



ELABORAZIONE DELLE IMMAGINI

OPENCV

INTRODUZIONE

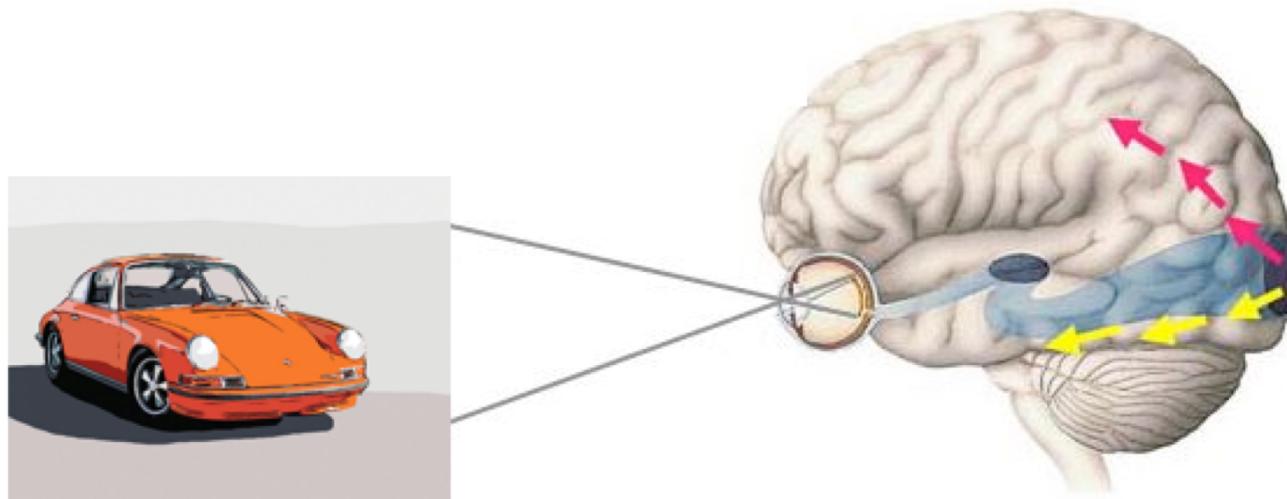
- OpenCV è una libreria open source per la computer vision e l'elaborazione delle immagini
 - Disponibile all'indirizzo <http://opencv.org>
- Il progetto nasce da Gary Bradsky nel 1999 presso la Intel Corporation
- La libreria è scritta in C e C++
- Può essere compilata ed eseguita sotto Linux, Windows e MacOS
- Esistono porting per Android e iOS
- La libreria mette a disposizione interfacce Python, Java e Matlab (ed altri)
- Poichè c'è una forte sinergia tra computer vision e machine learning, OpenCV dispone di un modulo di ML

Elim Parte II – Prof. A. Ferone

INTRODUZIONE

- La computer vision consiste nella **trasformazione** di immagini o video in una **decisione** o in una **nuova rappresentazione**
 - Es. di decisione: c'è una persona o una macchina nella scena
 - Es. di nuova rappresentazione: rimuovere imperfezioni dall'immagine
- Per un essere umano la **vista** è uno dei sensi più sviluppati e ci potrebbe far credere che questi **task** della computer vision siano **semplici**
- Attraverso un complesso sistema composto dagli occhi e dal cervello il compito risulta effettivamente molto semplice...

INTRODUZIONE



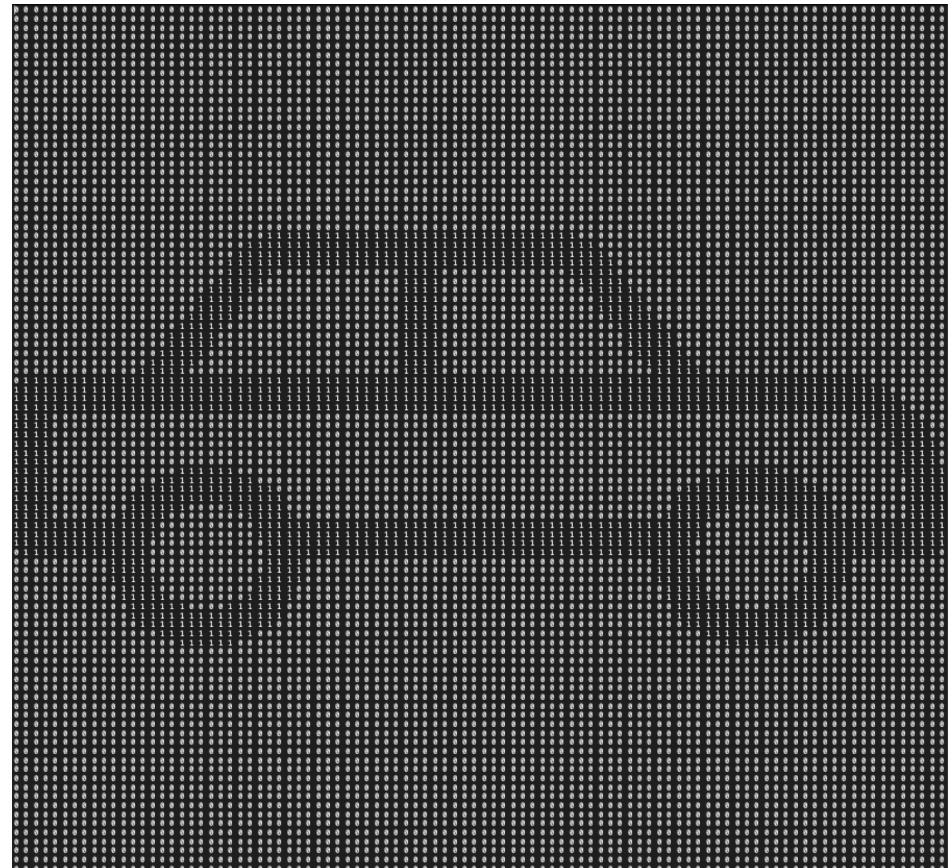
Elim Parte II – Prof. A. Ferone

INTRODUZIONE

- Ottenerlo stesso risultato in un sistema artificiale (computer) non è così semplice
- Cosa "vede" un computer?

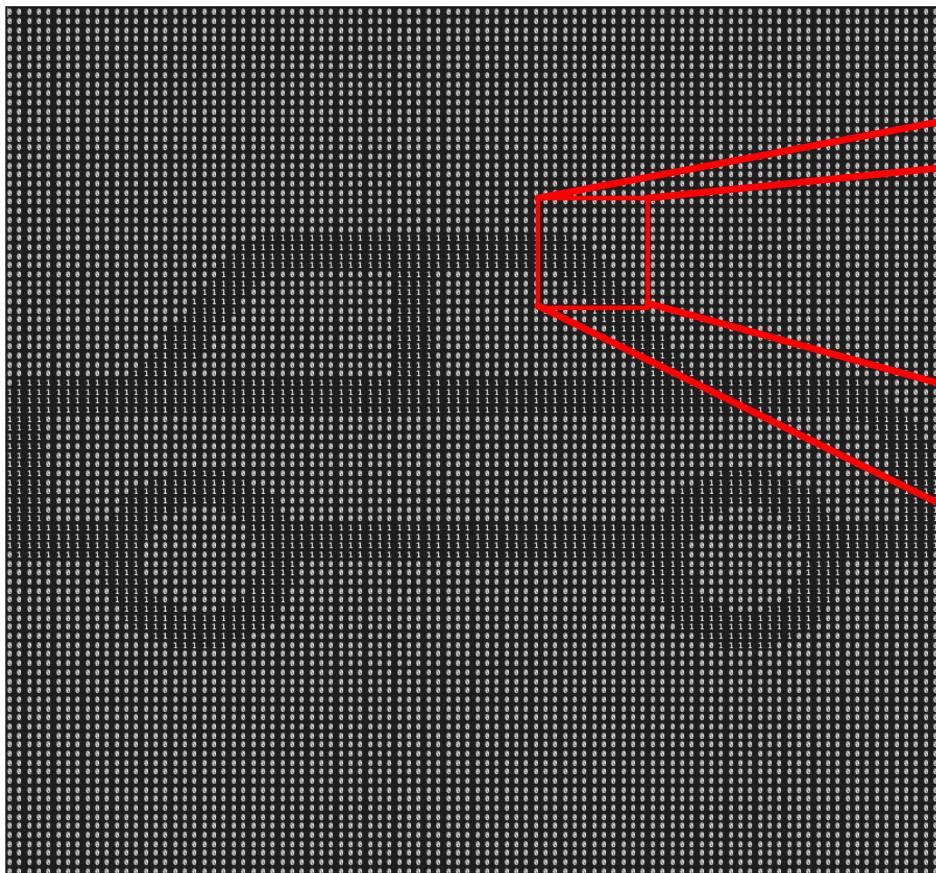


INTRODUZIONE



Elim Parte II – Prof. A. Ferone

INTRODUZIONE



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	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	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	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Per poter distinguere i punti bianchi e neri (0-1) e quindi distinguere i bordi ↓

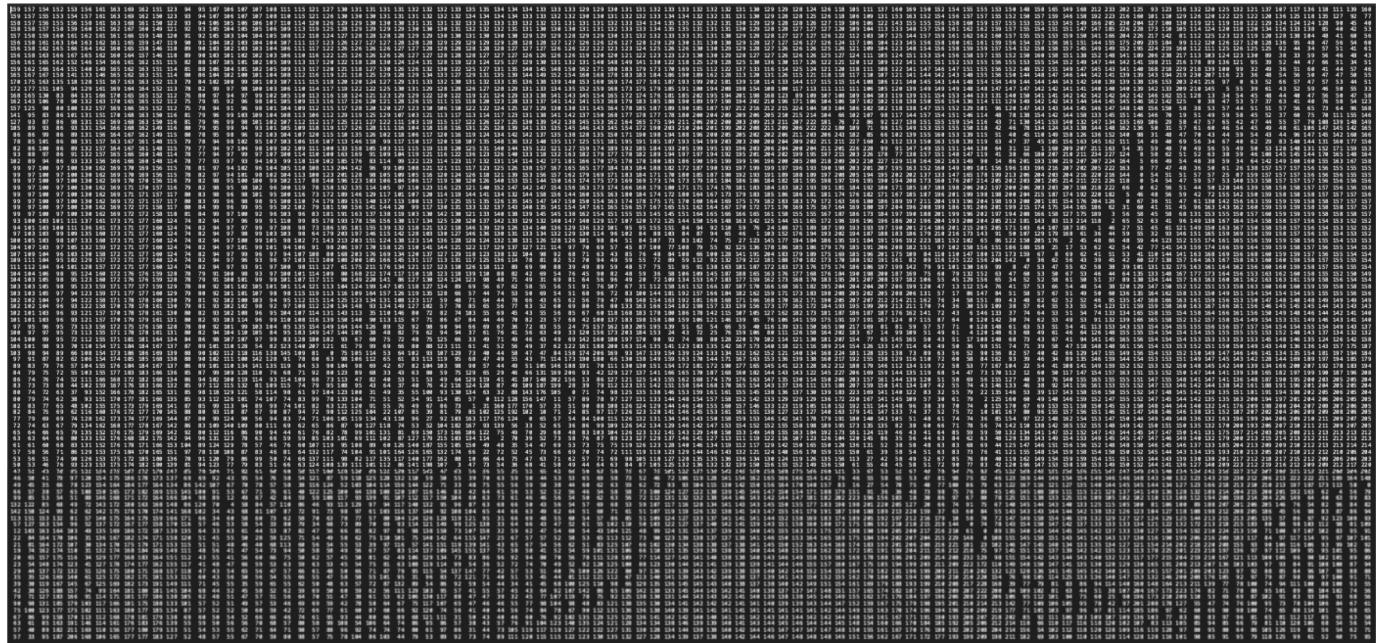
Prendo il punto e vedo se i suoi vicini nel suo intorno sono uguali o diversi

→ è questo l'interno che sto considerando - dettagliata è l'informazione

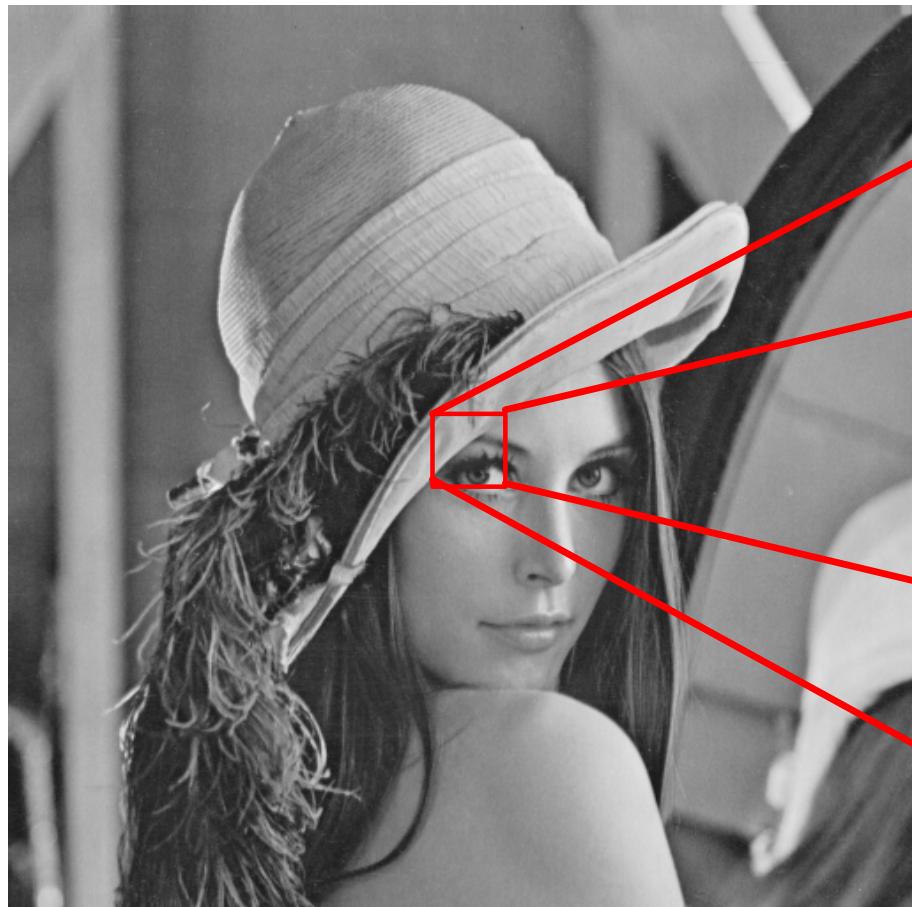
Ci interessa, oltreché la variazione dell'intensità, ma anche in che direzione si svilupperà la variazione.

Elim Parte II – Prof. A. Ferone

INTRODUZIONE



INTRODUZIONE



sezze di grigio

117	107	120	172	125	144	150	156	148	163	142	125	144
101	95	95	155	151	84	94	92	110	59	76	24	108
96	102	84	88	103	84	87	94	65	46	1	103	152
84	51	84	107	73	83	102	74	75	27	123	145	177
57	45	87	100	77	108	79	55	92	125	136	178	186
43	85	76	48	84	108	66	100	128	141	191	204	183
49	59	55	72	89	59	112	92	132	179	212	181	188
59	48	57	75	51	95	91	138	163	181	187	172	183
82	48	59	36	58	106	109	145	187	188	165	163	176
96	49	20	72	109	134	153	167	183	177	159	162	173
107	67	31	112	158	161	152	166	187	187	170	166	168
72	83	100	144	181	192	157	166	184	182	162	159	171
47	98	148	154	163	192	181	168	168	171	166	168	170
80	133	163	166	154	182	198	157	133	139	156	173	177
118	168	183	178	180	186	178	142	115	107	105	127	162
137	183	198	158	180	159	106	121	146	139	95	76	106
162	183	205	195	139	51	62	54	66	55	40	116	139
173	203	184	153	24	43	72	25	76	159	106	80	111
180	197	170	90	92	74	98	80	128	220	170	102	114
200	163	114	135	141	123	103	135	142	167	173	140	118
169	93	147	139	154	139	124	133	134	132	162	152	121
66	130	138	149	159	163	170	144	171	167	163	153	136
111	130	130	151	154	172	181	172	190	177	165	145	141
129	127	137	141	166	165	170	187	168	160	182	144	143

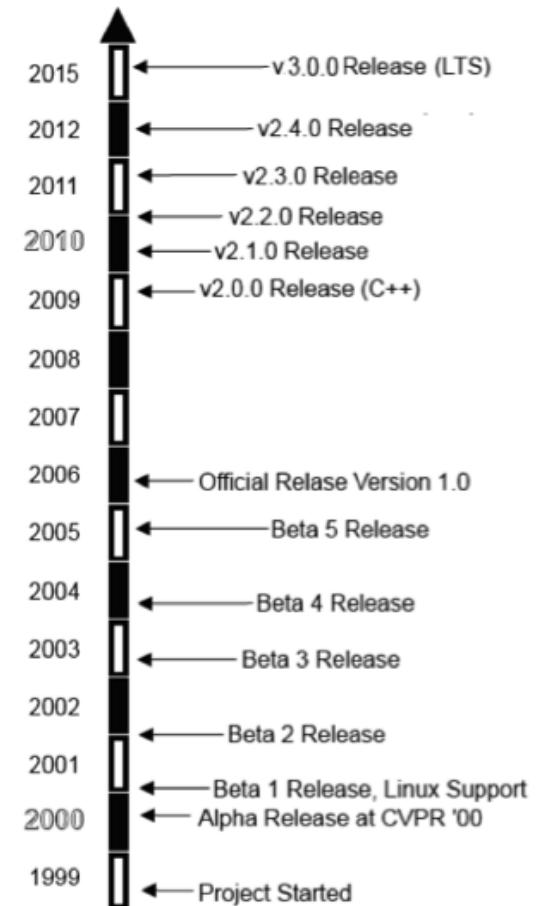
Elim Parte II – Prof. A. Ferone

OPENCV

- Cosa "vede" un computer?
- Il computer "vede" una matrice di numeri
- Obiettivo della computer vision è trasformare questa matrice in "percezione"
- OpenCV fornisce gli strumenti di base per risolvere i problemi nell'ambito della computer vision

OPENCV: TIMELINE

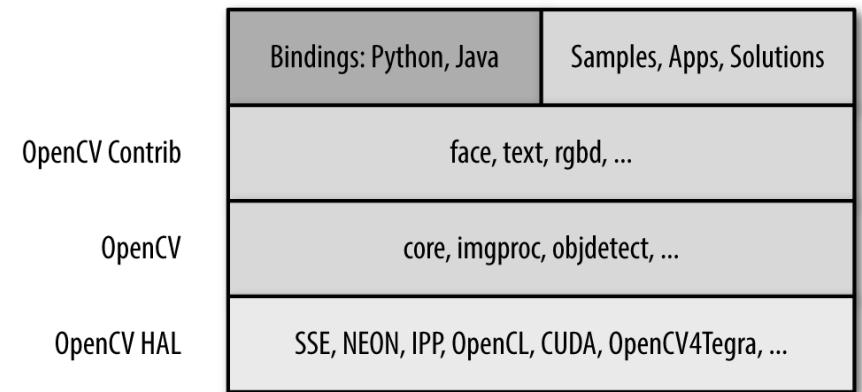
- L'ultima versione è la 4.x.x
- Per questo corso utilizzeremo la 3.4.9



Elim Parte II – Prof. A. Ferone

OPENCV: STRUTTURA

- Opencv ha una **struttura a livelli**
- Al di sotto del SO troviamo le interfacce dei linguaggio supportati e le applicazioni
- Nel livello successivo ci sono le funzionalità di alto livello
- Seguite dalle funzioni di basso livello
- Il livello più basso contiene le ottimizzazioni hardware (Hardware Acceleration Layer)



OPENCV: HEADER

- Per accedere alle funzioni messe a disposizione da OpenCV è necessario includere gli header opportuni
- `#include "opencv2/opencv.hpp"` *← usiamo sempre questi*
 - Consente di accedere a tutte le funzioni della libreria
- `#include "opencv2/imgproc/imgproc.hpp"`
 - Consente di accedere alle funzioni specifiche per l'elaborazione delle immagini

OPENCV: PRIMO PROGRAMMA

- Il primo passo è l'apertura di un'immagine
- OpenCV consente di leggere un'ampia varietà di tipi di immagini, video e videocamere
- Le relative funzioni di utilità sono contenute nel toolkit HighGUI (già incluso nel pacchetto OpenCV)

OPENCV: PRIMO PROGRAMMA

```
#include <opencv2/opencv.hpp> //Include file for every supported OpenCV function

int main( int argc, char** argv ) {
    cv::Mat img = cv::imread(argv[1], -1); //Carica l'immagine da file
    if( img.empty() ) return -1;
    cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
    cv::imshow( "Example1", img ); //Visualizza l'immagine
    cv::waitKey( 0 ); //Attendere la premiazione di un tasto
    cv::destroyWindow( "Example1" ); //Destruzione della finestra
    return 0;
}
```

il 'mat' è l'immagine in OpenCV
finestra grafica con nome e visualizzabile l'immagine
mostre l'immagine img nella finestra

come caricare l'immagine
-1 è carica come si è inserito all'origine
legge l'immagine argv[1] e il path

il nome delle finestre
è specificato nell'immagine

aspetta che premiamo un tasto sull'immagine

OPENCV: PRIMO PROGRAMMA

- Le funzioni OpenCV sono definite nel **namespace cv**
 - Vanno richiamate utilizzando il prefisso **cv::**
 - oppure utilizzare **using namespace cv;**
- cv::Mat img = cv::imread(argv[1], -1);
- Questa linea di codice serve caricare l'immagine il cui nome è passato come prima argomento della linea di comando
- La funzione **imread()**
 - determina automaticamente il tipo di file (BMP, JPEG, PNG, PPM, ...)
 - alloca la memoria necessaria per la struttura dati (**cv::Mat**) che conterrà i dati dell'immagine

OPENCV: PRIMO PROGRAMMA

- Successivamente viene richiamato il metodo **empty()** per verificare se l'immagine è stata letta correttamente

```
if( img.empty() ) return -1;  
cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
```

- Con la funzione **namedWindow()** viene creata una finestra che verrà utilizzata per visualizzare l'immagine (Example1 è il nome della finestra)

```
cv::imshow( "Example1", img );
```

- Infine viene mostrata l'immagine
- Le ultime due funzioni servono per attendere la pressione di un tasto e chiudere la finestra (**waitKey()**) e **destroyWindow()**)
- L'oggetto cv::Mat viene deallocated automaticamente quando esce dal contesto

OPENCV: INSTALLAZIONE

- Guide sul sito elearning





ELABORAZIONE DELLE IMMAGINI

OPENCV

OPENCV: IMREAD

- La funzione **imread()**
 - determina automaticamente il tipo di file (BMP, JPEG, PNG, PPM, ...)
 - alloca la memoria necessaria per la struttura dati (**cv::Mat**) che conterrà i dati dell'immagine

```
cv::Mat cv::imread(  
    const string& filename,           // Input filename  
    int          flags   = cv::IMREAD_COLOR // Flags set how to interpret file  
)
```

Parameter ID	Meaning	Default
cv::IMREAD_COLOR	Always load to three-channel array.	yes
cv::IMREAD_GRAYSCALE	Always load to single-channel array.	no
cv::IMREAD_ANYCOLOR	Channels as indicated by file (up to three).	no
cv::IMREAD_ANYDEPTH	Allow loading of more than 8-bit depth.	no
cv::IMREAD_UNCHANGED	Equivalent to combining: cv::IMREAD_ANYCOLOR cv::IMREAD_ANYDEPTH ^a	no

Elim Parte II – Prof. A. Ferone

OPENCV: IMWRITE

- La funzione **imwrite()**
 - determina il tipo di file (BMP, JPEG, PNG, PPM, ...) dal nome
 - Salvare sul filesystem l'immagine passata come secondo argomento

```
bool cv::imwrite(  
    const string&      filename,           // Input filename  
    cv::InputArray     image,              // Image to write to file  
    const vector<int>& params = vector<int>() // (Optional) for parameterized fmts  
);
```

Parameter ID	Meaning	Range	Default
cv::IMWRITE_JPG_QUALITY	JPEG quality	0-100	95
cv::IMWRITE_PNG_COMPRESSION	PNG compression (higher values mean more compression)	0-9	3
cv::IMWRITE_PXM_BINARY	Use binary format for PPM, PGM, or PBM files	0 or 1	1

OPENCV: STRUTTURE DATI

- OpenCV utilizza molte **strutture dati** che possono essere divise in **tre categorie**
 1. **Basic data types**: sono creati a partire dalle primitive del C++ (int, float, ...) e contengono le definizioni di vettori e matrici di piccole dimensioni (fino a 6x6) e semplici oggetti geometrici (punti, rettangoli, ...)
 2. **Helper objects**: rappresentano concetti più astratti come **garbage collector, range objects**, ecc.
 3. **Large array types**: contengono array e altri contenitori di primitive di dimensione arbitraria. A questa categoria appartiene la classe **cv:Mat** che viene usata per rappresentare array di dimensione arbitraria. Le immagini sono istanze di questa classe
- OpenCV usa anche la **STL** ed in particolare la classe **vector**

Elim Parte II – Prof. A. Ferone

OPENCV: CV::VEC

- Il tipo più semplice è la classe template **cv::Vec<>** che rappresenta un container di primitive (noto come **fixed vector class**)
- È utilizzata per **array piccoli** la cui **dimensione** è **nota** al tempo di compilazione
- Solitamente viene usata mediante **alias** che ne definiscono tipo e dimensione
- In generale ogni combinazione della seguente forma è valida

cv::Vec{2,3,4,6}{b,w,s,i,f,d}

- Dove {2,3,4,6} rappresenta la dimensione e {b,w,s,i,f,d} il tipo
- es.: **cv::Vec2i**, **cv::Vec3f**, **cv::Vec4d**

OPENCV: CV::MATX

- Esiste anche la versione per le matrici (**fixed matrix class**) **cv::Matx<>**
- È utilizzata per matrici piccole la cui dimensione è nota al tempo di compilazione
- Solitamente viene usata mediante alias che ne definiscono tipo e dimensione
- In generale ogni combinazione della seguente forma è valida

cv::Matx{1,2,3,4,6}{1,2,3,4,6}{f,d}

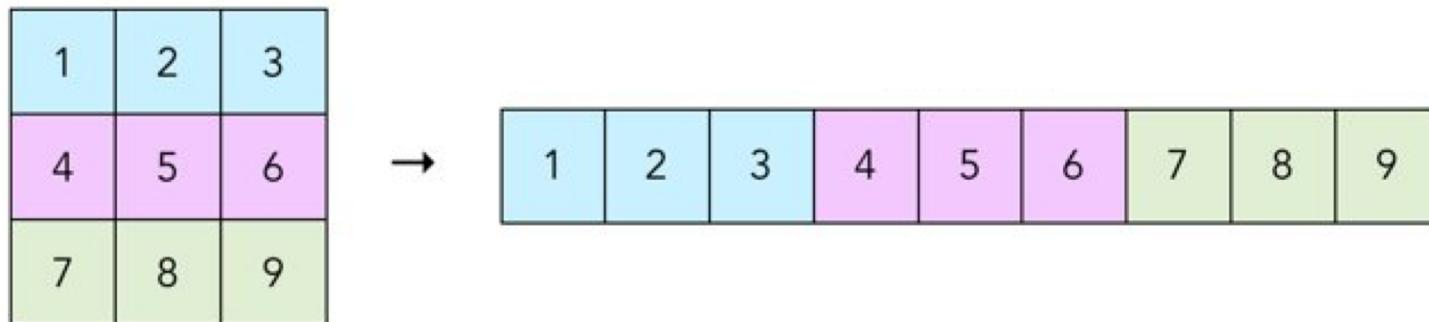
- Dove {1,2,3,4,6} rappresentano le dimensioni e {f,d} il tipo
- es.: **cv::Matx22f**, **cv::Matx33f**

OPENCV: CV::MAT

- La classe **cv::Mat** rappresenta la classe centrale dell'intera implementazione C++ della libreria OpenCV
- La maggior parte delle funzioni della libreria
 - sono membri di cv::Mat
 - ricevono cv::Mat come argomento
 - restituiscono cv::Mat
- Questa classe viene utilizzata per rappresentare **array densi di un qualunque numero di dimensioni**
 - L'aggettivo denso indica che per ogni posizione dell'array c'è un valore memorizzato nella corrispondente area di memoria
- In alternativa è possibile utilizzare **cv::SparseMat**

OPENCV: CV::MAT

- I dati sono memorizzati per riga, in maniera sequenziale



	Column 0	Column 1	Column ...	Column m		
Row 0	0,0	0,0	0,0	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1, m	1, m	1, m

0,0 0,0 0,0 0,1 0,1 ... 0, m 0, m 0, m 1,0 1,0 1,0 1,1 1,1 1,1 1, m 1, m 1, m

OPENCV: CV::MAT

- Ogni matrice ha diversi attributi
 - *flags* (campi di bit) specifica il contenuto
 - *dims* indica il numero di dimensioni
 - *rows* e *cols* indicano il numero di righe e colonne
 - *data* un puntatore all'area di memoria in cui sono memorizzati i dati

OPENCV: CV::MAT

- L'organizzazione della memoria in *data* è descritta dall'array *step[]*
- L'array *data* è organizzato in modo che l'indirizzo di un elemento di indici

$$(i_0, i_1, \dots, i_{N_d-1})$$

- è calcolato come segue

$$\begin{aligned} & \& \left(mtx_{i_0, i_1, \dots, i_{N_d-1}} \right) = mtx.data + mtx.step[0]*i_0 + mtx.step[1]*i_1 + \dots \\ & & + mtx.step[N_d-1]*i_{N_d-1} \end{aligned}$$

- che nel caso di un array bidimensionale è

$$\&(mtx_{i,j}) = mtx.data + mtx.step[0]*i + mtx.step[1]*j$$

CV::MAT ALLOCAZIONE

- È possibile creare un array semplicemente instanziando una variabile di tipo cv::Mat
 - In questo caso l'array non ha né tipo né dimensione
- Successivamente è possibile allocare memoria utilizzando il metodo **create()** passando come argomenti
 - il numero di righe e di colonne
 - un tipo
- Per quanto riguarda il **tipo** bisogna specificare il tipo fondamentale ed il numero di canali
- CV_32F3 corrisponde ad un array di float a 32 bit con tre canali

CV_{8U,16S,16U,32S,32F,64F}C{1,2,3}

8bit unsigned... Tipò n. di canali = profondità di colore
RGB

CV::MAT ALLOCAZIONE

- È possibile specificare gli argomenti anche nel costruttore

```
cv::Mat m( 3, 10, CV_32FC3, cv::Scalar( 1.0f, 0.0f, 1.0f ) );
```

↓ ↓
Righe Colonne

i uno SCALAR
in un vettore con 3
valori
↓
è usato per assegnare i valori
ai pixel con differenti
valori

Ogni pixel è un vettore di 3 posizioni

- Equivalenti a

```
cv::Mat m;
```

```
// Create data area for 3 rows and 10 columns of 3-channel 32-bit floats
m.create( 3, 10, CV_32FC3 );
↓
è lo stesso ma non sto initializzando la matrice
```

CV::MAT COSTRUTTORI

Constructor	Description
cv::Mat;	Default constructor
cv::Mat(int rows, int cols, int type);	Two-dimensional arrays by type
cv::Mat(int rows, int cols, int type, const Scalar& s);	Two-dimensional arrays by type with initialization value
cv::Mat(int rows, int cols, int type, void* data, size_t step=AUTO_STEP); <pre style="margin-left: 100px;">+ ↓ * • J per il cast a addizionare prendo i dati da un vettore esterno il puntatore data si muove fino</pre>	Two-dimensional arrays by type with preexisting data
cv::Mat(cv::Size sz, int type); <i>sostituisci rows e cols , è un oggetto di tipo Size</i>	Two-dimensional arrays by type (size in sz)
cv::Mat(cv::Size sz, int type, const Scalar& s);	Two-dimensional arrays by type with initialization value (size in sz)

- utilizzato data come puntatore agli elementi che dovrà andare a prendere da un vettore esterno (come una copia)
- se data è più grande, posso modificare AUTO_STEP e adattarlo alle attuali dimensioni

CV::MAT COSTRUTTORI

Constructor	Description
<pre>cv::Mat(cv::Size sz, int type, void* data, size_t step=AUTO_STEP)</pre>	Two-dimensional arrays by type with preexisting data (size in sz)
<pre>cv::Mat(int ndims, const int* sizes, int type)</pre>	Multidimensional arrays by type
<pre>cv::Mat(int ndims, const int* sizes, int type, const Scalar& s)</pre>	Multidimensional arrays by type with initialization value
<pre>cv::Mat(int ndims, const int* sizes, int type, void* data, size_t step=AUTO_STEP)</pre> <p style="text-align: center;">↑ <i>sono le dimensioni</i></p>	Multidimensional arrays by type with preexisting data

Elim Parte II – Prof. A. Ferone

CV::MAT COSTRUTTORI DI COPIA

Constructor	Description
<code>cv::Mat(const Mat& mat);</code>	Copy constructor
<code>cv::Mat(const Mat& mat, const cv::Range& rows, const cv::Range& cols);</code> ↳ è una classe	Copy constructor that copies only a subset of rows and columns <i>Estraggo solo un blocco delle mie immagini</i>
<code>cv::Mat(const Mat& mat, const cv::Rect& roi);</code> ↳ Oggetto rettangolare: coordinate del pixel in alto a sx e un'altezza n e un'ampiezza w Definito in termini di:	Copy constructor that copies only a subset of rows and columns specified by a region of interest
<code>cv::Mat(const Mat& mat, const cv::Range* ranges);</code>	Generalized region of interest copy constructor that uses an array of ranges to select from an n -dimensional array
<code>cv::Mat(const cv::MatExpr& expr);</code> ↳ espressione matematica che crea una matrice ↳ è differente da cv::Mat, alloca spazio	Copy constructor that initializes m with the result of an algebraic expression of other matrices

Elim Parte II – Prof. A. Ferone

CV::MAT ALTRE FUNZIONI

- La classe cv::Mat fornisce anche dei metodi statici per creare degli array di uso comune

- zeros() : crea una matrice di tutti 0
- ones() : // " " " 1
- eye() : // ' ' identity

Function	Description
<code>cv::Mat::zeros(rows, cols, type);</code>	Create a cv::Mat of size <code>rows × cols</code> , which is full of zeros, with type <code>type</code> (CV_32F, etc.)
<code>cv::Mat::ones(rows, cols, type);</code>	Create a cv::Mat of size <code>rows × cols</code> , which is full of ones, with type <code>type</code> (CV_32F, etc.)
<code>cv::Mat::eye(rows, cols, type);</code>	Create a cv::Mat of size <code>rows × cols</code> , which is an identity matrix, with type <code>type</code> (CV_32F, etc.)

CV::MAT ACCESSO AGLI ELEMENTI

- Ci sono diversi modi per accedere agli elementi di una matrice
- Le opzioni principali per **accedere ai singoli elementi** sono
 - locazione/posizione
 - iterazione
- Per l'**accesso diretto**, l'approccio principale è il metodo **at<>()** di cui esistono diverse varianti
- Il metodo templato viene **specializzato con i tipi** di dati contenuti nella matrice e successivamente si utilizzano gli indici di riga e colonna per accedere al dato

CV::MAT ACCESSO AGLI ELEMENTI

- Es. matrice 10x10 single channel

```
cv::Mat m = cv::Mat::eye( 10, 10, 32FC1 );  
  
printf(  
    "Element (3,3) is %f\n",  
    m.at<float>(3,3)  
);
```

- Es. matrice 10x10 multichannel

```
cv::Mat m = cv::Mat::eye( 10, 10, 32FC2 );  
  
printf(  
    "Element (3,3) is (%f,%f)\n",  
    m.at<cv::Vec2f>(3,3)[0],  
    m.at<cv::Vec2f>(3,3)[1]  
);
```

CV::MAT ACCESSO AGLI ELEMENTI

Example	Description
<code>M.at<int>(i);</code>	Element <i>i</i> from integer array M
<code>M.at<float>(i, j);</code>	Element (<i>i</i> , <i>j</i>) from float array M
<code>M.at<int>(pt);</code> <i>è un oggetto punto, identifica un punto / pixel in mat</i>	Element at location (<i>pt.x</i> , <i>pt.y</i>) in integer matrix M
<code>M.at<float>(i, j, k);</code>	Element at location (<i>i</i> , <i>j</i> , <i>k</i>) in three-dimensional float array M
<code>M.at<uchar>(idx);</code> <i>vettore di indici</i>	Element at <i>n</i> -dimensional location indicated by <i>idx[]</i> in array M of unsigned characters

CV::MAT ACCESSO ALLE RIGHE

- È anche possibile accedere ad un'intera riga usando il metodo templato **ptr<>()** che prende come argomento l'indice della riga e restituisce un puntatore al primo elemento

$\text{ptr} \leftarrow (\begin{matrix} & \\ & \end{matrix})$
sto accedendo alla 20° riga

CV::MAT ITERATORI

- Un altro approccio per l'**accesso sequenziale** agli elementi della matrice è attraverso l'uso di iteratori
- OpenCV fornisce due iteratori templati (const e non-const)
 - **cv::MatIterator<>**
 - **cv::MatConstIterator<>**
- La classe cv::Mat mette a disposizione i metodi **begin()** e **end()** che restituiscono oggetti iteratori
- Ogni iteratore deve essere dichiarato dello stesso tipo di oggetto contenuto nell'array

CV::MAT ITERATORI

```
int sz[3] = { 4, 4, 4 };
cv::Mat m( 3, sz, CV_32FC3 ); // A three-dimensional array of size 4-by-4-by-4
cv::randu( m, -1.0f, 1.0f ); // fill with random numbers from -1.0 to 1.0

float max = 0.0f;           // minimum possible value of L2 norm
cv::MatConstIterator<cv::Vec3f> it = m.begin();
while( it != m.end() ) {
    len2 = (*it)[0]*(*it)[0]+(*it)[1]*(*it)[1]+(*it)[2]*(*it)[2];
    if( len2 > max ) max = len2;
    it++;
}

} 
```

*punto ad un vettore
di 3 dimensioni:
(0,1,2)*

*iteratore espone
ponte al 1°
elemento delle matrice*

*n.b. non sono le dimensioni
delle matrici, ma i canali
dei pixel*

CV::MAT ACCESSO A BLOCCHI

- Un'altra situazione tipica consiste nell'accedere ad un sottoinsieme dell'array
 - Esistono diversi metodi nella classe cv::Mat
 - **IMPORTANTE:** i metodi che accedono a sottoinsiemi di dati non creano una copia dei dati originali ma accedono ai dati originali
 - Il modo più semplice è tramite i metodi **row()** e **col()** che ricevono come argomento un intero e restituiscono la riga o la colonna specificata
 - Le varianti **rowRange()** e **colRange()** consentono di estrarre un array di righe o colonne contigue
 - come argomento ricevono due interi (il primo è incluso e l'ultimo è escluso)
 - o un oggetto Range
- Prendono in input
un range di righe / colonne*

CV::MAT ACCESSO A BLOCCHI

- L'ultimo modo per estrarre sottomatrici è tramite **operator()** passando come argomento
 - due cv::Range, uno per le righe ed uno per le colonne
 - un oggetto di tipo cv::Rect

CV::MAT ACCESSO A BLOCCHI

Example	Description
<code>m.row(i);</code>	Array corresponding to row <code>i</code> of <code>m</code>
<code>m.col(j);</code>	Array corresponding to column <code>j</code> of <code>m</code>
<code>m.rowRange(i0, i1);</code>	Array corresponding to rows <code>i0</code> through <code>i1-1</code> of matrix <code>m</code>
<code>m.rowRange(cv::Range(i0, i1));</code>	Array corresponding to rows <code>i0</code> through <code>i1-1</code> of matrix <code>m</code>
<code>m.colRange(j0, j1);</code>	Array corresponding to columns <code>j0</code> through <code>j1-1</code> of matrix <code>m</code>
<code>m.colRange(cv::Range(j0, j1));</code>	Array corresponding to columns <code>j0</code> through <code>j1-1</code> of matrix <code>m</code>
<code>m.diag(d);</code>	Array corresponding to the <code>d</code> -offset diagonal of matrix <code>m</code>
<code>m(cv::Range(i0,i1), cv::Range(j0,j1));</code>	Array corresponding to the subrectangle of matrix <code>m</code> with one corner at <code>i0, j0</code> and the opposite corner at <code>(i1-1, j1-1)</code>
<code>m(cv::Rect(i0,i1,w,h));</code>	Array corresponding to the subrectangle of matrix <code>m</code> with one corner at <code>i0, j0</code> and the opposite corner at <code>(i0+w-1, j0+h-1)</code>
<code>m(ranges);</code>	Array extracted from <code>m</code> corresponding to the subvolume that is the intersection of the ranges given by <code>ranges[0] - ranges[nDim-1]</code>

Elim Parte II – Prof. A. Ferone

CV::MAT OPERAZIONI ALGEBRICHE

- OpenCV sfrutta le potenzialità del C++ (overload) per fornire delle operazioni algebriche sulle matrici
- Il risultato di queste operazioni è assegnato all'array di destinazione con l'operatore **operator=()**
- Questo operatore non restituisce un cv::Mat ma un **cv::MatExpr()** che può essere assegnato ad un cv::Mat
- Questa distinzione è stata introdotta perchè il risultato di un'operazione algebrica implica anche l'allocazione dell'array che contiene il risultato
- Nella tabella seguente sono elencati anche altri operatori oltre alle operazioni algebriche

CV::MAT OPERAZIONI ALGEBRICHE

Example	Description
<code>m0 + m1, m0 - m1;</code>	Addition or subtraction of matrices
<code>m0 + s; m0 - s; s + m0, s - m1;</code>	Addition or subtraction between a matrix and a singleton
<code>-m0;</code>	Negation of a matrix
<code>s * m0; m0 * s;</code>	Scaling of a matrix by a singleton
<code>m0.mul(m1); m0/m1;</code>	Per element multiplication of m0 and m1, per-element division of m0 by m1
<code>m0 * m1;</code>	Matrix multiplication of m0 and m1
<code>m0.inv(method);</code>	Matrix inversion of m0 (default value of method is DECOMP_LU)
<code>m0.t();</code>	Matrix transpose of m0 (no copy is done)
<code>m0>m1; m0>=m1; m0==m1; m0<=m1; m0<m1;</code>	Per element comparison, returns uchar matrix with elements 0 or 255

CV::MAT OPERAZIONI ALGEBRICHE

Example	Description
<code>m0&m1; m0 m1; m0^m1; ~m0; m0&s; s&m0; m0 s; s m0; m0^s; s^m0;</code>	Bitwise logical operators between matrices or matrix and a singleton
<code>min(m0,m1); max(m0,m1); min(m0,s); min(s,m0); max(m0,s); max(s,m0);</code>	Per element minimum and maximum between two matrices or a matrix and a singleton
<code>cv::abs(m0);</code>	Per element absolute value of m0
<code>m0.cross(m1); m0.dot(m1);</code>	Vector cross and dot product (vector cross product is defined only for 3×1 matrices)
<code>cv::Mat::eye(Nr, Nc, type); cv::Mat::zeros(Nr, Nc, type); cv::Mat::ones(Nr, Nc, type);</code>	Class static matrix initializers that return fixed $N_r \times N_c$ matrices of type type

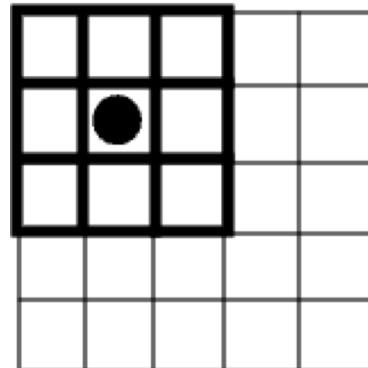
CV::MAT UTILITY

Example	Description
<code>m1 = m0.clone();</code>	Make a complete copy of <code>m0</code> , copying all data elements as well; cloned array will be continuous
<code>m0.copyTo(m1);</code>	Copy contents of <code>m0</code> onto <code>m1</code> , reallocating <code>m1</code> if necessary (equivalent to <code>m1=m0.clone()</code>)
<code>m0.copyTo(m1, mask);</code>	Same as <code>m0.copyTo(m1)</code> , except only entries indicated in the array <code>mask</code> are copied
<code>m0.convertTo(m1, type, scale, offset);</code>	Convert elements of <code>m0</code> to <code>type</code> (e.g., <code>CV_32F</code>) and write to <code>m1</code> after scaling by <code>scale</code> (default <code>1.0</code>) and adding <code>offset</code> (default <code>0.0</code>)
<code>m0.assignTo(m1, type);</code>	Internal use only (resembles <code>convertTo</code>)
<code>m0.setTo(s, mask);</code>	Set all entries in <code>m0</code> to singleton value <code>s</code> ; if <code>mask</code> is present, set only those values corresponding to nonzero elements in <code>mask</code>
<code>m0.reshape(chan, rows);</code>	Changes effective shape of a two-dimensional matrix; <code>chan</code> or <code>rows</code> may be zero, which implies "no change"; data is not copied
<code>m0.push_back(s);</code>	Extend an $m \times 1$ matrix and insert the singleton <code>s</code> at the end
<code>m0.push_back(m1);</code>	Extend an $m \times n$ by k rows and copy <code>m1</code> into those rows; <code>m1</code> must be $k \times n$
<code>m0.pop_back(n);</code>	Remove <code>n</code> rows from the end of an $m \times n$ (default value of <code>n</code> is 1) ^a

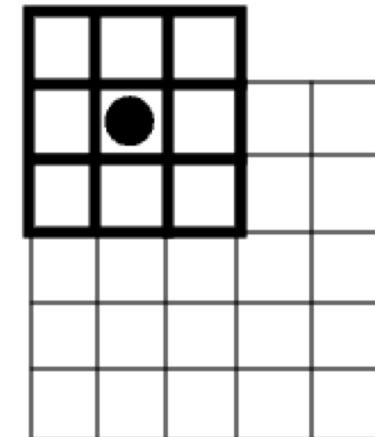
Elim Parte II – Prof. A. Ferone

CV::MAT PADDING

- Il padding è un'operazione che consiste nell'aggiungere ad un'immagine righe e colonne extra
- Viene utilizzata per generalizzare alcune operazioni quando vengono applicate ai bordi dell'immagine
- Es. sostituire ad ogni pixel il valore medio calcolato nell'intorno  3×3
- Cosa succede per i pixel sul bordo dell'immagine?



Elim Parte II – Prof. A. Ferone



CV::MAT PADDING

- Il padding è un'operazione che consiste nell'aggiungere ad un'immagine righe e colonne extra

- Scelta tutti 0 sui bordi:
- Selego un intorno di dimensioni dispari (in modo essere al più duro +) rispetto esterna in più
- Quando voglio analizzare un determinato pixel, sposto l'intorno un corrispondente di quel pixel
- A questo punto x le righe e colonne che si trovano fuori, vedo quanto differisce il loro valore
- Ancor point: punto in cui piatto la matrice
 - nelle dispari x default è centrale

PADDING

↓

Image

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

Image

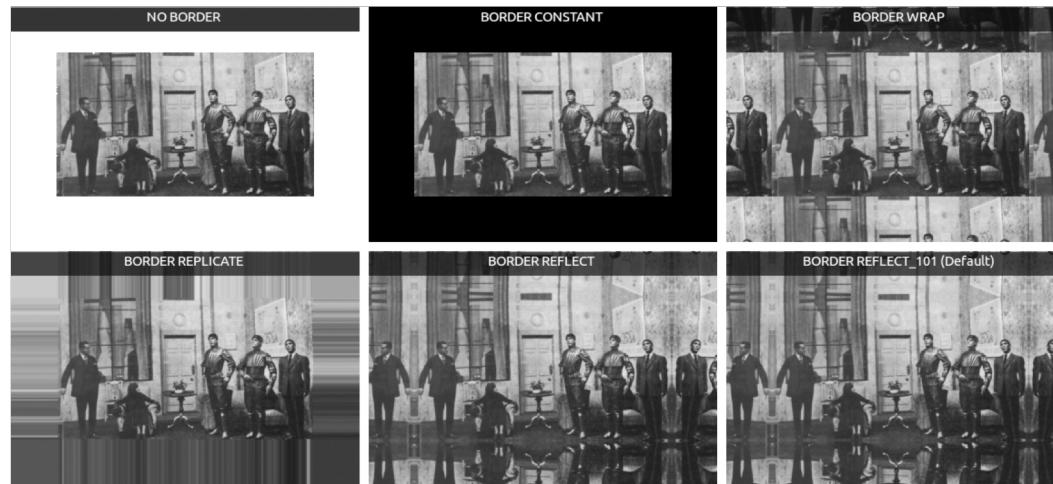
0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

Elim Parte II – Prof. A. Ferone

CV::MAT PADDING

- OpenCV fornisce una funzione per eseguire il padding

```
void cv::copyMakeBorder(
    cv::InputArray    src,                      // Input image
    cv::OutputArray   dst,                      // Result image
    int              top,                      // Top side padding (pixels)
    int              bottom,                     // Bottom side padding (pixels)
    int              left,                       // Left side padding (pixels)
    int              right,                      // Right side padding (pixels)
    int              borderType,                  // Pixel extrapolation method
    const cv::Scalar& value = cv::Scalar()       // Used for constant borders
);
```



Elim Parte II – Prof. A. Ferone

CV::MAT ESERCIZI

■ Esercizi

- Realizzare una funzione che effettui il padding di un'immagine
- Sostituire al valore di ogni pixel il valore medio dei livelli di grigio in un intorno

