# Convolution & Pooling The Way I Understand

Many of you would have heard of Convolutional Neural Networks also known as CNN or ConvNet. It is a class of deep neural networks used for computer vision projects. Through this post I wish to share my understanding of two core building blocks of CNN which are Convolution and Pooling. This post is for people who recently stepped into this technology and are wondering what these two things really are. As I did for a long time. Many of resource that I referred to understand this technology did not really explain these two terms enough only to leave me with more questions than answers. Again and again I only saw two matrices of numbers where the larger matrix represented an image also known as input tensor and smaller matrix called a kernel and this kernel would span the larger matrix (convolve) with the numbers getting multiplied and summed up. The calculated value would be shown as stored in another matrix called feature map. For e.g. in the image below.

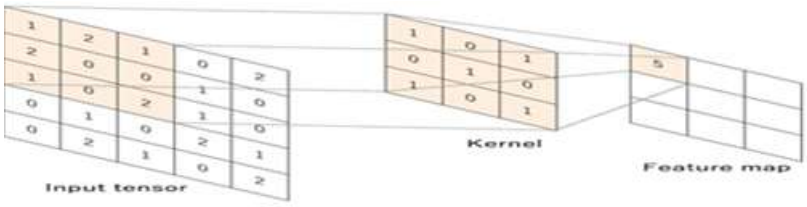



fig1

The process of the kernel matrix convolving with the input tensor matrix is called convolution and some sources did make an attempt to explain what convolution is using the mathematical approach. They usually refer to what happens in signal processing when two signals are mixed to give effect such as echoing in a song. The mathematical expression would look something like shown below.

$$(f * g_N)[n] \equiv \sum_{m=0}^{N-1} \left( \sum_{k=-\infty}^{\infty} f[m + kN] \right) g_N[n - m]$$

Source: https://en.wikipedia.org/wiki/Convolution

The equation would be preceded by various mathematical steps containing more integrals and other derivations. The best part is, in some cases they would also mention that the convolution in CNN is strictly not the same as it is known in signal processing!!! Then why did I go thru all this? After many fruitless explorations and resulting frustrations I went back to the basics. I realised that my struggle with CNN was more to do with the lack of basics than the quality of the materials available.

This decision helped me finally overcome the challenges in understanding the CNN and today am in a position to share with others what I learnt on the way to understanding the CNN. Assuming you have similar background as I i.e. no prior exposure to signal processing and hence no idea of what that equation means, I will keep this post at conceptual level. The mathematics behind the concept is not difficult at all. If you are keen on the mathematics, I suggest you read about feature extraction in image processing.

This post is by no way a comprehensive coverage on CNN. The main objective is to explain the two core concept of Convolution and Pooling.

To understand the concepts, we need to be clear on some building blocks of which I have listed only a few here that are directly related to our objective. I cover many more in my class before I start explaining the CNN. Including all in this post will make it too long and hence leaving them out.

1. Digital Image (grey scale and colour)
2. Digital Image as a function
3. Edges as image features
4. Convolution for detecting the edges
5. Kernels for convolving
6. Digital noise the villain
7. Denoising digital images
8. Pooling the features

## Digital Image

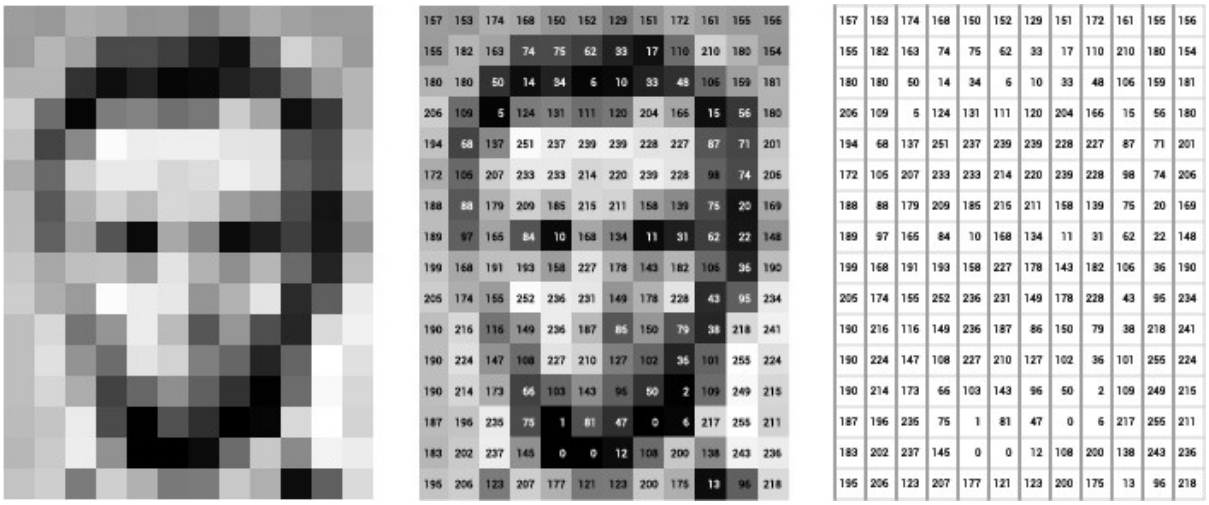
A digital image is a discrete representation of the real world which is analogue in nature. When we point our digital camera at a beautiful scene such as in the image below (Sunset in Coorg) and click a picture, the resulting digital image is a discrete representation of the analogue scene.


Fig 2

The reason why it is a discretized representation is that the resulting picture of the camera consists of a grid of tiny dot called pixels. Assume that the white grid of cells shown above in the camera represent the grid of pixels. Each cell being one pixel (in reality pixel is much smaller than this).

Each pixel has certain degree of brightness reflecting the amount of light captured by the camera at that location (of the pixel). The degree of brightness is measured in range of 0 to 255 (only integer values). A value of 0 indicates absolute dark/black and 255 indicates absolute white. Depending on the amount of light at each pixel location, their brightness will range from 0 to 255.

All digital images are represented by a grid of pixel intensity numbers. What we see as human recognizable is the output of plotting libraries such as matplotlib. What computers see is numbers. For example in the picture below. The one on the left is what humans see. But this is the result of a plotting library which took as input a gird of numbers were numbers correspond to pixel intensity (middle picture). What is actually stored on the system is a grid of numbers.



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |


| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |


Fig 3

Image source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html (beautiful introduction to computer vision)

Any colour is a combination of three primary colours the Red, Blue and Green (RGB). Each color digital image pixel is made of three colour pixels of RGB. The ratio of the intensities of the RGB pixels decide the colour of the pixel in the digital image.

In this post, I am going to refer only to grey scale image. Pl. refer to http://olympus.magnet.fsu.edu/primer/digitalimaging/cmosimagesensors.html for further information on colour images.

In a grey scale digital image, each pixel can take a value between 0 and 255, depending on the degree of brightness. Values between 0 and 255 reflect the degree of greyness. For the basic computer vision tasks such as classification, object detection etc., grey scale images usually suffice. Grey scale image take less storage space and processing time compared to coloured images. Hence, we often end up using grey scale image in computer vision tasks.

## Image as a function

Images are often known as complex functions. What does that mean? To understand that we first need to understand what a function is? The word “function” is borrowed from the world of physics and maths. In those fields a function represents the relation between input and output expressed in a

mathematical equation form or geometrically as a line, surface or hypersurface for e.g. in the picture below, y = f(x1, x2)

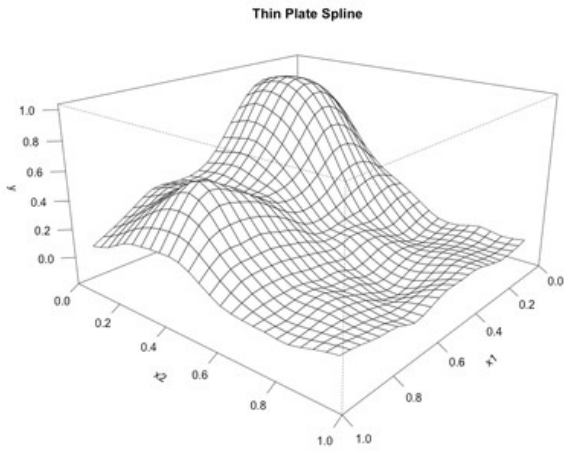



Fig 4

A digital image too can be represented as a function. Keep the two original dimension of the image as it is (height and width) and add a third dimensions orthogonal to the two. Let the third dimensions reflect the intensity of the pixel. We thus can transform a digital image into a function as shown below,


Fig 5

The greyscale image of camera man with X1 and X2 is shown in isometric view (middle one) which is subsequently transformed to three dimensional picture with the third dimension being pixel intensity in each combination of X1 and X2.

Code for the conversion from 2D grey scale to 3D function is given in the box below.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from skimage.transform import resize



cat =  resize(image,(100, 100) )

# create the x and y coordinate arrays (here we just use pixel indices)
xx, yy = np.mgrid[0:cat.shape[0], 0:cat.shape[1]]

# create the figure
fig = plt.figure(figsize= [10,10])
ax = fig.gca(projection='3d')
ax.plot_surface(xx, yy, cat ,rstride=1, cstride=1, cmap=plt.cm.gray,
      linewidth=0)
```

Let us look at the original greyscale image and compare it with the 3D image function.

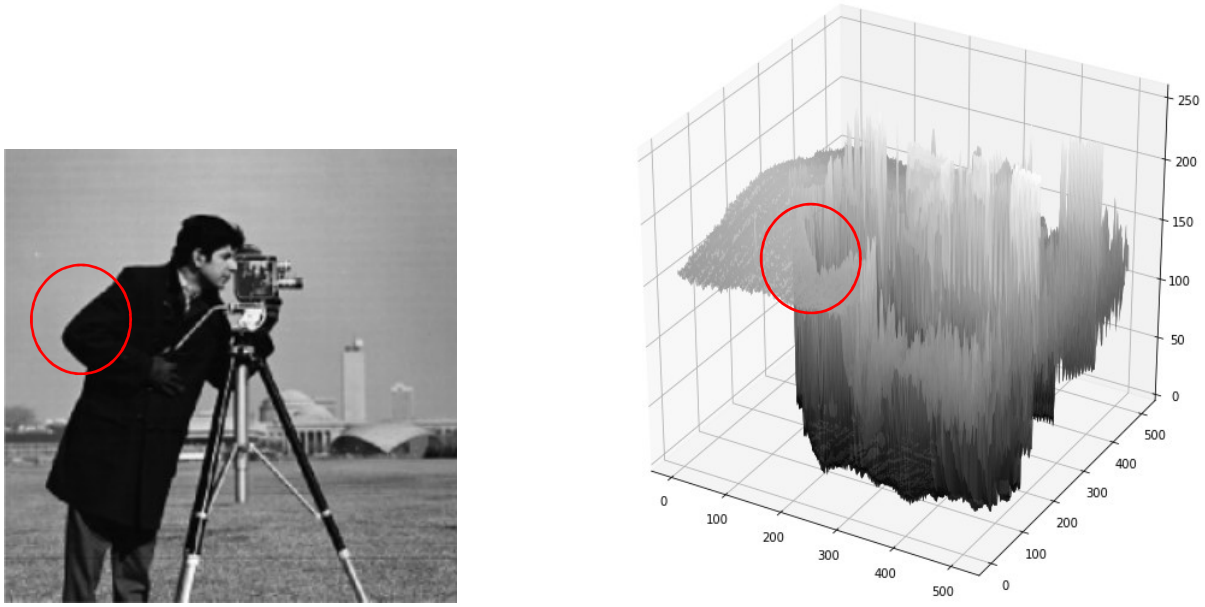



Fig 6

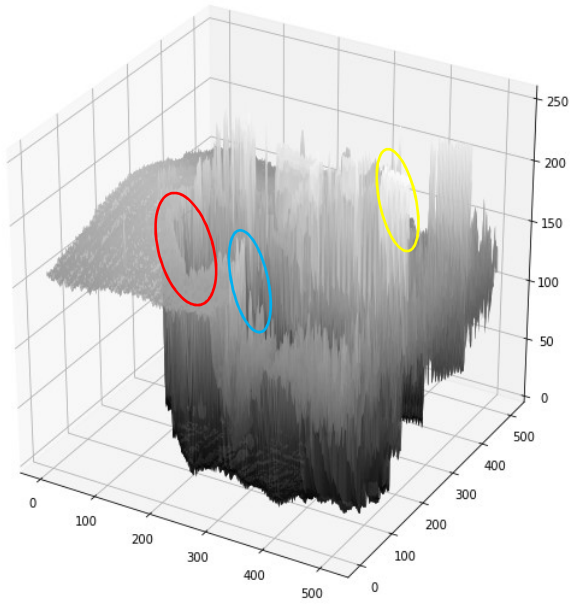
Observe the two carefully. You will notice that wherever the pixels are dark/black in the 2D picture, the corresponding pixels in 3D are on the floor. 2D pixels which are relatively bright such as the base of the camera where the handle is fixed or the legs of the tripod or the face of the person etc, corresponding 3D pixels have a high intensity (white spikes). The sky appears as greyish background.

Trace the elbow of the cameraman in 2D, you will notice a corresponding edge where the pixel intensity changes suddenly from grey to black in the 3D. This comparison is shown with circles.

The 3D view is a function representing the relation between pixel intensity (y) and input pixel positions (X1 and X2). This function has many sharp peaks and valleys, hence it is classified as a complex function (functions which are not straight or linear).

## Edges as Image Features

The X1,X2 positions in the 3D space where pixel intensities change drastically from close to black to some degree of grey or white appear as sudden fall in the surface. For e.g. when we look at the background sky (where pixel intensity is high (close to 200) approach the black coat of the person, the pixel intensity drops drastically to 0. Wherever the pixel intensities change drastically, there we see the edges. Edges that define the camera man and the camera!




1. Red – elbow
2. Coat
3. Tripod leg

Fig 7

The three circles show the coordinates where pixel intensity drastically changes reflecting presence of an edge. These edges belong to various aspects of the camera man of the digital image. The camera man gets a distinct boundary at overall level. Further, within the boundary there may be sub-boundaries marking separate regions such as eye.

When we locate such edges and keep only the edges while ignoring all other pixels, we get an image with edges defining the original subject. Look at the picture on the right which shows the features of original Lenna image on the left.

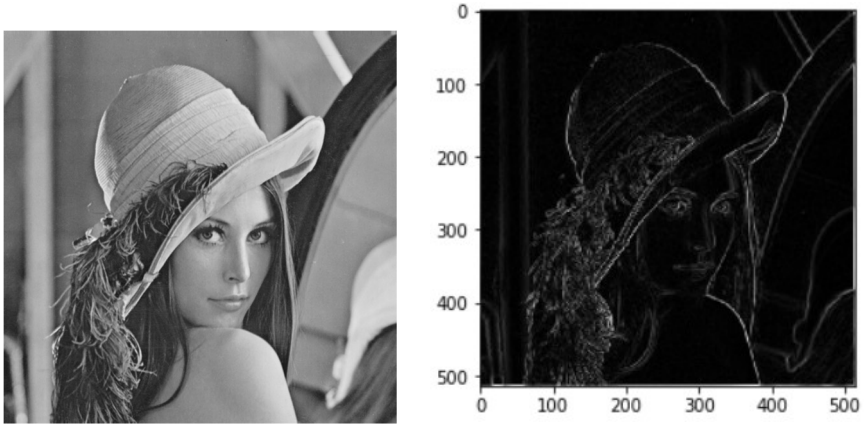


Fig 8

Thus the edges define the subject and the edges act like features i.e. predictor variables while the dependent variable is a human face. This way of defining an object based on the edges as features turns out to be much more reliable technique than the older techniques such as pixel intensity distributions of the subject. The reason for that is, the true features of the subject will appear as edges (pixel intensity changes) irrespective of whether the picture is taken in Sunlight or Moonlight. That is not the case with the older method of pixel intensity distribution to identify objects.

How are the edges in the original picture detected? For that we use the concept of convolution!

## Convolution for detecting the edges

To understand convolution, let us play with a toy known as the Pin Art. You may have played with this in your childhood. Some play with it even at my age ☺ . If you have not yet laid your hands on it, I suggest you do.


' Fig 9

Image source : https://www.amazon.in/Pin-Art-Impression-Stainless-Chrome/dp/B00MEIPKXS

When we place objects underneath, the pins rise up. Assume each pin as a pixel and height of the pin as brightness of the pixel. The pins will approximately map the underlying edges of the object.

After multiple iterations, one can even with blindfold, scan the pin art surface with hand and by merely feeling the edges and mentally combining the edges identify the object underneath.

The process of scanning the surface by hand is Convolution. The hand acts like a function which interacts with another function (2D surface of 3D) to detect presence of edges. The mental process of accumulating the edges to get a holistic picture is equivalent of Pooling. The edges define the object and act like features /independent variables.

## Kernels for Convolving

Kernels are special functions whose sole purpose is to operate on the 2D image and identify coordinates in terms of X1 and X2 where edges lie. Any other pixel which does not form the edge of the object, such as the pixels deep inside the

black coat of the camera man shown below in red ring are useless pixels. They should be ignored.

Example of useless pixels. They do not define the camera man and hence they cannot act as features

Fig 10

Let us define a simple kernel as [-1,1] and operate it on the greyscale image shown below. This kernel will move horizontally one pixel at a time and will not cross the boundary of the image. At every step it has to do the following

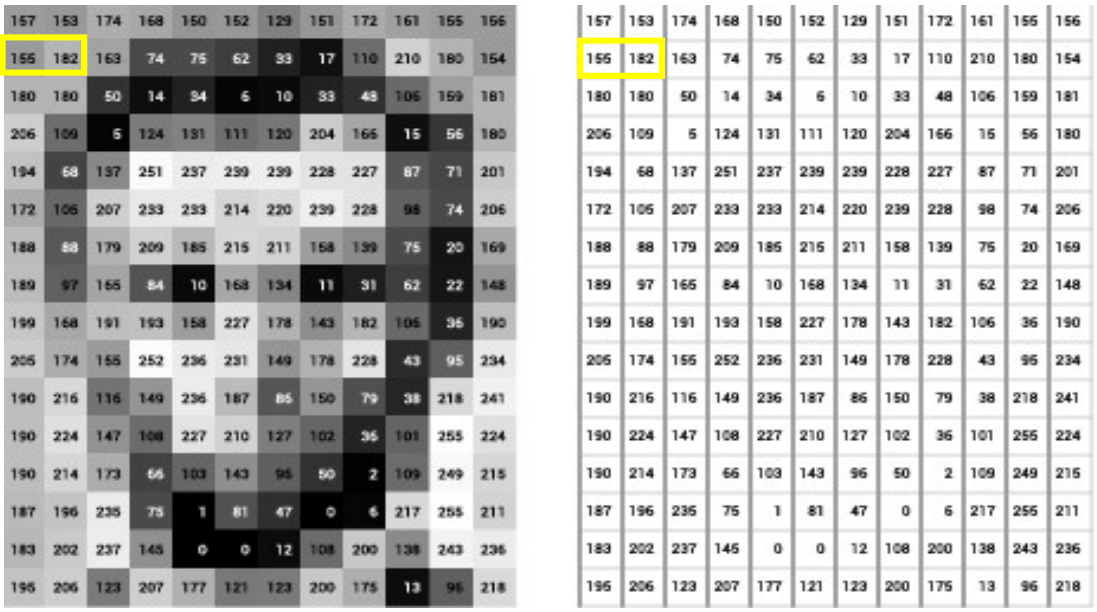
1. Multiply corresponding underlying pixel intensity in the image with the value in the kernel. For e.g. if the pixel intensities were 200,210. Then the kernel should do (200 X -1) , (210 X 1)
2. Add the two multiplications i.e. -200 + 210 = 10
3. The value of 10 shows a small change in pixel intensity
4. Repeat steps till last but one pixel
5. Repeat step 4 for next row till last row

All those coordinates where pixel intensities fluctuate in small magnitudes will appear as flat regions (check Fig 7) in the output 3D surface indicating no useful feature in that region.

The simple kernel of [-1,1] acts like a function that calculates dy/dx where dy being change in pixel intensity and dx is change in pixel position. Since we move the kernel one pixel at a time, dx is always = 1. Hence the difference that we saw in bullet point 2 is dy and equivalent of dy/dx.

dy/dx is also known as gradient at X. Gradient refers to the direction in which something changes (increases) and also reflects the magnitude of change. Let

the yellow box represent the kernel. Let us look at the effect of that sliding horizontally on second row of the image below.



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

Fig 11

Image source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html (beautiful introduction to computer vision)

The dy/dx values for second row would be (try it yourself if you like to)

26 , -19 , -89 , 1 , -13 , -29,-16, 93 , 100 , -30, -26

The sign of the number shows the direction of gradient at a pixel. While magnitude shows the strength of the change.

In this list -89 is a significant difference between 163 at (second row third column from top left) and 74 (second rows and fourth column from top left). The difference is due to the hairline of the person vis a vis the background.

Convolving the filter across the entire image will give us locations where there are strong gradients and their direction together indicating presence of an edge. Why the other values such as 19 are not edges? The smaller the magnitude, more likely it is caused by digital noise than by a real edge defining the object.

**Note:** The numbers -1,1 in the kernel are known as weights of the kernel.

## Digital noise the villain

Though in the example above a simple kernel of [-1,1] seems to be working, in practice we do not use such simple kernels. This was only for explaining the concept of feature detection being same as locating gradients (sudden change in pixel intensities). Why would this approach not work in practical world? The reason is digital noise. To understand digital noise one needs to understand the process of digitizing i.e. how analogue signals are converted into digital images by a digital camera which is not in scope of this post.

Digital noise is the random fluctuations in pixel intensities caused by dark current that is generated by the camera itself. The pixels intensities are supposed to represent the analogue light intensities falling on the lens of the cameral. However, dark current generated by the camera itself also impacts pixel intensities. To observe the unwanted changes in pixels take a digital camera and point it at the night sky where it is darkest. You may have to focus on star before clicking but click pointing in direction of darkest sky. Zoom into the picture shot and you will notice many tiny white specks. They are not stars, they are pixels with intensities greater than 0 (dark) caused by sources other than light through the lens (in fact there was not much light entering the lens).

Because of the pixel level fluctuations, two neighbouring pixels can end up being different in intensity and this change in intensity gets captured by the simple kernels as gradient. For e.g. in the picture below the difference between first row, first column and first row second column ((157 X -1) + (153X1)) will appear as a weak gradient of -4 but this is not caused by the object photographed (the face), it is caused by digital noise. The kernel should not pick this up as a feature describing the object because it does not represent the object.

You can observe many such pixel changes which within the face also. For e.g. 299, 297 in the forehead. What defines the subject is the significant gradients. The insignificant gradients are likely caused by digital noise (unexplained variance in pixel intensities)

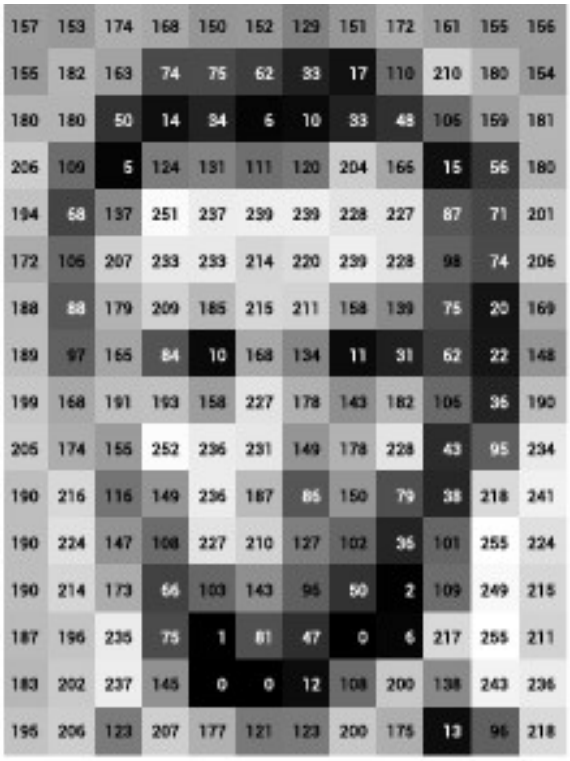


Fig 12

Image source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html (beautiful introduction to computer vision)

Because of this digital noise, when you run the simple kernel of [-1,1] on the cameraman greyscale image what you get is shown below. We can still see some edges but most of them are fake edges caused by noise.

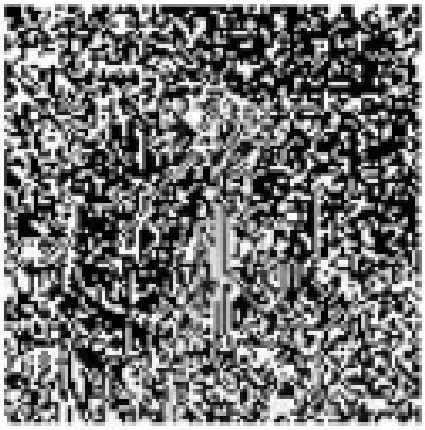


Fig 13

## Denoising digital images

An important step in preparing digital images for computer vision modelling is to de-noise the images i.e. remove the digital noise effect from the digital image. Statistically noise can be removed to a great extent using averaging. The same is done here too.

The effect of denoising a digital image is the removal of insignificant gradients across pixels. The smaller the gradients the more likely they are going to be noise not the real feature / edge. There are various tools available to do that and one of the most famous ones is the Gaussian filter. This tool takes as in put an image and outputs another new image whose pixel values are averaged pixel values of the input image.

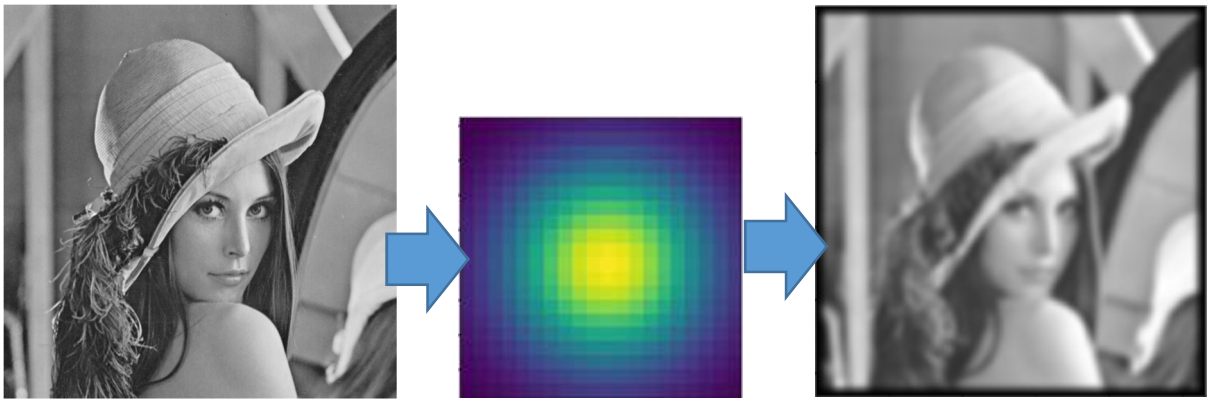



Fig14

If you are wondering, 'who in his right mind would do this given the denoised image looks so blurred!', what we need to understand is that the original image though very clear to human vision, has many noisy edges at pixel level. The noisy edges have been erased through an averaging mechanism. As result, the output image has less noise but is blurred. The strong gradients which are likely to be true features too get blurred but they still exist strong enough to be picked up by the convolution process. The objective of denoising is similar to PCA, improve signal to noise ratio where signal refers to to true features and noise refers to the fake features caused by the device. The truth is, noise can never be entirely eliminated. It can be minimized but cannot be made zero unless we wish to loose even good features.

Thus, for convolution which works at pixel level, the denoised image is more appropraite than the original image which is good to human vision that do not work at pixel level.

I have not touched upon the various kinds of noise and the various techniques to address those noise in this post. You will find plenty of online resources for that. The bottom line is, before we convolve, we have to remove the fake edges caused by random fluctuations in pixel intensities. Not doing that will lead to etraction of lot of useless features along with useful ones and may even lead to overfitting i.e. good performance in training but poor results in test images.

In the early days of DNN for image processing, the kernels were hand crafted. For e.g. Sobel kernel was a well known kernel to extract features from an image. This was further replaced by Canny edge detector. However, today, thanks to the DNN with back propagation methods, the weights in the kernels are also learnt thru training. We do not have to pre-define them. In fact, they start with random numbers but over epochs settle down to appropriate weights leading to specialization of each kernel in detecting different features. The kernels usually are 3X3 or 5X5 and rarely 7X7 size. I have not discussed these in this post as that was not the objective.

**Pooling Features**

The output of convolution process is a new image called the feature map. The feature map, as the name suggests is a map that reflects the positions on the input image where there are pixel intensity gradients i.e. likely features that represent the object in the picture.

However, the first few feature maps contain not only useful information but also redundant useless information. For e.g. in the gradient vector that we saw earlier i.e [26 , -19 , -89 , 1 , -13 , -29,-16, 93 , 100 , -30, -26], there are strong gradients such as 89, 93, 100 along with other that are not large enough to be true features. How do we extract only the useful information and pass it on to next stage for processing?

Look at the following pictures of a dog.

Fig 15

Cut vertical stripes in the original picture. Re arrange the stripes. You get two pictures of the dog (third from left). Next cut the two dog pictures horizontally and re-arrange the rectangular pieces, you will get four images (two complete and two incomplete) images of the dog!

What does this indicate? This tells us that the original picture had many redundant pixels. Pixels that do not define the dog. Even pixels within dog's neck for instance, do not define the dog. What defines the dog is the pixels that form the edges of the dog and its eyes and ears etc. So, when we cut the picture in stripes, the redundant pixels on resulting two images were reduced without losing the real information (edges of the dog).

Pooling does the same thing on the feature maps generated by the convolution process. It extracts gradient information and reduces the redundant pixels (which do not contain any real features) in the output of image of the pooling process.

However, to understand pooling in a technically accurate way, we need to revisit the feature map. What is a feature map? Feature map is a map of where in the original image the pixel intensities change significantly. This we saw is mathematically expressed as dy/dx.

dy/dx is also known as gradient or slope in geometrical terms. Gradient is a term used in mathematics and physics to represent a direction and magnitude of increase. In which direction is the pixel intensity changing and how much. Negative direction means against the direction of dx and positive direction means in the direction of dx. For e.g. in the list of gradients [26 , -19 , -89 , 1 , -13 , -29,-16, 93 , 100 , -30, -26], the first one +26 is a change from left to right (in direction of dx) while the second gradient -19 is in direction opposite to the dx. So, if we look at the picture again and use an arrow to represent the gradients it will look as shown below.

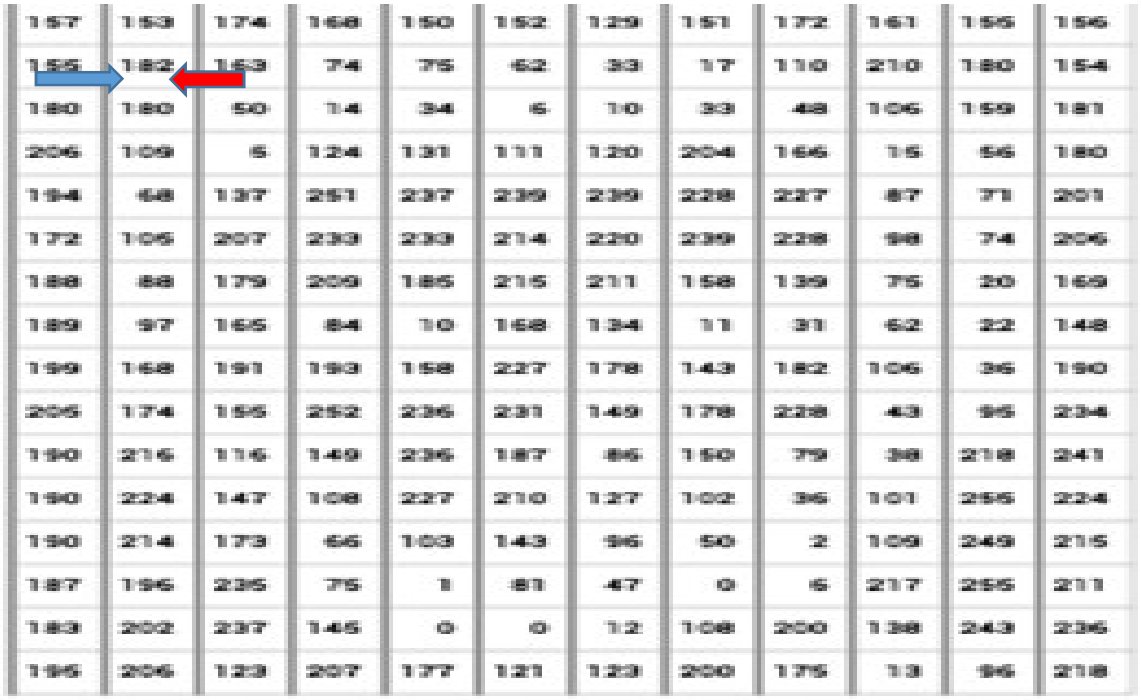



Fig 16

Blue is +26 gradient and red one is -19 gradient, direction of dx is from left to right. Blue one is longer because its magnitude is larger.

So if we look at feature maps as a field of arrows, it will look like the one below

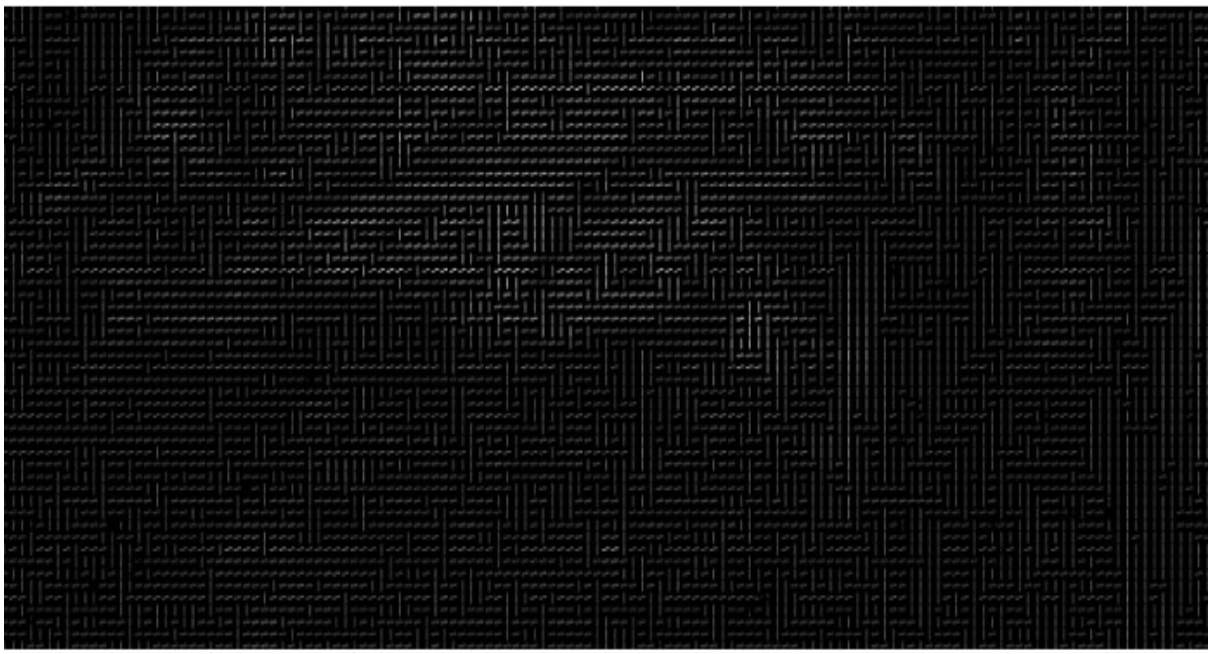


Fig 17

The beauty of working with pixel intensity changes (gradients) rather than pixel intensity distributions (older approach) is, under varying lighting conditions the true gradients reflecting the true features will remain the same i.e. they will have same magnitude and direction. The fictitious gradients caused by fluctuations will not remain the same. Further, when we look at pixel intensity distributions, they will never be the same under different lighting conditions and hence building models using pixel intensity distribution characteristics never really worked. The models were too fragile and often fail in real world where lighting would be different.

Instead of having to look at gradient vectors at each pixel, if we get an overall general flow or directions of the gradients we would have sufficient information to extract the features. This is where the pooling comes in. Pooling gives us a overall general flow in the gradient map i.e. feature map.

To get the overall flow, break the feature map into small squares of mXm which is usually 2X2 blocks i.e. 4 pixels at a time. In the 2X2 grid there will be four vectors. For e.g. In the figure below each coloured quadrant is a 2X2 region.

Max Pooling - Select the gradient with largest magnitude and let that represent the overall gradient in that 2X2 block. This is called max pooling. This is often show as

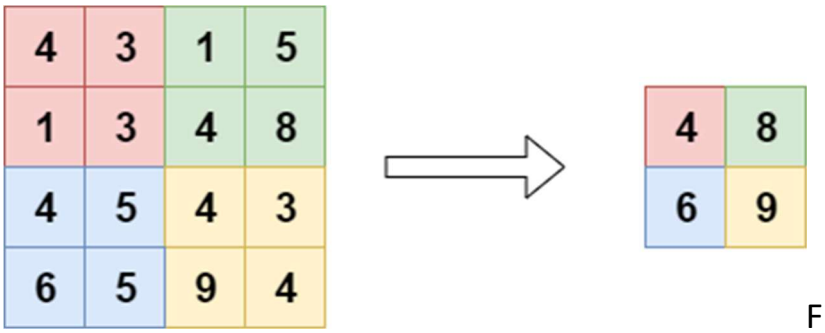



Fig 18

Image source: https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/

The 4 in the first quadrant is the largest gradient (in magnitude) and hence it is used to represent that region. Similarly is the case in other four quadrants.

Average Pooling - take average of all the four gradients (vector addition), the resultant average vector will represent the gradient in the 2X2 region. This is average pooling.

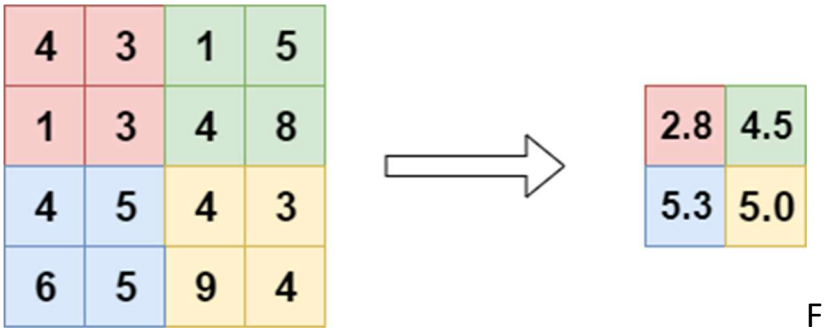



Fig 19

Image source: https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/

As you may have guessed, the problem with average pooling is, the smaller gradients such as 1 in first quadrant may be due to noise! (noise cannot be completely eliminated). That is being used to define the feature! Thus, if the image dataset is too noisy, average pooling may lead to less optimal models.

The result of pooling is another new feature map where the coarse level features start to take shape as in the image below which is the result of max pooling. Do you see a face?

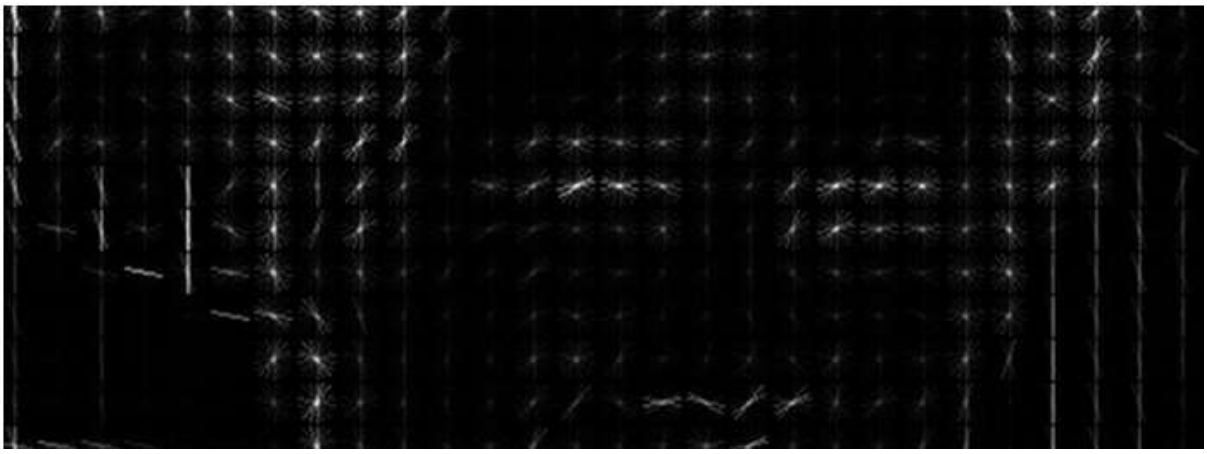


Fig 20

This feature map may now be used as input for next convolutional layer where the entire cycle will repeat again. Finally, at the last stage of the Convnet model we will have a feature map packed with useful features (dense feature maps). That feature map will be used by a dense layer along with the training labels to build the model.

**Note:** Convnet is only a feature extractor. It cannot do classification or regression on its own. It needs the dense layer or other conventional algorithm such as SVC to do the classification.

Assuming you are in the same position that I was when I started to explore CNN, I hope the concept of convolution and pooling is now slightly clearer that it was when you started to read this post. ☺