Модель процесса искажения изображения

Изображение часто бывает искажено вследствие несовершенства оптических аппаратов, дефокусировки снимков, из-за влияния среды между объектом и аппаратом, в силу ряда других причин. Задача восстановления изображения является, как правило, некорректной (сильно неустойчивой) и для ее решения необходимо использовать современные, устойчивые методы.

Постановка задачи восстановления изображения достаточно проста и наглядна. В то же время, решение этой задачи требует использования довольно разнообразных и сложных инструментов, в том числе математического моделирования, статистической теории оценивания, преобразования Фурье, методов оптимизации, численных методов линейной алгебры и др.

А теперь перейдем к более формальному и научному описанию этих процессов искажения и восстановления. Будем рассматривать только полутоновые черно-белые изображения в предположении, что для обработки полноцветного изображения достаточно повторить все необходимые шаги для каждого из цветовых каналов RGB. Введем следующие обозначения:

f(x, y) – исходное неискаженное изображение

h(x, y) – искажающая функция

n(x, y) – аддитивный шум

g(x, y) – результат искажения, т.е. то, что мы наблюдаем в результате (смазанное или расфокусированное изображение)

Сформулируем модель процесса искажения следующим образом:

g(x, y) = h(x, y) \* f(x, y) + n(x, y) (1)

Задача восстановления искаженного изображения заключается в нахождении наилучшего приближения f'(x, y) исходного изображения. Рассмотрим каждую составляющую более подробно.

С f(x, y) и g(x, y) все достаточно понятно. А вот про функцию h(x, y) нужно сказать пару слов – что же она из себя представляет? В процессе искажения каждый пиксель исходного изображения превращается в пятно для случая расфокусировки и в отрезок для случая простого смаза.

Либо же можно сказать наоборот, что каждый пиксель искаженного изображения «собирается» из пикселей некоторой окрестности исходного изображения. Все это друг на друга накладывается и в результате мы получаем искаженное изображение.

То, по какому закону размазывается или собирается один пиксель и называется функцией искажения. Другие синонимы – **PSF** (Point spread function, т.е. функция распределения точки), ядро искажающего оператора, kernel и другие.

Размерность этой функции, как правило меньше размерности самого изображения – к примеру, в начальном рассмотрении примера «на пальцах» размерность функции была 2, т.к. каждый пиксель складывался из двух.

Существующие подходы для деконволюции

Есть подходы, которые учитывают учитывают наличие шума на изображении – один из самых известных и самых первых, это фильтр Винера (Wiener). Он рассматривает изображение и шум как случайные процессы и находит такую оценку **f'** для неискаженного изображения **f**, чтобы среднеквадратическое отклонение этих величин было минимальным. Минимум этого отклонения достигается на функции в частотной области:

https://habrastorage.org/storage2/168/e0e/8d2/168e0e8d2e2755fd412bf5af8e51051a.png (6)

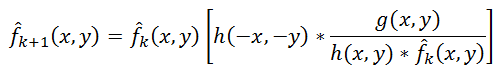
Этот результат был получине Винером в 1942 году. Подробный вывод здесь приводить не будем. Функцией **S** здесь обозначаются энергетические спектры шума и исходного изображения соответственно – поскольку, эти величины редко бывают известны, то дробь **Sn / Sf** заменяют на некоторую константу **K**, которую можно приблизительно охарактеризовать как соотношение сигнал-шум.

Следующий метод, это «сглаживающая фильтрация методом наименьших квадратов со связью», другие названия: «фильтрация по Тихонову», «Тихоновская регуляризация». Его идея заключается в формулировке задачи в матричном виде с дальнейшем решением соответствующей задачи оптимизации. Это решение записывается в виде:

https://habrastorage.org/storage2/bbb/acf/473/bbbacf473cac4d25f7f1370a6180d7da.png (7)

Где y – параметр регуляризации, а P(u, v) – Фурье-преобразование оператора Лапласа (матрицы 3 \* 3). Как известно, классическая тихоновская регуляризация не позволяет качественно восстановить негладкое, т.е. недифференцируемое решение, т.к. приводит к заглаживанию тон кой структуры решения.

Еще один интересный подход предложили независимо Ричардосн [Richardson, 1972] и Люси [Lucy, 1974]. Метод так и называется «метод Люси-Ричардсона». Его отличительная особенность в том, что он является нелинейным, в отличие от первых трех – что потенциально может дать лучший результат. Вторая особенность – метод является итерационным, соответственно возникают трудности с критерием останова итераций. Основная идея состоит в использовании метода максимального правдоподобия для которого предполагается, что изображение подчиняется распределению Пуассона. Формулы для вычисления достаточно простые, без использования преобразования Фурье – все делается в пространственной области:

 (8)

Здесь символом «\*», как и раньше, обозначается операция свертки. Этот метод широко используется в программах для обработки астрономических фотографий – в них использование деконволюции (вместо unsharp mask, как в фоторедакторах) является стандартом де-факто. В качестве примера можно привести Astra Image, вот примеры деконволюции. Вычислительная сложность метода очень большая – обработка средней фотографии, в зависимости от количества итераций, может знанимать многие часы и даже дни.

Последний рассматриваемый метод, а вернее, целое семейство методов, которые сейчас активно разрабатываются и развиваются – это слепая деконволюция (blind deconvolution). Во всех предыдущих методах предполагалось, что искажающая функция PSF точно известна, в реальности это не так, обычно PSF известна лишь приблизительно по характеру видимых искажений. Слепая деконволюция как раз является попыткой учитывать это. Принцип достаточно простой, если не углубляться в детали – выбирается первое приближение PSF, далее по одному из методов делается деконволюция, после чего некоторым критерием определяется степень качества, на основе нее уточняется функция PSF и итерация повторяется до достижения нужного результата.

% Load image

I = im2double(imread('Khamukhin\_Photo.jpg'));

figure(1); imshow(I); title('Исходное изображение');

% Blur image

PSF = fspecial('disk', 15);

Blurred = imfilter(I, PSF,'circular','conv' );

% Add noise

noise\_mean = 0;

noise\_var = 0.00001;

Blurred = imnoise(Blurred, 'gaussian', noise\_mean, noise\_var);

figure(2); imshow(Blurred); title('Размытое изображение');

estimated\_nsr = noise\_var / var(Blurred(:));

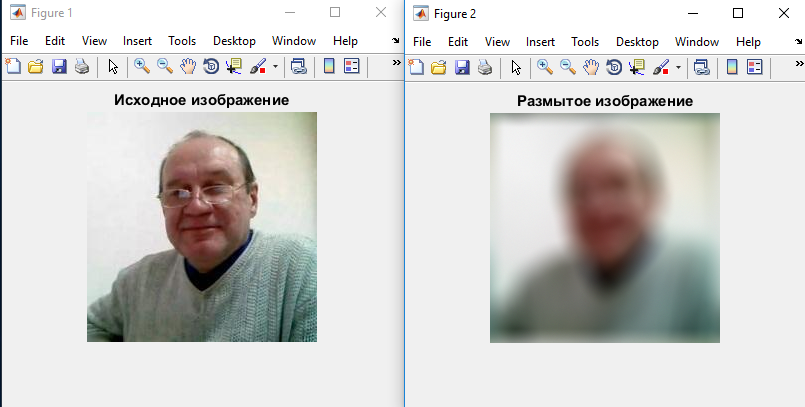
% Restore image

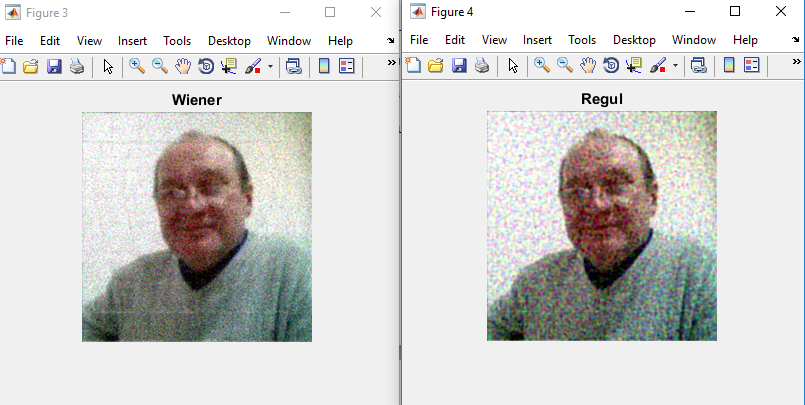
figure(3), imshow(deconvwnr(Blurred, PSF, estimated\_nsr)), title('Wiener');

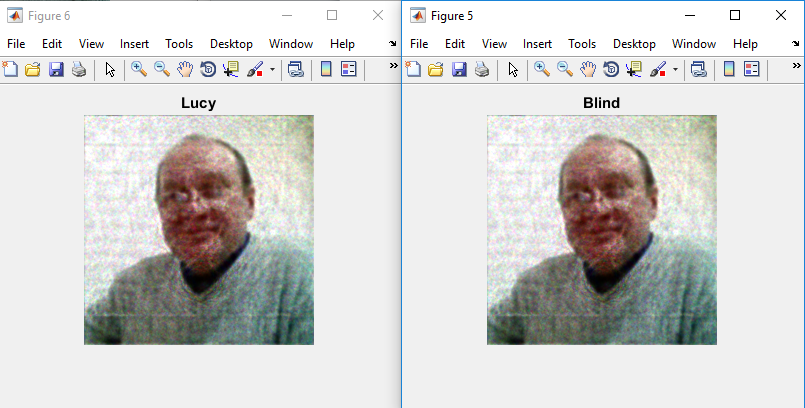
figure(4); imshow(deconvreg(Blurred, PSF)); title('Regul');

figure(5); imshow(deconvblind(Blurred, PSF, 100)); title('Blind');

figure(6); imshow(deconvlucy(Blurred, PSF, 100)); title('Lucy');







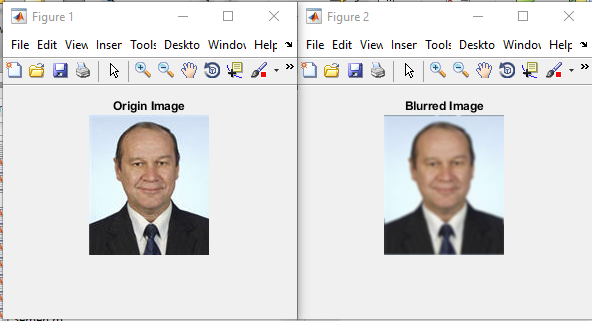
Main functions Основные функции

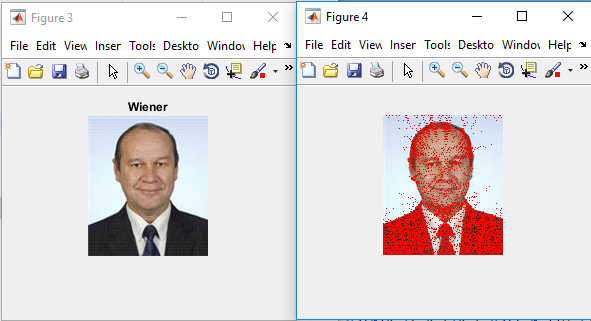
|  |  |
| --- | --- |
| J = deconvwnr(I,PSF,NSR) deconvolves image I using the Wiener filter algorithm, returning deblurred image J. Image I can be an N-dimensional array. PSF is the point-spread function with which I was convolved. NSR is the noise-to-signal power ratio of the additive noise. NSR can be a scalar or a spectral-domain array of the same size as I. Specifying 0 for the NSR is equivalent to creating an ideal inverse filter.  The algorithm is optimal in a sense of least mean square error between the estimated and the true images. | J = deconvwnr (I, PSF, NSR) восстанавливает из свертки изображение I, использую алгоритм фильтра Винера, возвращая очищенный образ J. Изображение I может быть N-мерным массивом. PSF - это функция точечного распространения, с которой I был свернут. NSR - отношение мощности шума к сигналу аддитивного шума. NSR может быть скалярным или массивом спектрального домена того же размера, что и I. Задание 0 для NSR эквивалентно созданию идеального обратного фильтра.  Алгоритм оптимален в смысле наименьшей среднеквадратической ошибки между оцененными и истинными изображениями. |
| J = deconvreg(I, PSF) deconvolves image I using the regularized filter algorithm, returning deblurred image J. The assumption is that the image I was created by convolving a true image with a point-spread function PSF and possibly by adding noise. The algorithm is a constrained optimum in the sense of least square error between the estimated and the true images under requirement of preserving image smoothness.  I can be a N-dimensional array | J = deconvreg(I, PSF) восстанавливает из свертки изображение I, использую алгоритм регуляризованного (по Тихонову) фильтра. Предполагается, что изображение I было создано путем свертки истинного изображения с помощью функции распределения по точкам PSF и, возможно, путем добавления шума. Алгоритм представляет собой ограниченный оптимум в смысле наименьшей квадратной ошибки между оцененными и истинными изображениями при необходимости сохранения гладкости изображения.  I может быть N-мерным массивом |
| [J,PSF] = deconvblind(I,INITPSF) deconvolves image I using the maximum likelihood algorithm, returning both the deblurred image J and a restored point-spread function PSF. The restored PSF is a positive array that is the same size as INITPSF, normalized so its sum adds up to 1. The PSF restoration is affected strongly by the size of the initial guess INITPSF and less by the values it contains. For this reason, specify an array of 1's as your INITPSF.  I can be an N-dimensional array.  To improve the restoration, deconvblind supports several optional parameters, described below. Use [] as a placeholder if you do not specify an intermediate parameter.  [J,PSF] = deconvblind(I, INITPSF, NUMIT) specifies the number of iterations (default is 10) | [J,PSF] = deconvblind(I,INITPSF) восстанавливает из свертки изображение I, использую алгоритм максимального правдоподобия, возвращая как дефлорированное изображение J, так и восстановленную функцию распределения по точкам PSF. Восстановленный PSF - это положительный массив с таким же размером, как и INITPSF, нормализованный, поэтому его сумма добавляет до 1. Восстановление PSF сильно зависит от размера исходного предположения INITPSF и меньше значений, которые он содержит. По этой причине укажите массив из 1 как ваш INITPSF.  [J, PSF] = deconvblind (I, INITPSF, NUMIT) указывает количество итераций (по умолчанию 10). |
| J = deconvlucy(I, PSF) restores image I that was degraded by convolution with a point-spread function PSF and possibly by additive noise. The algorithm is based on maximizing the likelihood of the resulting image J's being an instance of the original image I under Poisson statistics. | J = deconvlucy (I, PSF) восстанавливает изображение I, которое деградировалось сверткой с помощью функции разнесения по точкам PSF и, возможно, с помощью аддитивного шума. Алгоритм основан на максимизации вероятности того, что полученный образ J является экземпляром исходного изображения I по статистике Пуассона.NUMIT указывает количество итераций (по умолчанию 10). |

Preliminary functions Вспомогательные функции

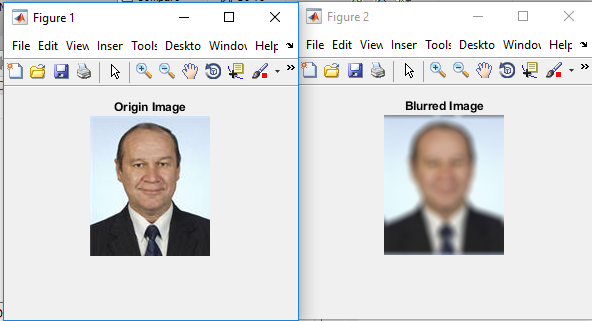
|  |  |
| --- | --- |
| Imread  Read image from graphics file. Read more "Help" about different file formats | Imread  Чтение изображения из графического файла. Подробнее «Help» о разных форматах файлов |
| im2double  I2 = im2double(I) converts the intensity image I to double precision, rescaling the data if necessary. I can be a grayscale intensity image, a truecolor image, or a binary image. If the input image is of class double, then the output image is identical. | im2double  I2 = im2double (I) преобразует изображение интенсивности I в двойную точность, при необходимости масштабируя данные. I можеть быть изображением интенсивности в оттенках серого, изображением truecolor или бинарным изображением. Если входное изображение имеет класс double, то выходное изображение идентично. |
| Imshow  Display image. Read more "Help" about arguments of this function. | Imshow  Показать изображение. Подробнее «Help» об аргументах этой функии. |
| Fspecial  h = fspecial(type) creates a two-dimensional filter h of the specified type. Some of the filter types have optional additional parameters, shown in the following syntaxes. fspecial returns h as a correlation kernel, which is the appropriate form to use with imfilter.  h = fspecial('disk',radius) returns a circular averaging filter (pillbox) within the square matrix of size 2\*radius+1. | Fspecial  h = fspecial (type) создает двумерный фильтр h указанного типа. Некоторые типы фильтров имеют дополнительные дополнительные параметры, показанные в следующих синтаксисах. fspecial возвращает h как ядро корреляции, которое является подходящей формой для использования с imfilter.  h = fspecial («диск», радиус) возвращает круговой фильтр усреднения (дота) в квадратной матрице размером 2 \* радиус + 1. |
| Imfilter  N-D filtering of multidimensional images. A neutral-density filter, or N-D filter, is a filter that reduces or modifies the intensity of all wavelengths, or colors, of light equally, giving no changes in hue of color rendition. | Imfilter  N-D фильтрация многомерных изображений. Фильтр с нейтральной плотностью или фильтр N-D - это фильтр, который уменьшает или изменяет интенсивность всех длин волн или цветов света одинаково, не изменяя оттенка цветопередачи. |
| Imnoise  Add noise to image. Read more "Help" about arguments of this function. | Imnoise  Добавить шум в изображение. Подробнее «Help» об аргументах этой функии. |
|  |  |

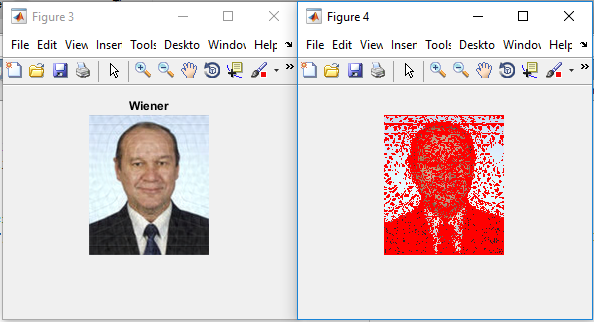
Попиксельное сравнение изображений



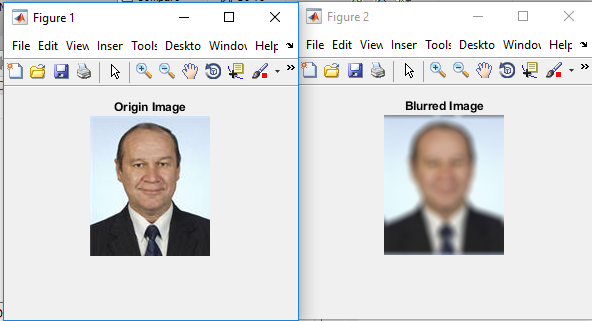


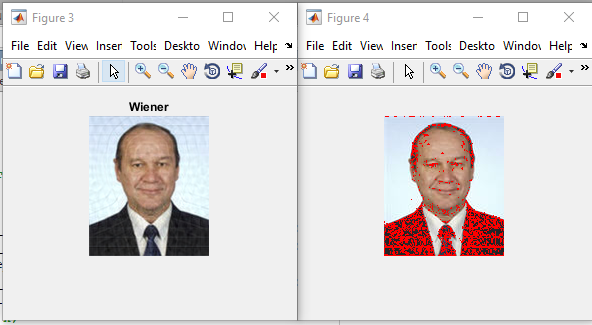
Справа – несовпадающие пиксели в пределах 2%, размытие 'disk', **2**





Справа – несовпадающие пиксели в пределах **2**%, размытие 'disk', **5**





Справа – несовпадающие пиксели в пределах **10**%, размытие 'disk', **5**

Program

clear; clc;

% Load image

IMG\_1= im2double(imread('61741.jpg')); % Read origin photo

figure(1); imshow(IMG\_1); title('Origin Image');

% Blur image

PSF = fspecial('disk', 5);% creates a two-dimensional filter

Blurred = imfilter(IMG\_1, PSF,'circular','conv' );% performs multidimensional filtering according to the specified options

% Add noise

noise\_mean = 0;

noise\_var = 0.00001;

%Blurred = imnoise(Blurred, 'gaussian', noise\_mean, noise\_var);% Add noise to image

figure(2); imshow(Blurred); title('Blurred Image');

estimated\_nsr = noise\_var / var(Blurred(:));

% Restore image by Wiener

figure(3); IMG\_2=deconvwnr(Blurred, PSF, estimated\_nsr);

imshow(IMG\_2);

title('Wiener');%reconstructs image I from convolution using the Wiener filter algorithm,

% compare images

%IMG\_2=IMG\_1;

IMG\_size\_y=size(IMG\_1,1);

IMG\_size\_x=size(IMG\_1,2);

IMG\_size\_y2=size(IMG\_2,1);

IMG\_size\_x2=size(IMG\_2,2);

%Разделим трехмерный массив изображения на двумерные массивы цветовых компонент RGB:

R\_1=IMG\_1(:,:,1);

G\_1=IMG\_1(:,:,2);

B\_1=IMG\_1(:,:,3);

R\_2 =IMG\_2 (:,:,1);

G\_2 =IMG\_2 (:,:,2);

B\_2 =IMG\_2 (:,:,3);

% Проиндексируем те пиксели, которые полностью одинаковы, т.е. те, у которых значения для всех цветовых компонент совпадают:

% зададим небольшое несовпадение 10%

ns=0.1;

ind\_R=uint32(find(((B\_1>=B\_2-B\_2\*ns)&(B\_1<=B\_2+B\_2\*ns)) & ((R\_1>=R\_2-R\_2\*ns)&(R\_1<=R\_2+R\_2\*ns)) & ((G\_1>=G\_2-G\_2\*ns)&(G\_1<=G\_2+G\_2\*ns))));

ind\_G=uint32(find(((B\_1>=B\_2-B\_2\*ns)&(B\_1<=B\_2+B\_2\*ns)) & ((R\_1>=R\_2-R\_2\*ns)&(R\_1<=R\_2+R\_2\*ns)) & ((G\_1>=G\_2-G\_2\*ns)&(G\_1<=G\_2+G\_2\*ns))));

ind\_G=ind\_G+IMG\_size\_y\*IMG\_size\_x;

ind\_B=uint32(find(((B\_1>=B\_2-B\_2\*ns)&(B\_1<=B\_2+B\_2\*ns)) & ((R\_1>=R\_2-R\_2\*ns)&(R\_1<=R\_2+R\_2\*ns)) & ((G\_1>=G\_2-G\_2\*ns)&(G\_1<=G\_2+G\_2\*ns))));

ind\_B=ind\_B+2\*IMG\_size\_y\*IMG\_size\_x;

%ind\_R=ind\_R+IMG\_size\_y\*IMG\_size\_x;

% Создадим новое изображение, в котором яркими точками будем показывать места неодинаковости пикселей:

IMG=IMG\_1;

% Определим цвет, который будет означать, что пиксели различаются.

IMG(:,:,1)=255;

IMG(:,:,2)=0;

IMG(:,:,3)=0;

%Места, где пиксели различны, будут выделяться красным цветом.

%Запишем одинаковые значения цветовых компонент на свои места.

IMG(ind\_R)=IMG\_1(ind\_R);

IMG(ind\_G)=IMG\_1(ind\_G);

IMG(ind\_B)=IMG\_1(ind\_B);

%Выведем на экран изображение, которое ясно показывает несовпадения пикселей.

figure;

imshow(IMG);

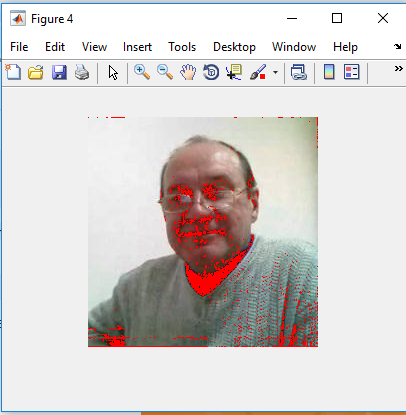
Попиксельное сравнение дает некоторую информацию о качестве восстановления (можно посчитать процент несовпадающих (в заданном диапазоне) пикселей от их общего количества) . Однако главный критерий – это возможность распознать восстановленное фото.

Задание. Подсчитать относительнле количество несовпадающих пикселей

PS. Sourse

<http://gr1g0ry.blogspot.com/2011/03/images-comparison.html>

|  |  |
| --- | --- |
| **"Find"**  indices and values of nonzero elements  If X is a multidimensional array, then find returns a column vector of the linear indices of the result.  X = [1 0 2; 0 1 1; 0 0 4]  k = find(X) | «Найти» индексы и значения ненулевых элементов  Если X - многомерный массив, то поиск возвращает вектор-столбец линейных индексов результата.  120\*140 = 16800  Это размер фото '61741.jpg'  Количество совпадающих пикселей ( в заданных пределах) это размер массива IND\_R  У фото 'Khamukhin\_photo.jpg' размер  230\*230 = 52900 |
| uint32  32-bit unsigned integer arrays  If you have an array of a different type, such as double or single, then you can convert that array to an array of type uint32 by using the uint32 function. | 32-разрядные целые массивы без знака  Если у вас есть массив другого типа, например double или single, то вы можете преобразовать этот массив в массив типа uint32 с помощью функции uint32. |
|  |  |
|  |  |



несовпадающие пиксели в пределах **10**%, размытие 'disk', **5**

s=size(ind\_R)/(IMG\_size\_x\*IMG\_size\_y)\*100;

fprintf('Процент совпадения пикселей заданных пределах %s\n ', s(1));

**2. Denoising by Neural network** **(pretrained)**

* Please insert the next text of program in Matlab and read description of new functions for you (use Help). Please study these functions by varying the function arguments and running this code.

% Load image

I = rgb2gray(imread('Khamukhin\_Photo.jpg')); % Read origin photo and transfer to grayscale

figure(1); imshow(I); title('Origin Image');

% Add noise

noise\_mean = 0;

noise\_var = 0.01;

Blurred = imnoise(I, 'gaussian', noise\_mean, noise\_var);% Add noise to image

figure(2); imshow(Blurred); title('Blurred Image');

% Restore image by

net = denoisingNetwork('dncnn'); %returns a pretrained image denoising deep neural

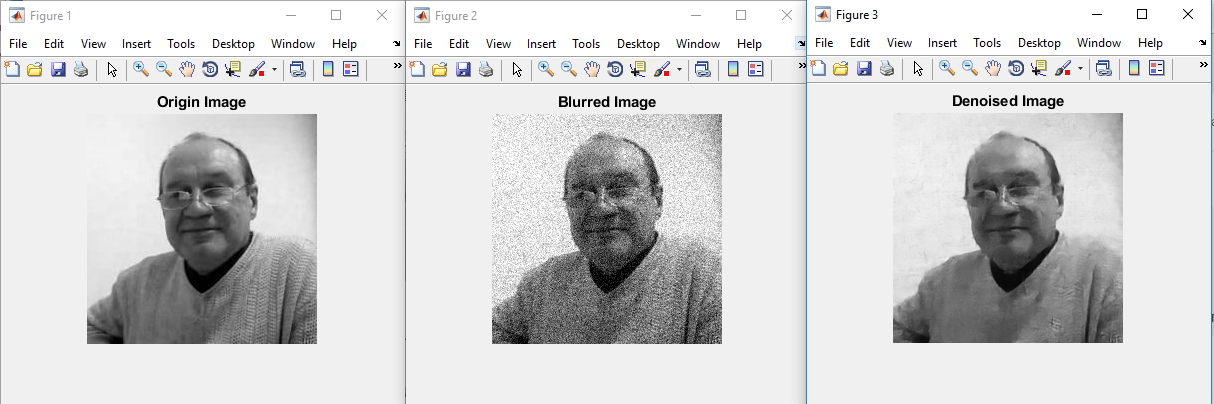
denoisedI=denoiseImage(Blurred,net);%estimates denoised image from noisy image using a pretrained denoising deep neural network specified by net

figure(3);

imshow(denoisedI);

title('Denoised Image');

The result should be



It was noise.

And now we check to restore blurred image

clear; clc;

% Load image

I = rgb2gray(imread('Khamukhin\_Photo.jpg')); % Read origin photo and transfer to grayscale

figure(1); imshow(I); title('Origin Image');

%Blur image

PSF = fspecial('disk', 10);% creates a two-dimensional filter

Blurred = imfilter(I, PSF,'circular','conv' );% performs multidimensional filtering according to the specified options

figure(2); imshow(Blurred); title('Blurred Image');

% Restore image by

net = denoisingNetwork('dncnn'); %returns a pretrained image denoising deep neural network specified by modelName.

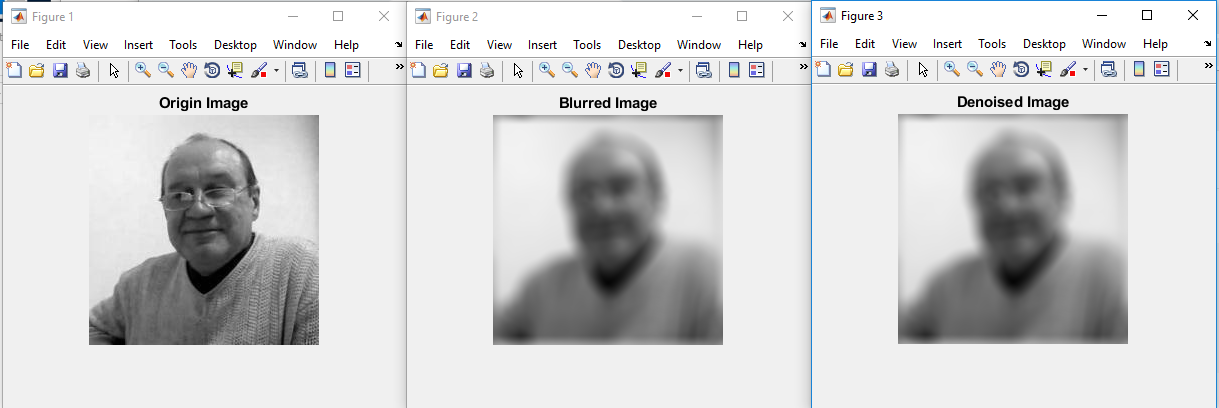
denoisedI=denoiseImage(Blurred,net);%estimates denoised image from noisy image using a pretrained denoising deep neural network specified by net

figure(3);

imshow(denoisedI);

title('Denoised Image');

Result



What will be the conclusion about this?