione delle features da parte della rete neurale, concludendo l'addestramento con una precisione del 10%.