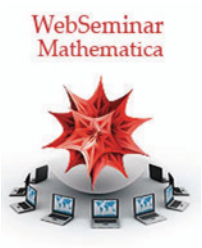
Wolfram Mathematica

*Il software di riferimento per la Didattica, la Ricerca e lo Sviluppo*




Lezione 8

# Elaborazione immagini

*Crescenzio Gallo – Università di Foggia*

*crescenzio.gallo@unifg.it*

*Note:*

- Il materiale visualizzato durante questo seminario è disponibile per il download all'indirizzo http://www.crescenziogallo.it/unifg/seminario-mathematica-2014/
- Per una migliore visione ingrandire lo schermo mediante il pulsante in alto a destra "Schermo intero"

# Agenda

Introduzione

- Creare le immagini
- Proprietà elementari delle immagini

Manipolazione di immagini

- Alcune funzioni elementari
- Operazioni dinamiche su immagini

Operazioni sui singoli punti o sui canali

- Applicare operazioni aritmentiche o più generali
- Tonalità, saturazione e luminosità

Esempi di applicazioni dell'image processing

Conclusioni

# Introduzione: creare le immagini

## » I dati di partenza

Le immagini su cui operare possono essere create o importate. Per creare immagini si possono usare diversi strumenti e diversi metodi. Vediamo alcuni esempi.

Creare immagini a partire dai dati.

La funzione da usare per creare immagini si chiama `Image`. Una matrice genera un'immagine con un singolo canale:

```
Image[im = RandomReal[1, {5, 5}]]
```

```
MatrixForm[im]
```

$$\begin{pmatrix} 0.9496 & 0.246 & 0.6897 & 0.2778 & 0.0008519 \\ 0.02118 & 0.1241 & 0.9008 & 0.4715 & 0.2208 \\ 0.3325 & 0.2758 & 0.04001 & 0.4491 & 0.745 \\ 0.3279 & 0.6973 & 0.3631 & 0.9131 & 0.01821 \\ 0.2287 & 0.4849 & 0.9854 & 0.9885 & 0.1079 \end{pmatrix}$$

Mentre un array con profondità 3 genera un'immagine multicanale:

```
Image[im3 = RandomReal[1, {5, 5, 3}]]
```

```
MatrixForm[im3]
```

$$\begin{pmatrix} \begin{pmatrix} 0.7822 \\ 0.7286 \\ 0.8888 \end{pmatrix} & \begin{pmatrix} 0.3924 \\ 0.8106 \\ 0.9703 \end{pmatrix} & \begin{pmatrix} 0.2983 \\ 0.1872 \\ 0.2135 \end{pmatrix} & \begin{pmatrix} 0.6028 \\ 0.7981 \\ 0.001201 \end{pmatrix} & \begin{pmatrix} 0.4054 \\ 0.4053 \\ 0.1843 \end{pmatrix} \\ \begin{pmatrix} 0.1362 \\ 0.2984 \\ 0.7251 \end{pmatrix} & \begin{pmatrix} 0.9537 \\ 0.3925 \\ 0.5633 \end{pmatrix} & \begin{pmatrix} 0.09337 \\ 0.9643 \\ 0.2743 \end{pmatrix} & \begin{pmatrix} 0.488 \\ 0.347 \\ 0.6209 \end{pmatrix} & \begin{pmatrix} 0.3443 \\ 0.9956 \\ 0.06542 \end{pmatrix} \\ \begin{pmatrix} 0.8993 \\ 0.4791 \\ 0.9067 \end{pmatrix} & \begin{pmatrix} 0.7779 \\ 0.9484 \\ 0.4202 \end{pmatrix} & \begin{pmatrix} 0.03734 \\ 0.2066 \\ 0.8046 \end{pmatrix} & \begin{pmatrix} 0.6868 \\ 0.6291 \\ 0.9401 \end{pmatrix} & \begin{pmatrix} 0.2821 \\ 0.2589 \\ 0.739 \end{pmatrix} \\ \begin{pmatrix} 0.6172 \\ 0.887 \\ 0.716 \end{pmatrix} & \begin{pmatrix} 0.2946 \\ 0.1044 \\ 0.8756 \end{pmatrix} & \begin{pmatrix} 0.5564 \\ 0.4806 \\ 0.2773 \end{pmatrix} & \begin{pmatrix} 0.7746 \\ 0.1772 \\ 0.847 \end{pmatrix} & \begin{pmatrix} 0.2485 \\ 0.9913 \\ 0.4784 \end{pmatrix} \\ \begin{pmatrix} 0.5983 \\ 0.3588 \\ 0.9509 \end{pmatrix} & \begin{pmatrix} 0.4268 \\ 0.7746 \\ 0.2838 \end{pmatrix} & \begin{pmatrix} 0.4493 \\ 0.3488 \\ 0.4344 \end{pmatrix} & \begin{pmatrix} 0.8212 \\ 0.9901 \\ 0.454 \end{pmatrix} & \begin{pmatrix} 0.186 \\ 0.447 \\ 0.2619 \end{pmatrix} \end{pmatrix}$$

Per default le immagini create con **Image** sono di tipo “Real” e non assumono uno specifico valore per lo spazio colori, ossia l’opzione **ColorSpace** assume valore **Automatic**. Comunque si possono specificare sia il tipo di immagine sia il suo spazio colori:

```
im4 = Image[{{128, 255, 0}, {255, 128, 255}, {0, 255, 128}}, "Byte"]
```

```
InputForm[im4]
```

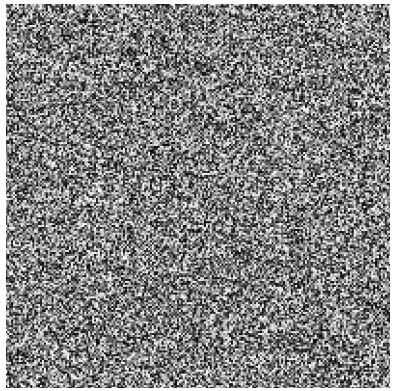
```
Image[RawArray["Byte", {{128, 255, 0}, {255, 128, 255}, {0, 255, 128}}], "Byte",
 ColorSpace -> Automatic, Interleaving -> None]
```

```
im5 = Image[CellularAutomaton[30, {{1}, 0}, 20], "Bit"]
```

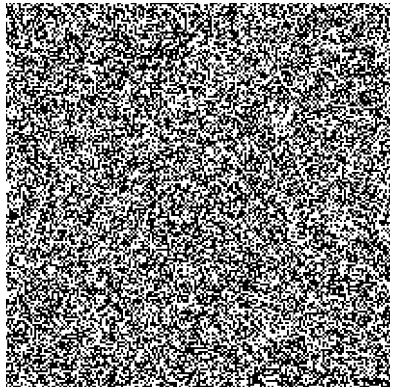
```
InputForm[im5]
Image[RawArray["Byte", {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1
  0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0
  0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0
  0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0,
  0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0
  0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0
  1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0
  1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
  0}, {0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0
  1, 0, 0, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1
  1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0
  1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 0
  1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
  1, 0, 0, 0}, {0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1
  1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1
  1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0}, {1, 1, 0, 0, 1, 0, 0, 0, 1
  1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1}}
 "Bit", ColorSpace -> Automatic, Interleaving -> None]
```

Si può anche usare `RandomImage` per generare immagini random a partire da dati di una certa distribuzione:

```
RandomImage[UniformDistribution[], {200, 200}]
```



```
RandomImage[BinomialDistribution[1, .5], {200, 200}]
```



# Introduzione: creare le immagini

## » Il tipo “Real”

In una matrice che genera un’immagine di tipo “Real” i valori possono anche essere esterni all’intervallo [0, 1] ma nella visualizzazione dell’immagine essi verrano considerati comunque nel range [0,1]:

```
dati = Table[i / 10., {20}, {i, -10, 20}];
```

```
dati[[1]]
```

```
{-1., -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0., 0.1, 0.2, 0.3,
 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.}
```

```
im = Image[dati]
```

Comunque i valori internamente alla variabile non sono modificati fisicamente, ossia sono sempre quelli originali:

```
ImageData[im][[1]] === dati[[1]]
```

```
True
```

Se si desidera scalare i valori originali nell’intervallo [0, 1] si può usare la funzione `ImageAdjust`

```
im1 = ImageAdjust[im]
```

Questi sono i valori scalati:

```
ImageData[im1][[1]]
```

```
{0., 0.03333, 0.06667, 0.1, 0.1333, 0.1667, 0.2, 0.2333, 0.2667, 0.3,
 0.3333, 0.3667, 0.4, 0.4333, 0.4667, 0.5, 0.5333, 0.5667, 0.6, 0.6333,
 0.6667, 0.7, 0.7333, 0.7667, 0.8, 0.8333, 0.8667, 0.9, 0.9333, 0.9667, 1.}
```

# Introduzione: creare le immagini

## » I tipi “Integer”

I tipi di immagini “Bit”, “Byte”, “Bit16” possono accettare solo valori interi in specifici intervalli. Altri valori sono arrotondati (reali) o tagliati (esterni)

Ad esempio il tipo `"Bit"` può contenere solo valori 0 e 1

```
Quit[]
```

```
i = Image[{{-1, -0.3, 0, .3, .5, 1, 1.5, 2}}, "Bit"]
```



```
ImageData[i]
```

```
{{0, 0, 0, 0, 1, 1, 1, 1}}
```

Come si vede dalla InputForm i dati sono effettivamente modificati

```
InputForm[i]
```

```
Image[RawArray["Byte", {{0, 0, 0, 0, 1, 1, 1, 1}}], "Bit", ColorSpace -> Automatic,
 Interleaving -> None]
```

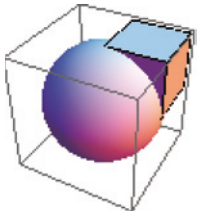
# Introduzione: creare le immagini

## » Immagini a partire da grafici di *Mathematica*

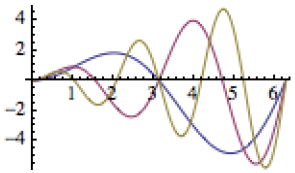
Si possono anche generare immagini con le funzioni native di *Mathematica* dedicate alla grafica:

```
im = Image[Graphics[{Pink, Disk[], Cyan, Rectangle[]}, ImageSize → 100]]
```

```
Image[Graphics3D[{Sphere[], Cuboid[]}], ImageSize → 100]
```



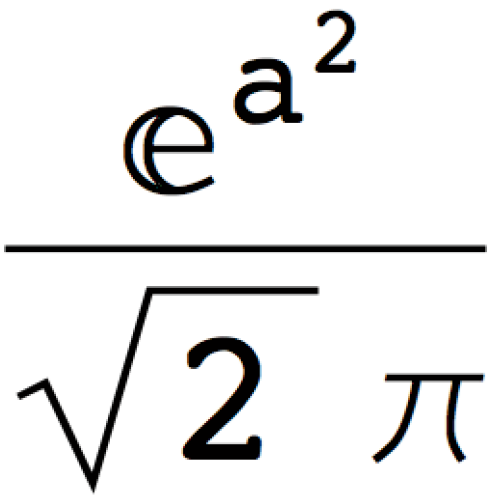
```
i = Image[Plot[Evaluate[Table[x Sin[n x], {n, 1, 3}]], {x, 0, 2 Pi}, ImageSize → 150]]
```




Mentre **Image** necessita di una matrice di valori o di un oggetto **Graphics**, `Rasterize` fornisce analoghe funzionalità

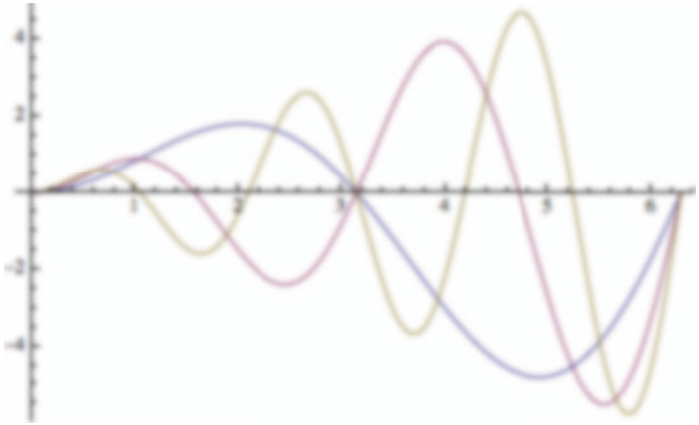
per qualsiasi oggetto di *Mathematica* e dunque può essere usata per creare immagini da espressioni:

```
Rasterize[Exp[a^2]/(π Sqrt[2]), "Image", ImageResolution → 400]
```



Molte funzioni della sezione “image processing” sono in grado di accettare sia immagini che grafici di *Mathematica*. Esempio di filtro di Gauss:

```
GaussianFilter[Plot[Evaluate[Table[x Sin[n x], {n, 1, 3}]], {x, 0, 2 Pi}], 3]
```



# Introduzione: creare le immagini

## » Import da file esterni

Ovviamente le immagini possono anche essere importate da file esterni a *Mathematica*

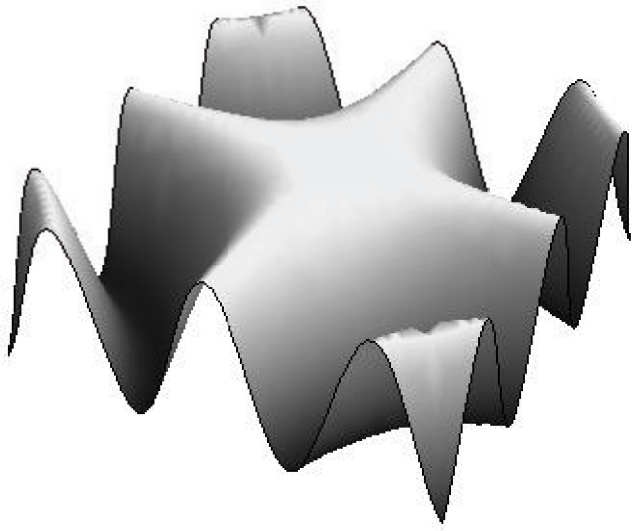
```
$ImportFormats
```

```
{3DS, ACO, Affymetrix, AgilentMicroarray, AIFF, ApacheLog, ArcGRID, AU, AVI, Base64,
 BDF, Binary, Bit, BMP, Byte, BYU, BZIP2, CDED, CDF, Character16, Character8, CIF,
 Complex128, Complex256, Complex64, CSV, CUR, DBF, DICOM, DIF, DIMACS, Directory,
 DOT, DXF, EDF, EPS, ExpressionML, FASTA, FASTQ, FCS, FITS, FLAC, GenBank, GeoTIFF,
 GIF, GPX, Graph6, Graphlet, GraphML, GRIB, GTOPO30, GXL, GZIP, HarwellBoeing,
 HDF, HDF5, HIN, HTML, ICC, ICNS, ICO, ICS, Integer128, Integer16, Integer24,
 Integer32, Integer64, Integer8, JCAMP-DX, JPEG, JPEG2000, JSON, JVX, KML, LaTeX,
 LEDA, List, LWO, MAT, MathML, MBOX, MDB, MGF, MIDI, MMCIF, MOL, MOL2, MPS, MTP, MTX,
 MX, NASACDF, NB, NDK, NetCDF, NEXUS, NOFF, OBJ, ODS, OFF, OpenEXR, Package, Pajek,
 PBM, PCX, PDB, PDF, PGM, PLY, PNG, PNM, PPM, PXR, QuickTime, RawBitmap, Real128,
 Real32, Real64, RIB, RSS, RTF, SCT, SDF, SDTS, SDTSDEM, SFF, SHP, SMILES, SND, SP3,
 Sparse6, STL, String, SurferGrid, SXC, Table, TAR, TerminatedString, Text, TGA, TGF,
 TIFF, TIGER, TLE, TSV, UnsignedInteger128, UnsignedInteger16, UnsignedInteger24,
 UnsignedInteger32, UnsignedInteger64, UnsignedInteger8, USGSDEM, UUE, VCF, VCS,
 VTK, WAV, Wave64, WDX, XBM, XHTML, XHTMLMathML, XLS, XLSX, XML, XPORT, XYZ, ZIP}
```

**Import** legge sia dal file system del computer sia da qualsiasi URL accessibile:

```
SetDirectory[NotebookDirectory[]];
```

```
Import["Plot3D.jpg"]
```



```
Import["http://photojournal.jpl.nasa.gov/jpeg/PIA12708.jpg"]
```

```
Import[
 "http://1.bp.blogspot.com/_FJsabaNYeEU/SwguHlv0JYI/AAAAAAAAAAs/mDcZOJ-wzCo/s1600/leone-
   leonessa.jpg"]
```

Ci sono anche alcune immagini di test in *Mathematica* nella banca dati ExampleData

```
Import["ExampleData/Lena.tif"]
```

```
ExampleData["TestImage"]
```

```
Map[ExampleData, ExampleData["TestImage"]]
```

◄ | ►

# Introduzione: proprietà elementari delle immagini

## » Lo spazio colori

Quando si generano immagini dai dati e non si specifica lo spazio colori, *Mathematica* cerca di definire uno schema di mix dei canali per visualizzare al meglio l'immagine. Comunque si può specificare lo spazio colori che si vuole usare:

```
im = Image[RandomReal[1, {5, 5, 3}], ColorSpace → "RGB"]
```

Se non specifichiamo un **ColorSpace** a 4 canali l'opzione **ColorSpace→Automatic** fa sì che si assuma la quarta dimensione come il valore per il canale *alfa*:

```
im = Image[RandomReal[1, {5, 5, 4}]]
```

```
InputForm[im]
```

```
Image[{{{0.27822194314005455, 0.7715826543364765, 0.5698971892552291, 0.64428992497823 08},
 {0.9349743295042146, 0.4312182544747678, 0.47445938010077 48, 0.655359092582922},
 {0.32420882025515496, 0.5455108628574248, 0.949891609813563, 0.6981174431914985},
 {0.8755537860199141, 0.7325444866291708, 0.9129333119720855, 0.23804104498597267},
 {0.8914610904061833, 0.9389534005774358, 0.12748012116730534, 0.0664416108397976}},
 {{0.4052984059280422, 0.257562846454646, 0.0560800885316137 25, 0.68705942304494 69},
 {0.5924307391912695, 0.6490324776221594, 0.2741944029556902, 0.9362432991871166},
 {0.4180474787354713, 0.20277071177593053, 0.6413169850589575, 0.7442402746171948},
 {0.3821825010824804, 0.8636918084354428, 0.2345246806514356, 0.063331822334443 01},
 {0.5948573237055126, 0.6554134959614826, 0.4902479634971433, 0.4545487571482041}},
 {{0.014966265186419303, 0.6020517015595677, 0.4588869364283803, 0.1062321399423 4826},
 {0.8283824164355829, 0.09581980143731128, 0.27850497382870154, 0.4101214297695266},
 {0.9075846542387502, 0.2571077251439273, 0.8108984296469461, 0.7737481982334917},
 {0.6656388636095101, 0.3772628606749098, 0.11700069707951744, 0.8186395066298098},
 {0.388447560013528, 0.13444455945315537, 0.6754969467956269, 0.4303913470382 9873}},
 {{0.6062131012984813, 0.9419407384879166, 0.8721107316630121, 0.8884329272726015},
 {0.17689163750018677, 0.5110781190938312, 0.064289190390153 46, 0.4734817944085 5276},
 {0.14781566870986307, 0.15509333512173962, 0.8593586325799847, 0.7947246474409488},
 {0.6037200053720804, 0.8989578205769504, 0.5456064403133429, 0.6147087692719306},
 {0.5369626432633072, 0.8863049540175378, 0.8167283951580722, 0.0861479793940882 5}},
 {{0.7136912026240156, 0.9493970022873035, 0.15699090990520692, 0.8249596774247905},
 {0.6016808458336464, 0.9706413930774098, 0.25379507432819337, 0.0287903787482382},
 {0.05537660160540647, 0.46176273608270435, 0.6072703158000872, 0.970791295554436},
 {0.14159733643239347, 0.49791649426173823, 0.36933844186477405, 0.6084020034654707},
 {0.4180450334618222, 0.24103732192060234, 0.12504408531670852, 0.8347480951083 05}}}, "Real",
 ColorSpace -> Automatic, Interleaving -> True]
```

```
im = Image[RandomReal[1, {5, 5, 4}], ColorSpace → "CMYK"]
```

Con profondità 5 si può specificare il grado di opacità (canale *alfa*) per gli spazi colore a 4 canali:

```
im = Image[RandomReal[1, {5, 5, 5}], ColorSpace → "CMYK"]
```

Gli spazi colore riconosciuti da **Image** sono:

Grayscale → Livelli di grigio
RGB → Rosso, verde, blu
CMYK → Ciano, Magenta, Giallo, Nero
HSB → Tonalità, saturazione, luminosità

# Introduzione: proprietà elementari delle immagini

## » Interallacciata vs. planare

*Mathematica* supporta due modi di combinazione dei canali: *interleaved* (interallacciata) e *planar* (planare). Per default, i dati sono considerati nella forma interallacciata:

```
im = {{{1, 0, 0}, {0, 1, 0}}, {{0, 0, 1}, {1, 1, 0}}};
ii = Image[im, ColorSpace → "RGB", Interleaving → True]
```

```
MatrixForm[im]
```

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \end{pmatrix}$$

Un'immagine in forma "planare" può essere rappresentata da una lista di matrici, ciascuna rappresentante un singolo canale:

```
pm = {{{1, 0}, {0, 1}}, {{0, 1}, {0, 1}}, {{0, 0}, {1, 0}}};
pi = Image[pm, ColorSpace → "RGB", Interleaving → False]
```

```
MatrixForm[pm]
```

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix}$$

```
MatrixForm /@ pm
```

$$\left\{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\right\}$$

`ImageData` restituisce sempre i dati in forma interallacciata, anche se l'immagine è in forma planare:

```
ImageData[pi] // MatrixForm
```

$$\begin{pmatrix} \begin{pmatrix} 1. \\ 0. \\ 0. \end{pmatrix} & \begin{pmatrix} 0. \\ 1. \\ 0. \end{pmatrix} \\ \begin{pmatrix} 0. \\ 0. \\ 1. \end{pmatrix} & \begin{pmatrix} 1. \\ 1. \\ 0. \end{pmatrix} \end{pmatrix}$$

Volendo si può usare anche in **ImageData** l'opzione **Interleaving**:

```
ImageData[pi, Interleaving → False] // MatrixForm
```

$$\begin{pmatrix} \begin{pmatrix} 1. \\ 0. \end{pmatrix} & \begin{pmatrix} 0. \\ 1. \end{pmatrix} \\ \begin{pmatrix} 0. \\ 1. \end{pmatrix} & \begin{pmatrix} 0. \\ 1. \end{pmatrix} \\ \begin{pmatrix} 0. \\ 0. \end{pmatrix} & \begin{pmatrix} 1. \\ 0. \end{pmatrix} \end{pmatrix}$$

`ColorSeparate` genera una lista di immagini ciascuna rappresentante i singoli canali:

```
i =
```

```
;
```

```
ColorSeparate[i]
```

Al contrario, `ColorCombine` combina una lista di immagini di singoli canali in un'immagine multicanale:

**ColorCombine**[{  ,  ,  }]

# Introduzione: proprietà elementari delle immagini

## » Dimensione immagine e zoom

`ImageSize` e `Magnification` sono due opzioni che modificano il modo in cui un'immagine viene visualizzata. Per default `ImageSize` è impostata al valore **Automatic**, che generalmente ingrandisce immagini piccole e ridimensiona immagini che non entrano nelle dimensioni della finestra del notebook.

```
Image[{{0, 1, 0}, {1, 0, 1}, {0, 1, 0}}]
```

Ad esempio il valore **Automatic** permette di visualizzare un'immagine ad una dimensione più grande se si trova da sola in una cella:

Mentre la ridimensiona se si inserisce in una riga di testo, ad esempio per un input:

```
Image[ , ImageSize → All]
```

`Magnification` solitamente si usa per scalare l'immagine o fare uno zoom di ingrandimento:

```
Image[ , Magnification → 2]
```

Se si usano entrambi le opzioni, **ImageSize** prevale:

```
Image[  , ImageSize → 10, Magnification → 3]
```

# Manipolazione di immagini: proprietà elementari delle immagini

## » Alcune funzioni elementari

Usando il menu contestuale che compare su un'immagine, si può selezionare l'opzione Get Coordinates dal menù contestuale per prelevare le singole coordinate di un punto nell'immagine o di una regione definita da due o più coordinate:

```
i =
```

`ImageTrim` estrae una regione definita in termini di coordinate:

```
ImageTrim[i, {{10, 10}, {107, 143}}]
```

`ImageCrop` può essere usata allo stesso modo definendo le righe/colonne da eliminare

```
ImageCrop[i, {100, 120}, Bottom]
```

`ImageCrop` è una delle funzioni disponibili nel menu contestuale delle immagini

```
i
```

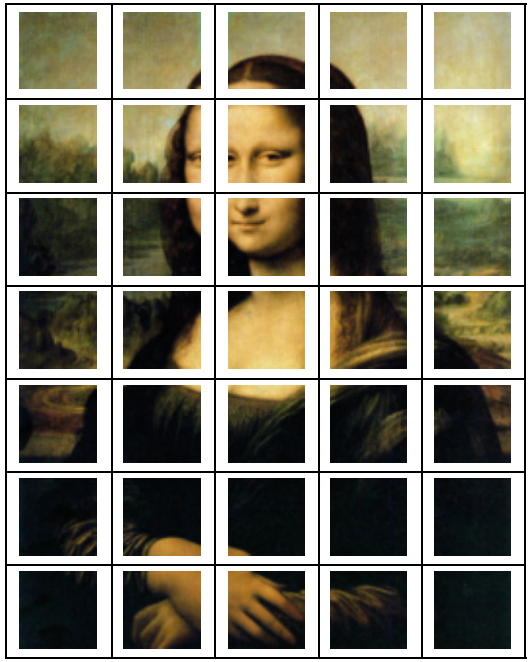
`ImageValue` consente di estrarre valori numerici dei singoli pixel:

```
ImageValue[i, {100, 100}]
{0.01176, 0.04706, 0.04706}
```

`ImagePartition` può essere utilizzata per suddividere un'immagine in tante piccole immagini:

```
g = ImagePartition[i, 40];
```

```
Grid[g, Frame → All]
```



`ImageAssemble` ricostruisc e un'immagine da una lista di "tasselli":

```
ImageAssemble[Partition[RandomSample[Flatten[g], Length[Flatten[g]]], 5]]
```

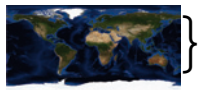
`ImageRotate` ruota un'immagine di un angolo arbitrario. Il centro dell'immagine è il centro di rotazione e l'angolo è considerato antiorario:

```
Manipulate[ImageRotate[ , θ], {θ, 0, 2 π}]
```

# Manipolazione di immagini: proprietà elementari delle immagini

## » Operazioni dinamiche su immagini

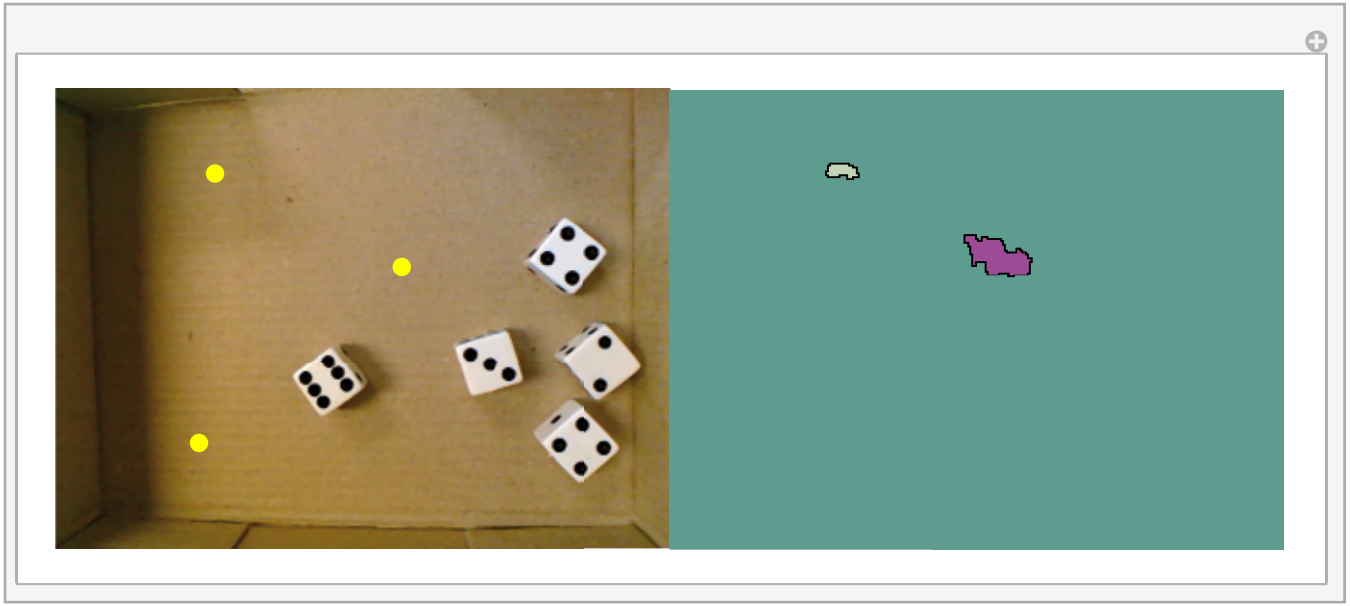
Molte operazioni si possono eseguire dinamicamente sulle immagini così da rendere più semplice lo studio di particolare caratteristiche delle immagini:

```
Module[{i =  },
 Manipulate[
  ImageResize[ImageCrop[i, {300, 300} / zoom, -pts],
   {300, 300}, Resampling → ControlActive["Nearest", "Lanczos"]],
  {{pts, {0, 0}}, {-1, -1}, {1, 1}},
  {{zoom, 1}, 1, 10}
 ]]
```


Un esempio di tecniche di segmentazione dove **Manipulate** consente di collocare i punti focali di una segmentazione "watershed" (bacino) per evidenziare un particolare da una certa immagine. Nell'esempio, il particolare da evidenziare con la tecnica di *watershed segmentation* sono i due tubercoli articolari che si trovano in corrispondenza dei bulbi oculari nell'immagine:

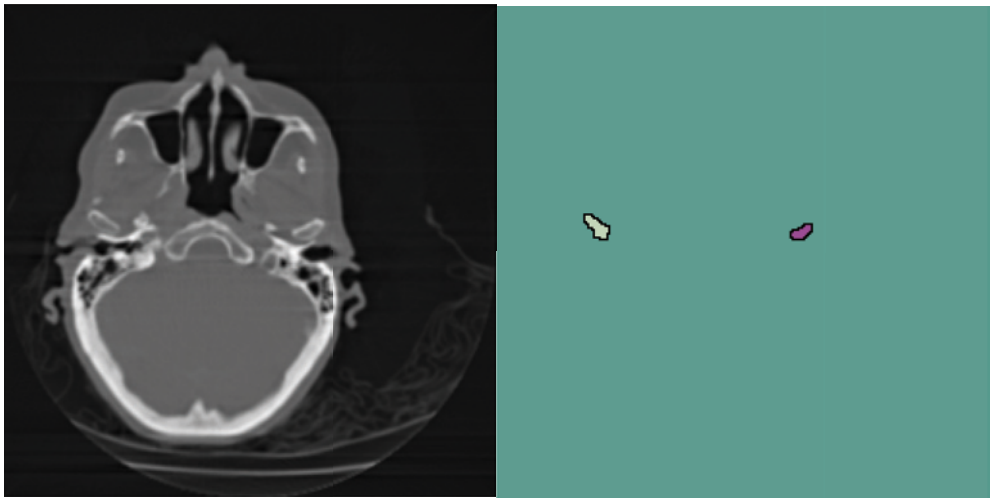
```
i = <image>;
```

```
Manipulate[
 Row[{Show[i], Image[WatershedComponents[i, pts] // Colorize, ImageSize → All]}],
 {{pts, RandomReal[Min[ImageDimensions[i]], {3, 2}]}, {0, 0}, ImageDimensions[i],
  Locator, Appearance -> Graphics[{Yellow, Disk[{0, 0}]}, ImageSize → 10],
  LocatorAutoCreate → {2, 10}}]
```



Una volta individuata la posizione utile, si usa **Paste Snapshot** dal menù della **Manipulate** per prelevare i valori numerici dei locator mossi manualmente:

```
DynamicModule[
 {pts = {{154.5`, 137.5`}, {51.5`, 141.5`}, {167.46081666362903`, 40.241060512337015`}}},
 Row[{Show[i], Image[Colorize[WatershedComponents[i, pts]], ImageSize → All]}]]
```

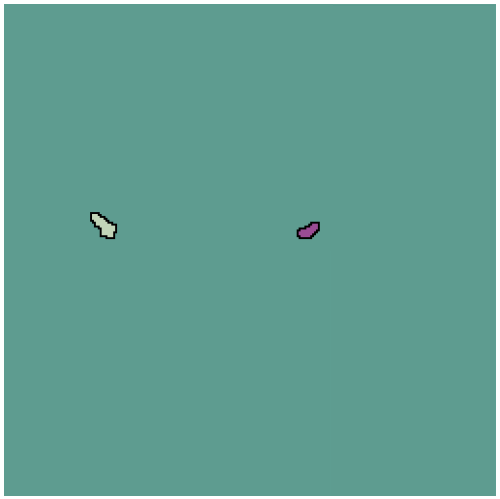


```
pts = {{154.5`, 137.5`}, {51.5`, 141.5`}, {167.46081666362903`, 40.241060512337015`}}
```

```
{{154.5, 137.5}, {51.5, 141.5}, {167.5, 40.24}}
```

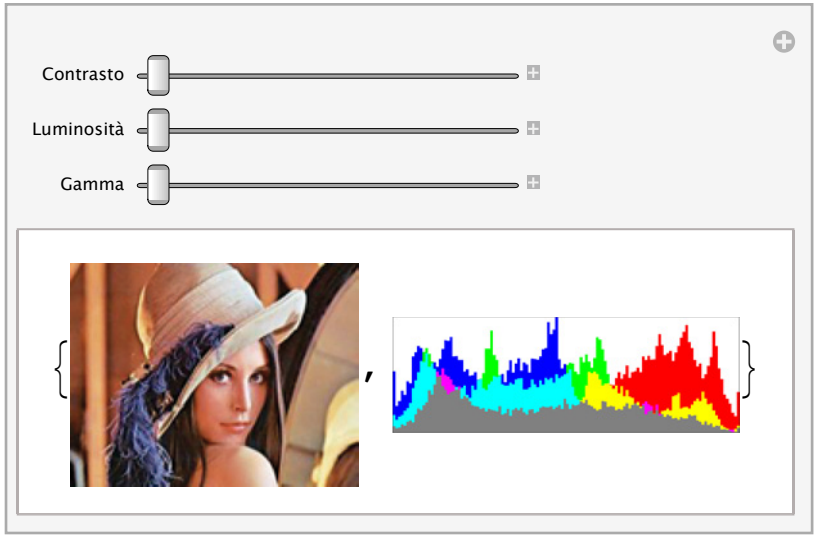
```
label = WatershedComponents[i, pts];
```

**Colorize[label]**



Un altro esempio con **ImageAdjust** che modifica i valori dei singoli canali applicando particolari formule. Alcune operazioni comuni sono contrasto, luminosità, gamma.

```
Manipulate[With[{i = ImageAdjust[ , {c, b, g}]}, {i, ImageHistogram[i]}],
 {{c, 0, "Contrasto"}, 0, 1}, {{b, 0, "Luminosità"}, 0, 1}, {{g, 1, "Gamma"}, 1, 4}]
```



# Operazioni sui singoli punti o sui canali

## » Applicare operazioni aritmetiche o più generali

`ImageAdd`, `ImageSubtract`, e `ImageMultiply` consenono di eseguire semplici operazioni aritmetiche sui valori dei singoli canali:

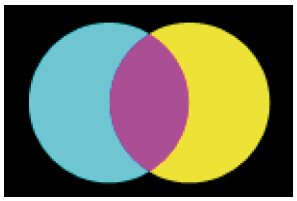
**i =**  **;**

**ImageAdd[i, .5]**

**ImageMultiply[i, .5]**

`ImageDifference` restituisce l'immagine con le distanze tra pixel corrispondenti nelle due immagini:

```
ImageDifference[ ● , ● ]
```



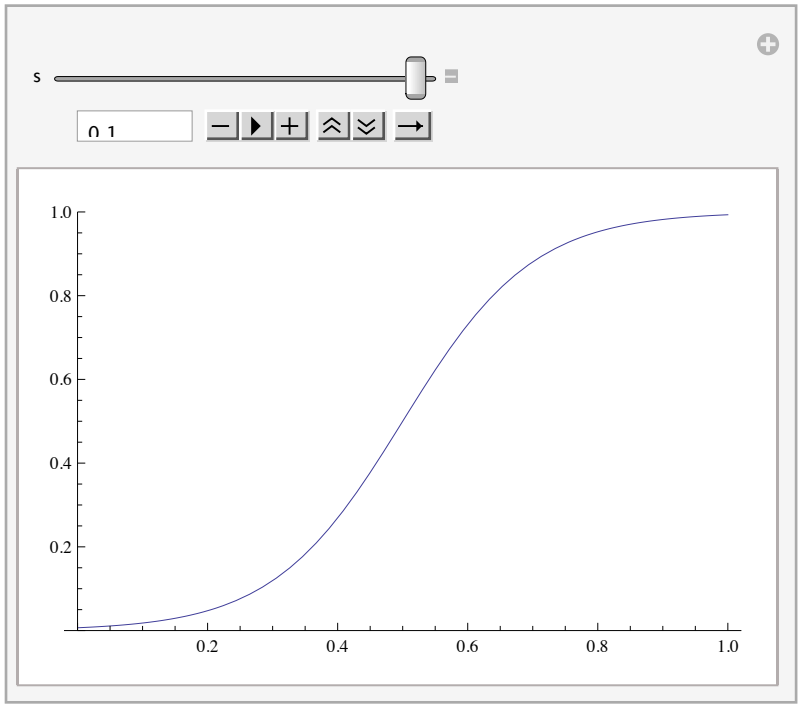
`ImageCompose` sovrappone la seconda alla prima immagine combinando i valori sui canali delle due immagini (opera sul canale *alfa* con un'eventuale fattore di trasparenza, nell'esempio del 50%):

```
ImageCompose[ <image> , { <image> , .5 } ]
```

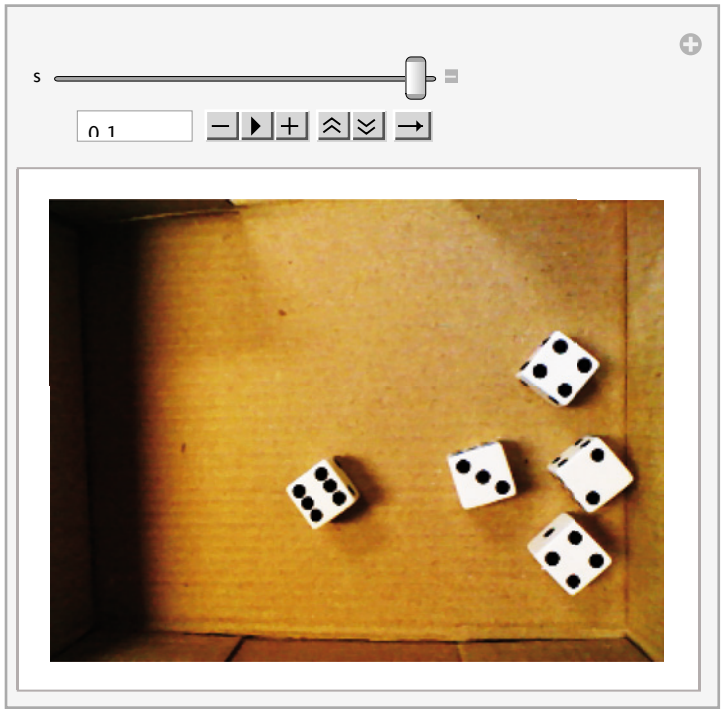
`ImageApply` consente di applicare qualsiasi funzione a ciascun pixel:

$$f[x\_, \sigma\_] := \frac{1}{1 + e^{-\frac{x-0.5}{\sigma}}};$$

```
Manipulate[Plot[f[x, s], {x, 0, 1}, PlotRange → {0, 1}], {{s, 0.1}, .01, 0.1}]
```



```
Manipulate[ImageApply[f[#, s] &, i], {{s, 0.1}, .01, 0.1}]
```




Se l'immagine ha i canali nella forma interallacciata, la funzione **Max** applicata con **ImageApply** restituisce il massimo valore dei canali per ciascun pixel; come risultato si ha l'immagine con un solo canale:

```
ImageApply[Max, i]
```

Se la funzione è planare il massimo si applica a ciascun singolo canale, restituendo di fatto il suo valore stesso, dunque l'immagine di partenza:

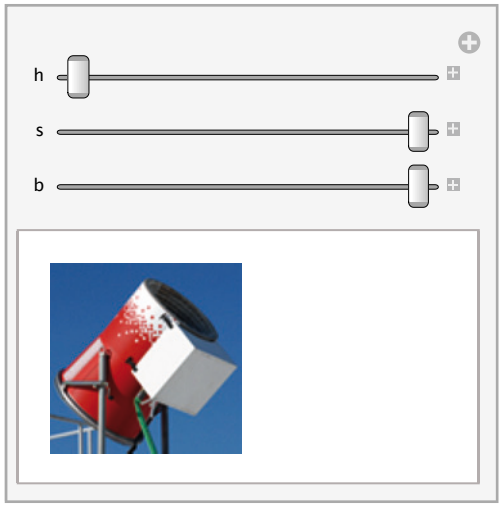
```
ImageApply[Max, i, Interleaving → False]
```

# Operazioni sui singoli punti o sui canali

## » Tonalità, saturazione e luminosità

In questo esempio si converte un'immagine nello spazio colore `"HSB"` moltiplicando i valori di ciascun canale per uno scalare e poi combinandoli insieme:

```
With[{hsb = ColorConvert[ , "HSB"]},
 Manipulate[ImageApply[{Mod[#[[1]] + h, 1], s #[[2]], b #[[3]]} &, hsb],
  {{h, 0}, 0, 1}, {{s, 1}, 0, 1}, {{b, 1}, 0, 1}]]
```

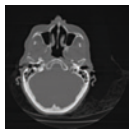


# Esempi di applicazioni dell'image processing

## » Trovare le dimensioni di una parte dell'immagine

In questo esempio, a partire dall'immagine

```
i =
```



```
;
```

bisogna calcolare la dimensione della regione del cervello.

Si utilizza prima RegionBinarize, che evidenzia regioni connesse che sono all'interno di una fissata distanza di colore (in questo caso .1).

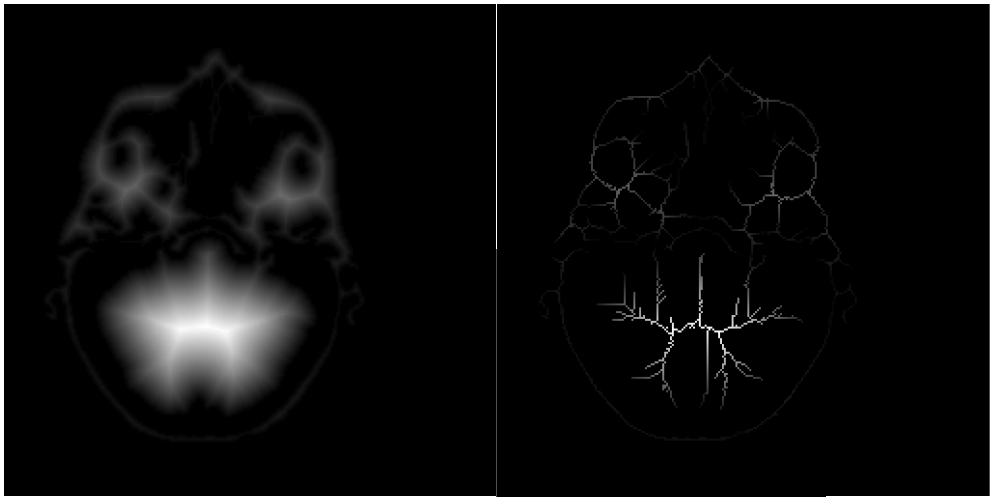
```
rb = Image[RegionBinarize[i, {{92, 91}}, .1], ImageSize → All]
```

```
? RegionBinarize
```

RegionBinarize[*image*, *marker*, *d*] gives a binary version of *image* that includes the foreground pixels of *marker* and also connected regions whose pixel values are within a distance *d*.
RegionBinarize[*image*, *marker*, *d*, {$t_1$, $t_2$}] grows regions in *marker* by adding pixels whose average intensity is also constrained within an interval {$t_1$, $t_2$}. ≫

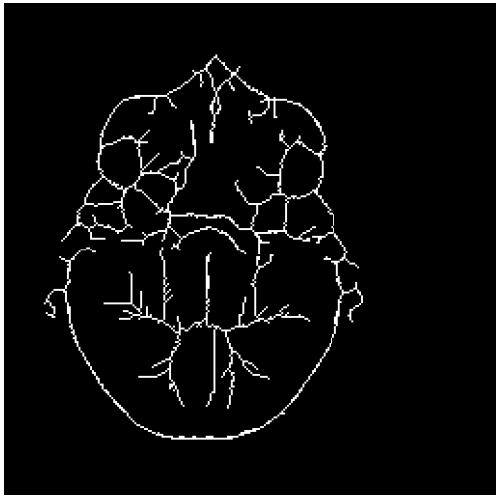
L'esempio di sopra non risulta molto efficace, perchè nella parte di destra non ha separato correttamente il cervello dal resto dell'immagine. Pertanto si può provare con altre funzioni, quali DistanceTransform (che calcola la distanza tra ciascun pixel ed il colore dello sfondo) e SkeletonTransform che restituisce una traformazione "skeleton" dell'immagine:

```
Row[ImageAdjust /@ {DistanceTransform[rb], skeleton = SkeletonTransform[rb]}]
```



A questo punto, usando un'altra funzione del menu contestuale delle imamgini chiamata GetIndices, possiamo determinare che il pixel di coordinate {132, 132} può essere quello giusto per individuare la regione del cervello sulla parte anteriore destra, staccando così l'area del cervello dal resto dell'immagine:

```
dat = ImageData[skeleton];
dat[[132, 132]] = 0;
cut = Image[dat]
```



```
(segments = MorphologicalComponents[cut]) // Colorize
```

```
? MorphologicalComponents
```

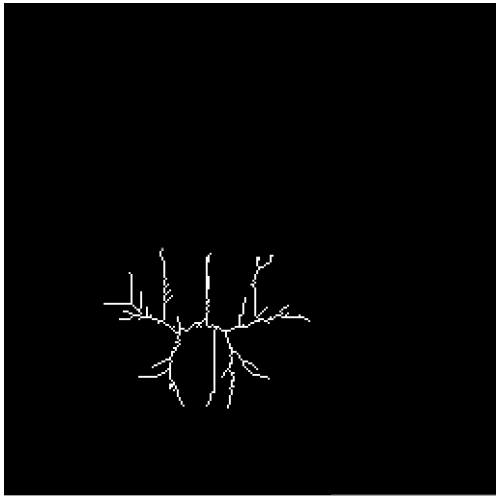
MorphologicalComponents[*image*] gives an array in which each pixel of *image* is replaced by an integer index representing the connected foreground image component in which the pixel lies.
MorphologicalComponents[*image*, *t*] treats values above *t* as foreground. ≫

```
? Colorize
```

Colorize[*m*] generates an image from an integer matrix *m*, using colors for positive integers and black for non−positive integers.
Colorize[*image*] replaces intensity values in *image* with pseudocolor values. ≫

Moltiplicando il segmento così estratto con lo skeleton prima memorizzato, otteniamo una ricostruzione della sola area di pertinenza del cervello:

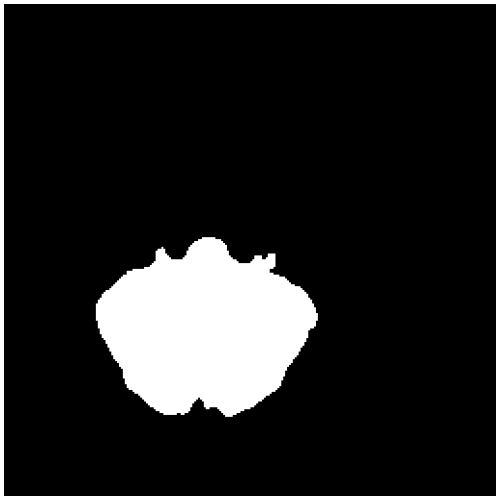
```
brainskeleton =
 ImageMultiply[skeleton, Image[SelectComponents[segments, "Label", # == 2 &], "Bit"]]
```



```
? ImageMultiply
```

ImageMultiply[*image*, *x*] multiplies each channel value in *image* by a factor *x*.
ImageMultiply[$image_1$, $image_2$] gives an image in
which each pixel is the product of the corresponding pixels in $image_1$ and $image_2$. ≫

```
brainregion = InverseDistanceTransform[brainskeleton]
```



```
? InverseDistanceTransform
```

InverseDistanceTransform[*image*] gives the inverse distance transform of *image*, returning the result as a binary image. ≫

Questa è la sua misura:

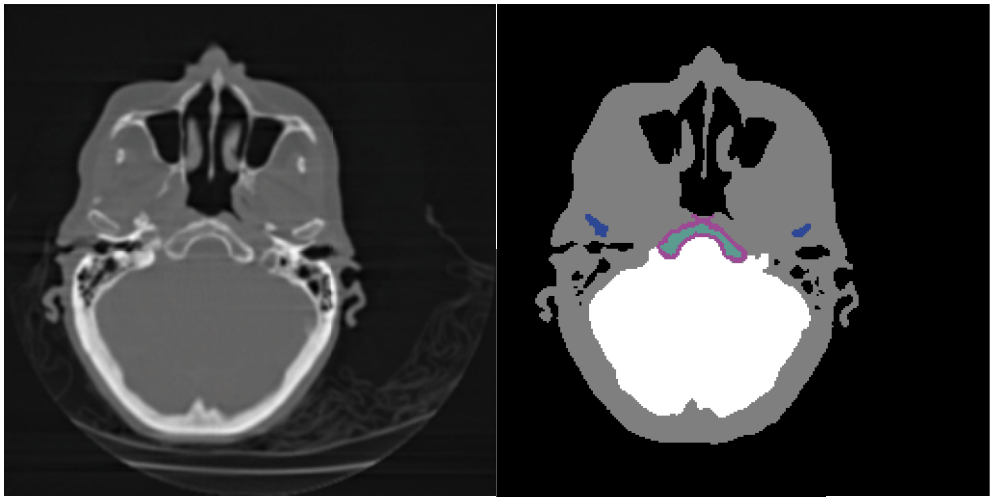
```
ComponentMeasurements[brainregion, "Count"]
```

```
{1 → 7412}
```

```
? ComponentMeasurements
```

ComponentMeasurements[*m*, "*prop*"] computes the values of
property *prop* for each component of a label matrix *m* that consists of identical elements.
ComponentMeasurements[*image*, "*prop*"] uses the connectivity of nonzero pixels in *image* to compute the label matrix.
ComponentMeasurements[{*m*, *image*}, "*prop*"] uses pixel values of *image* to compute property *prop*.
ComponentMeasurements[…, "*prop*", *crit*] only returns measurements that satisfy a criterion *crit*. ≫

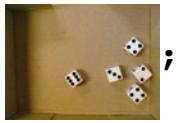
```
Row[Image[#, ImageSize → All] & /@ {i,  }]
```



# Esempi di applicazioni dell'image processing

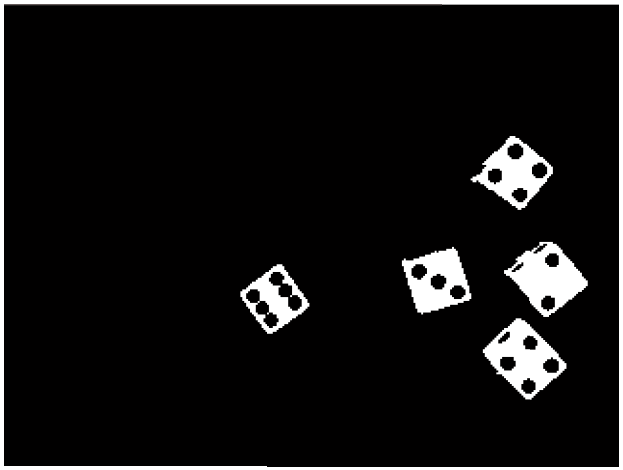
## » Riconoscere il punteggio ottenuto dai dadi

Questo esempio mostra un'altra serie di funzioni di image processing, applicate al caso di riconoscimento automatico del punteggio ottenuto da una serie di dadi ricavato dalla manipolazione della fotografia dei dadi stessi.

```
i =
```

```
;
```

Il primo passo effettua ancora una segmentazione per ricavare i riquadri relativi ai singoli dadi:

```
b = DeleteBorderComponents[MorphologicalBinarize[i, {.65, .8}]]
```



```
? MorphologicalBinarize
```

MorphologicalBinarize[*image*, {$t_1$, $t_2$}] creates a binary image from *image* by replacing all values above the upper threshold $t_2$ with 1, also including pixels with intensities above the lower threshold $t_1$ that are connected to the foreground.
MorphologicalBinarize[*image*, $t$] uses $t$ as the upper threshold, automatically choosing a suitable value for the lower threshold.
MorphologicalBinarize[*image*] chooses the lower and the upper threshold automatically. ≫

```
? DeleteBorderComponents
```

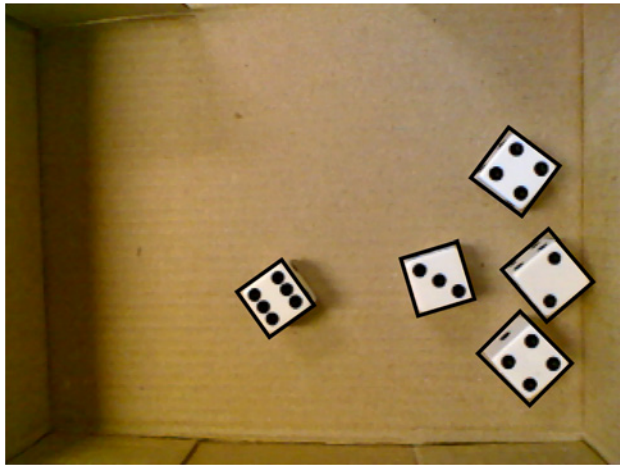
DeleteBorderComponents[*image*] replaces connected components adjacent to the border in a binary image *image* with background pixels.
DeleteBorderComponents[$m$] replaces components adjacent to the border in a label matrix $m$ with 0. ≫

ComponentMeasurements permette di trovare molte proprietà delle componenti, tra cui una molto utile in questo caso che si chiama "MinimalBoundingBox" che ci aiuta a trovare i riquadri di maggiore interesse nell'immagine:

```
bounds = #[[2]] & /@ ComponentMeasurements[b, "MinimalBoundingBox"]
```

```
{{{262.1, 175.4}, {242.4, 149.2}, {269., 129.3}, {288.6, 155.5}},
 {{306.1, 95.38}, {282.8, 122.6}, {258.5, 101.7}, {281.8, 74.55}},
 {{206., 107.4}, {214.9, 77.67}, {244.3, 86.48}, {235.3, 116.2}},
 {{137.6, 66.57}, {160.9, 83.19}, {144., 106.8}, {120.7, 90.22}},
 {{268., 80.}, {246., 58.}, {273., 31.}, {295., 53.}}}
```

```
Show[i, Graphics[{EdgeForm[Thick], Transparent, Polygon@# & /@ bounds}]]
```



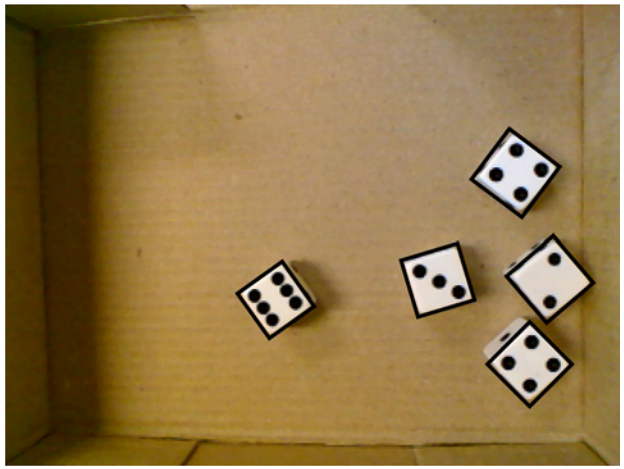
I lati dei dadi non sono ancora ripuliti da valori di “disturbo” che mostrano i punteggi delle altre facce del dado. Applicando ulteriori filtri si rimuove questo problema:

```
squareBB[box_List, center_List] := Module[{ratio, p1, p2, p3, p4},
  ratio = Divide @@
    Sort[{EuclideanDistance[box[[1]], box[[2]]], EuclideanDistance[box[[2]], box[[3]]]}];
  p1 = SortBy[box, EuclideanDistance[#, center] &][[4]];
  {p1, p2, p3, p4} = SortBy[box, EuclideanDistance[#, p1] &];
  p3 = ratio p3 + (1 - ratio) p1;
  p4 = ratio p4 + (1 - ratio) p2;
  {p1, p2, p4, p3}
 ]
```

```
squarebounds = squareBB[#, ImageDimensions[i] / 2] & /@ bounds
```

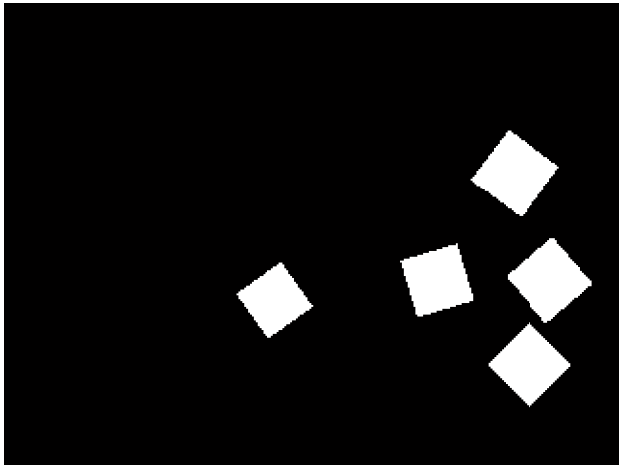
```
{{{288.6, 155.5}, {269., 129.3}, {242.7, 149.}, {262.4, 175.2}},
 {{306.1, 95.38}, {281.8, 74.55}, {261., 98.85}, {285.3, 119.7}},
 {{244.3, 86.48}, {214.9, 77.67}, {206.1, 107.}, {235.4, 115.8}},
 {{137.6, 66.57}, {160.9, 83.19}, {144.2, 106.5}, {121., 89.84}},
 {{295., 53.}, {273., 31.}, {251., 53.}, {273., 75.}}}
```

```
Show[i, Graphics[{EdgeForm[Thick], Transparent, Polygon@# & /@ squarebounds}]]
```

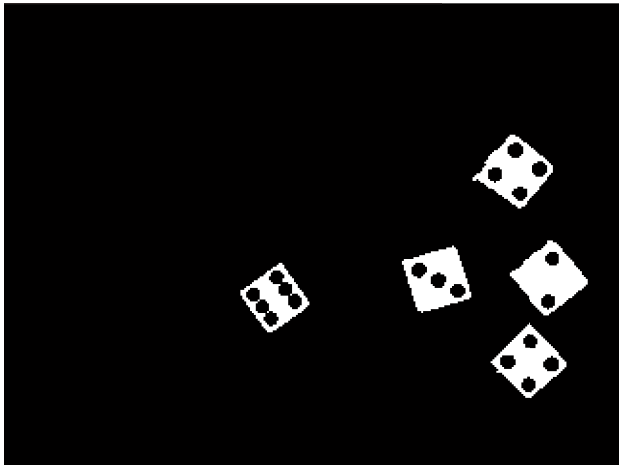


Una volta “ripulita” l’immagine con alcune altre trasformazioni si possono ricavare i soli punti corrispondenti ai punteggi dei dadi:

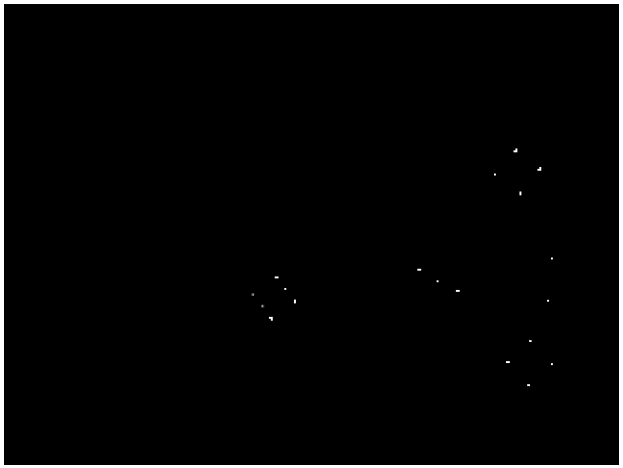
```
boxmask = Binarize[ColorNegate@Graphics[{Polygon@# & /@ squarebounds},
    PlotRange → Transpose[{{1, 1}, ImageDimensions[i]}], ImageSize → ImageDimensions[i]]]
```



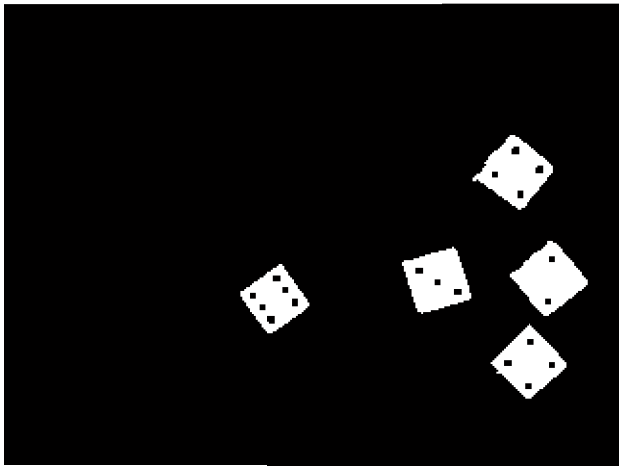
```
cleaned = ImageMultiply[FillingTransform[boxmask], b]
```



```
dots = DeleteBorderComponents[ColorNegate[cleaned]] // DistanceTransform // MaxDetect
```



```
dotteddice = ImageSubtract[FillingTransform[cleaned], Dilation[dots, 1]]
```



L'opzione **Dilation** (dilatazione) rimuove le caratteristiche scure più piccole.

```
ComponentMeasurements[dotteddice, "Holes"][[All, 2]]
```

```
{4, 2, 3, 6, 4}
```

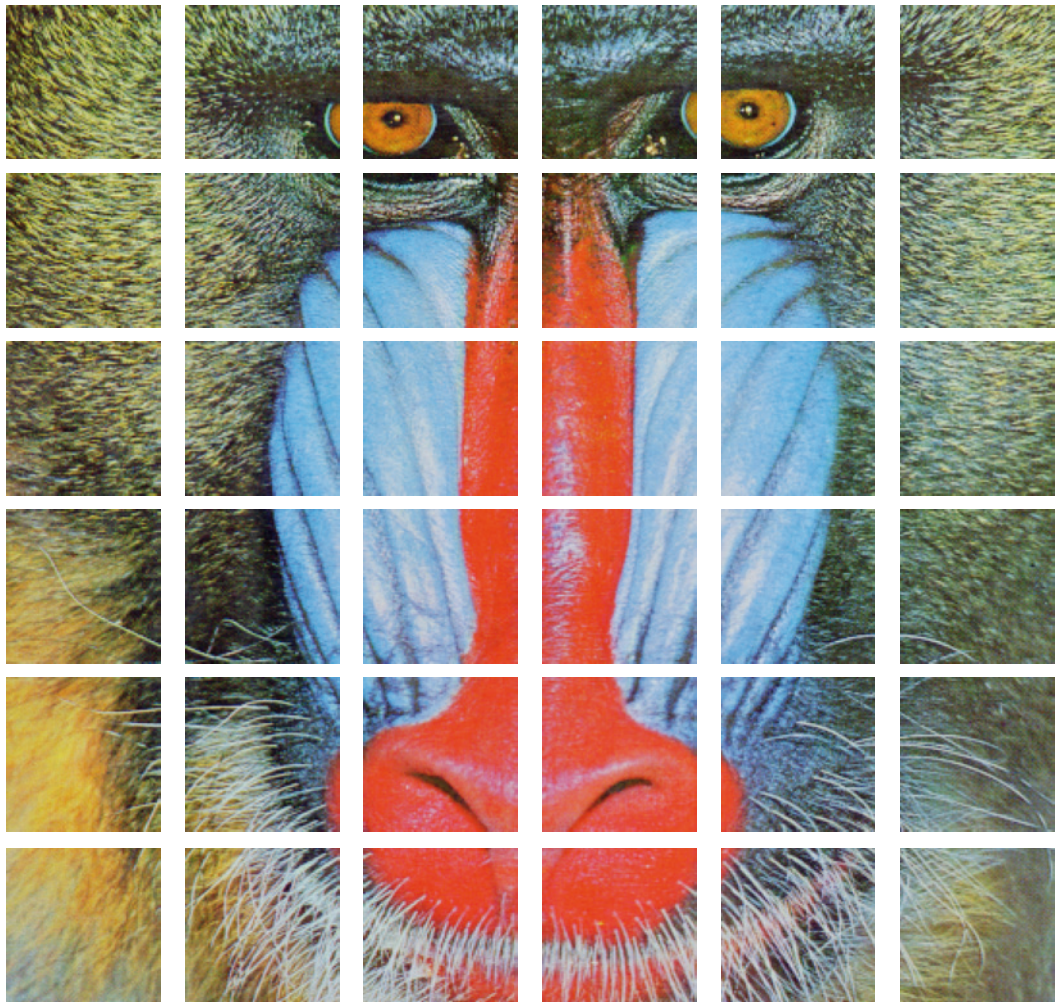
Infine, conteggiando i buchi neri (“Holes”) per ciascuna componente dell’immagine, otteniamo i valori delle facce dei dadi ;-)

# Esempi di applicazioni dell'image processing

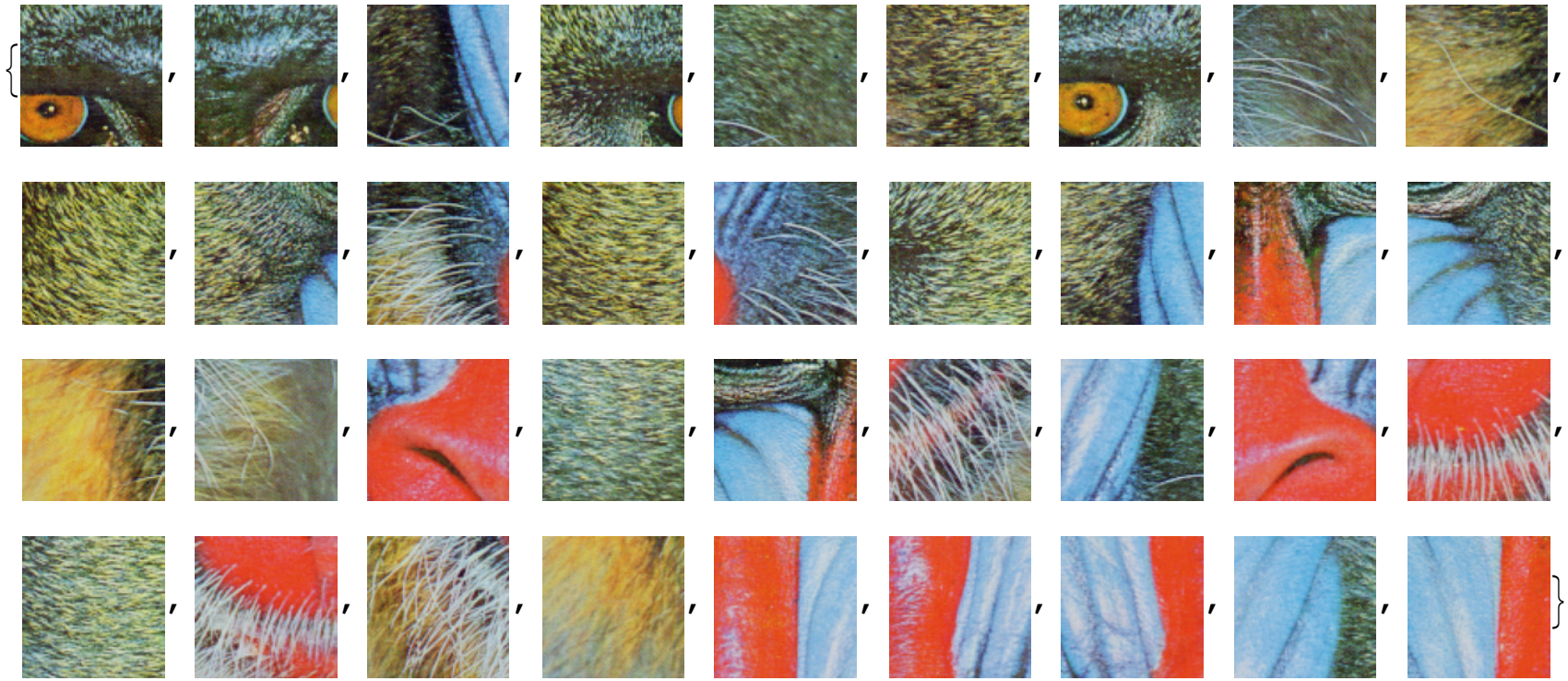
» Clustering di tasselli di immagine

```
im =  ;
```

```
ImagePartition[im, 80] // Grid
```



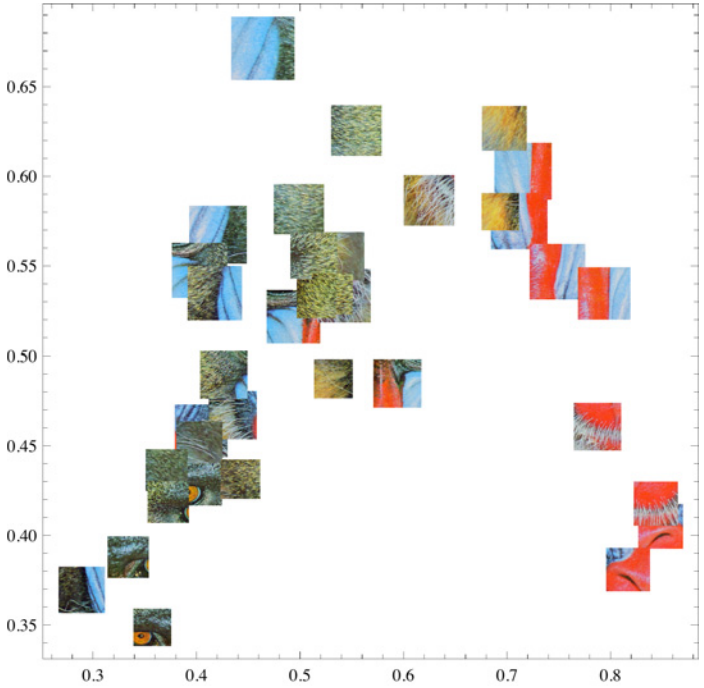
```
im2 = SortBy[Flatten[ImagePartition[im, 80]], Total[ImageData[#], 3] &]
```



```
colorMeans = Map[Mean[Mean[ImageData[#]]] &, im2]
```

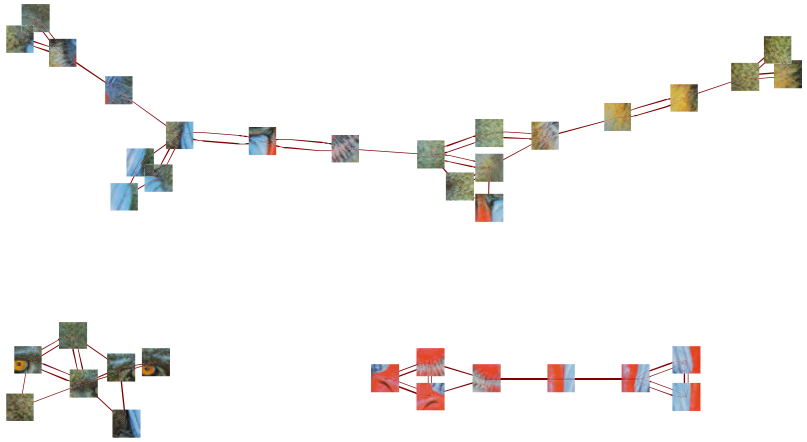
```
{{0.3573, 0.3488, 0.2721}, {0.3341, 0.3879, 0.3298}, {0.2891, 0.3697, 0.415},
 {0.3726, 0.4187, 0.332}, {0.3711, 0.4365, 0.3374}, {0.4424, 0.4314, 0.2993},
 {0.4043, 0.4285, 0.3422}, {0.4027, 0.4509, 0.4092}, {0.5325, 0.4874, 0.2896},
 {0.5159, 0.5318, 0.2944}, {0.4263, 0.4895, 0.4293}, {0.435, 0.4671, 0.4478},
 {0.5312, 0.5344, 0.3241}, {0.4046, 0.4584, 0.5364}, {0.5132, 0.5558, 0.4192},
 {0.4178, 0.5349, 0.5675}, {0.5941, 0.4848, 0.4473}, {0.4029, 0.5475, 0.5885},
 {0.6936, 0.5803, 0.2707}, {0.5388, 0.5555, 0.4544}, {0.817, 0.381, 0.3678},
 {0.4989, 0.5817, 0.4856}, {0.4937, 0.5218, 0.5512}, {0.5425, 0.5334, 0.5471},
 {0.4208, 0.5675, 0.643}, {0.8484, 0.4051, 0.3849}, {0.8437, 0.4177, 0.3775},
 {0.5545, 0.6255, 0.4919}, {0.7875, 0.4606, 0.4369}, {0.6246, 0.5866, 0.4928},
 {0.6974, 0.6268, 0.3838}, {0.794, 0.5347, 0.5313}, {0.7489, 0.5468, 0.594},
 {0.7118, 0.5751, 0.6164}, {0.464, 0.6715, 0.7737}, {0.7155, 0.6029, 0.6294}}
```

```
BubbleChart[colorMeans, ChartElements -> im2]
```

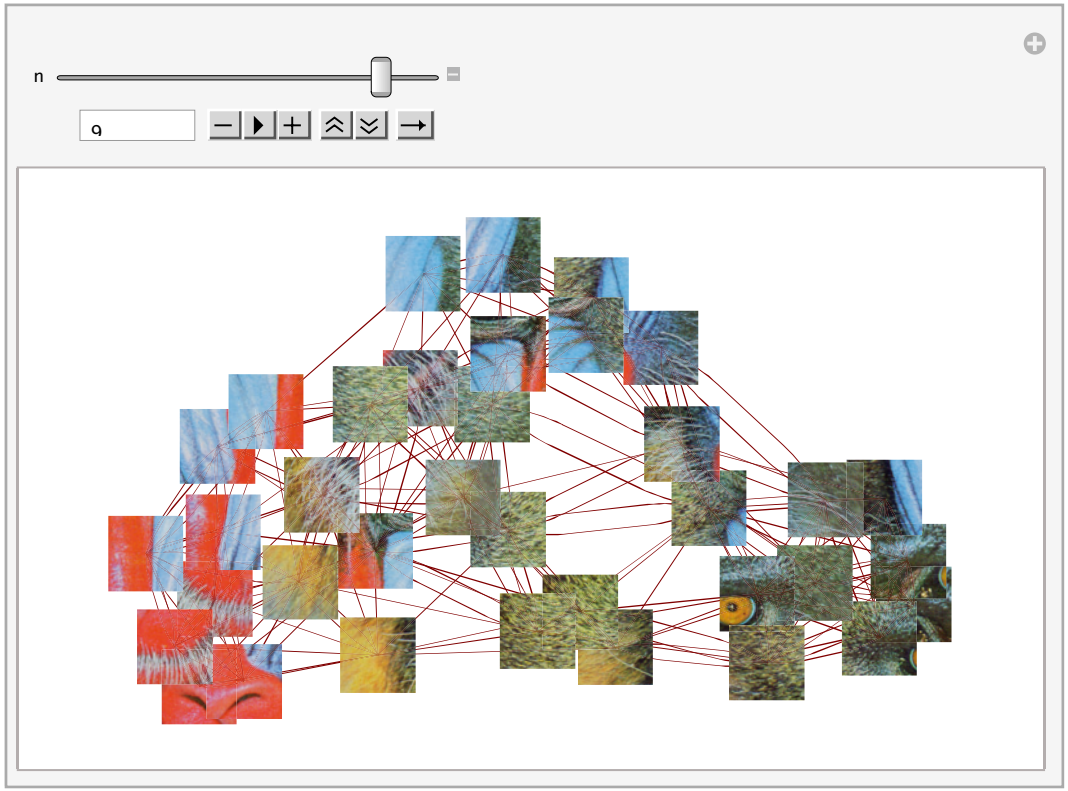



```
nf = Nearest[colorMeans → im2];
```

```
GraphPlot[Flatten[Table[Thread[im2[[i]] → nf[colorMeans[[i]], 3]], {i, Length[im2]}]],
 VertexRenderingFunction → (Inset[#2, #, Center, .5] &),
 SelfLoopStyle → None, ImageSize → 500]
```



```
Manipulate[
 GraphPlot[Flatten[Table[Thread[im2[[i]] → nf[colorMeans[[i]], n]], {i, Length[im2]}]],
  VertexRenderingFunction → (Inset[#2, #, Center, .5] &),
  SelfLoopStyle → None, ImageSize → 500], {n, 1, 10, 1}]
```



# Conclusioni

*Mathematica* offre non solo funzionalità *general purpose* ma anche verticalizzazione in specifici settori quali questo appena visto della elaborazione delle immagini. Il valore aggiunto di strumenti come *Mathematica* in tali casi diventa proprio il fatto di avere a disposizione non solo i migliori algoritmi e le più avanzate funzionalità di uno specifico settore ma anche centinaia di funzioni aggiuntive. Ad esempio l'interattività dei controller o la enorme flessibilità dei documenti consente di realizzare applicazioni anche complesse in qualsiasi campo.

Infine, in particolare per quanto riguarda l'elaborazione delle immagini va sottolineato che sono disponibili molte funzioni avanzate di image processing implementate in linguaggio Cuda direttamente come funzioni native di *Mathematica*.