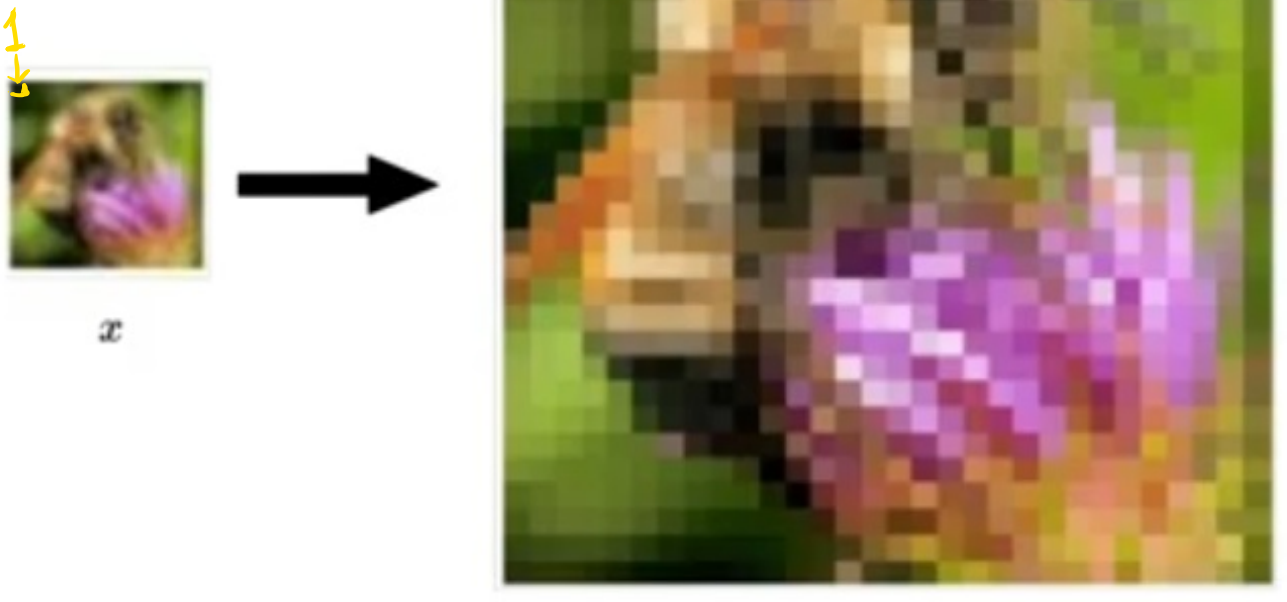
Interpolation or upsampling



(10 x 10) 10 x

Sub-sampling is one type of mechanim applied on the image.

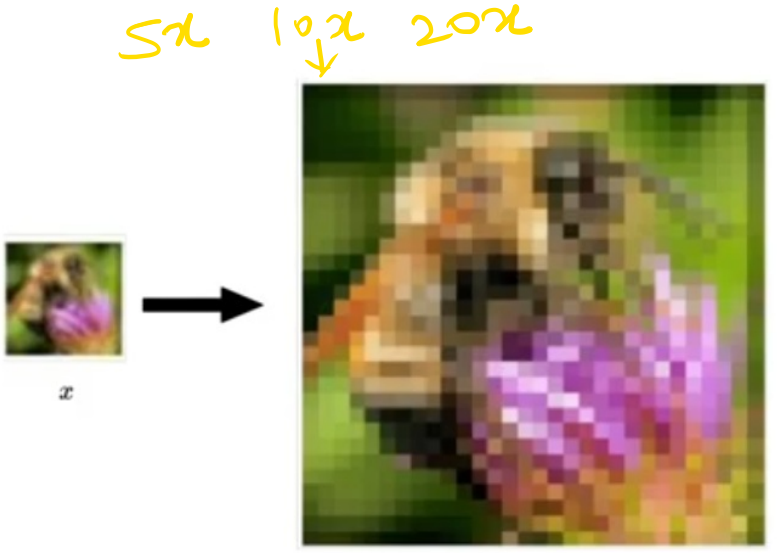
The other type of mechanism reverse approach is upsample.

Sometimes you convert a low-resolution image to a high resolution image. This can be performed with a standard interpolation operation.

A least linear interpolation is on the line joining those two points. That means any two values here is simply values that lie in the line connecting.

So the simplest form of interpolation that we can do, if we want to upsample an image is what is known as nearest neighbour interpolation.

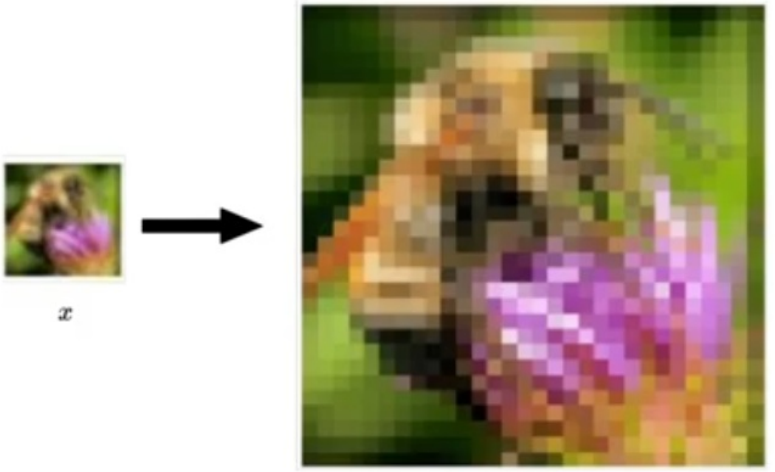
linear interpolation



Repeat each row and column 10 times. So every pixel, you repeat it 10 times along the column and 10 times along the row, so it almost becomes like one entire 10 by 10 block has exactly the same value.

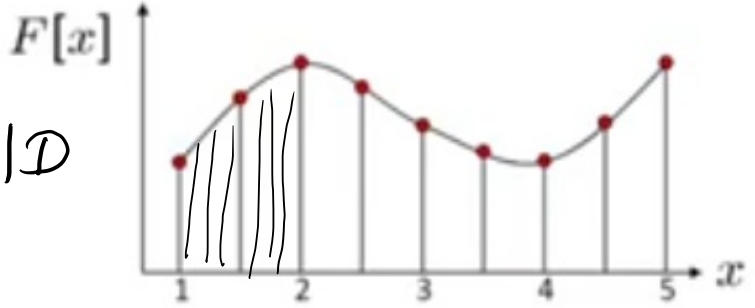
Then you take the next pixel repeat it for 10 columns, repeat it for 10 rows which means the next 10 by 10 block has the same value so and so forth.

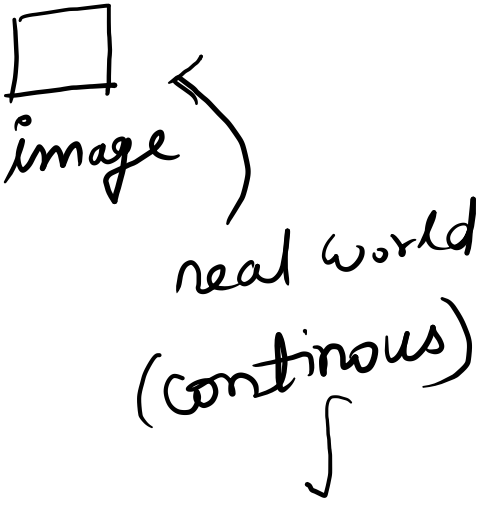
However, your resolution goes up. But you are going to have such artifacts across the image because you took several blocks, several pixels and simply expanded them into blocks.



How can we do better than this?

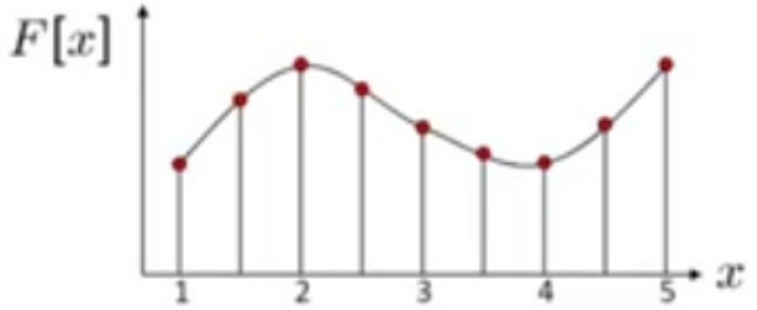
Clearly we do not want to upsample images like this and have a blocky effect in the upsampled image.





$F[x, y] = \text{quantize}\{f(x_d, y_d)\}$

original image



$$F[x,y] = \text{quantize}\{f(x_d, y_d)\}$$

It is a discrete point sampling of a continuous function.

An image being a discrete sampled version of the original continuous function or the original continuous signal.

So you have an original signal f, you quantize it at certain locations which gives you your image or your signal F [x, y] . In our case it is going to be an image.

So when we do interpolation, we are simply asking the question can we somehow reconstruct that original continuous function from the discrete samples.

That is the question that we are trying to ask because if we do that, we could interpolate and go to any resolution that we want to go for.

Unfortunately, in practice we do not know the small f which is your continuous function.

We do not know that continuous function, we do not know how, what was the real world seeing from which your image was captured in a sampled manner.

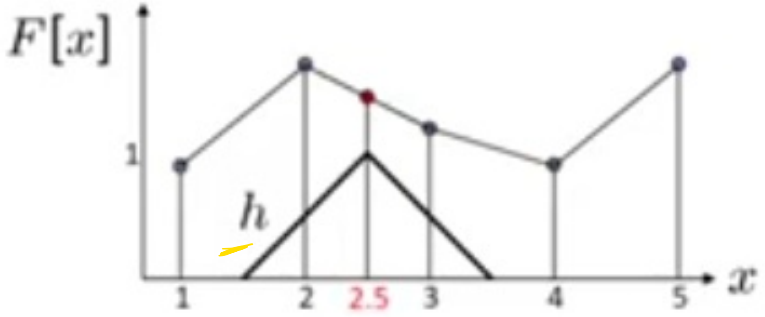
However, we can try to approximate this in a principled way and the way we are going to do that is through a convolution approach.

In the first step, we convert continuous function F which is discrete sample version to a continuous function.

Because we do not know the original continuous function we are going to define a pseudo continuous function which simply states that wherever you know the value, you simply put the images value there, everywhere else you just say it is 0.

This would be an automatic version of a continuous function.

This approximation you guess via filtering



$$f_F(x) = \begin{cases} F\left(\frac{x}{d}\right) & \text{if } \frac{x}{d} \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}$$

Reconstruct $\hat{f} = h * f_F$

We try to reconstruct $\hat{f}$ which is the resolution at which you want to take f_F to, by convolving this pseudo continuous image with some filter.

If we have this function, continuous function, f_F , can we get some kernel h in such a way that we can construct your original, not original but at least an upsampled signal?

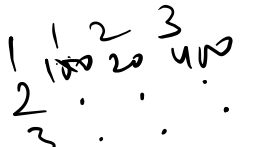
What do you think ‘h’ should be?

This image should give you some understanding, maybe a signal such as a tent, in a shape of a tent could help you with the 1D signal perhaps

But that still does not answer how convolution can take us from one resolution to a higher resolution because the way we saw it, convolution can maintain the same resolution if you include padding and if you do not pad, you probably will lose a few pixels on the boundaries.

But now we want to go to a higher resolution then we use convolution.

Interpolation as convolution



- To interpolate (or upsample) an image to a higher resolution, we need an interpolation kernel

original image ↓ , interpolation kernel ↓

$$g(i,j) = \sum_{k,l} f(k,l)\, h(i - rk,\ j - rl)$$

r ↓ upsampling rate $r = 10$, $r = 2$

Convolution formula

$$g(i,j) = \sum_{k,l} f(k,l)\, h(i - k,\ j - l)$$

To perform interpolation and then upsample an image to a higher resolution using convolution, we are going to do it using an interpolation kernel.

The main difference between two kernel expressions (interpolation and convolution), is the quantity r here, that is multiplying k and l, and r is what we call as the upsampling rate.

So if you want to double your image, your r is 2, if you want to quadruple your size of your image, remember we are doing up sampling, so if you want to quadruple the size of your image, r is going to be 4, so on and so forth.

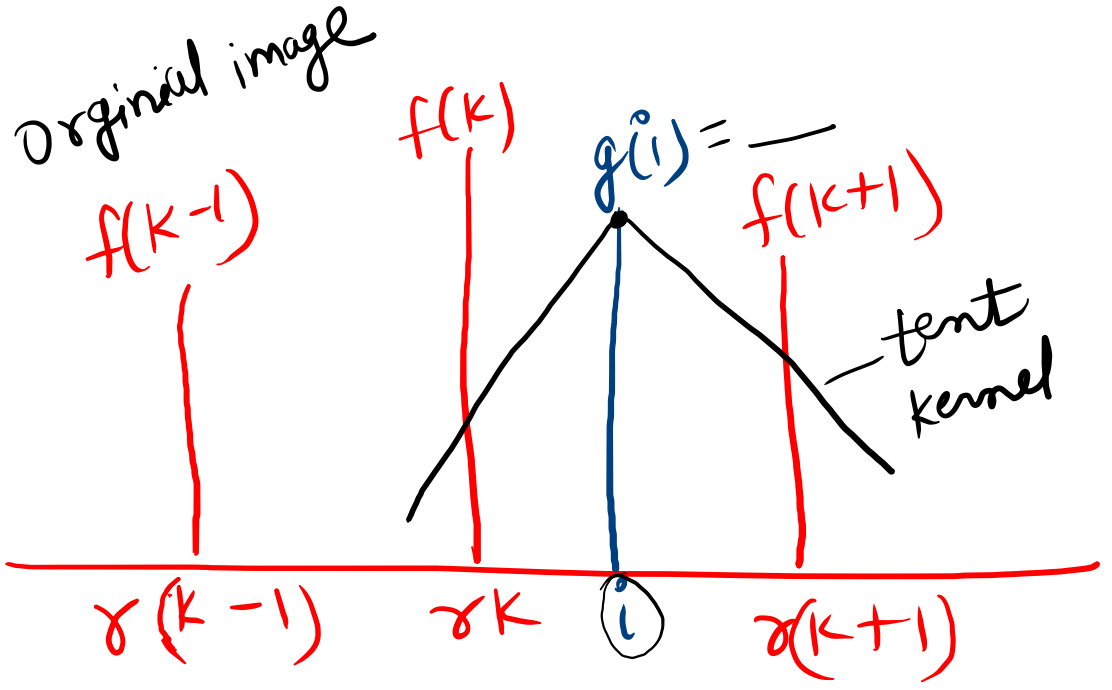
Let us try to understand how this actually works.

So let us consider an x axis, let us assume that there are some signals f(k-1), that is it, there is another signal f (k), there is another signal here f (k+1), so on and so forth.

The vertical lines represent the various values of f that I have with me, that is my input image which I want to increase the resolution.

If I take some values in between, what would be g(i)?

g is the output, f is the input. So in the g axis the tricky part here is that the coordinates, the indices in g are different from the indices in f, because f has only a certain number of indices whereas g has double those number of indices. So this would be rk-1 in g indices, this will be rk and this would be rk+1 in the indices of g.



h

	1	2	3
1			
2			
3			

3x3

$$g(i) = \sum_{k} f(k)\, h(i - \gamma k)$$

As you take many k values, so k can range from k-1 to k+1. If you take a 3x3 window or a higher depending on k simply defines the length of the window that you have.

So in this case, so what we need now is h(i - rk), the h was a one of the interpolation kernels, in this case it is called the tent kernel because it looks like a tent.

And what this kernel does is as you go further out from the central pixel on which you place the kernel, the effect of those pixels fall off linearly, not exponentially but linearly. So that is your tent kernel.

That is how this interpolation kernels work.

Complex kernels such as B-splines, those are one of the most complex kernels.

The first one is an effect of doing the nearest neighbour interpolation, where you have the blockyness effects.

If you do bilinear interpolation which is the linear interpolation that we spoke about but along the two axis and you get something like this

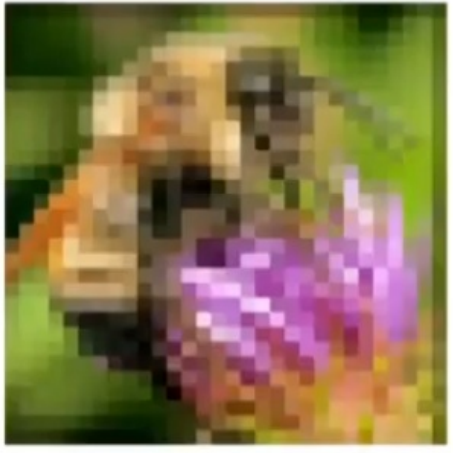
if you going to bicubic interpolation you get something like this, so on and so forth.

So the main difference in these interpolations is the h kernel that you use, in one case it is nearest neighbour, in the second case it is bilinear, third case is bicubic.

original image

Upsampled Images:

1



2

3

Left to right: Nearest Neightbour Interpolation, Bilinear Interpolation, Bicubic Interpolation.

Interpolation and Decimation

- To interpolate (or upsample) an image to a higher resolution, we need an interpolation kernel (r = upsampling rate)

$$g(i,j) = \sum_{k,l} f(k,l)\, h(i - rk,\ j - rl)$$

↑ $g(i,j)$

Decimation (sub-sampling)

- To decimate (or sub-sample) an image to a lower resolution, we need an decimation kernel with which to convolve the image (γ is downsampling rate)

$$g(i,j) = \sum_{k,l} f(k,l)\, h\left(i - \frac{k}{\gamma}, j - \frac{l}{\gamma}\right)$$

γ = downsampling

$\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$

So interpolation and decimation are the complementary operations.

Decimation means sub-sampling, you simply omit every other pixel where if you do Gaussian pre-filtering, you also ensure that you get a smoother output.

You can also achieve sub-sampling using a convolution like operation where it would be very similar, the only thing is instead of rk you would have k/r and l/r here because you now want to sub-sample and not interpolate.