Introduction to

Convolutional Neural Networks(CNNs)

With the help of CNNs we can incorporate domain knowledge into the architecture of a Neural Network.

**Motivation—Image Data**

Image data has important structures, such as;

* ”Topology” of pixels
* Translation invariance
* Issues of lighting and contrast
* Knowledge of human visual system
* Nearby pixels tend to have similar values
* Edges and shapes
* Scale Invariance—objects may appear at different sizes in the image.

Fully connected would require a vast number of parameters. MNIST images are small (32 x 32 pixels) and in grayscale. Color images are more typically at least (200 x 200) pixels x 3 color channels (RGB) = 120,000 values. A single fully connected layer would require (200x200x3)2 = 14,400,000,000 weights! Variance (in terms of bias-variance) would be too high.

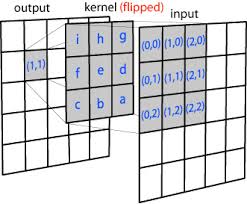
So we introduce “bias” by structuring the network to look for certain kinds of patterns.

Features need to be “built up". Edges let us know the shapes from which we can generalise relations between shapes. Also textures need to followed. For eg. a cat = two eyes in certain relation to one another + cat fur texture; whereas eyes = dark circle (pupil) inside another circle.

**Kernels**

* A kernelis a grid of weights “overlaid” on image, centered on one pixel
* Each weight multiplied with pixel underneath it
* Output over the centered pixel is *p*
* Used for traditional image processing techniques:
* Blur
* Sharpen
* Edge detection
* Emboss

**Kernel: 3x3 Example**



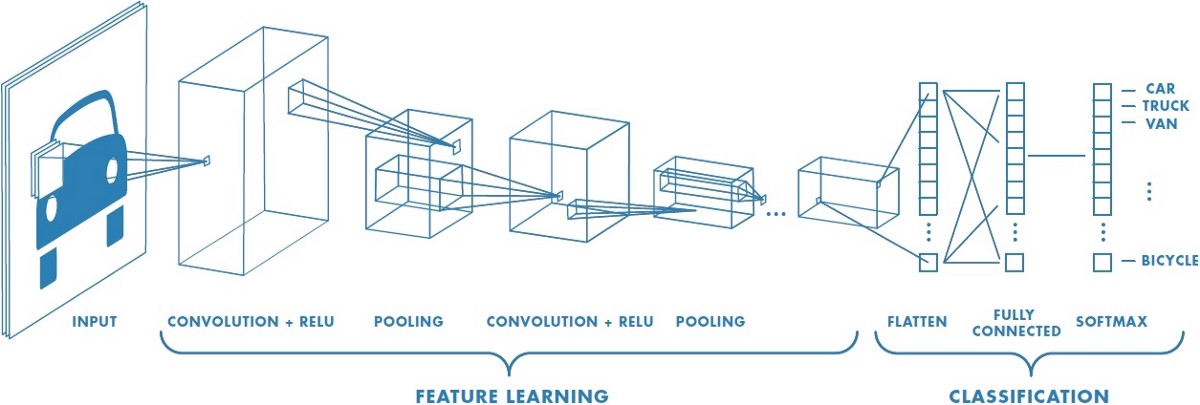
**Kernels as Feature Detectors**

We can think of kernels as a ”local feature detectors”

**Convolutional Neural Nets**

Primary Ideas behind Convolutional Neural Networks:

* Let the Neural Network learn which kernels are most useful
* Use same set of kernels across entire image (translation invariance)
* Reduces number of parameters and “variance” (from bias-variance point of view)



**Convolution Settings**

**Grid Size (Height and Width):**

* The number of pixels a kernel “sees” at once
* Typically use odd numbers so that there is a “center” pixel
* Kernel does not need to be square

**Padding:**

* Using Kernels directly, there will be an “edge effect”
* Pixels near the edge will not be used as “center pixels” since there are not enough surrounding pixels
* Padding adds extra pixels around the frame
* So every pixel of the original image will be a center pixel as the kernel moves across the image
* Added pixels are typically of value zero (zero-padding)

**Stride**:

* The ”step size” as the kernel moves across the image
* Can be different for vertical and horizontal steps (but usually is the same value)
* When stride is greater than 1, it scales down the output dimension

**Depth:**

* In images, we often have multiple numbers associated with each pixel location.
* These numbers are referred to as “channels” – RGB image—3 channels – CMYK—4 channels
* The number of channels is referred to as the “depth”
* So the kernel itself will have a “depth” the same size as the number of input channels
* Example: a 5x5 kernel on an RGB image – There will be 5x5x3 = 75 weights
* The output from the layer will also have a depth
* The networks typically train many different kernels
* Each kernel outputs a single number at each pixel location  So if there are 10 kernels in a layer, the output of that layer will have depth 10.

**Pooling:**

* Idea: Reduce the image size by mapping a patch of pixels to a single value.
* Shrinks the dimensions of the image.
* Does not have parameters, though there are different types of pooling operations.
* Max-pool - For each distinct patch, represent it by the maximum
* Average-pool - For each distinct patch, represent it by the average