Back

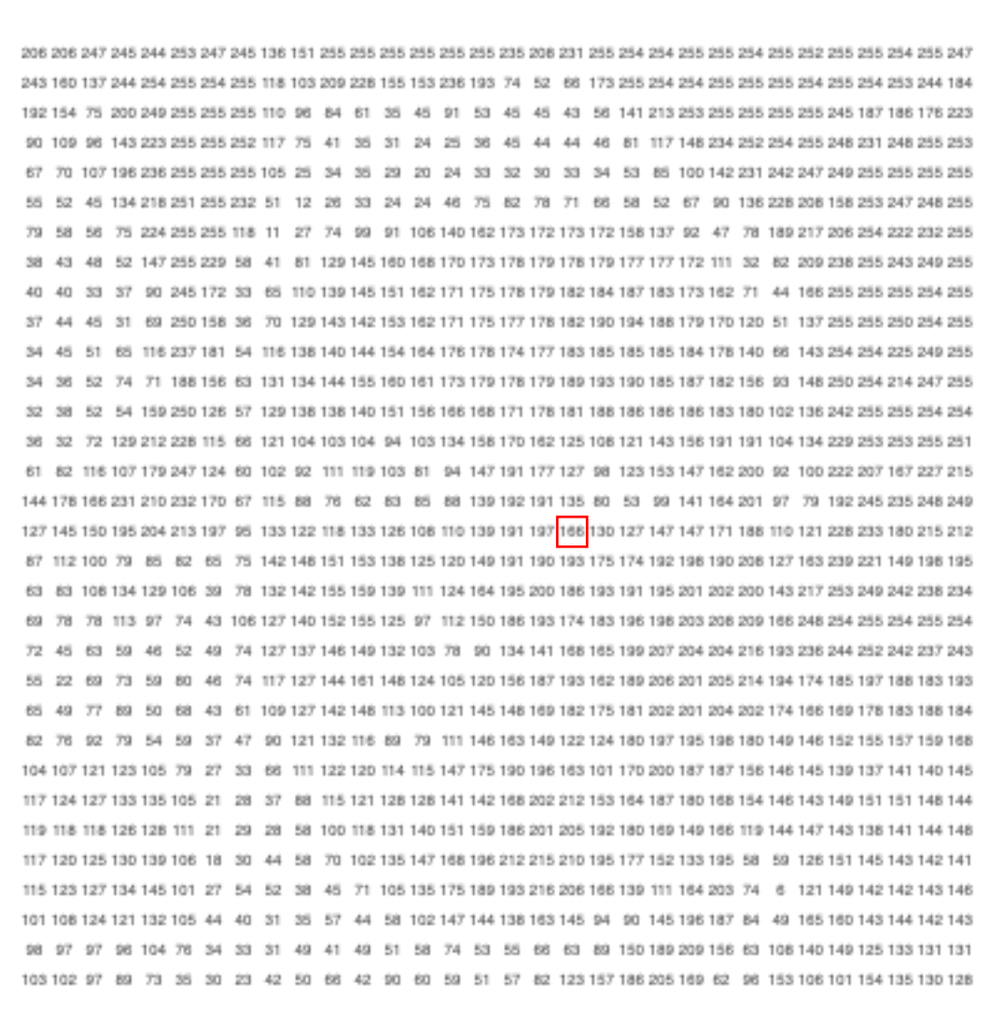
Image Kernels

Explained Visually

By Victor Powell

An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as "convolution" (see: convolutional neural networks.)

To see how they work, let's start by inspecting a black and white image. The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face. The large, granulated picture has been blown up to make it easier to see; the last image is the "real" size.

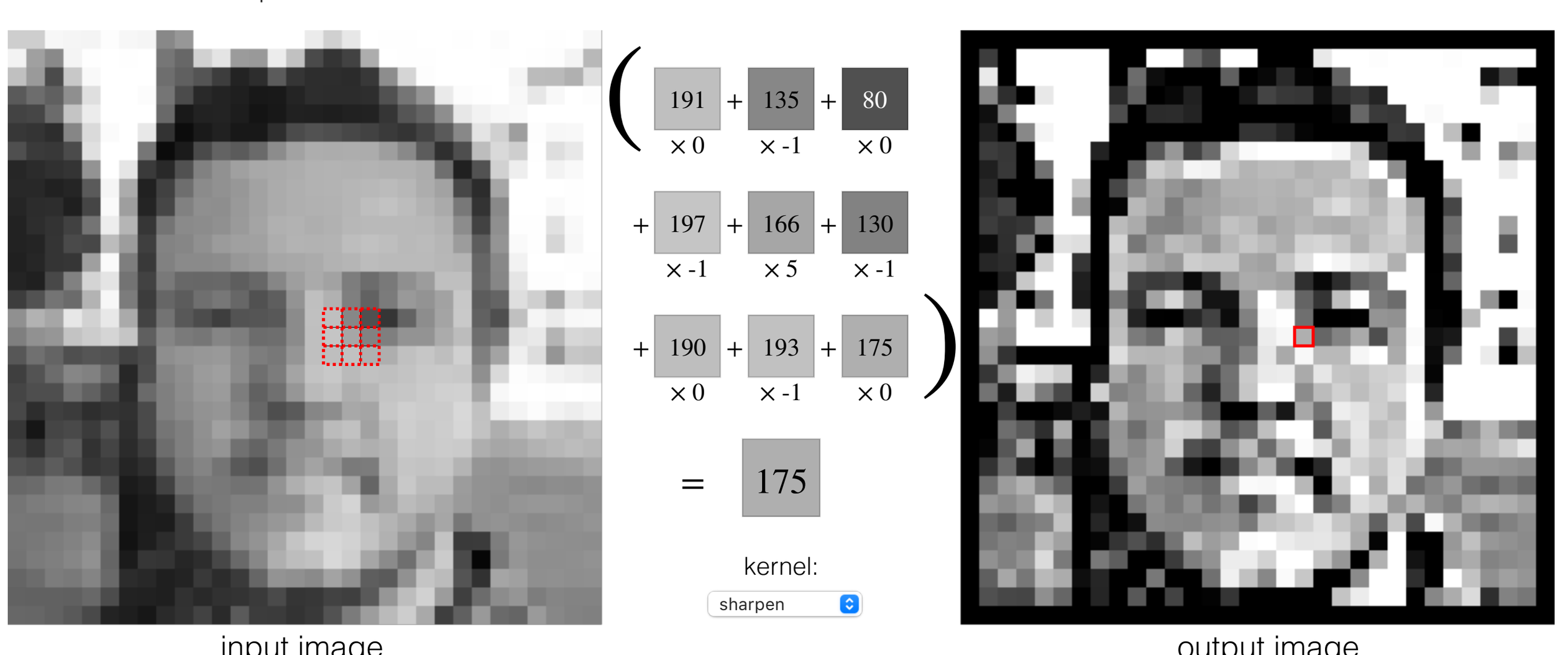


Let's walk through applying the following 3x3 **sharpen** kernel to the image of a face from above.

sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



input image | output image

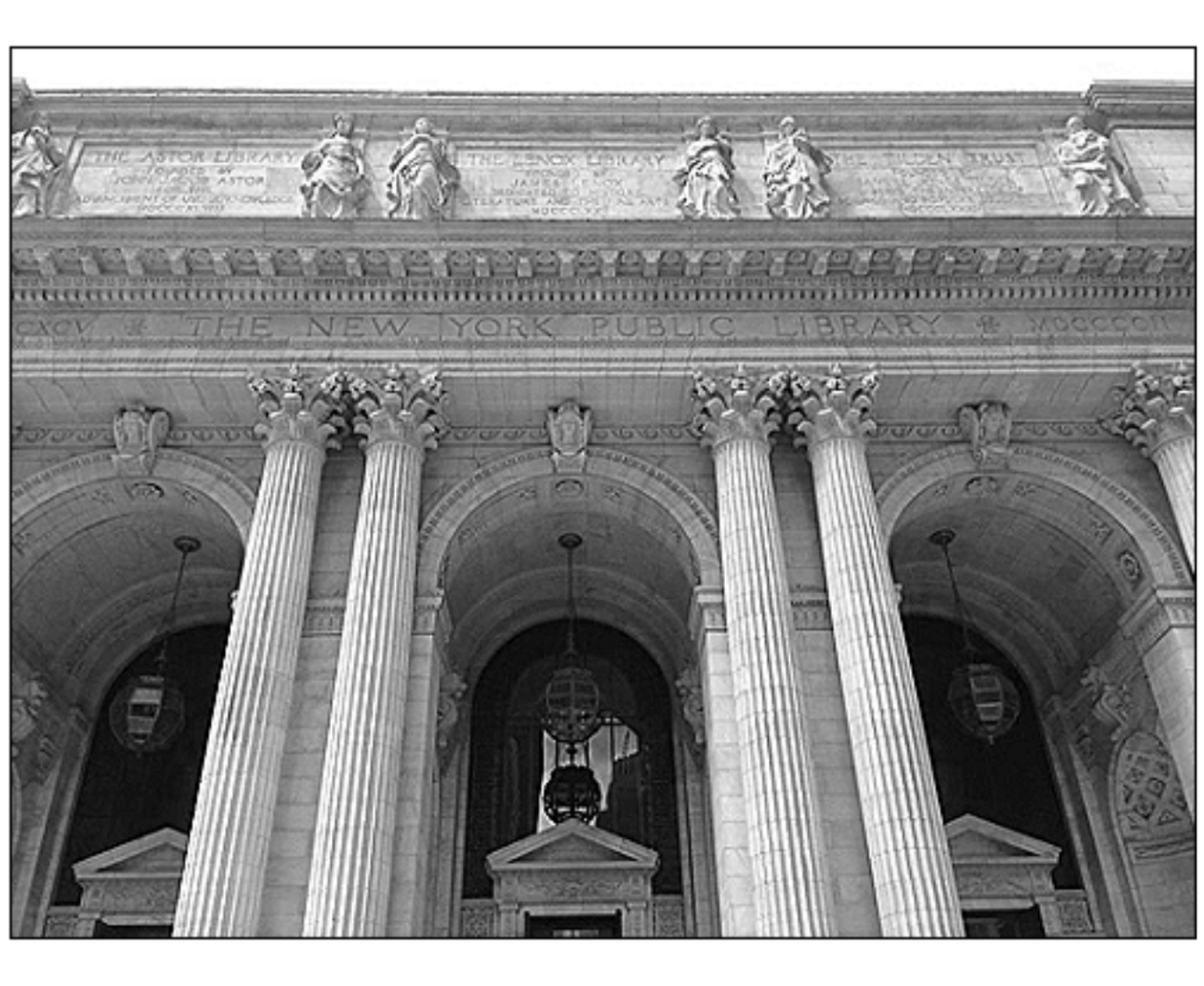
One subtlety of this process is what to do along the edges of the image. For example, the top left corner of the input image only has three neighbors. One way to fix this is to extend the edge values out by one in the original image while keeping our new image the same size. In this demo, we've instead ignored those values by making them black.

Here's a playground were you can select different kernel matrices and see how they effect the original image or build your own kernel. You can also upload your own image or use live video if your browser supports it.

Choose File no file selected

0	-1	0
-1	5	-1
0	-1	0

sharpen



The **sharpen** kernel emphasizes differences in adjacent pixel values. This makes the image look more vivid.

For more, have a look at Gimp's excellent documentation on using Image kernel's. You can also apply your own custom filters in Photoshop by going to Filter -> Other -> Custom...

For more explanations, visit the Explained Visually project homepage.

Or subscribe to our mailing list.

Email address | Subscribe

97 Comments Explained Visually Disqus' Privacy Policy 1 Login

Recommend 81 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

harveen · 3 years ago
Amazing Visualization. But one question, How do you come up with these kernel values?
37 | 3 · Reply · Share ›

Ong Beng Seong → harveen · 3 years ago
I don't know the full answer, but for some I can explain. For sharpen & Sobel for example, the goal is to see the edge better. So a difference operator (would be an approximation to derivative here) would help you see the change in neighboring pixels. So a partial derivative can be across or horizontal, so can the difference operator. For example, horizontal difference operator using central difference would be [1 0 -1]. In the case of Sobel, it differentiates and average, hence [1 2 1]^T * [1 0 -1] where [1 2 1] is an averaging operator with more emphasis in the center pixel. A second derivative, approximated with central difference would take on 5 star stencil , [[0 1, 0] [1 -4 1] [0 1 0]]. Similarly for 3x3 mean smoothing , you average all your neighbors, so it will be 3x3 matrix of 1's divided by 9. These then serve as a motivation for the shape/pattern of the kernel, you can then do something fancier, like [[0 1, 0] [1 -5 1] [0 1 0]] instead of [[0 1, 0] [1 -4 1] [0 1 0]] to emphasize mid pixel's difference to neighbor or a mean smoothing that is less affected by neighbors, so has kernel [[0.1 0.1, 0.1] [0.1 1 0.1] [0.1 0.1 0.1]] / 1.8. Others kernel probably has it's own logic and interpretation as well.
12 | · Reply · Share ›

Ali Panahi → harveen · 2 years ago
Gradient Descent will learn them through an iterative process of trying to change the kernel values so the final output of the neural network be as close to the target output.
1 | · Reply · Share ›

sina seyfi → Ali Panahi · 2 years ago
hello. excuse me. do you have CNN base code in Matlab for image compression?I'm industrial engineering, PHD student at Iran technology and science university and need to this Code.
5 | 5 · Reply · Share ›

Steve C. Miller → Ali Panahi · 7 months ago
But gradient descent works on the fully connected network. The convolutional layers are only feed forward
| · Reply · Share ›

Hadi Zand → harveen · 2 years ago · edited
For one case,
if you set all elements equal to 1/9, it gives sth like an averaging filter, and if we input a noisy image, the output would be smooth image (Denoising)
1 | · Reply · Share ›

sina seyfi → Hadi Zand · 2 years ago
hello. do you have CNN base code in Matlab for image compression?I'm industrial engineering, PHD student at Iran technology and science university and need to this Code.
| · Reply · Share ›

Sameen → harveen · 7 months ago
You use a CNN.
1 | 1 · Reply · Share ›

Redonnet → harveen · 3 years ago
One rule of thumb, if you want to play with the custom kernel, would be to always have the sum of coefficients sum to 1 if they are positive (as in the blur kernel), or to be between 0 and 1, to maintain a balance between the values and no end up all black, or all white.
| · Reply · Share ›

bikashg → harveen · 3 years ago
same question, someone please reply.
| · Reply · Share ›

Ong Beng Seong → bikashg · 3 years ago
see above
| · Reply · Share ›

Keeguon · 7 years ago · edited
Just as a side note, though image kernels are a good way to produce a blurring effect on pictures it's not really the most efficient way to do it on large kernels. If you want to achieve more complex blurs efficiently it's more efficient to use a Fourier transform.
6 | 1 · Reply · Share ›

Zz Tux → Keeguon · 5 years ago · edited
I prefer using firwin. Using one or more FIR filter works better than ifft.
After all a kernel is just a filter :D.
27 | · Reply · Share ›

tar van krieken → Keeguon · 6 years ago
well, simply running the same kernel over the output picture multiple times gives the desired effect aswell and is slightly more efficient than large kernels :)
| · Reply · Share ›

renan jegouzo → tar van krieken · 5 years ago
for the blur it's faster to do it with 2 passes, one horizontally, one vertically
| · Reply · Share ›

Adam Hancock → renan jegouzo · 5 years ago
And it's faster still to use FFT to perform the convolution of the blur kernel in the frequency domain
| 1 · Reply · Share ›

Chris → Adam Hancock · 2 years ago
No, it's faster to use an FIR filter in the image domain, since the filter kernel is much smaller than the image
| · Reply · Share ›

Christopher Silvia · 7 years ago
So how do Convolutional Neural Nets use these Kernels to detect features?
5 | 1 · Reply · Share ›

pwais → Christopher Silvia · 7 years ago
CNNs will learn a weighted combination of these kernels in order to match parts of objects (e.g. part of the wheel on a car). This is a nice visualization! Other helpful resources:
* See DeepViz: https://github.com/bruckner...
* See Layers >=2 plotted here: http://arxiv.org/pdf/1311.2...
* Another tool one can use to play with convolutional kernels is ShaderToy (https://www.shadertoy.com/)
8 | · Reply · Share ›

Rafael Espericueta → Christopher Silvia · 5 years ago
Filters can be designed to find edges, which are places in an image with maximum information, as an edge implies a spatial change. Areas of an image with the same color contain very little information. So filters are used in image processing (even by our own retinas!) to extract information from images.
1 | · Reply · Share ›

myJS → Rafael Espericueta · 3 years ago
Try the following in the custom kernel above for a good "find edges" kernel:

| -1 -1 -1 |
| -1 8 -1 |
| -1 -1 -1 |
| · Reply · Share ›

Tijn → myJS · 3 years ago · edited
If you want to also find the direction of that edge, you can use the canny edge detection for this.
1. Grayscale the image first.
2. Blur the image a bit with any blur kernel you want.
3. Now the difficult part. You have to use two kernels and store their data in two seperate numbers:
Result number A of Kernel 1 (Sobel) =
| 1 0 -1 |
| 2 0 -2 |
| 1 0 -1 |
Result number B of Kernel 2 (Sobel) =
| 1 2 1 |
| 0 0 0 |
| -1 -2 -1 |
4.1 (pixel power) To get the Pixel Color (power) of that (x,y) pixel spot you use: square root(Number A * Number A + Number B * Number B).
4.2 (pixel direction) To get the Direction of that (x,y) pixel spot you use: Atan2(Number A, Number B) * 180 / PI

------------(optional even more difficult part for the complete version of the canny edge detection)------------

see more
2 | · Reply · Share ›

Deepak Sadulla → Christopher Silvia · 4 years ago
See this through http://course.fast.ai/start...
| · Reply · Share ›

Gyeongho Kim · 9 months ago
Great visualization
1 | · Reply · Share ›

Hamza Ansari · 2 years ago · edited
it's a great for understaning the inside working of metrics operations ... but how we can train it as with NN
1 | · Reply · Share ›

Lorenzo Del Signore · 5 days ago
Hello what if I have not a 3x3 kernel? if I have different size of kernels what happens? for example 1x7 kernel etc
| · Reply · Share ›

Yousef Baradaran sadeghi · 2 months ago
nice work 👌
| · Reply · Share ›

Juli · 4 months ago
Hi, can someone explain what to do along the edges of the image? How can I extend the edge values out by one? Thank you!
| · Reply · Share ›

Robert Lugg → Juli · 4 months ago
There is no correct answer to your question. First, the problem and the image boundary is addressed by "padding". There are several ways to pad. The first, most simple is just use zeros. If you have a kernel that is 7x7, then you make the image 3 pixels bigger. On those extra pixels, you set the value to zero. As you can imagine, the accuracy will be bad there. You can also do fancier value setting like using the outer pixels, or mirroring the edge pixels. The other type of padding is called "none", which means the outer pixels are not determined, thus the output image is 3 pixels (per edge) smaller than the input image. Please see this https://www.tensorflow.org/... and https://machinelearningmast... for nice information on padding. All machine learning libraries and image processing libraries should have pad operations.
| · Reply · Share ›

I write on things i like · 6 months ago
This is an awesome website to be at. Thank you for making this.
| · Reply · Share ›

Steve C. Miller · 7 months ago
Best visual explanation ever.
| · Reply · Share ›

Muhammad Ahmed · 10 months ago
Nice one. Can somebody tell me why image sizes are reduced when convolution is performed without padding?
| · Reply · Share ›

Jayden Stuckey → Muhammad Ahmed · 7 months ago
Let's say you are convolving on input image pixel (x, y) = (1, 1) (where the centre of the kernel, lets use a 3x3 kernel in this case, is required to be over pixel (1,1)). This would mean that the edge of the kernel would be hanging off the side of the image and the element wise multiplication between the kernel and the pixels of the input image would be impossible. To work around this issue, convolving the image often takes place from pixel (x,y) = (2,2) such that each entry of the kernel matrix is 'hovering' over the top of an entry from the input image. The end result of this is an output image with dimensions (n-1) and (m-1) where n is the height, and m is the width. A way to keep the original image size is to pad the image by replicating the exterior pixels or placing a layer of 1's or 0's.
| · Reply · Share ›

learner · a year ago · edited
Hi! Thanks for the visualization, you help me a lot.
I want to ask about the output pixel value in the convolution. If the output pixel >255 or <0 is it replaced by 255 or 0? Or are you doing normalization for the output?
| · Reply · Share ›

Robert Lugg → learner · 4 months ago
If the convolution happens in a neural network, then the output is typically filtered by an "activation" function. Using a function like tanh restricts output values to a range (such as -1 to 1). Other activations like relu have no such limit and so the output values can grow infinitely big. Usually there is other "normalization" method to keep the values controlled. For images, I guess cropping as you say is best. Otherwise, the overall brightness would be distorted if normalized.
| · Reply · Share ›

Thomas · a year ago
Awesome. This is the best, most straight-forward intuition-building kernel explanation I have ever seen. Very impressed.
| · Reply · Share ›

Eduardo Moya · a year ago
This was very helpful! Thank you!
| · Reply · Share ›

Hang_A_Guy · 2 years ago
wouldn't combining 9 pixel to one pixel shrink the height and weight of the image??
| · Reply · Share ›

Robert Lugg → Hang_A_Guy · 4 months ago
yes, it would reduce the image size by 1 pixel on all sides, unless another method is used to compensate, which is called "padding"
| · Reply · Share ›

Maged · 2 years ago
The kernal presented is 3 x 3, is their a way to increase the kernal size to 6 x 6 for example ?
| · Reply · Share ›

rwp · 2 years ago
Is it possible to get the source for "Image Kernels" page?
| · Reply · Share ›

physics physics · 2 years ago
there's a misspelling
| · Reply · Share ›

bash · 2 years ago
Thanks a lot for this amazing visualization!
This helped me get a clearer understanding.
| · Reply · Share ›

Sarthak Banerjee · 2 years ago
does that mean the output image resolution will be decreased?
| · Reply · Share ›

Nishant Mandavkar → Sarthak Banerjee · 2 years ago
No, as you could see above, for every pixel we have calculated a new, replacement value. We have not dropped or added any pixels. This is my assumption for given transformations.
| · Reply · Share ›

david zu · 2 years ago
Genius.
| · Reply · Share ›

Angga · 2 years ago
hi, when i try to implement image kernel in python. why its slower than your playground? are you use c++?
| · Reply · Share ›

justanotherboob · 2 years ago
thank you for a GREAT explanation. Does it make sense to think about an image kernel for convolution whose values actually depend on the image? For example, instead of having a 'sharpen' kernel
010
141
010

change the value in the center to be a number between 1 and 5 depending on the brightness of the pixel which is two places to the left?

The reason I ask this is because I've noticed that when I look at an image, I actually look at 'neighborhoods'. not just individual pixels. While bright spots might catch my attention. I recognize parts of an image before it 'clicks'. So if I am using some parts of an image to inform me about the 'neighborhood' that part lives in, I am wondering if this sort of 'dynamic kernel' could produce a matrix that informs us more meaningfully about the image. I am sorry to bother you but am wondering if you have thought about this
| · Reply · Share ›

Weega Week · 2 years ago
if you constantly feed the image back through the kernal you could make a cellular autometon
| · Reply · Share ›

Jhonarendra · 2 years ago
Wow amazing..
| · Reply · Share ›

Load more comments

Subscribe · Add Disqus to your site · Do Not Sell My Data · DISQUS