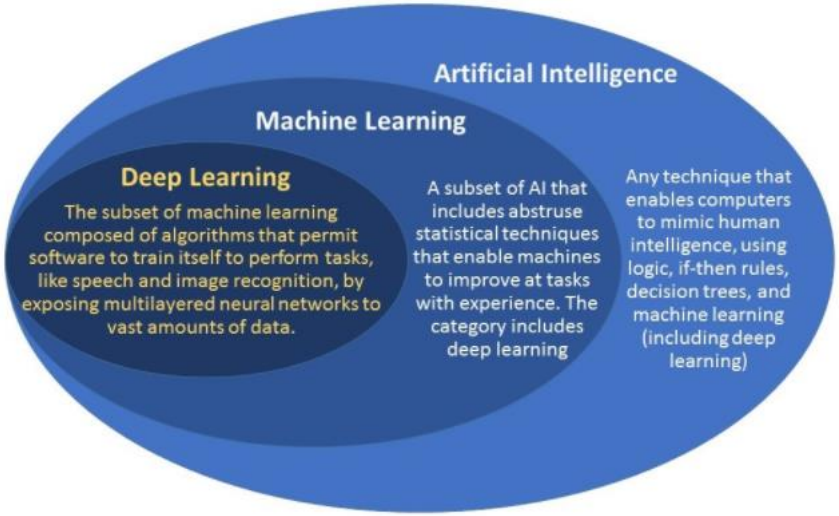
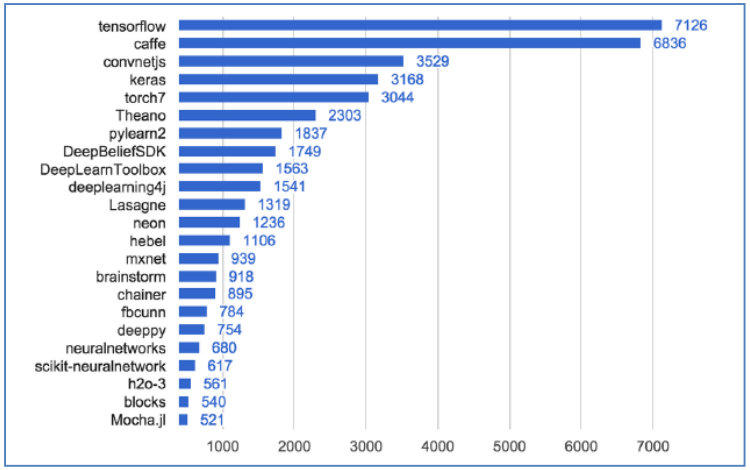
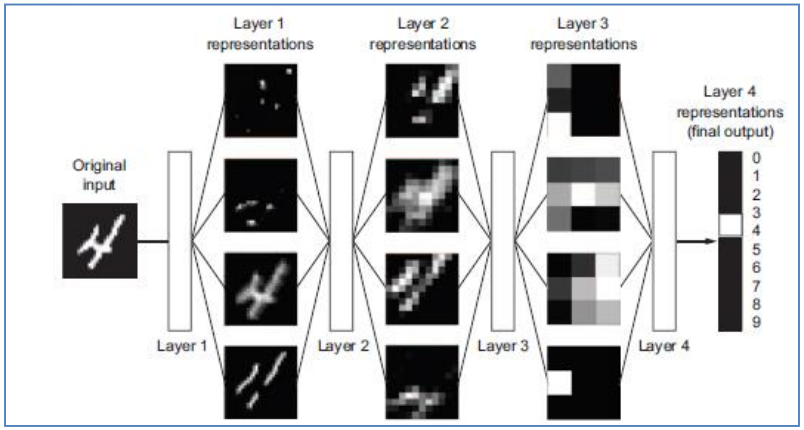
Deep Learning – Tools (open source)







keras :

Deep Learning Library in python for Theneo and Tensor flow for developing and evaluating deep learning models

It wraps the efficient numerical computation libraries Theano and TensorFlow

Deep Learning computation libraries:

TensorFlow : supported by Google

Theano : supported by University of Montreal's MILA

Torch : supported by Facebook, Twitter and NVIDIA

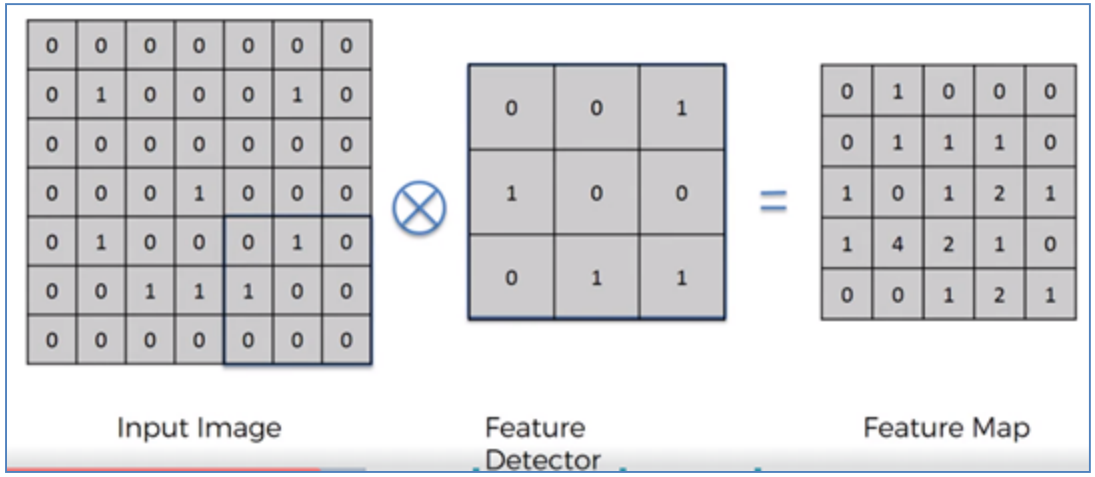
CNN - Convolution Neural Network

CNN: The **Convolutional Neural Network(CNN)** is a type of Supervised **Deep Learning** that is used to **analyze Computer Vision (Image & Video Recognition).**

- Like Neural Network, **CNN is also made up of Neurons** with learnable weights and Biases.
- In the **CNN** the input is **Multi Channeled Image**(RGB 64X64X3), unlike in **ANN** where the input is **vector**.

Convolution is basically a combined integration of two functions

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

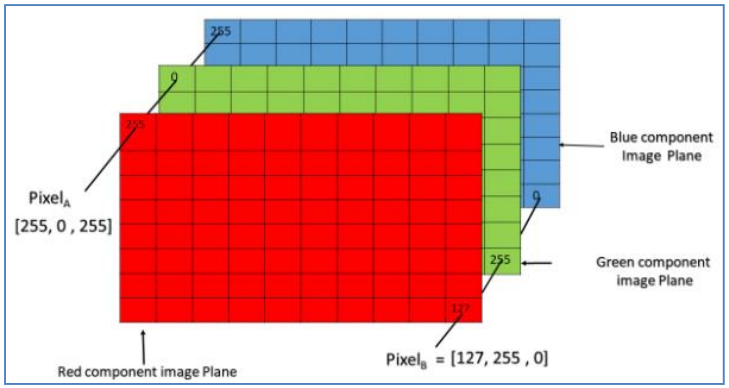


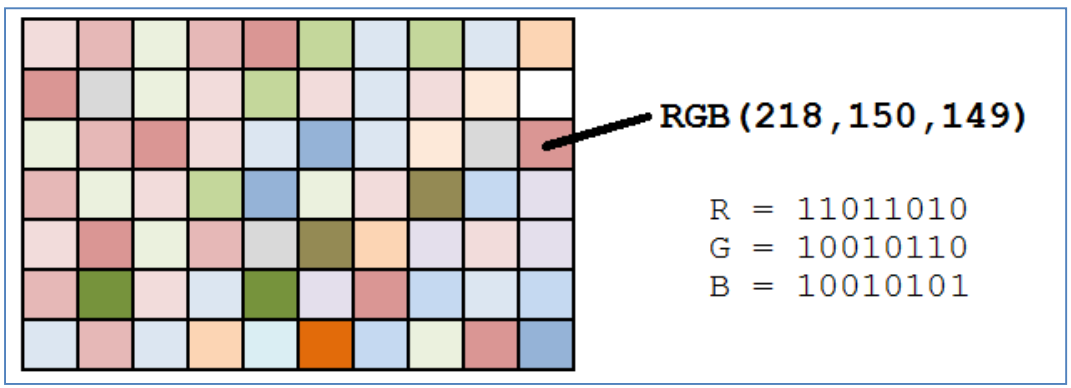
Yann LeCun - Founding Father of CNN
1988 – Back Propagation and CNN Theory

Image Pixel Values

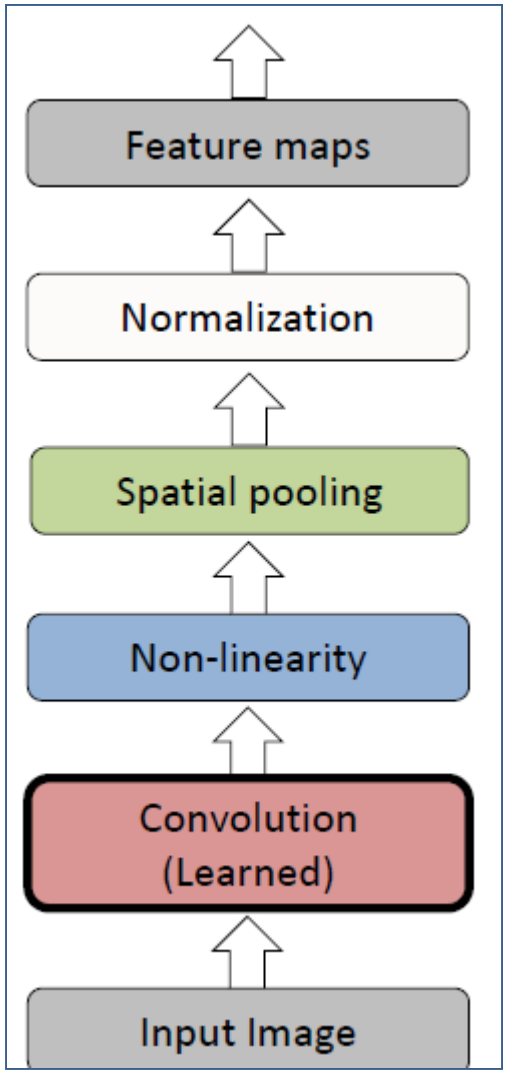
- Images are in the form of **arrays of Pixels**, depends on the resolution and size of images, Computer will see the image as an array of Pixel numbers (ex - 32X32X3 (RGB))
- Each of **these pixels** will be given a value from **0 to 255** which describe the **pixel intensity** at that point.

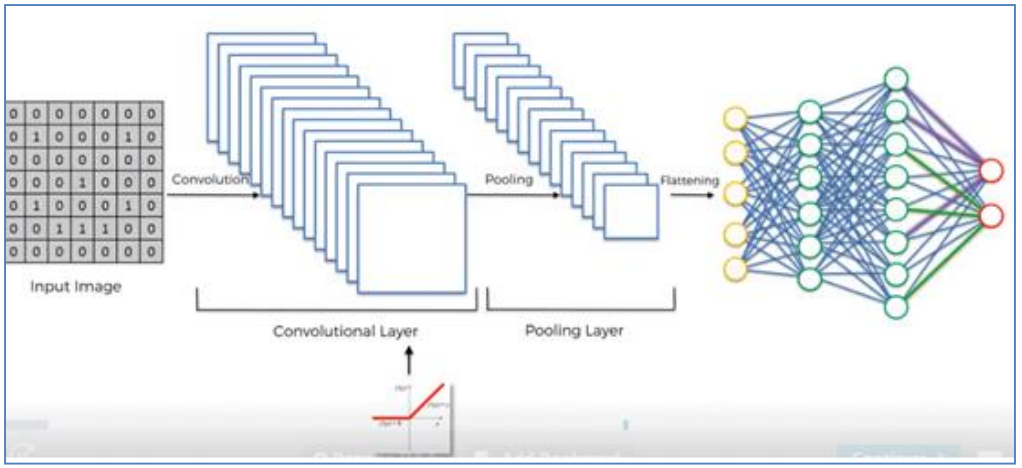
90	91	92	93	94	95	96	97	98	99
80	81	82	83	84	85	86	87	88	89
70	71	72	73	74	75	76	77	78	79
60	61	62	63	64	65	66	67	68	69
50	51	52	53	54	55	56	57	58	59
40	41	42	43	44	45	46	47	48	49

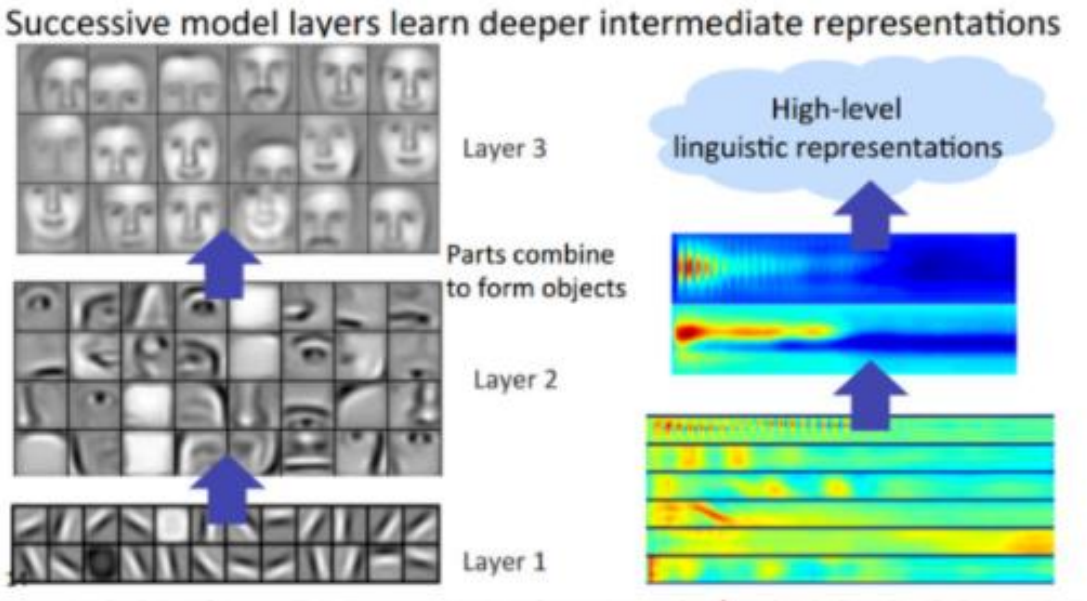




Convolutional Neural Network (Yann Lecun, Geoffrey Hinton)







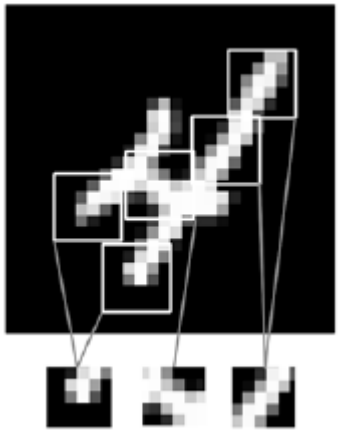
In Image recognition , CNN model identify the **unique features** very specific to particular image Class (Car, Mobile, Cat, Dog etc) by first **create low level of features** such as **edge & curves**.

CNN Learning

- For a small image of 64X64X3, there are **high number of input features** of 12288 variables (each pixel is a relevant variable) and for that Fully connected Neural Network has to estimate the **total 12288 weights for each Neuron** in next layer.

- The **convolution operation** brings a solution to this problem as it **reduces the number of free parameters**, allowing the network to be deeper with fewer parameters.

- **ANN** learns **global patterns** in their input feature space(patterns involving all pixels), whereas **CNN** Learns **local patterns** (edges, textures etc).

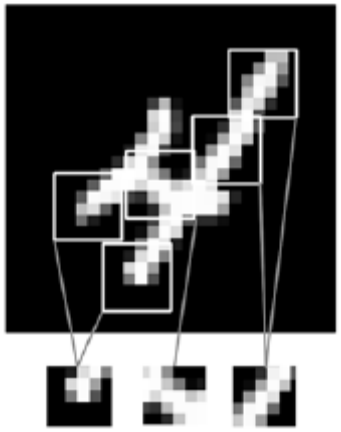


Images can be broken into local patterns such as edges , textures etc.

CNN Local Pattern Learning

Translation Invariant

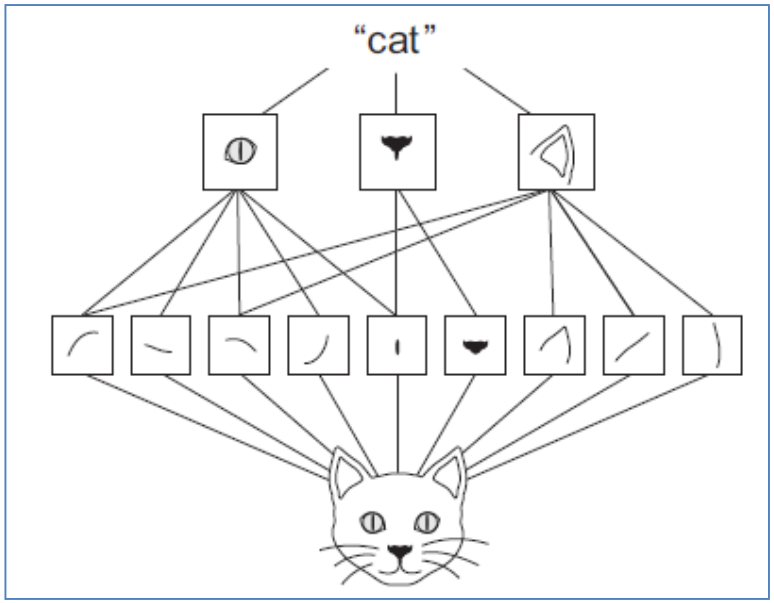
- Invariance means, recognize an object as an object, even when it appears varies in some way.
- After learning a certain patterns in an image, CNN can recognize it anywhere.
- ANN would have to learn the pattern anew if it appeared at a new location.
- This makes CNN data efficient when processing images.
- The visual world is fundamentally translation invariant.



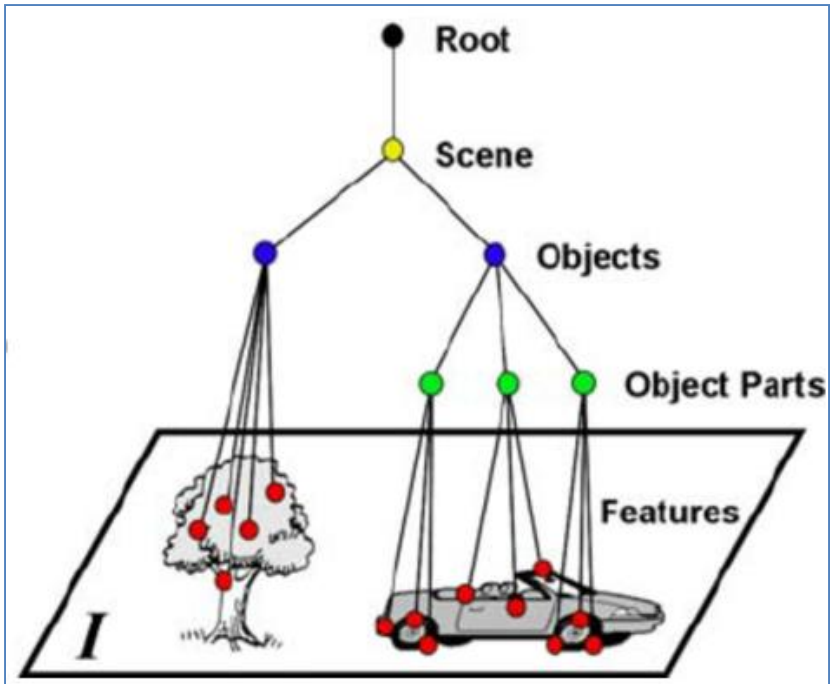
CNN Local Pattern Learning

Spatial Hierarchies

- CNN learn spatial hierarchies of visual image.
- A first CNN layer will learn small local patterns such as edges, textures etc.
- Second CNN will learn larger patterns made of the features from the first layer, and so on.
- This allows CNN to efficiently learn increasingly complex and abstract visuals.



Hyper local edges combines into local objects such as eyes or ears, which combines high level concepts such as "Cat".

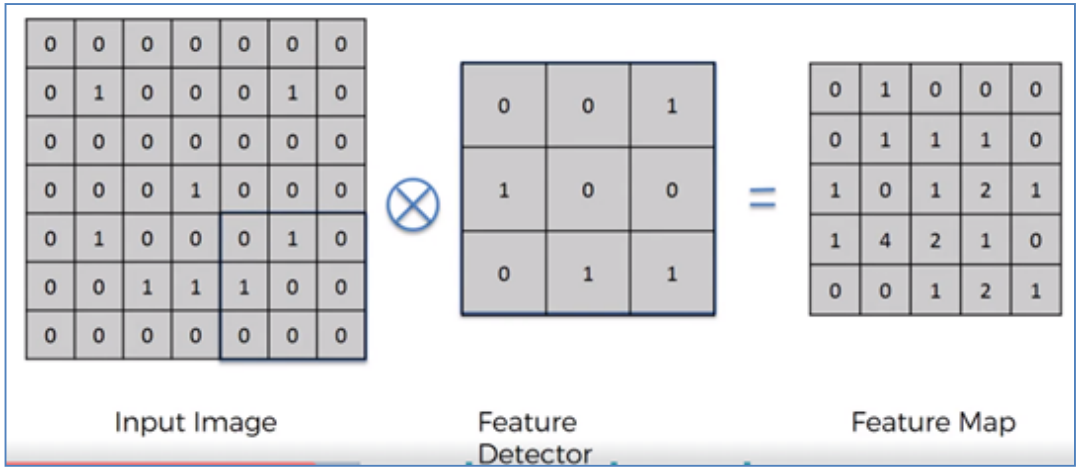


An illustration of the hierarchical spatial patterns present in an image.

Convolution Step

Convolution Steps:

- **Feature detector** will be multiply over part of input image to get the **Feature Map** (high number).
- Each **Feature Map** get some **highest number** that contain the **specific patterns** in input image.
- Convolution steps will use different feature detectors to generate **low level of Features** like **edges** and **curves**.



Input Image = 7X7
Feature Detector = 3X3
Feature Map = 5X5

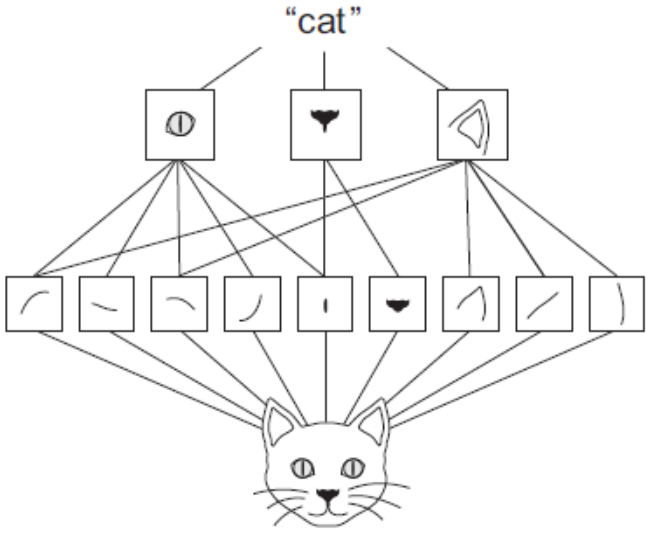


Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as "cat."

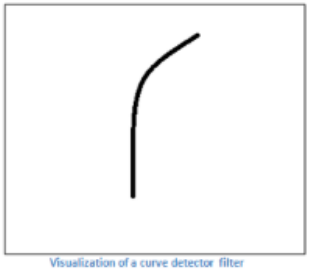
Convolution Step

Convolution Example: Edge detection convolution filter

Original image

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

High value of Multiplication Sum
(50*30 + 50*30 + ...) = Large Value

Visualization of the filter on the image

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Low value of Multiplication Sum
(0*30 + 0*30 + ...) = Low Value

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

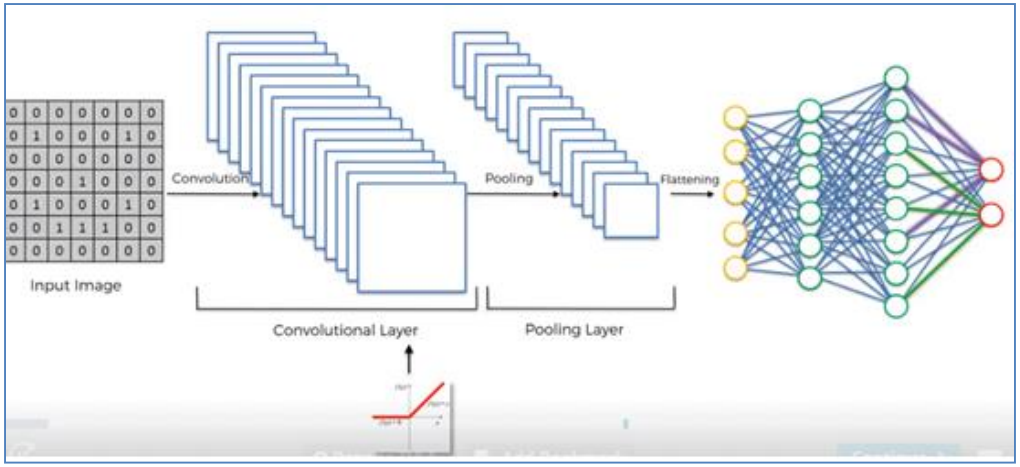
Pixel representation of receptive field

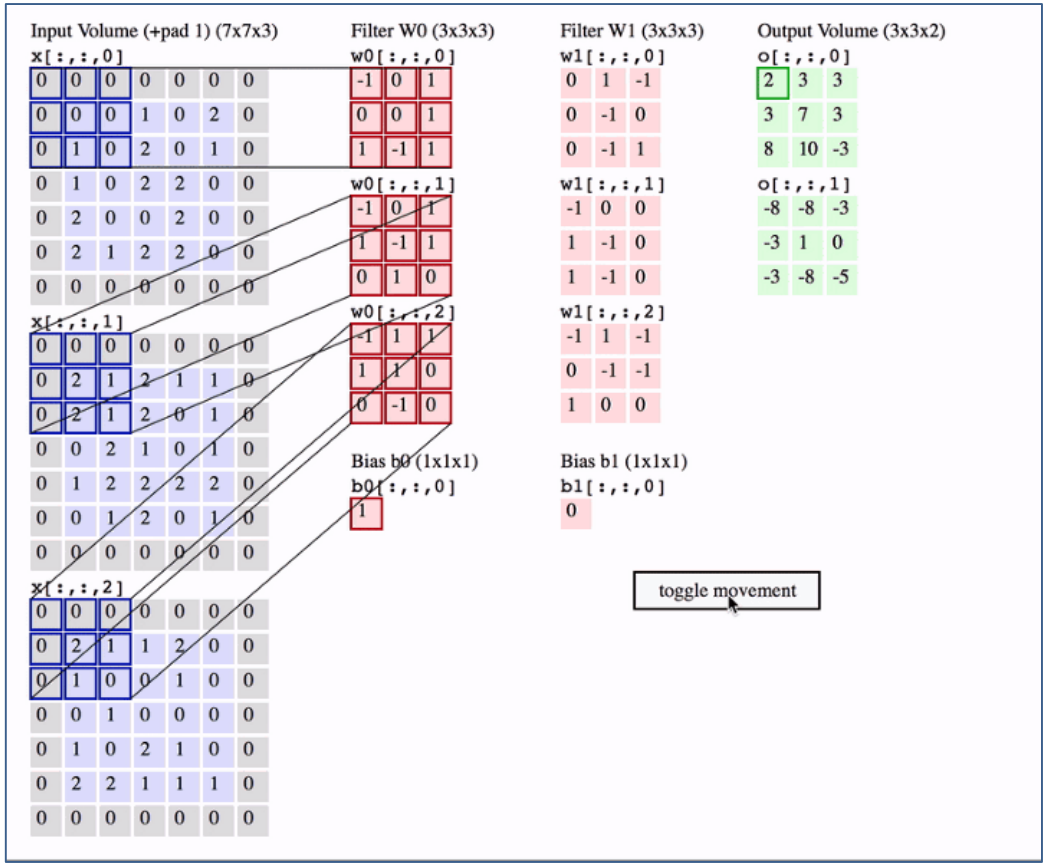
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

In the above example, after applying the convolution filter of curve detector, it will generate the feature Map where it will show the areas in which there most likely to be curve in the picture.





- **Feature detector** will slide over the entire input image and by a **Stride** and compute the **Dot Product** between **receptive field** and **filter** get the **Feature Maps.**

- **Feature Detector** or **Kernel** or **Filter** (Typical filter 5x5x3)

- **Feature Map** or **Convolved Map** or **Activation Map**

- **Zero-padding** – adds Zeros around the outside of input so that convolution should not loose the information at the borders layer.

- **Output Convolved Map Size** (W2 , H2, D2)

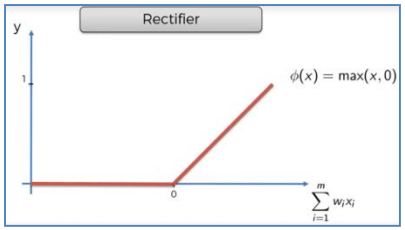
Input Image = 5 X 5 X 3 : W1 = 5, H1=5, D1=3
Two Filter = (3X 3 X 3) x 2, D2 = 2

Receptive Field(F) = 3
Zero-Padding (P) = 1
Stride (S) = 2 (Filter is shifting by 2 pixel amount)

W2 = (W1 – F + 2P)/ S + 1 = (5- 3 + 2*1)/2 + 1 = 3
H2 = (H1 - F + 2P) S + 1 = (5 - 3 + 2*1)/2 + 1 = 3
D2 = 2

Output = (3 x 3 x 2)

ReLu and Pooling

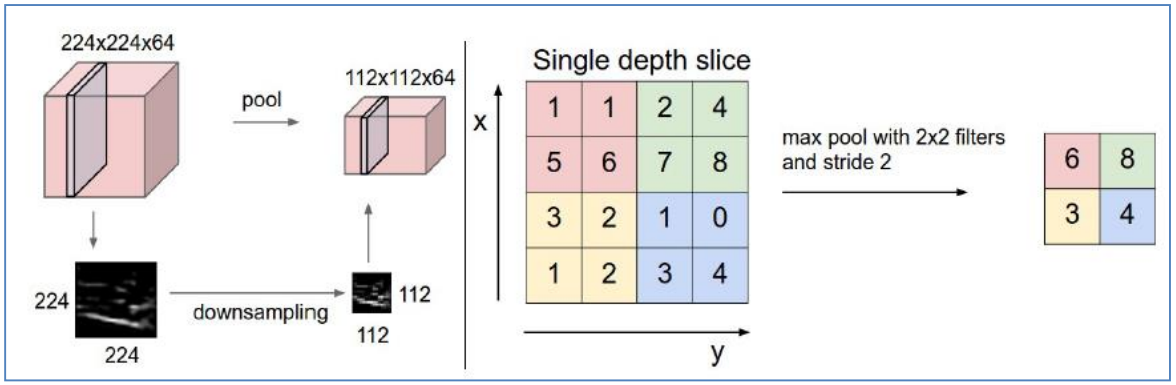


Relu (Rectifier Liner Unit) :
A non-linearity layer (ReLu) activation function that takes the feature map generated by the convolutional layer and creates the activation map as its output.

In CNN, ReLu has significant advantages as compare to other activation functions.
(1) ReLU propagate the **gradient efficiently** and therefore **reduce** the likelihood of a **vanishing gradient problem.**
(2) ReLu threshold **negative values to zero**, and therefore solve the **cancellation problem** as well as result in a much more **sparse activation** volume at its output and sparsity provide robustness to small change in input noise.

Pooling :

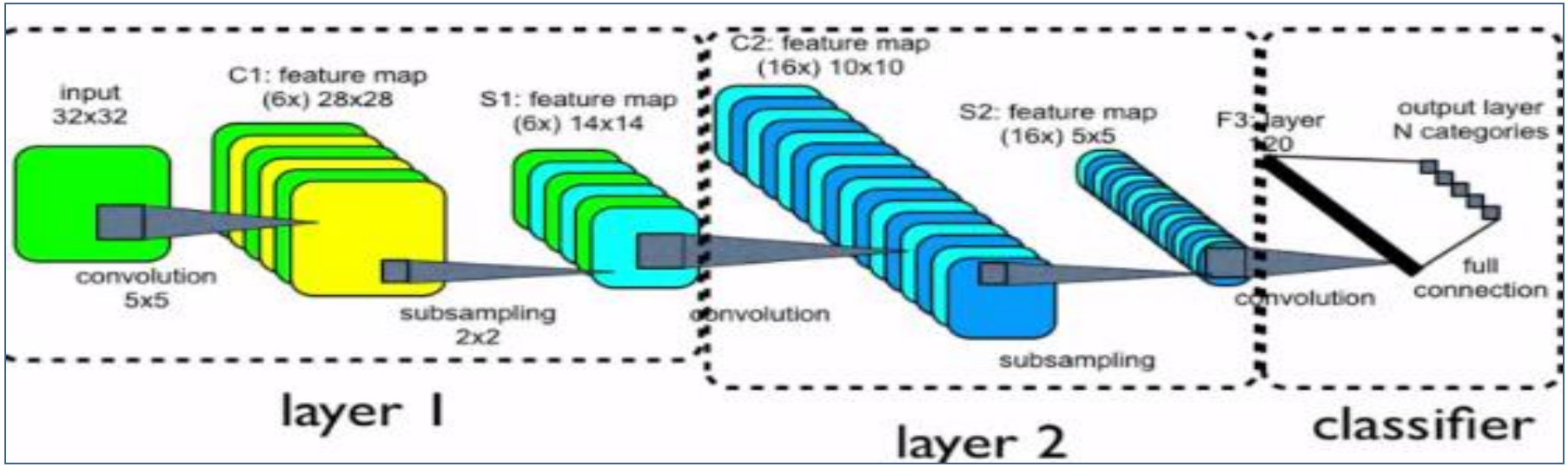
- Pooling is to **reduce the size of feature map** without **loosing the information**.
- Its function is to progressively **reduce the spatial size** of the representation to **reduce the amount of parameters** and **computation** in the network. It also control the over-fitting.
- There are different type of Pooling like **Max-pooling**, **Average-pooling**, L2-norm pooling.
- Example of Max pooling of 2x2 filter and stride = 2, Max-pooling takes the max value of the 4 pixels.

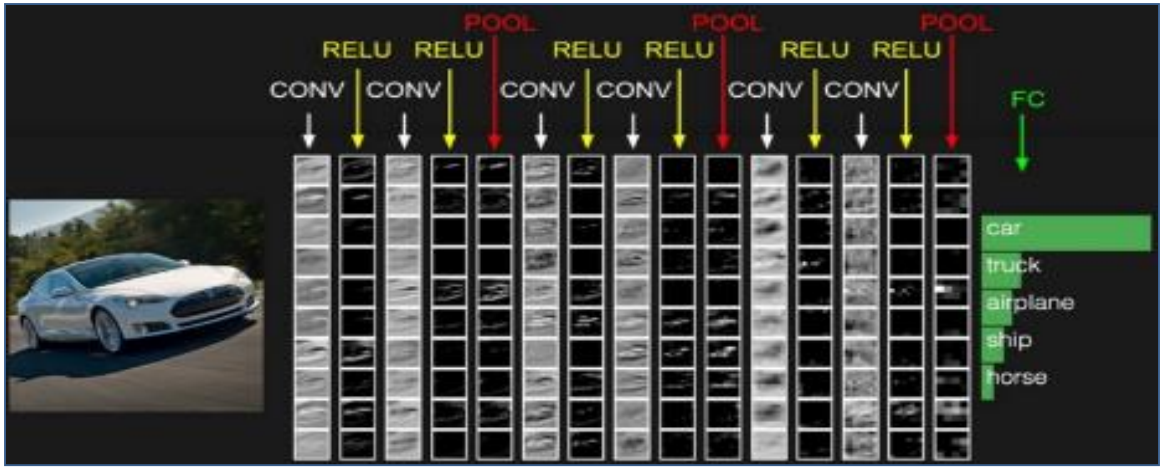


CNN Complete Flow

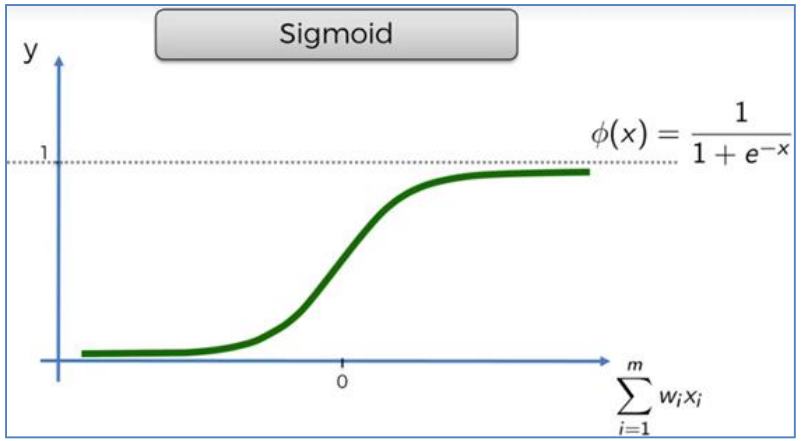
Flattening : is converting the **n-dimensional output** of convolution/pooling step into **1-dimensional vector** that can be used in the next step of **ANN Full connection** modeling for the classification.

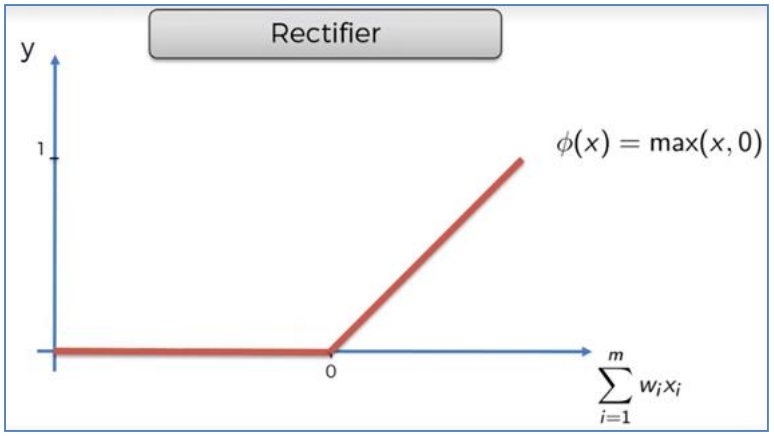
Fully Connected Layer (ANN): This is the last layer of CNN model that uses a **Softmax activation** function for **classifying** the **generated features** of the input image into various classes based on the training dataset.

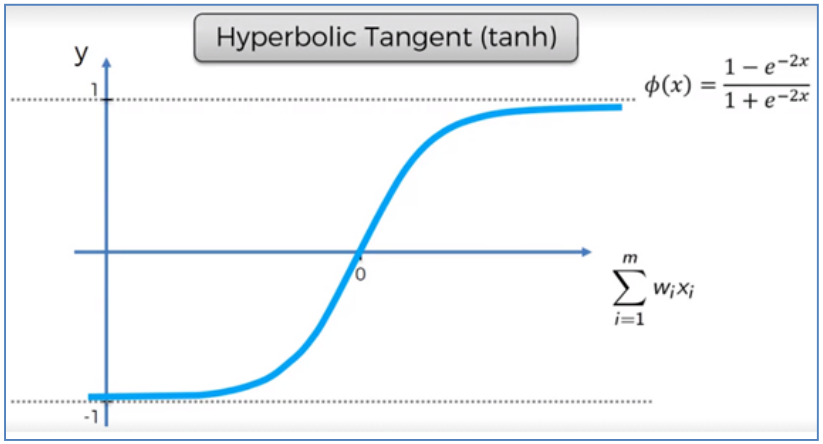


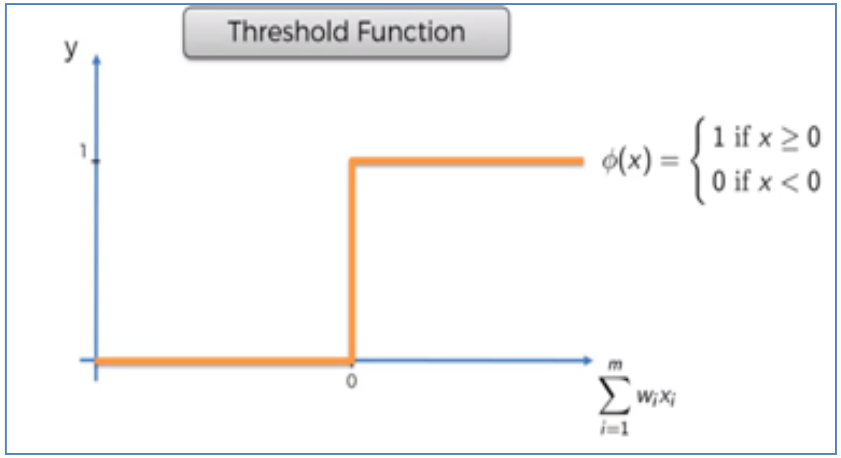


Activation Function









Neural Network Classification

- **Two key architecture** decisions that you need to make to make your model:
  (i) how many **layers** you're going to use
  (ii) how many "**hidden units**" you will chose for each layer

- Pre processing of predictors to standardize them before applying to neural network for best performance.

- Categorical variables need to transformed into dummy variables before applying to neural network.

- Neural Network back propagation weights updating gets stop when
  (i) when misclassification error reached to threshold.
  (ii) when the limit on the number of training epoch is reached.

- To avoid the over fitting, limit the number of epochs and not to over train the data and keep measuring the validation error after each epoch and select optimum epoch based on minimum validation error.

Neural Network Classification

- **Two key architecture** decisions that you need to make to make your model:
  (i) how many **layers** you're going to use
  (ii) how many "**hidden units**" you will chose for each layer
- Pre processing of predictors to standardize them before applying to neural network for best performance.
- Categorical variables need to transformed into dummy variables before applying to neural network.

MLP(Multiple Perceptron NN) Tuning Parameters:

(i) **hidden_layer_sizes :** Number of hidden layer in multiple NN **(default = 100)**

(ii) **activation** : Activation function for the hidden layer. {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'
'**identity**', no-op activation, useful to implement linear bottleneck, returns f(x) = x
'**logistic**', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
'**tanh**', the hyperbolic tan function, returns f(x) = tanh(x).
'**relu',** the rectified linear unit function, returns f(x) = max(0, x), It gives an output x if x is positive and 0 otherwise.

(iii) **Optimizers (Solver) :** Stochastic Gradient Descent (SGD), ADAM and RMSprop.
solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'-stochastic gradient descent.
learning rate : Learning rate to update weights {'constant', 'invscaling', 'adaptive'}, default 'constant'
(invscaling – gradually decrease scaling exponent of 'power_t', adaptive – decrease in case of no tol improvement
Loss Function: For regression - Mean Squared Error (MSE), For classification cross-entropy function.
Cross Entropy - $$E = -\sum_{i=1}^{nout} (t_i \log(y_i) + (1 - t_i) \log(1 - y_i))$$

(v) **Tolerance(tol) :** Tolerance for the optimization (default = .0001)

(vi) **Maximum iteration (max_iter)**: The solver iterates until convergence (determined by 'tol') or this number of iterations.(default = 200)

(vii) **validation_fraction :** Proportion of training data set for validation for early stopping.
float, optional, default 0.1.

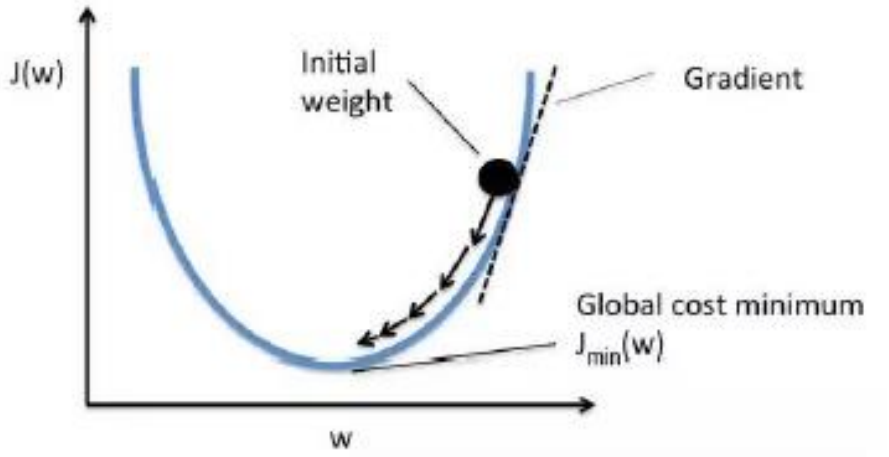
Neural Network Classification

Gradient Descent (Batch) : Updates the weights after each epoch.

- for one or more epochs:
  - for each weight j
    - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta \sum_i (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$

Stochastic Gradient Descent (Iterative) : Updates the weights for each training samples.

- for one or more epochs, or until approx. cost minimum is reached:
  - for training sample i:
    - for each weight j
      - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$



$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

Mean Squared Error

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Cross Entropy

Image Categorization: Testing phase