**Convolutional Neural Network**

Convolutions are kind of like wavelet transforms. It’s one way to reduce the dimensionality of a data set, but also extract the import features of the (image) data set.

**Convolutions**

So say we wanted to classify an image as being either an X or an O.

A picture containing square, rectangle, pattern, tile

Description automatically generated A black and white checkered pattern

Description automatically generated with low confidence

Since the image is 6×6 = 36 pixels, we could construct and train a classification neural network with 36 input nodes. Each node would take on the value black or white. In a real gray scale image, each node would have a pixel value between 0 and 255. We’d scale them so that the min/max values are -1/1 though. So for instance, these two pictures would be:

A black and white crossword puzzle

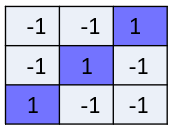
Description automatically generated A black and white crossword puzzle

Description automatically generated

And we could connect the 36 input nodes to some variety of internal hidden layers, and an output node(s) to figure out the identity of a given image. But for larger images, with thousands/millions of pixels, this brute force method would be too unwieldy. But to distinguish an O from an X, we don’t need all that information anyway. This is kind of like how we can use thermodynamics to effectively describe a system with a reduced set of variables (S, E, V, N), rather than recording the positions and momenta of all N ~ 1023 particles. Seems to me that we could describe these things with a reduced set of variables comprising, say, various moments of the pixel density distribution function. But instead, what we do is this:

**1**. **Convolution w/ Filter**

First we take the convolution of the image with a given filter/kernel, like this one for example.



We start by overlaying the filter, F, on top of the image, I, in the upper left corner. And we calculate a number to determine how well the filter matches the part of the image which it overlays. We can could add up all the times the filter and image had the same pixel values. But instead we just take a straight ‘dot’ product between the filter values and pixel values. I think this is called the ‘feature’.



If we have (r,g,b) values for each pixel, instead of just a single grayscale number, then I’m guessing we’d scale r, g, b each to the range (-1,1), and then take the dot product between the Image pixel vector and Filter pixel vector.



where **F**ij = (rij, bij, gih), and **I**ij = (rij, bij, gij). But not sure. Actually, I read (Jason Brownlee website) that it’s not necessarily a straight dot product **F**·**I** = F1I1 + F2I2 + F3I3, but some linear combination, **F**·**I** = w1F1I1 + w2F2I2 + w3F3I3. Apparently these weights are learned as part of the process of minimizing the loss. Moving on…And we see that f will be maximized when the feature is present in the filter. So for instance, the first four overlaps, below,

A picture containing square, rectangle, tile, mosaic

Description automatically generated A picture containing square, rectangle, tile

Description automatically generated A picture containing square, rectangle, tile, mosaic

Description automatically generated A black and white crossword puzzle

Description automatically generated with low confidence

would have the value (shortcut: white = dark blue = 1, light blue = grey = -1): 1, -3, 1, 9, respectively. The next row would look like,

A picture containing square, rectangle, colorfulness, tile

Description automatically generated A picture containing square, rectangle, colorfulness, tile

Description automatically generated A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, tile, mosaic

Description automatically generated

and have values -3, -3, 3, 1, respectively. And the next row would be:

A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, pattern, tile

Description automatically generated

with values, 1, 5, -3, -3. And the last row would be:

A picture containing square, rectangle, line, pattern

Description automatically generated A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, pattern, line

Description automatically generated A picture containing square, rectangle, tile

Description automatically generated

with values 9, 1, -3, 1. We can put all our values in a so-called *Feature Map*.

A grid of numbers with black text

Description automatically generated

But we’d want to normalize it by dividing everything by are max value, 9. So



So the closer it is to 1, the more the filter overlaps with the image in that region. The closer it is to -1, the less the filter overlaps with the image in that region. Can see that it overlaps quite strongly at the bottom left and top right corners. This makes sense. And in general it overlaps pretty well along that bottom left to top right diagonal. This is indeed appropriate for the character X.

**2. Pool the Feature Map Values**

Finally, we don’t want to deal with even this many inputs. So looks like we take our Feature Map, and break it down into sections. If we do 2×2 matrices, then this is called Stride = 2.

A grid of numbers and symbols

Description automatically generated

And either replace the sections with their max value (Max Pooling) or mean value (Mean Pooling). For instance, Max Pooling would give us:

A white rectangular box with black numbers

Description automatically generated

and Mean Pooling would be (if I’m mathing right):

A white rectangular box with black numbers

Description automatically generated

And so then we’d use this 2×2 matrix as input instead of the original 6×6 matrix.

**3. Repeat for other Filters**

Typically, you would then repeat this process for lots of other filters. You could have a diagonal filter going the other way. And you could have an O filter. Or a vertical line filter, or horizontal line filter. Each of filters would ‘detect’ different features of the image.

**4. Concatenate**

You could then unravel all these pooled feature matrices and glue them together to get a single long feature string. This should contain a lot of relevant information about the image, while not being as verbose as the image itself. One of the nice things about the convolutional approach is that if you shift the image up/down or right/left, that will basically shift the feature map values in the same direction. And will by and large leave the reduced Max/Mean Pooled feature map the same as before. And so our learning algorithm will be insensitive to simple displacements of the image, as it should be. I wonder if there’s something you can do that would make it invariant to rotations too.