

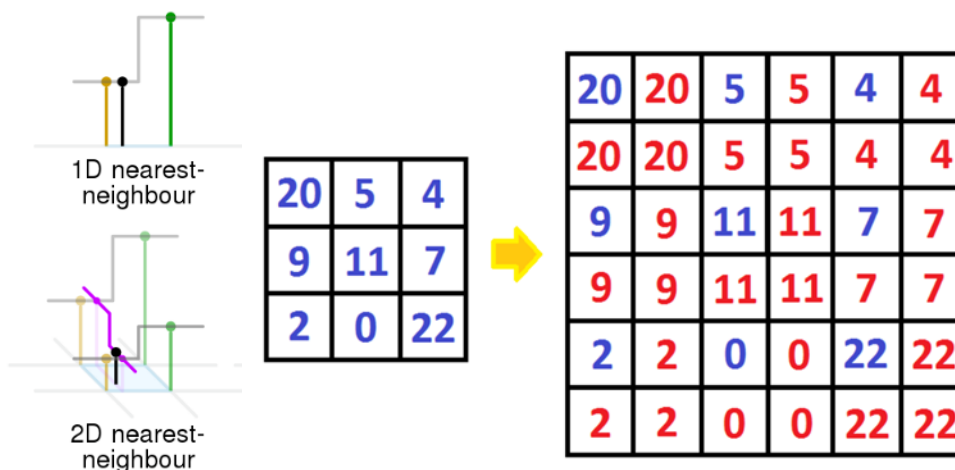
## 1 -

## روش درون‌یابی nearest neighbor :

ساده‌ترین روش درون‌یابی برای توابع چند متغیره در یک یا دو بعد با کمترین محاسبات و بیشترین سرعت ولی با ضعیف‌ترین کیفیت هست.

درون‌یابی یعنی پیدا کردن مقادیر تابع در نقاطی که، مقدار تابع در همسایگی آن نقاط را داریم. الگوریتم nearest neighbor مقدار تابع در نزدیک‌ترین نقطه را انتخاب کرده و نقاط همسایگی دیگر نیازی ندارد. این روش نزدیک‌ترین پیکسل را به پیکسل ناشناخته اختصاص می‌دهد. مقدار متغیر در نقطه مجهول را با مقدار متغیر در نزدیک‌ترین نمونه به آن تخمین می‌زند.

یکی از کاربردهای این روش در تغییر تعداد پیکسل‌های یک تصویر است. به عنوان مثال، تصویری که حاوی 9 پیکسل باشد، برای تبدیل شدن به یک تصویر بزرگتر با 36 پیکسل به 27 پیکسل دیگر نیاز دارد. از آنجایی که تصویر اصلی فاقد این پیکسل‌ها است، باید این پیکسل‌ها را با توجه به مقادیر پیکسل‌های موجود ایجاد کرد. در این روش در همسایگی هر پیکسل سه پیکسل اضافه می‌شود که هر سه پیکسل مقداری مشابه پیکسل اصلی دارند.



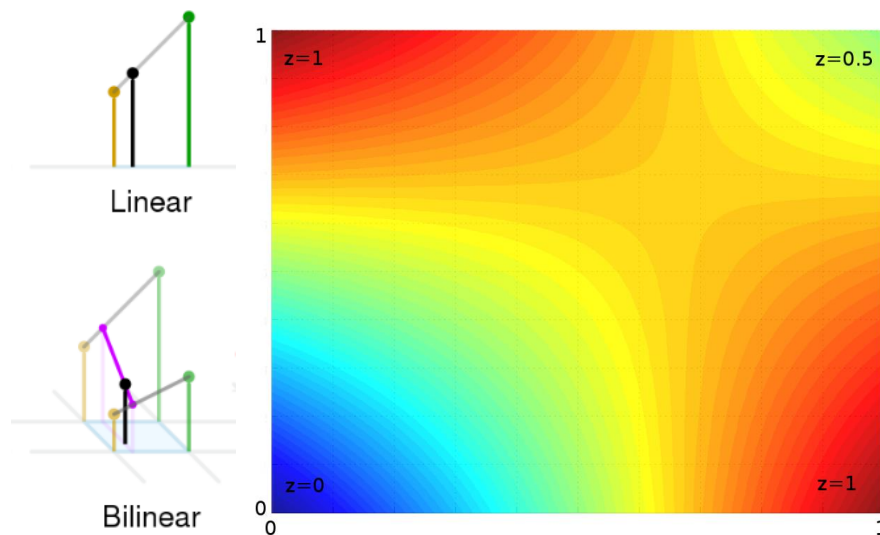
## روش درون‌یابی bilinear :

ایده اصلی درون‌یابی bilinear این است که در ابتدا درون‌یابی در یک جهت انجام شود و پس از این کار مجدداً درون‌یابی در جهت دیگر هم بدست بیاید.

با وجود این که هر گام الگوریتم خطی هستند پاسخ نهایی بدست آمده دیگر خطی نیست.

درون‌یابی خطی بر روی دو متغیر در جدول دو بعدی معمولی عمل می‌کند. تابع‌های درون‌یابی نباید از جمله‌های  $x^2$  یا  $y^2$  استفاده کند و تنها  $xy$  که حالت دو خطی دو متغیر است استفاده خواهند شد.

این روش از روش قبل بهتر است اما فرکانس‌های بالا را به درستی درون‌یابی نمی‌کند.



اگر بخواهیم مقدار تابع مجهول  $f$  را در نقطه  $(x,y)$  بیابیم. با داشتن مقدار  $f$  را در چهار نقطه  $Q_{11}=(x_1,y_1)$  و  $Q_{12}=(x_1,y_2)$  و  $Q_{21}=(x_2,y_1)$  و  $Q_{22}=(x_2,y_2)$ ، با روش درون‌یابی bilinear، خواهیم داشت :

ابتدا درون‌یابی در سمت  $x$  :

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

که  $R_1 = (x, y_1)$  :

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

که  $R_2 = (x, y_2)$  :

با درون یابی در سمت  $y$  :

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

مقدار تقریبی  $f(x, y)$  :

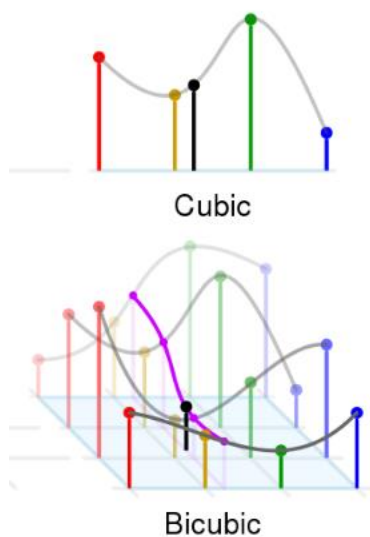
$$\begin{aligned} f(x, y) &\approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)} (x_2 - x)(y_2 - y) + \\ &\quad \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)} (x - x_1)(y_2 - y) + \\ &\quad \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)} (x_2 - x)(y - y_1) + \\ &\quad \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)} (x - x_1)(y - y_1) \\ &= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left( f(Q_{11})(x_2 - x)(y_2 - y) + \right. \\ &\quad \left. f(Q_{21})(x - x_1)(y_2 - y) + \right. \\ &\quad \left. f(Q_{12})(x_2 - x)(y - y_1) + \right. \\ &\quad \left. f(Q_{22})(x - x_1)(y - y_1) \right). \end{aligned}$$

## روش درون‌یابی bicubic :

درون‌یابی Bicubic شبیه به درون‌یابی Bilinear است ولی بر روی نقاط جدول معمولی دو بعدی عمل می‌کند و به جای ۴ نقطه درون‌یابی Bilinear از ۱۶ نقطه برای انجام درون‌یابی استفاده می‌کند.

سطح بدست آمده با استفاده از این درون‌یابی نرم‌تر از سطح بدست آمده توسط درون‌یابی‌های دیگر مانند درون‌یابی nearest neighbor یا درون‌یابی bilinear است.

در پردازش تصویر اگر سرعت محاسبات مهم نباشد معمولاً از این درون‌یابی استفاده می‌شود. این درون‌یابی به خاطر درگیر کردن نقاط بیشتر پیچیدگی بیشتری دارد و محاسبات طولانی‌تری دارد.



فرض کنید مقادیر تابع  $f$  و مشتقات  $f_x$  و  $f_y$  و  $f_{xy}$  در چهار گوشه  $(0,0)$ ،  $(0,1)$ ،  $(1,0)$  و  $(1,1)$  مربع واحد مشخص باشد. سطح درون‌یابی شده را به این صورت می‌توان نوشت:

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

از تطبیق  $p(x,y)$  با مقادیر تابع معادلات زیر بدست می آید :

$$f(0,0) = p(0,0) = a_{00}$$

$$f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30}$$

$$f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03}$$

$$f(1,1) = p(1,1) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij}$$

مشتق ها در سمت X و سمت Y :

$$f_x(0,0) = p_x(0,0) = a_{10}$$

$$f_x(1,0) = p_x(1,0) = a_{10} + 2a_{20} + 3a_{30}$$

$$f_x(0,1) = p_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13}$$

$$f_x(1,1) = p_x(1,1) = \sum_{i=1}^3 \sum_{j=0}^3 a_{ij}i$$

$$f_y(0,0) = p_y(0,0) = a_{01}$$

$$f_y(1,0) = p_y(1,0) = a_{01} + a_{11} + a_{21} + a_{31}$$

$$f_y(0,1) = p_y(0,1) = a_{01} + 2a_{02} + 3a_{03}$$

$$f_y(1,1) = p_y(1,1) = \sum_{i=0}^3 \sum_{j=1}^3 a_{ij}j$$

چهار معادله مشتق ترکیبی XY :

$$f_{xy}(0,0) = p_{xy}(0,0) = a_{11}$$

$$f_{xy}(1,0) = p_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31}$$

$$f_{xy}(0,1) = p_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13}$$

$$f_{xy}(1,1) = p_{xy}(1,1) = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij}ij$$

چهار مقدار یکسان زیر محاسبه می شود :

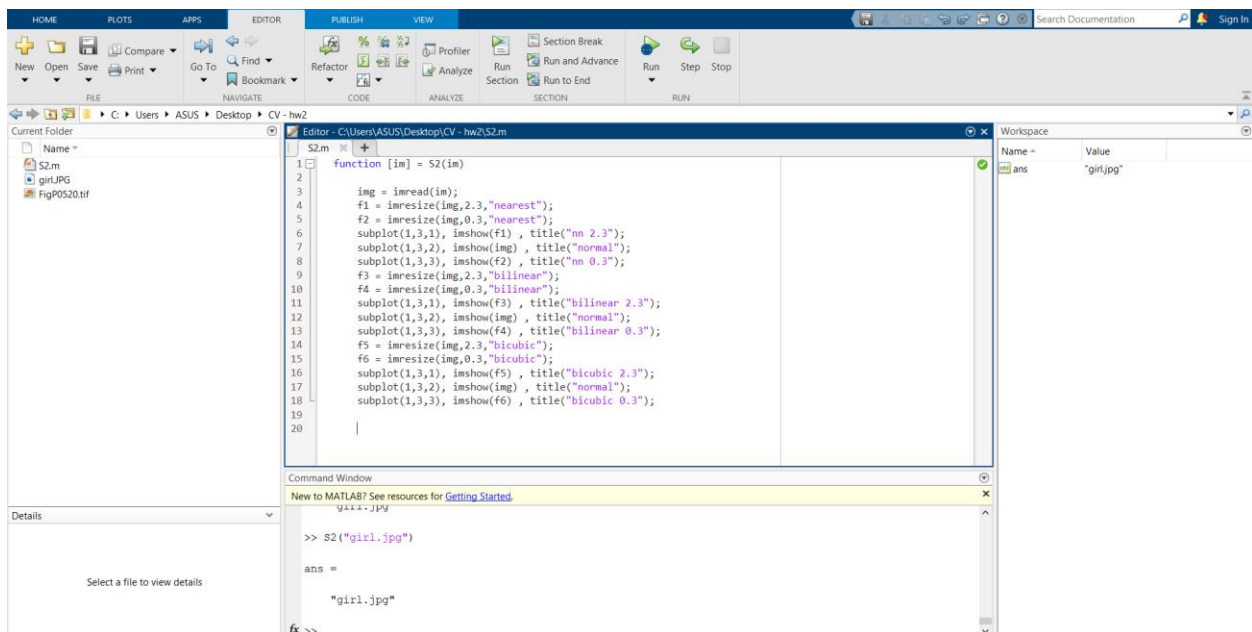
$$p_x(x, y) = \sum_{i=1}^3 \sum_{j=0}^3 a_{ij} i x^{i-1} y^j$$

$$p_y(x, y) = \sum_{i=0}^3 \sum_{j=1}^3 a_{ij} x^i j y^{j-1}$$

$$p_{xy}(x, y) = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij} i x^{i-1} j y^{j-1}.$$

**2-** در این تمرین باید با الگوریتم های گفته شده bilinear و bicubic و nearest neighbor سائز دو عکس دلخواه را به 0.3 و 2.3 برابر تغییر دهیم و تغییرات را بررسی کنیم.

کد : برای این سوال فقط تصویر مورد نظر را با imread() به تابع می دهیم و در تابع imresize با نوشتن الگوریتم و اسکیل مورد نظر، به جواب می رسیم.



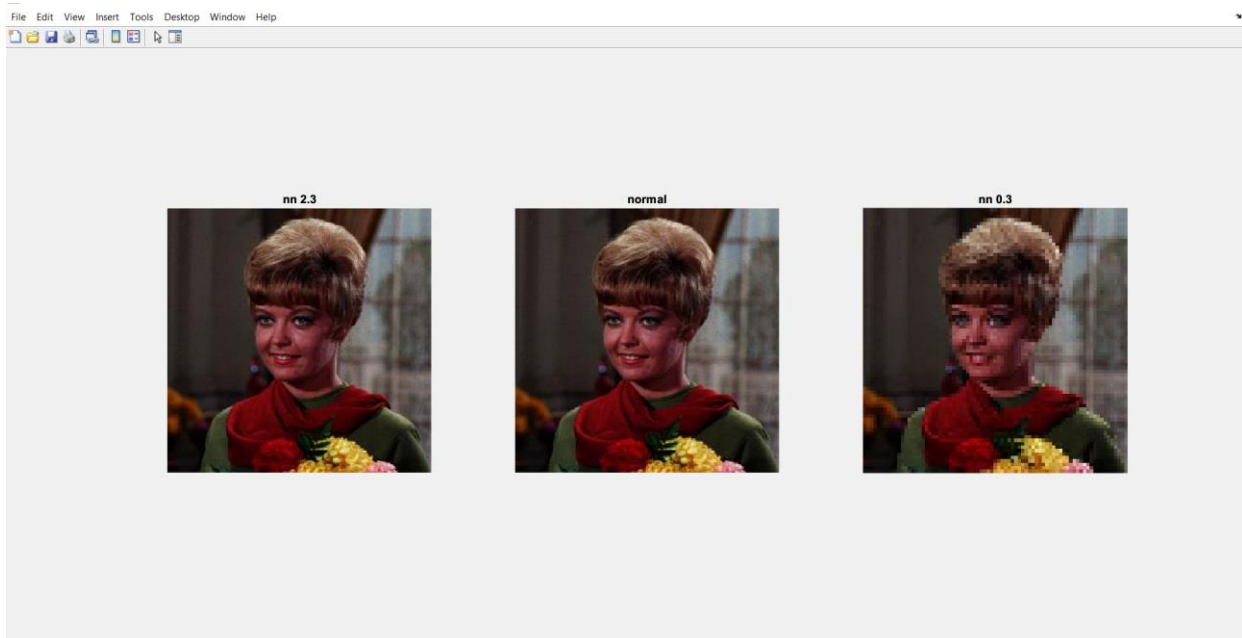
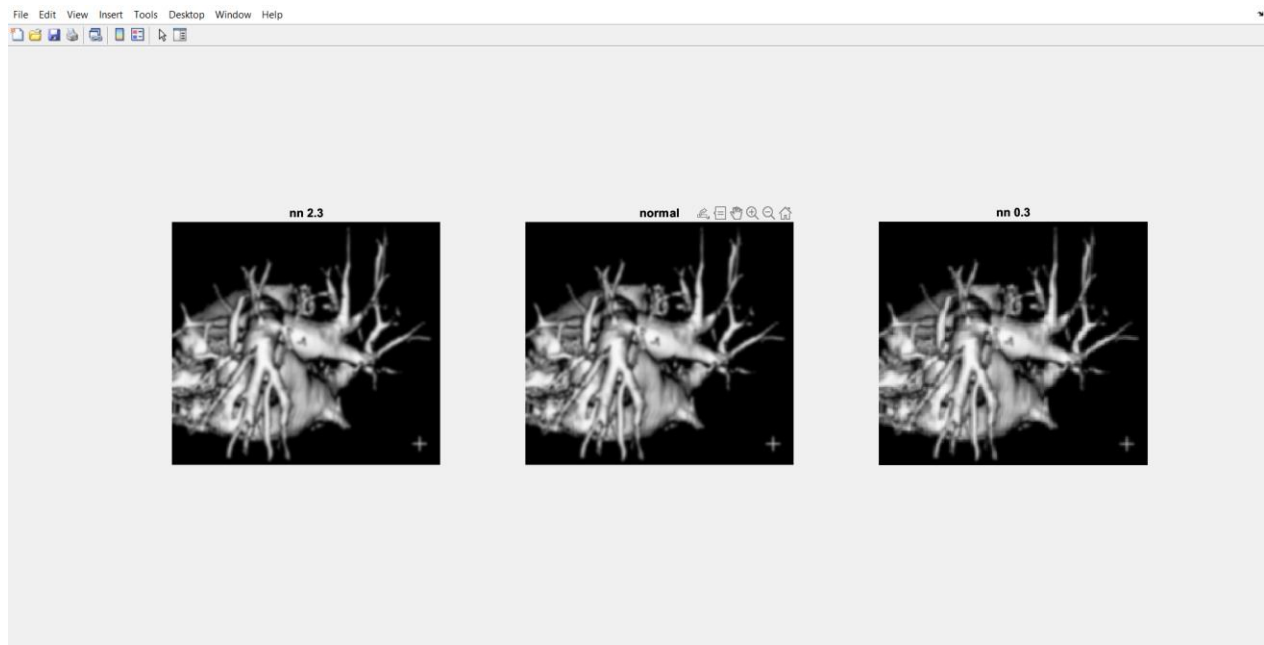
The screenshot shows the MATLAB R2020a environment. The Editor window displays a script named S2.m with the following code:

```
1 function [im] = S2(im)
2
3     img = imread(im);
4     f1 = imresize(img,2.3,"nearest");
5     f2 = imresize(img,0.3,"nearest");
6     subplot(1,3,1), imshow(f1), title("nn 2.3");
7     subplot(1,3,2), imshow(img), title("normal");
8     subplot(1,3,3), imshow(f2), title("nn 0.3");
9     f3 = imresize(img,2.3,"bilinear");
10    f4 = imresize(img,0.3,"bilinear");
11    subplot(1,3,1), imshow(f3), title("bilinear 2.3");
12    subplot(1,3,2), imshow(img), title("normal");
13    subplot(1,3,3), imshow(f4), title("bilinear 0.3");
14    f5 = imresize(img,2.3,"bicubic");
15    f6 = imresize(img,0.3,"bicubic");
16    subplot(1,3,1), imshow(f5), title("bicubic 2.3");
17    subplot(1,3,2), imshow(img), title("normal");
18    subplot(1,3,3), imshow(f6), title("bicubic 0.3");
19
20
```

The Command Window shows the execution of the command `>> S2('girl.jpg')`, resulting in the output `ans = 'girl.jpg'`. The Workspace window shows a variable `ans` with the value `'girl.jpg'`.

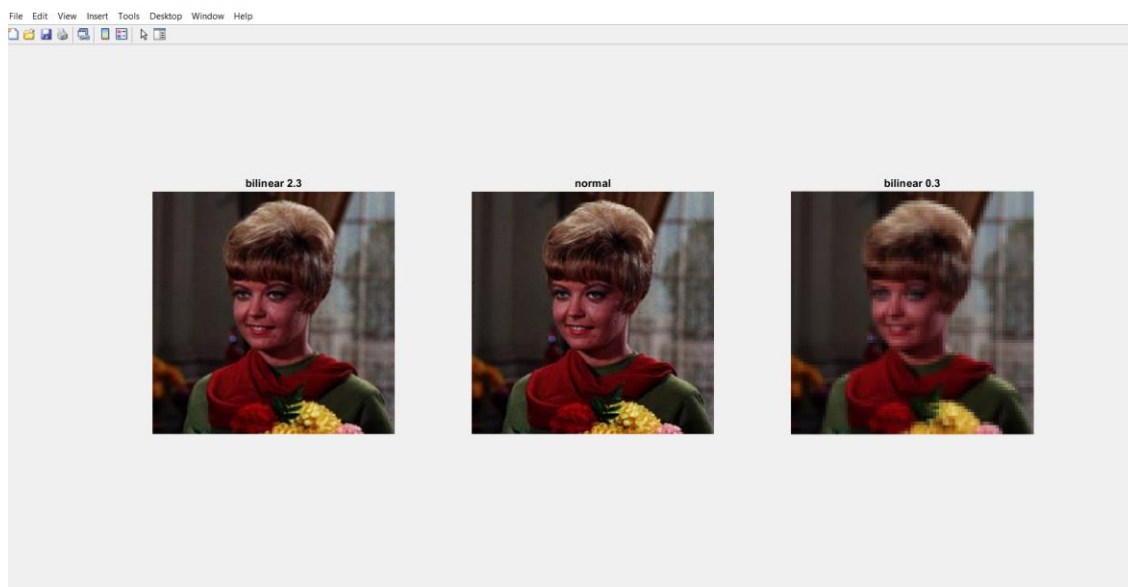
## بررسی خروجی برای nearest neighbor :

کیفیت عکس در 0.3 برابر بسیار کاهش یافته به نحوی که پیکسل های عکس به وضوح دیده می شود ولی در 2.3 برابر کیفیت عکس به نسبت کمتر دست خوش تغییر شده به نسبت 0.3 برابر و بهتر هست.



## بررسی خروجی برای bilinear :

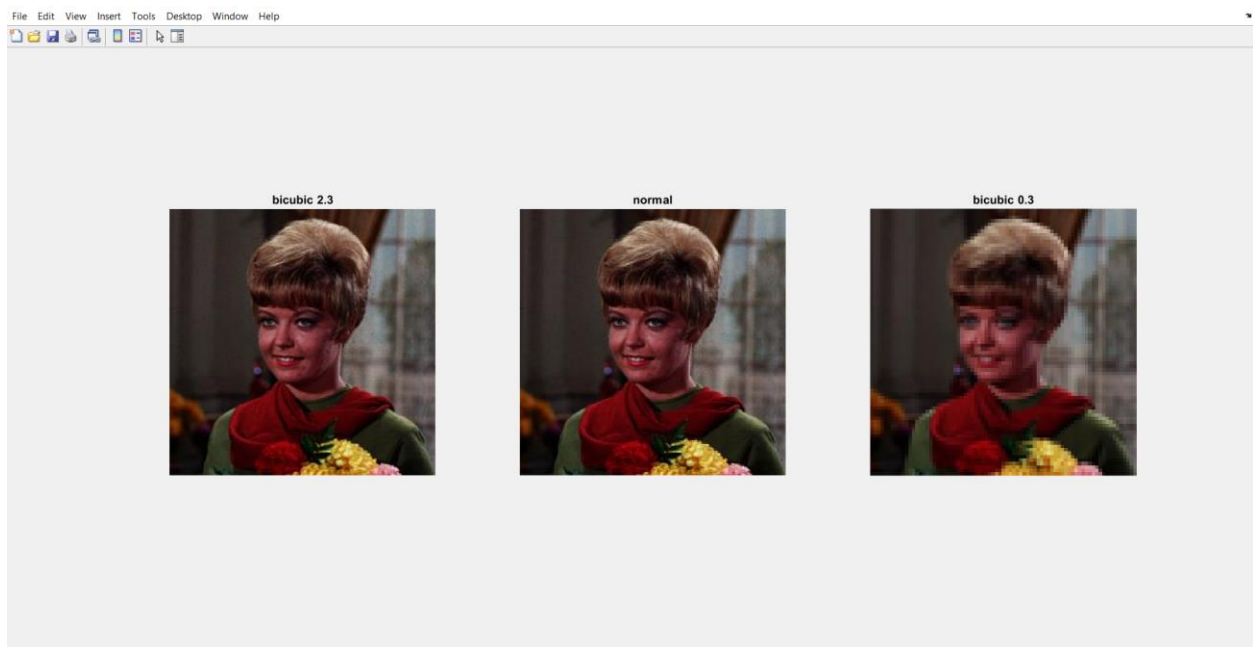
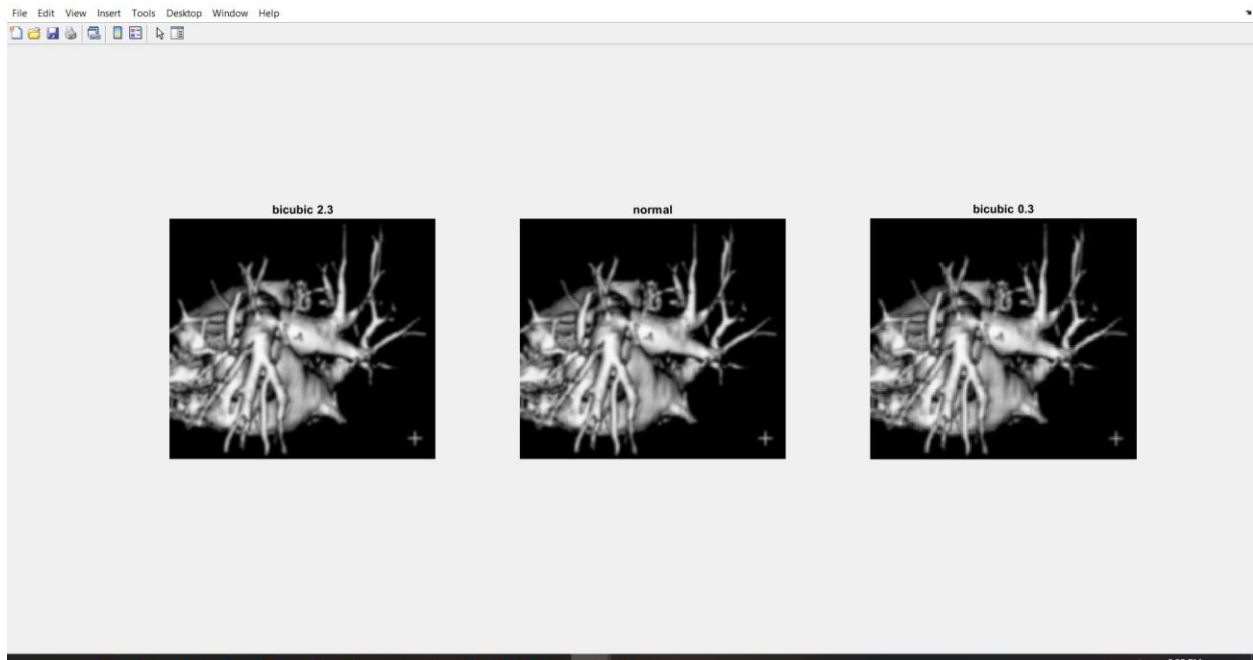
در این روش در حالت 0.3 بخاطر پایین آمدن کیفیت به صورت یکنواخت تصویر تار شده هست ولی در حالت 2.3 برابر کیفیت تصویر از حالت نرمال هم بیشتر شده است.





## بررسی خروجی برای bicubic :

در حالت 0.3 برابر مانند bilinear شاهد افت کیفیت بودیم و تصویر رو به تاری رفته است ولی در حالت 2.3 برابر تفاوت چندانی بین حالت نرمال تصویر و 2.3 برابر نیست و فقط کمی تا حدودی پیکسل ها هموار تر شده‌اند.



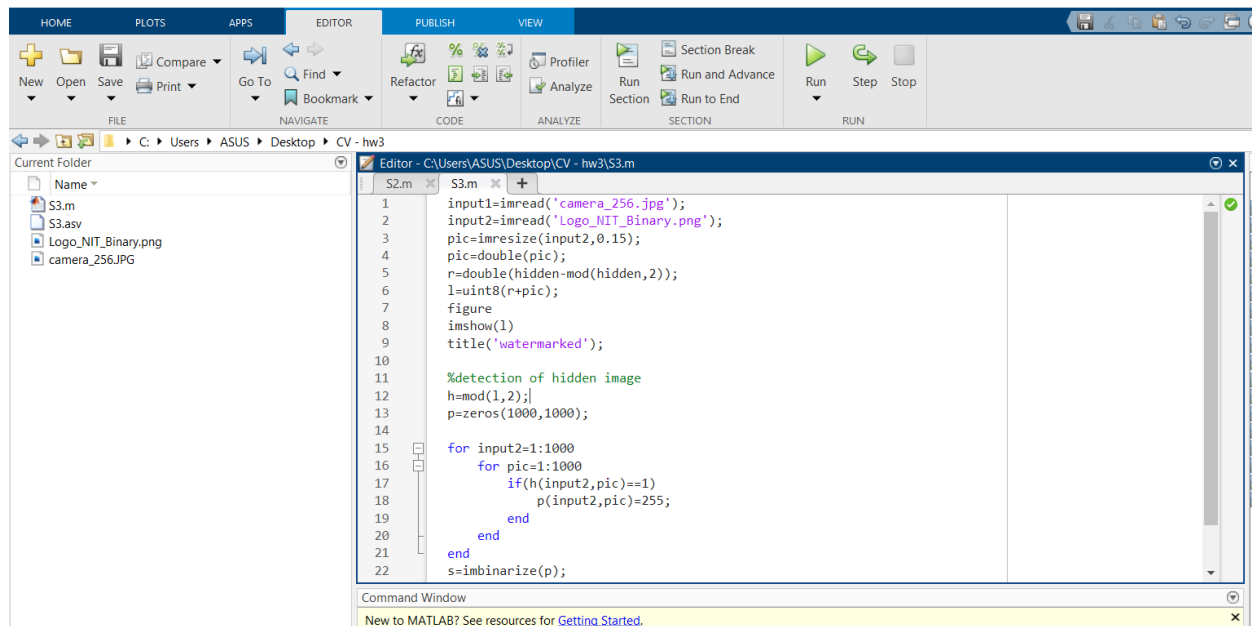
### نهان نگاری در بیت های کم ارزش

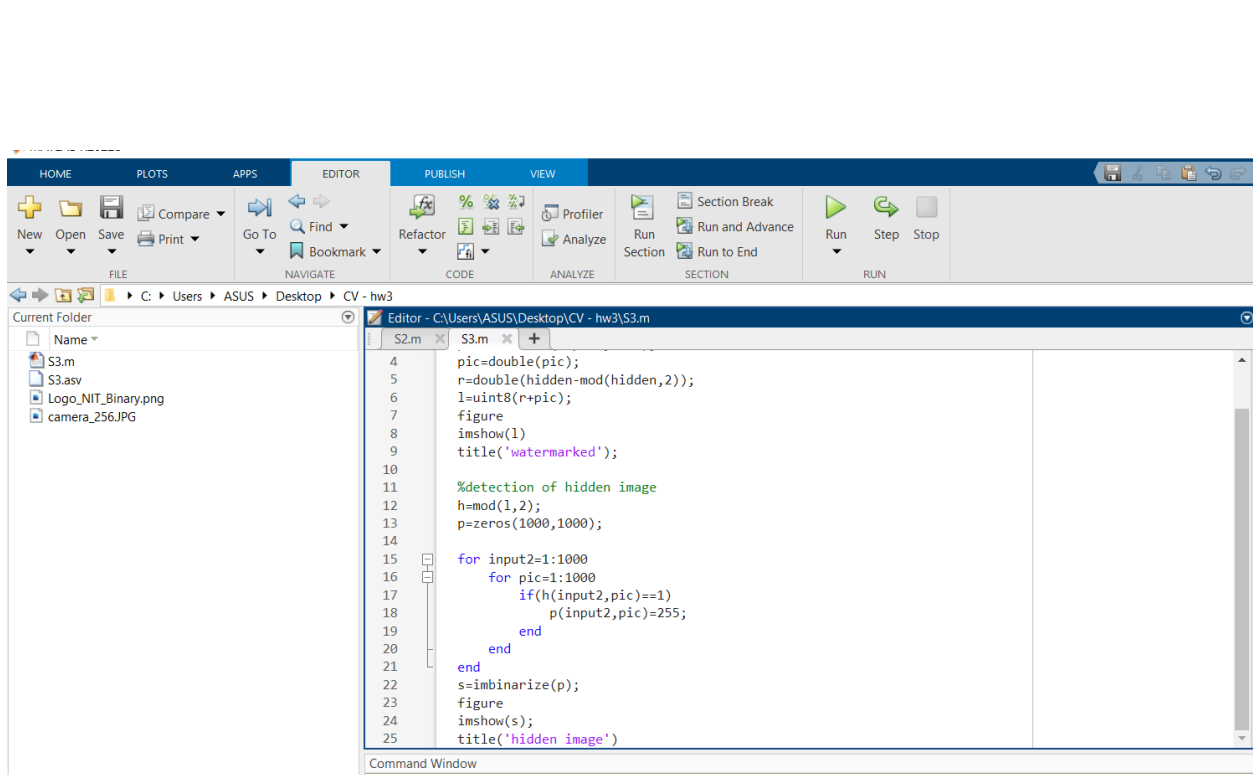
برای حل این سوال ابتدا تصویر camera man به عنوان input1 و تصویر لوگوی دانشگاه را به عنوان input2 از ورودی دریافت می کنیم. و سایز تصویر لوگو را با دستور imresize تغییر دادیم. سپس با بزرگ تر کردن سایز تصویر لوگو با double، نهان نگاری را انجام می دهیم. ابتدا با تفریق بیت های lsb (کم ارزش) را حذف کرده و سپس با جمع کردن بیت های لوگو را در آن جای می دهیم.

```
1 input1=imread('camera_256.jpg');
2 input2=imread('Logo_NIT_Binary.png');
3 pic=imresize(input2,0.15);
4 pic=double(pic);
5 r=double(hidden-mod(hidden,2));
6 l=uint8(r+pic);
7
```

و بعد از نهان نگاری تصویر پنهان شده را هم بازبایی کردیم (در صورت سوال خواسته نشده بود)

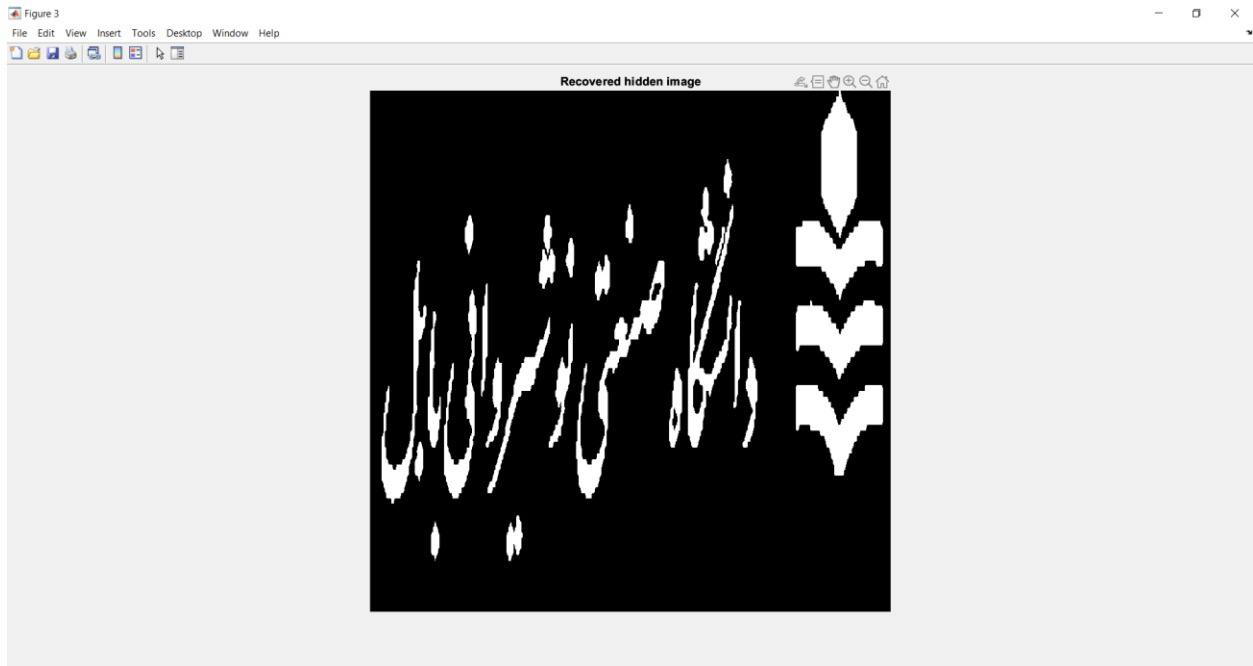
کد :





خروجی :





نهان نگاری بین پیکسل ها

برای این قسمت ابعاد ماتریس را از کاربر دریافت می کنیم و طبق فرمول فواصل محاسبه می شود.

```

1  d = input('dimension d*d => ');
2  m_data=str2double(d);
3  t=floor(m_data(1)*0.5);
4  z=floor(m_data(1)*0.5);
5  dist_a=zeros(m_data);
6  dist_c=zeros(m_data);
7  dist_8=zeros(m_data);
8  for i=0:m_data-1
9      for j=0:m_data-1
10         dist_a(i+1,j+1)=sqrt((t-i)^2 + (z-j)^2);
11         dist_c(i+1,j+1)=(abs(t-i) + abs(z-j));
12         dist_8(i+1,j+1)=max(abs(t-i) , abs(z-j));
13     end
14 end
15 disp(['dimension : ',num2str(m_data),'* ',num2str(m_data)])
16 disp('dist_a = ')
17 disp(num2str(dist_a))
18 disp(' ')
19 disp('-----')
20 disp('dist_c = ')
21 disp(' ')
22 disp(num2str(dist_c))
23 disp('-----')
24 disp('dist_8 = ')
25 disp(' ')
26 disp(num2str(dist_8))
27

```

خروجی :

```
Command Window
New to MATLAB? See resources for Getting Started.

>> S3_2
dimension d*d => 4
dimension : 4*4
dist_a =
2.8284    2.2361    2    2.2361
2.2361    1.4142    1    1.4142
         2         1    0         1
2.2361    1.4142    1    1.4142

-----
dist_c =

4  3  2  3
3  2  1  2
2  1  0  1
3  2  1  2
-----
dist_8 =

2  2  2  2
2  1  1  1
2  1  0  1
2  1  1  1
fx >>
```

#### – 4

**MSE** : روش برآورد میزان خطا است که تفاوت بین مقدار تخمینی و آنچه تخمین زده شده هست را محاسبه می‌کند.

مقدار آن به طور تقریبی در همه جا مثبت هست زیرا تصادفی است و دیتا هایی که قابلیت تخمین دقیق تری دارند را محاسبه نمی‌کند. مقدار این شاخص همواره نامنفی است و هر چقدر به صفر نزدیک تر باشد میزان خطا کمتر هست و هر چقدر بیشتر باشد مقدار خطا بیشتر هست.

در **MSE** وزن دهی به خطا های بزرگ تر بیشتر هست برای مثال بین چندین داده، داده‌ای که خطای بیشتری دارد اثرگذاری بیشتری خواهد داشت.

همچنین **MSE** دارای بُعد هست و شامل بایاس و واریانس است.

## : PSNR

SNR نسبت سیگنال به نویز هست که برابر با نسبت توان سیگنال به توان نویز آن سیگنال هست که معیاری برای بیان عملکرد بهینه سیستم پردازش سیگنال هست که معمولاً بر حسب دسیبل بیان می‌شود و هر چقدر نسبت سیگنال به نویز بیشتر باشد، بهتر است چون اطلاعات بیشتری نسبت به نویز دریافت کرده است.

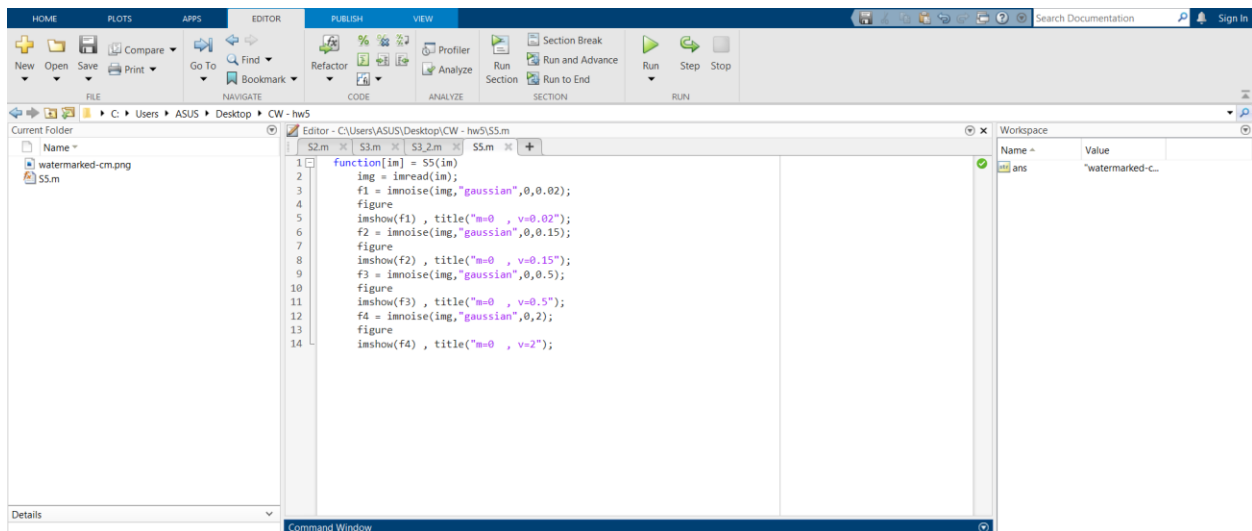
اگر سیگنال Deterministic توان از فرمول زیر محاسبه می‌شود :

$$P_s = \frac{1}{T} \int_0^T s^2(t) dt$$

PSNR در پردازش تصویر این معیار تعریف متفاوتی دارد و صورت کسر برابر با مربع مقدار پیک سیگنال هست و مخرج آن توان نویز یا واریانس آن هست. یک تصویر 8 بیتی دارای مقادیر 0 تا 255 هست به همین علت صورت پیک نسبت سیگنال به نویز در تمام موارد  $255^2$  هست.

## – 5

برای این سوال تصویر به دست آمده‌ی نهان نگاری شده از سوال سوم را ذخیره می‌کنی و آن را به عنوان ورودی این سوال به تابع S5 می‌دهیم سپس با دستور imnoise که امکان انتخاب نوع نویز و واریانس و میانگین را به ما می‌دهد، نویزهای صورت سوال را به تصویر اضافه می‌کنیم.



و نتیجه به صورت زیر خواهد بود :







m=0 , v=0.5



m=0 , v=2

