27-10-2022

L'INTERPOLAZIONE

Lo **zooming** può essere eseguito per **ingrandire** o per **rimpicciolire**
Per esempio:

- Data una matrice **4x4** e si vuole **ingrandire 2x**, la matrice diventa **8x8**.
- Quindi il numero di pixel è **quadruplicato**.
  Supponiamo che la matrice d'esempio sia:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Ingrandendo la matrice (2x), distribuisco i pixel sulla nuova immagine, ottenendo una 8x8 come segue:

1		2		3		4	
5		6		7		8	
9		10		11		12	
13		14		15		16	

> Quindi avrò dei posti vuoti, cioè righe vuote e colonne vuote.

L'interpolazione lavora sull'immagine ingrandita e si stabilisce quanto valgono i vari pixel vuoti. Essa fa uso di diversi algoritmi:

1. Nearest neighbor(replication);
2. Bilinear;
3. Bicubic;

Nearest neighbor(replication)

- Questo algoritmo **vede i vicini** di un pixel.
  La procedura è la seguente:

1. Considero la posizione [0,1];
2. Ricopio un numero dei vicini, ma quale fra quello a destra e quello a sinistra?
3. E' una scelta arbitraria e va rispettata per ogni casella successiva;
4. Decido di copiare da sinistra. Quindi copio **sempre a sinistra**

1	1	2	2	3	3	4	4
5	5	6	6	7	7	8	8
9	9	10	10	11	11	12	12
13	13	14	14	15	15	16	16

> Esaminando la matrice noto che **riempio in righe dispari e colonne pari**.
> Nelle le **righe pari e colonne dispari** e decido di copiare da sopra e rispetto questa condizione per tutte le altre rispettive posizioni.

Fin quando è possibile uso i pixel di partenza, altrimenti prendo quelli dell'iterazione precedente. Decido, allora, di copiare da sopra:

1	1	2	2	3	3	4	4
1		2		3		4	
5	5	6	6	7	7	8	8
5		6		7		8	
9	9	10	10	11	11	12	12
9		10		11		12	
13	13	14	14	15	15	16	16
13		14		15		16	

- Ma in **riga pari colonna pari** cosa faccio? Uso pixel in alto a sinistra.

> Se ci sono i pixel originali li prendo anche in diagonale altrimenti no

1	1	2	2	3	3	4	4
1	1	2	2	3	3	4	4
5	5	6	6	7	7	8	8
5	5	6	6	7	7	8	8
9	9	10	10	11	11	12	12
9	9	10	10	11	11	12	12
13	13	14	14	15	15	16	16
13	13	14	14	15	15	16	16

Il risultato ottenuto è un'immagine scalettata perchè un pixel diventa un blocco 2x2 e questo effetto crea questo difetto.

- Qual è il **pregio**? non si inserisce mai un valore non presente nell'immagine di partenza.


(a)


(b)

Bilinear

Si torna alla situazione della matrice 8x8 iniziale.

1		2		3		4	
5		6		7		8	
9		10		11		12	
13		14		15		16	

→Si deve trovare dei pixel che nel proprio **intorno abbiano 4 valori**.

- Nella prima iterazione trovo sempre i valori originali della matrice di partenza ma, nelle prossime iterazioni, potrei non trovare tutti e 4 i valori.

Allora il primo pixel che nel proprio intorno ha 4 valori ha la riga pari e la colonna pari, ovvero la posizione [2,2] (*con indici che partono da 1*)
La funzione $v(x, y) = ax + by + cxy + d$ è la **formula del bilineare**.

> Quindi la funzione prevede che io conosca a, b, c, d.
> Quanto vale la funzione in posizione [2,2]?

$v(2, 2) = 2a + 2b + 4c + d$
Non conosco quanto vale tale funzione nel punto [2,2] ma sicuramente conosco quanto vale la funzione nei 4 punti nell'intorno di esso, ovvero:
$v(1, 1) = 1 = a + b + c + d$
$v(1, 3) = 2 = a + 3b + 3c + d$

$v(3,1) = 5 = 3a + b + 3c + d$
$v(3,3) = 6 = 3a + 3b + 9c + d$
Quindi posso imporre il seguente sistema lineare

$$\begin{cases} 1 = a+b+c+d \\ 2 = ax+3b+3c+d \\ 5 = 3a+b+3c+d \\ 6 = 3a+3b+9c+d \end{cases}$$

Le soluzioni di questo sistema mi danno le soluzioni a, b, c, d.

> Suppongo che il risultato sia: a = 1 b=2 c=3 d=4 (realmente sono a=2 b=0.5 c=0 d=-1.5). Questi valori **si arrotondano SOLO dopo aver applicato v(2,2)**.
> Se i risultati fossero questi, allora:

v(2,2) = 2+4+12+4=22 e quindi lo scriverei in posizione (2,2)

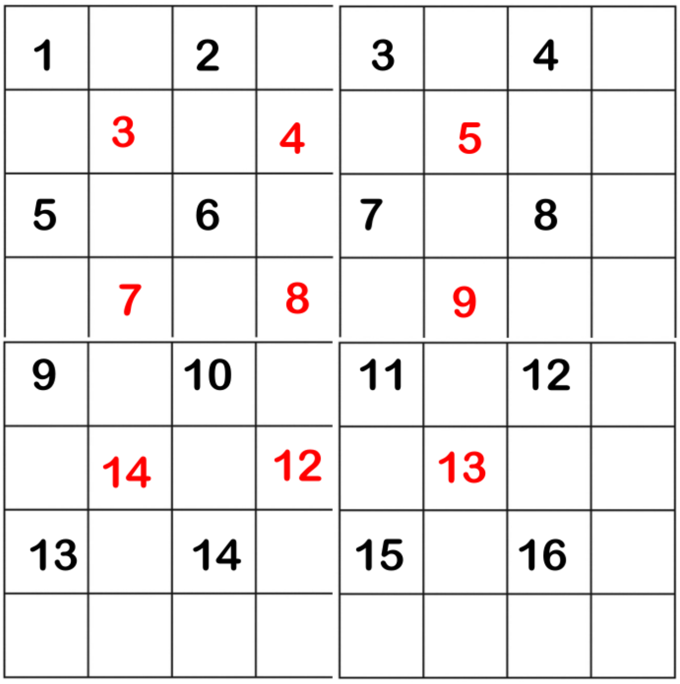
1		2
	22	
5		6

Andando di questo passo, allora, per ogni pixel dovrei trovare le soluzioni del sistema.

> 22 è palesemente errato perchè a b c d sono inventati. Inoltre viene una media in pos (2,2) (1+2+5+6)/4 = MEDIA DEI VALORI floorata

Risolvendo tutti questi sistemi, si può allora definire che nella posizione dove esistono 4 posizioni nell'intorno di un pixel va inserita la media "floorata" dei 4 valori dell'intorno
Quindi, se applico questa definizione, in posizione [2,2] metto il valore

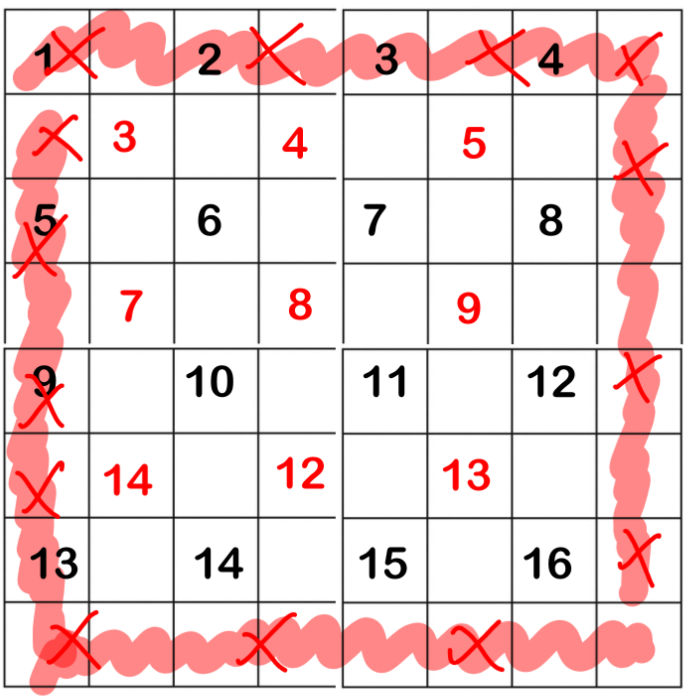
$\lfloor(1+2+5+6)/4\rfloor = 3$



> Quindi alla prima passata risolvo 1/3 dell'immagine perchè nei punti dove ho un intorno di 4 elementi in diagonale.

E nei restanti punti dei bordi? cioè dove non ho 4 punti nell'intorno del pixel in considerazione.

In tali punti **ritaglio l'immagine** di una colonna a sinistra e a destra e una riga sopra e una sotto perchè è una matrice ravvicinata ($2\times$).



- Si può ritagliare senza problemi perchè in genere si trattano di **Mega Pixel** e quindi è totalmente impercettibile.

3		4		5	
	6		7		8
7		8		9	
	10		11		12
14		12		13	
	14		15		16

- Visto che l'output deve essere obbligatoriamente di una determinata dimensione allora **nei pixel mancanti applico una sorta di replication** nei bordi. Guardo i bordi e solo i bordi li tratto con una replication dove ricopio i valori più prossimi.

> cosa faccio righe pari colonne dispari? vale anche righe dispari e colonne pari quindi il metodo vale per entrambi.

Devo copiare per forza quelli dello step precedente, perchè non avrei 4 valori nell'intorno del singolo pixel cosi posso fare il sistema lineare.
Quindi ottengo:

3	3	4	4	5	5
6	6	7	7	8	8
7	7	8	8	9	9
10	10	11	11	12	12
14	14	12	12	13	13
14	14	15	15	16	16

I sistemi si possono scrivere in forma algebrica per colonna ottenendo la **matrice T** moltiplicata per il **vettore riga**:

$$[a \quad b \quad c \quad d] \times \begin{matrix} i-1 & i-1 & i+1 & i+1 \\ j-1 & j+1 & j-1 & j+1 \\ (i-1)(j+1) & (i-1)(j+1) & (i+1)(j-1) & (i+1)(j+1) \\ 1 & 1 & 1 & 1 \end{matrix}$$

in funzione del pixel i j che è al centro della prima iterazione, nel primo caso i = 2 j = 2
Prodotto riga per colonna si ottiene una matrice 1x4 che è un vettore riga:

$[a \quad b \quad c \quad d] \times T = [p_1 \quad p_2 \quad p_3 \quad p_4]$ dove p_1 p_2 p_3 p_4 sono i risultati dei singoli risultati ottenuti da riga per colonna
Ma non ho a, b, c, d quindi applico la **formula**: $[a \quad b \quad c \quad d] = [p_1 \quad p_2 \quad p_3 \quad p_4] \times inversa(T)$

- Questa regola vale anche per i pixel dove non ho ancora inserito nulla visto che si tratta di scrivere la matrice degli indici relativi a tale posizione.
- Per ogni pixel si fa questo lavoro.
- L'immagine ottenuta è **meno seghettata** rispetto a prima ma **è più sfocata**.


(a)


(b)

Bicubic

La formula è

$$v(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

- $a_{i,j}$ è l'equivalente delle incognite da calcolare. Dovrei trovare 1 punto buono per risolvere un sistema a **16 equazioni di 16 incognite** perchè una sommatoria la faccio 4 volte e per ogni ciclo applico la sommatoria più interna altre 4 volte.

- Quindi devo trovare un punto dove **sicuramente ho 16 valori nell'intorno**, quindi il centro andrebbe bene ma comunque resta un'operazione molto complicata da svolgere.
Partendo dal centro, allargo il cerchio finché trovo 16 valori buoni da prendere in considerazione.

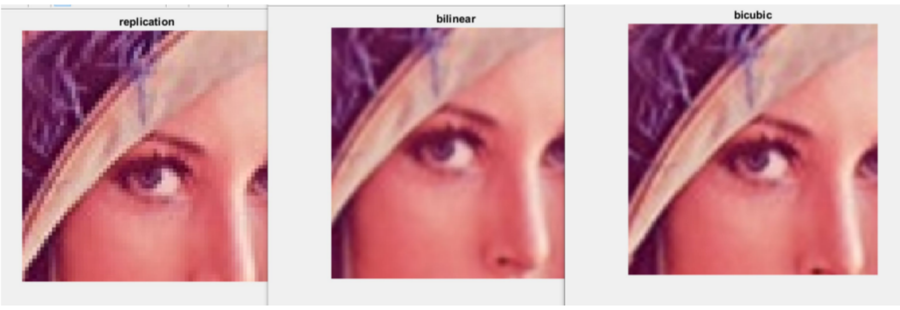
> Si dice **bicubico** perchè x e y sono elevati a 3 al massimo.

Il risultato, rispetto al bilinear, è **meno sfocato** ma sfoca perchè comunque verranno numeri non interi, non presenti nell'immagini di partenza.

"*Pesano*" di più i pixel più vicini e pesano meno via via se ci si allontana.
Il **peso è in funzione della distanza** quindi è una sorta di media pesata.
I pesi tengono conto della distanza del pixel preso in considerazione e quello calcolato. E questo evita l'effetto sfocatura

> Il bicubico rende **un po' più nitida l'immagine** rispetto alle altre interpolazioni.

Il risultato, diverso per ogni tipo di interpolazione, è il seguente:



Interpolazione in MATLAB

Usiamo la forma della matrice.

```
i=double(rgb2gray)(imread)

[m n] = size(I);
O=zeros(2*m,2*n); //matrice output di zeri

for i=1:m
        for j=1:n
                O(2*i-1,2*j-1)=I(i,j); //prende i valori e li ricopia nella
matrice nuova
        end
end

figure ,imshow(O,[]);
//interpolazione bilineare
for i=2:2*m-1
        for j=2:2*n-1
                if mod(i,2) == 0 & mod(j,2) == 0 //riga pari colonna pari
                        P=[O(i-1,j-1),O(i-1,j+1),O(i+1,j-1),O(i+1,j+1)];
                        T=[i-1 i-1 i+1 i+1; j-1 j+1 j-1 j+1; (i-1)*(j-1) (i-1)*
(j+1) (i+1)*(j-1) (i+1)*(j+1); 1 1 1 1];
                        vett=P*inv(T);
                        a=vett(1,1);
```

```
                    b=vett(1,2);
                    c=vett(1,3);
                    d=vett(1,4);
                    O(i,j)=a*i+b*j+c*i*j+d;
                end
        end
end

//mi sposto nelle posizioni sopra sotto dx e sx
for i=2:2*m-1
        for j=2:2*n-1
                if (mod(i,2) ~= 0 & mod(j,2) == 0) | mod(i,2) == 0 & mod(j,2) ~=
0 //riga pari colonna dispari oppure riga dispari colonna pari
                        P=[O(i-1,j),O(i,j-1),O(i,j+1),O(i+1,j)];
                        T=[i-1 i i i+1; j j-1 j+1 j; (i-1)*j (i)*(j-1) (i)*(j+1)
(i+1)*(j+1); 1 1 1 1];
                        vett=P*inv(T);
                        a=vett(1,1);
                        b=vett(1,2);
                        c=vett(1,3);
                        d=vett(1,4);
                        O(i,j)=a*i+b*j+c*i*j+d;
                end
        end
end
```

> Questo è zoom in (ingrandimento). Lo zoom out è rimpiciolimento.

DECIMAZIONE

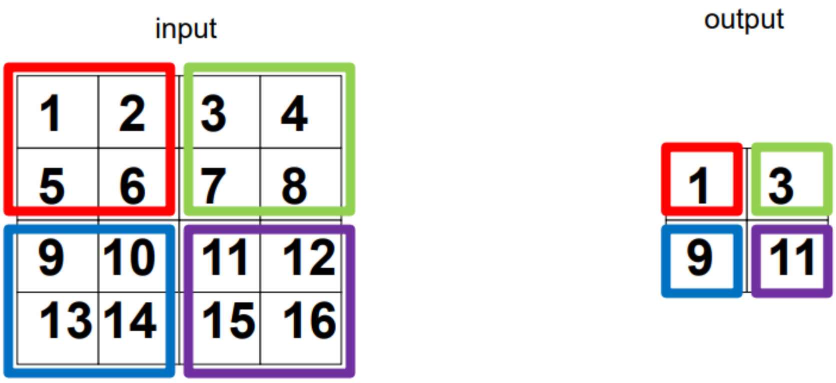
Per decimazione si intende lo **zooming out** dell'immagine.

> Se l'immagine è 4x4, dopo uno zoom out di x0.5, si ha una 2x2

Scegliere arbitrariamente un pixel

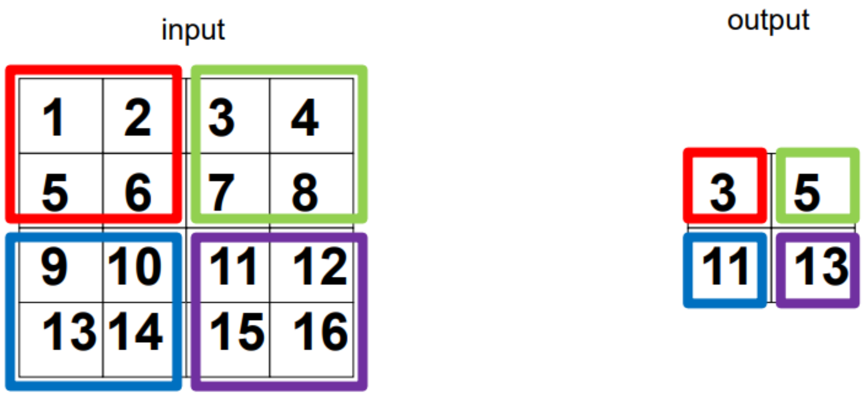
1. Si prende un intorno 2x2 ottenendo una zona di 4 pixel. **Di questi 4 pixel se ne sceglie solamente uno**.
2. Ne tengo uno già di partenza.
3. Scelgo quello in alto a sinistra (arbitrariamente) e rispetto la scelta per le possibili sottomatrici 2x2 (**senza sovvraposizione**).

4. E' una sorta di **replication al contrario**.



Media dei 4 pixel

Tra 4 pixel potrei fare la media fra i 4 valori:



Decimazione in MATLAB

```
//Codice sicuramente non completo
//replication x2
O=imresize(I,2,'nearest');
figure, imshow(O2,[]), title ('replication');

O2=imresize(I,2,'bilinear');
figure, imshow(O2,[]), title ('bilinear');

O3=imresize(I,2,'bicubic');
figure, imshow(O3,[]), title ('bicubic');

//da editare il seguito
I1=imresize(I,0.5,[],'decimazione');
figure, imshow(I1,[]), title ('decimazione');

I2=imresize(I,0.5,[],'decimazione');
figure, imshow(I1,[]), title ('decimazione');

I3=imresize(I,0.5,[],'decimazione');
figure, imshow(I1,[]), title ('decimazione');
```

SISTEMI DI MISURA QUALITA' ALGORITMO INTERPOLAZIONE

MSE

MSE sta per Mean Square Error.

1. Ho 2 matrici $A[m, n]$ e $B[m, n]$;
2. Si fa la differenza punto a punto fra le 2 matrici.
3. Applico $(A[i, j] - B[i, j])^2$ così non ho numeri negativi e positivi che si annullano quindi faccio la sommatoria di tutti i pixel
   $\sum_{i=1}^{n} \sum_{j=1}^{n} (A[i, j] - B[i, j])^2$

- Sostanzialmente si fa un numero meno l'altro. Se questi due numeri sono uguali allora **non c'è errore**.
- L'errore sta qua $(A[i, j] - B[i, j])^2$
  La media viene (***mean***):
  $MSE = \frac{1}{mn} \times \sum_{i=1}^{n} \sum_{j=1}^{n} (A[i, j] - B[i, j])^2$
- MSE **cerca un errore**, cioè due immagini uguali, quindi lo zero. In caso di immagini identiche
- MSE dice **quanto sono diverse le immagini fra loro**.
- Più è alto questo numero e più le due immagini sono diverse.
- Più è basso questo numero più le due immagini sono uguali.

> **L'immagine di output** deve essere **quanto più simile** alla matrice di input quindi **più è basso MSE**, allora **migliore è l'algoritmo**.

PSNR

PSNR sta per Peak Signal to Noise Ratio e la sua formula è $PSNR = -log_{10} \frac{MSE}{S^2}$

- dove **S** è il **valore più alto** che posso usare **in un pixel**. Nel caso di scala di grigi è 255.
- $\frac{MSE}{S^2}$ è sempre compreso fra $[0, 1]$.
- il **segno meno** davanti al logaritmo serve per cambiare segno visto che sarà negativo.

Ci si aspetta MSE basso e PSNR alto

- Questo ci dice quanto sono diverse le immagini ma non quanto una è migliore dell'altra.

PSNR **si può applicare solo su immagini di uguale dimensione**.
In caso di zooming non posso farlo perchè le due immagini (input/output) hanno

dimensioni diverse.
Quindi in questo caso si agisce nel seguente modo:

1. Prendo la matrice iniziale, faccio replication o decimazione, e ottengo un'immagine nuova (per esempio).

- Poi faccio lo zooming della nuova immagine e posso fare **PSNR** con la nuova immagine.
- Ora posso vedere quanto il mio output sia simile all'immagine di partenza.

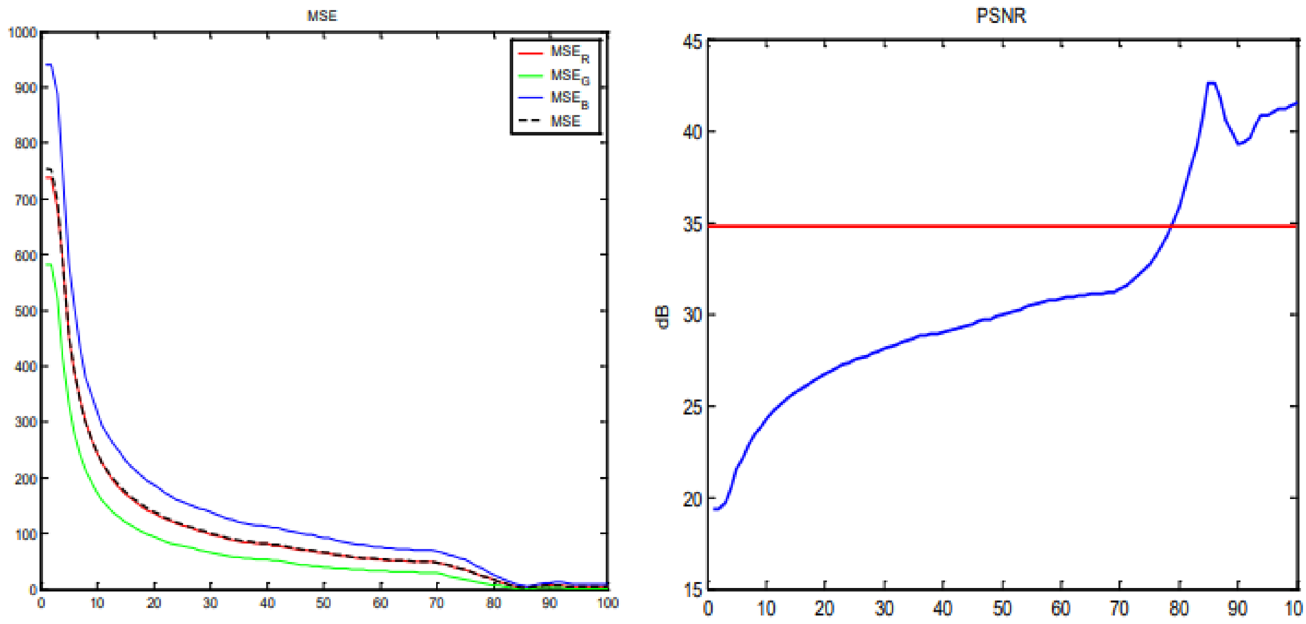
Riassumendo **prima la riduco**, **poi la ingrandisco** e **vedo se assomiglia a quella di partenza**.
Dopo aver implementato tale algoritmo su MATLAB ci si accorge che:

- decimazione con media porta un PSNR più alto;
- decimazione senza media porta un PSNR più basso rispetto alla precedente.

Osservazioni

- La replication non inventa valori nuovi ma riutilzza valori già presenti.
- Tra 4 numeri uno sicuramente vale 0 e quindi questo processo penalizza MSE.
- Vi è un annullamento nelle sommatorie di un quarto dei numeri e quindi l'MSE non è adatto.
- Quindi, in caso di replication, conoscendo la caratteristica di tenere valori di partenza, nel calcolo di MSE e $PSNR$ potrebbero generare risultato falsati proprio per questa caratteristica della replication.
- In caso di interpolazione bilineare e bicubico l'MSE ha un significato più decente, quindi anche $PSNR$ ha un significato migliore.

Valore massimo PSNR



Guardando il grafico si può denotare che:

- $PSNR$ non supera mai **45**.
- Se maggiore $PSNR > 30$ esso è un $PSNR$ buono e più si avvicina a 45 e più è migliore.
- Tale numero non dipende dalla dimensione dell'immagine.
- Si osserva, inoltre, che se, dopo aver scritto l'algoritmo, si ottiene un $PSNR > 45$ allora sicuramente c'è qualche errore nella scrittura dell'algoritmo.