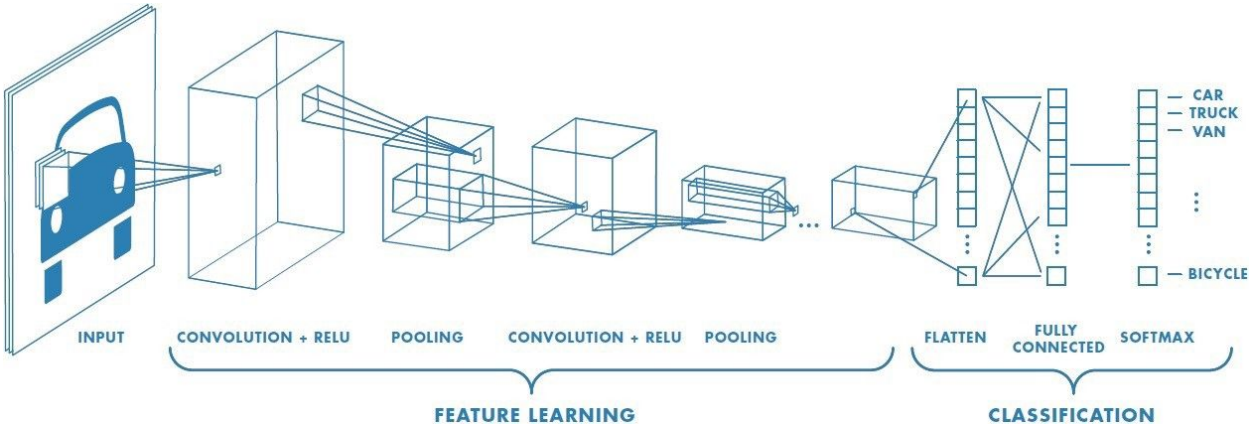
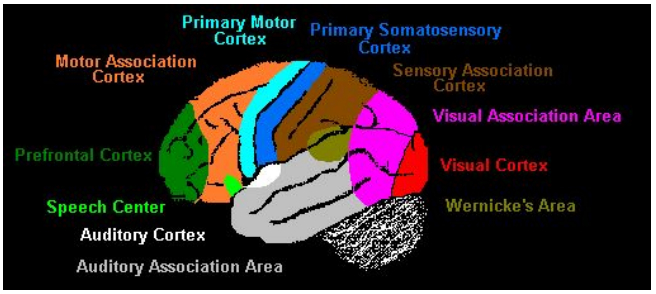
Convolutional Neural Networks

Fully connected layers are an essential component of Convolutional Neural Networks (CNNs), which have been proven very successful in recognizing and classifying images for computer vision. The CNN process begins with convolution and pooling, breaking down the image into features, and analyzing them independently. The result of this process feeds into a fully connected neural network structure that drives the final classification decision.



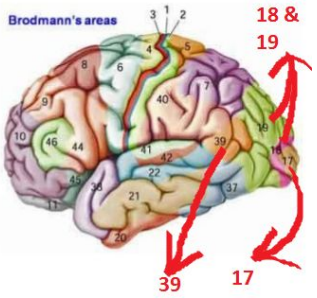
Biological Inspiration of Convolutional Neural Network (CNN)

Where is Visual Cortex Located in Humans Brain?



Visual Cortex is the part of the Cerebral Cortex of the Brain that processes the visual information. Visual nerves from the eyes run straight to the primary visual cortex. Based on the structural and the functional characteristics it is divided into different areas, as shown in the following picture:

1. Primary visual area (area17)
2. Visual association area (area 18 & 19)
3. Higher visual association area (area 39)



Visual Cortex: Functions

The visual information is passed from one cortical area to another and each cortical area is more specialized than the last one. The neurons in the specific field only respond to specific actions. Some of them with their functions are as follows:

1. Primary Visual Cortex or V1: It preserves the spatial location of visual information i.e. orientation of edges and lines. It is the first one to receive the signals form what eyes have captured.
2. Secondary Visual Cortex or V2: It receives strong feedforward connections from V1 and sends strong connections to V3, V4, and V5. It also sends a strong feedback network to V1. Its function is to collect spatial frequency, size, color and shape of the object.
3. Third Visual Cortex or V3: It receives inputs from V2. It helps in processing global motion and gives complete visual representation.
4. V4: It also receives inputs from V2. It recognizes simple geometric shapes and also forms recognition of the object. It is not tuned for complex objects as Human Faces.
5. Middle Temporal (MT)Visual Area or V5: It is used to detect speed and direction of moving visual object i.e. motion perception. It also detects the motion of complex visual features. It receives direct connections from V1.
6. Dorsomedial (DM) Area or V6: used to detect wide field and self-motion stimulation. Like V5 it also receives direct connections from V1. It has an extremely sharp selection of the orientation of visual contours.

Edge Handling

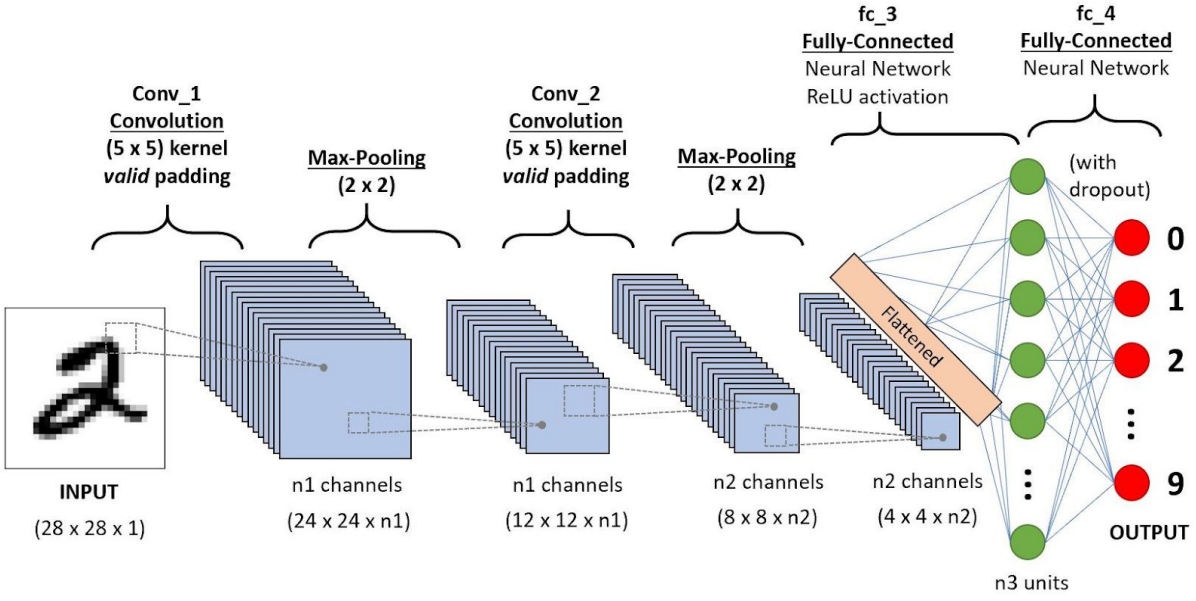
Kernel convolution relies on centering a pixel and looking at it's surrounding neighbors. So, what do you do if there are no surrounding pixels like on an image corner or edge? Well, there are a number of ways to process the edges, which are

listed below. It’s most common to use padding, cropping, or extension. In extension, the border pixels of an image are copied and extended far enough to result in a filtered image of the same size as the original image.

Extend The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

Padding The image is padded with a border of 0's, black pixels.

Crop Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

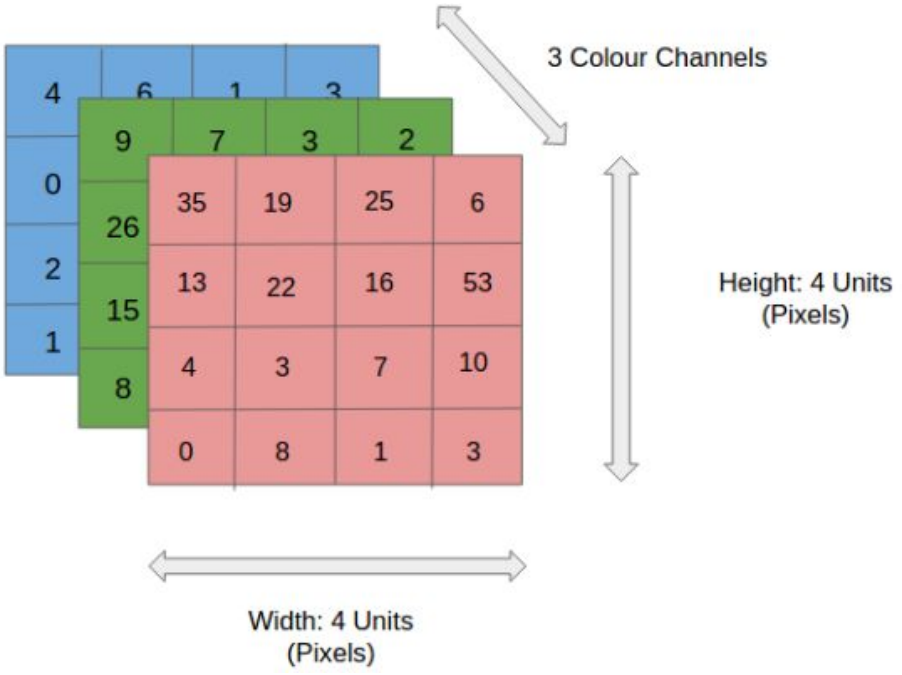


A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification

algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Input Image



In the figure, we have an RGB image that has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form that is easier to process, without losing features which are critical for getting a good

prediction. This is important when we are to design an architecture that is not only good at learning features but also is scalable to massive datasets.

Convolution Layer — The Kernel

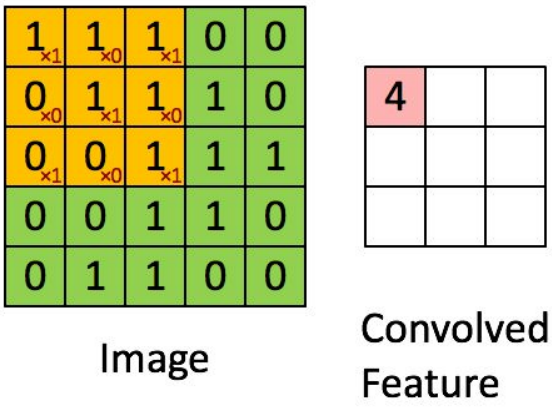
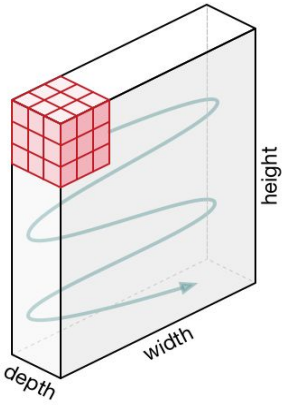
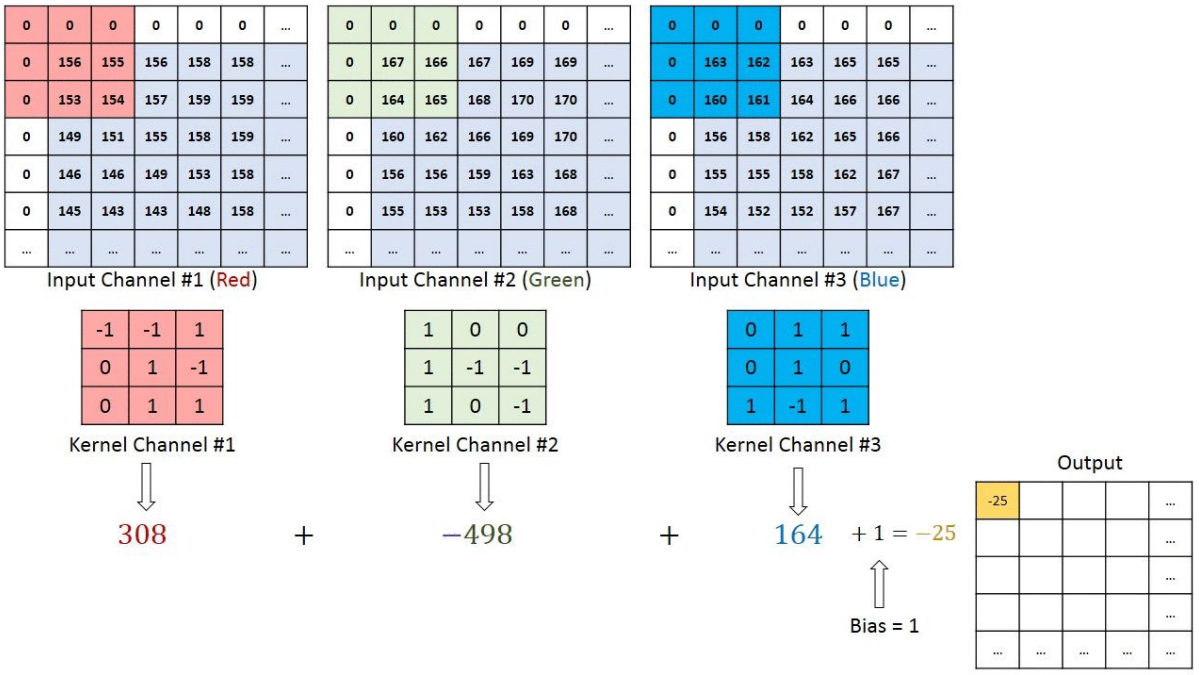
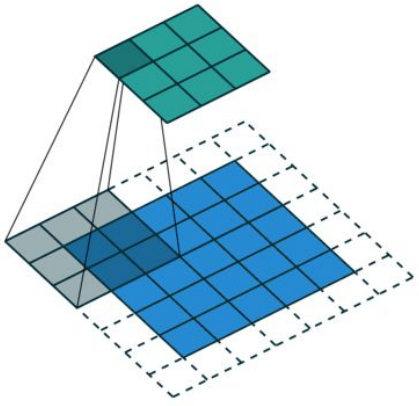


Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB).

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.** The Kernel shifts 9 times because of Stride Length = 1 (Non-Stride), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

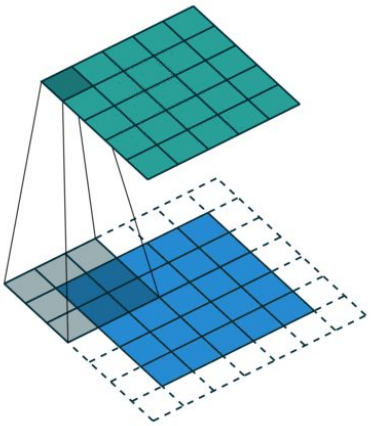






The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network that has the wholesome understanding of images in the dataset, similar to how we would.

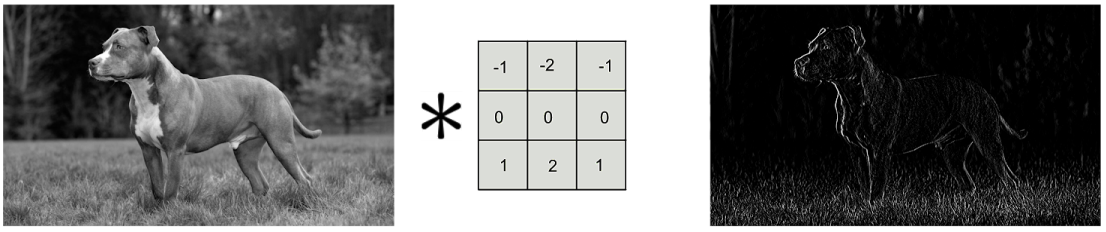
There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.



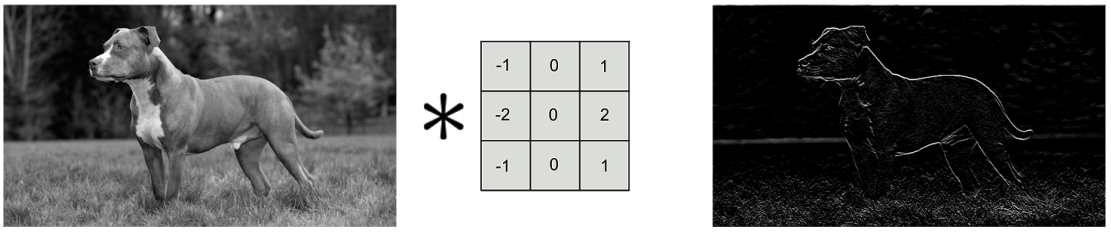
Padding is a term relevant to convolutional neural networks as it refers to the number of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.

Then, we move the convolution kernel from horizontally to the right by one pixel, we make a new element-wise product then added up to get a new coefficient of the output image. Once at the end of a line, the kernel makes a vertical stride down and starts again from the left, we iterate likewise until the kernel has covered all the matrix image. It is important to note that the kernel always remains on the initial matrix, without overflowing. For sure, we cannot use any filter, the coefficients of our kernel will depend on the features we want the filter to highlight. Let’s see the result of a convolution with some well-known filters.

Vertical Sobel filter, Its action is to highlight the vertical lines of the object. Applied to the initial image on the left side, here is the result,



Horizontal Sobel filter, This time it is to highlight the horizontal contours of the image. Here is the result applied to the left-side image,

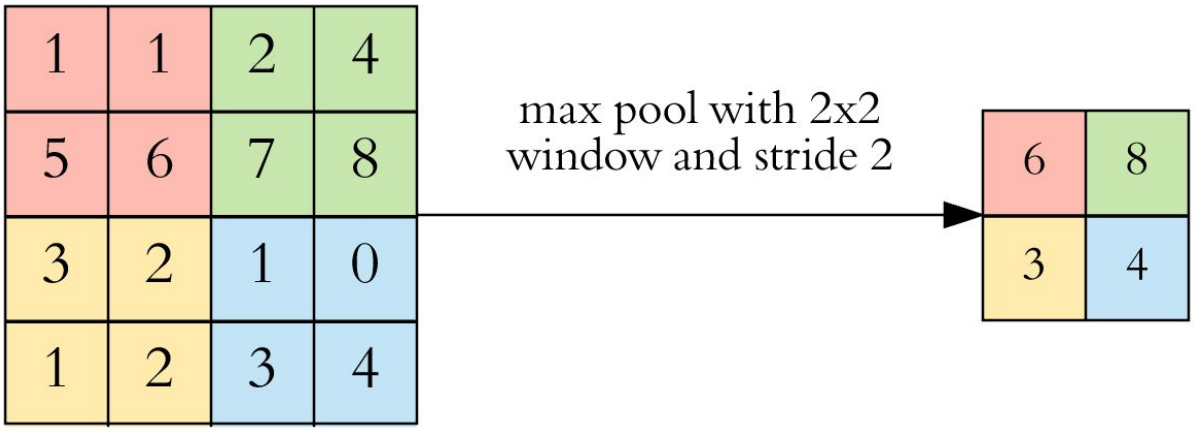


The actions of these filters can be combined to perform more complex operations. There are several more filters already listed which can be directly used this way, depending on the task to be solved: average filter, Gaussian filter, etc. Before the emergence of Deep Learning, human experts had to calculate and determine the right filters to use in order to perform specific image processing actions: face detection, photo editing, like Snapchat filters, etc. Now with Deep Learning, to determine these filters is automatically done by learning, the model will find according to the problem to solve, the good filters from the training data. In a

cat or dog classification problem, for example, the filters will make it possible to highlight the determining characteristics for the classification: forms of the ears, the shape of the eyes, the shape of the muzzle, the contours, etc.

Pooling Layer

Max Pooling: It is used to detect, where the objects are located in the Image based on the output of the each cluster of neurons in previous layer. As the face is detected where ever it is; it doesn't depend on the location of the face in image.



Max Pooling Operation

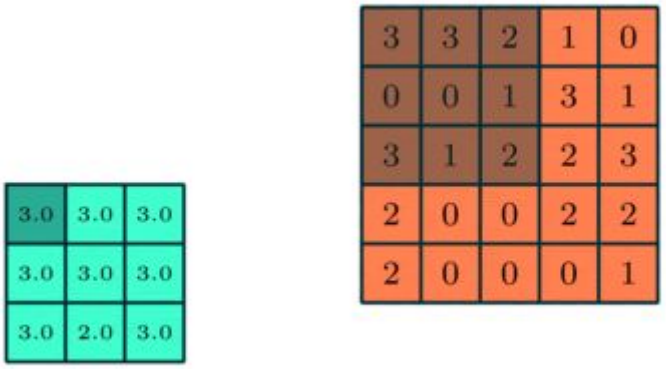
ReLU (Rectified Linear Unit): As the human brain never stops learning, it (brain) always learns from the observations and experiences; i.e. the inputs which it receives from the sensory organs, are utilized at some or another point; but the learning never becomes "Zero". To add this feature to the neural networks ReLU is used. The activation function is : f(x) = max(0,x). For any activation function, we must be able to take the derivative of that

function and with ReLU we can do that. But the derivative at zero is not defined for the ReLU. Due to zero, we can have the problem of a dead activation state. This implies there will be no weight change meaning no learning. But in humans, it doesn't happen often. To tackle this problem the concept of Leaky ReLU is used.

Leaky ReLU : The function is : f(x) = if (x > 0) then x else 0.01*x. With this, we are avoiding the problem of dead states. That is the network can continue to learn, but it can face the problem of Vanishing gradient.

Data Augmentation: We humans can recognize the face even if it is inverted, rotated, flipped, reflected or skewed. Using the Data Augmentation technique we can convert a single image into different types of images and use the newly formed images for training CNN. After that, the CNN will be able to detect In-variance based data such as rotated faces, large and small faces, flipped faces, etc (i.e. the objects will be recognized even if they are not in their original position).

Dropouts: Do all the neurons present in our brain fire to learn something? The answer is **‘’NO’’**. It is not necessary that the fire in a linear fashion or in back-propagation. Some of the neurons may stay inactive in one phase of learning and may get active in another phase of learning or vice-versa. This gives the capability of independent learning to the neurons. To have this in the networks, the concept of dropouts is introduced. After applying dropout with probability p, the randomly selected individual nodes/neurons are dropped out of that epoch for the learning process and the respective incoming and outgoing edges are also dropped out. It is much used to avoid over-fitting in the network.



Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

Pooling algorithm

In a natural image, there is a strong local correlation between pixels. It means that with an image, if you know that a pixel is red, it is very likely that its four closest pixels are also shades of red. In the case of a grayscale image, if a pixel has an intensity of 180 for example, its nearest pixels will also have intensities around 180. It is, therefore, possible to reduce the dimensions of the image by keeping only a local representative per pixel local block, this is called pooling. Generally speaking, we will take the pixel with maximum intensity (so highest value) in the block — **max pooling** — , we can also average the intensities of pixels in the block — **average pooling** — and hold it as a representative.

As it can be seen above, pooling halves each dimension (height and width) of the input image. One could think that pooling strongly degrades the initial image by representing a block of pixels by only one; but in fact, the output image (feature map) is certainly half as great, but it contains the main characteristics of the input image.

For example, let’s apply max pooling to the feature map outputted by the horizontal Sobel filter, “pooled” images have been enlarged for comparison.

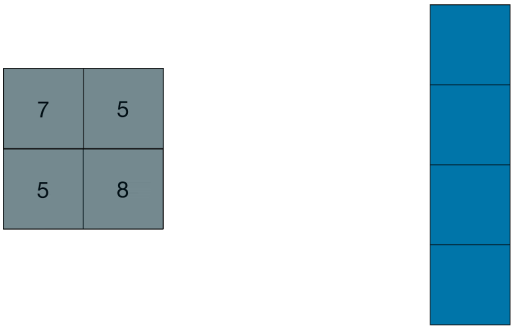
Feature map after two max-pooling

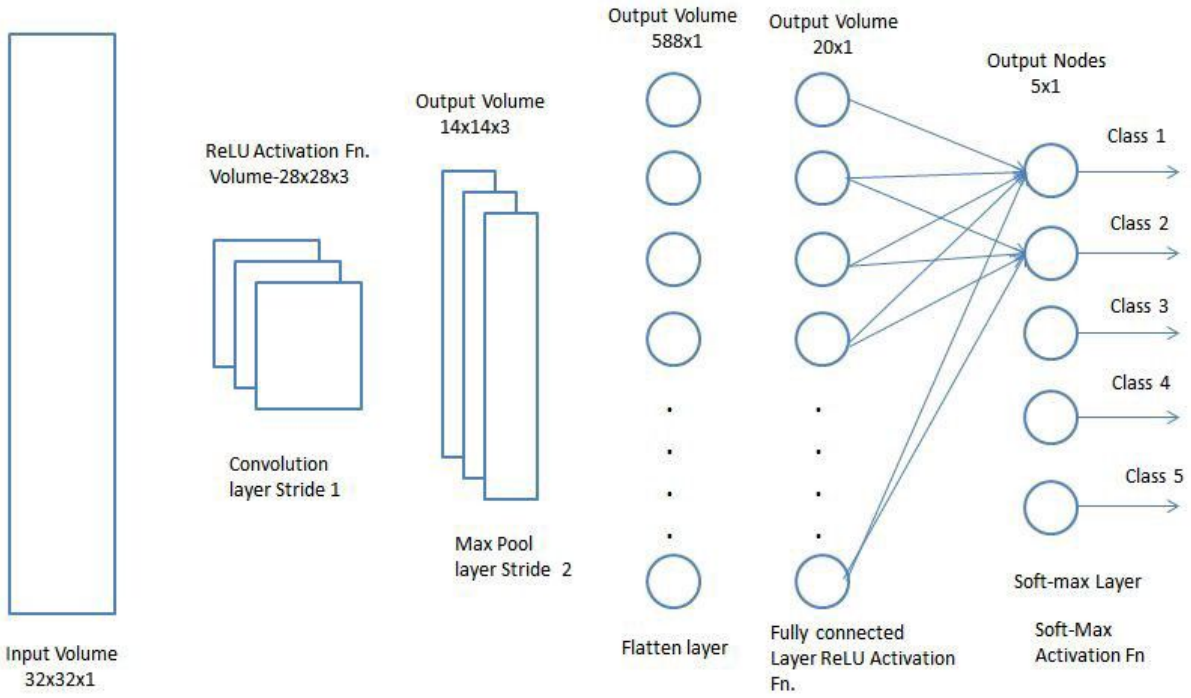
Note that even after two pooling, the contours are clearly visible, and the image is less rich in detail. Pooling is not just about resizing but also keeping only the meaningful features of the input image.

Typically, in a convolutional network, there is a sequence of operations; convolution-pooling layer-convolution-pooling layer and so on. By repeating these operations many times, we finally get feature maps with only the meaningful (according to the problem so solve) characteristics of the input image. We can now exploit the power of a multilayer perceptron to achieve the classification task.

Flatten the feature map

To finally classify the image into a category, say cat or dog, we will set up a multilayer perceptron (Multi-Layer Perceptron) on top of the last convolution layer. The previous convolution and pooling operations have greatly reduced the size of the input image to keep uniquely the meaningful characteristics for the classification. Since feeding an MLP requires input vectors (one-dimension arrays or 1d arrays), we need to “flatten” the output feature map. The MLP, therefore, receives small-sized feature maps as 1d array and chooses the corresponding category with regard to those feature maps.





Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

1. LeNet

2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet