Image Classification on CIFAR-10

By Arulselvan, Deepen, Ashutosh & Mavez

04/27/2016

Agenda

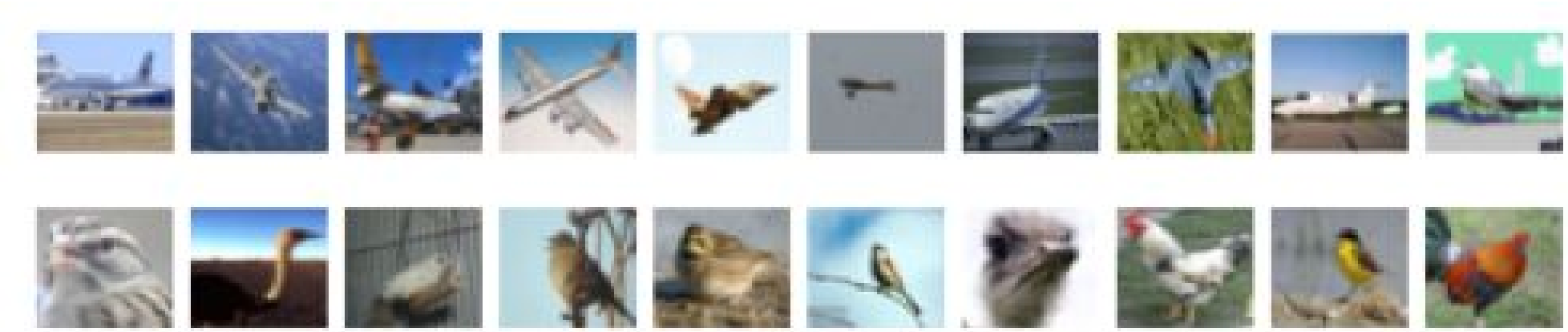
- Introduction
- Goal
- Dataset Description
- Feature Extraction techniques
- Classifiers - KNN
- Classifiers - SVM
- Classifiers - Softmax
- Convolutional Neural Networks - Intuition
- Convolution Layer
- Activation Layer
- Pooling Layer
- Gradient Update Functions
- Evaluation and Analysis of Results
- Future Work

Goal

Build a robust Image Classification system for CIFAR-10 dataset

Why is this problem interesting and challenging?

- Computer Vision has become ubiquitous in our society, with applications in search, image understanding, mapping, medicine, drones, and self-driving cars.
- Getting the granularity for classification is hard. The objects within classes are extremely varied. For example, the bird class contains many different types of birds(big and small).
- Deal with different orientations, partial objects in images, magnified objects etc.,



Cifar-10 Dataset

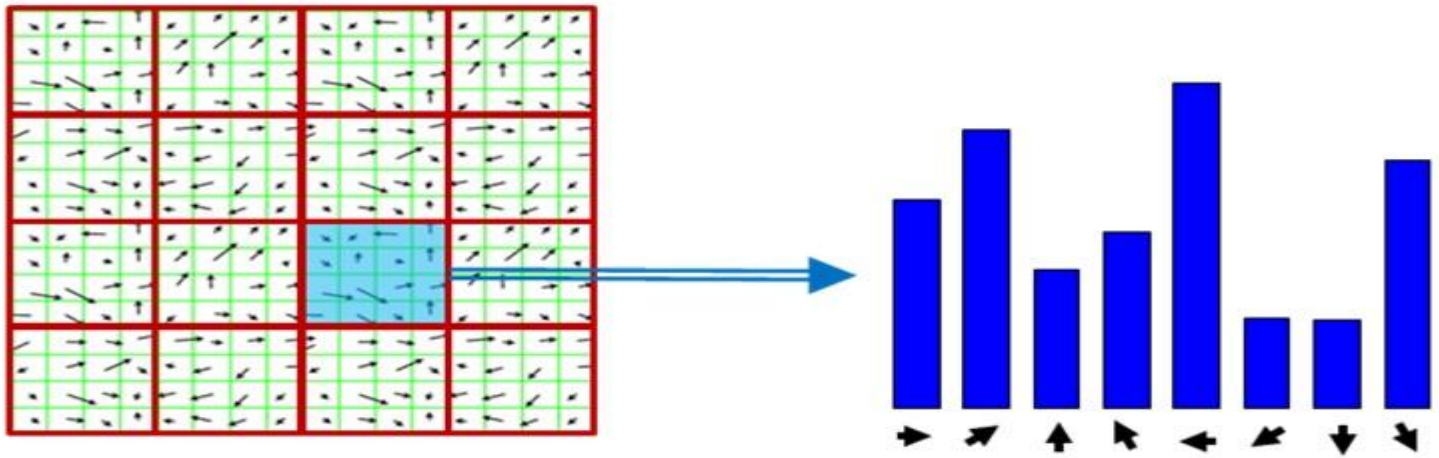
- **Type**: Image(Color: RGB)
- **Image Size:** 32x32 pixels. Each pixel has RGB information
- **Feature Vector:** 1 x 32 x 32 x3 = 1 x 3072
- **Description**:
  - Dataset consists of 60000 32x32 color images of 10 classes
  - Each class consists of 6000 images
  - 50000 training images and 10000 test images
  - The classes are completely mutually exclusive

Feature Descriptors

Histogram of Oriented Gradients (HOG)

Feature Descriptors - HOG

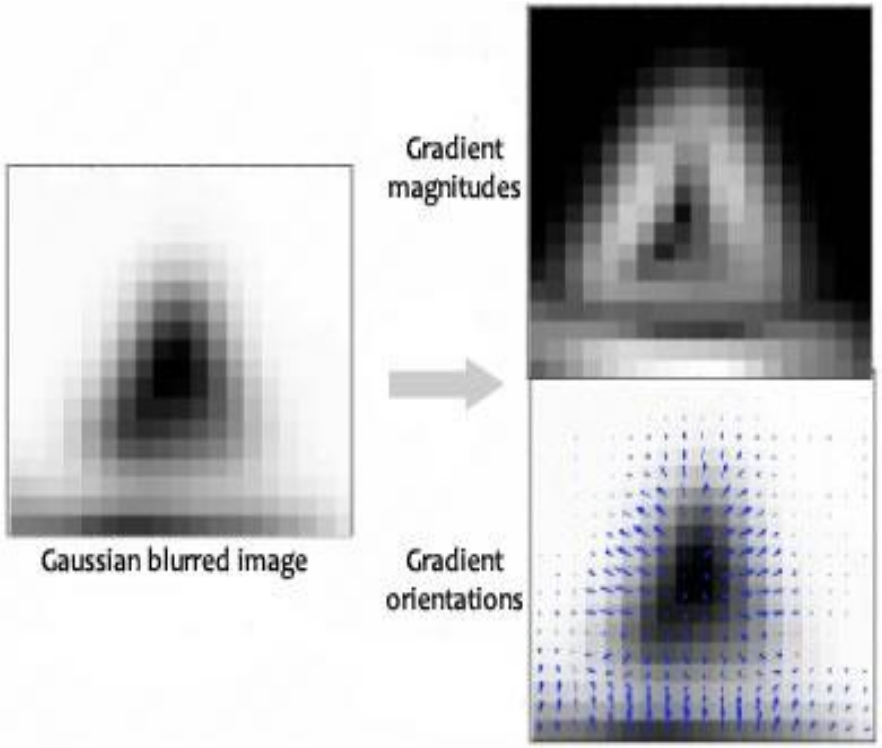
- HOG captures the texture of the image.
- Image is of size: 32x32 pixels.
- Divide the image into blocks of size: 8x8. (16 blocks in total)
- Compute the gradient orientation for each block.(0 and 180 degrees)
- Create 9 bins by dividing the orientation range.



Source: Mubarak Shah's lecture for University of Central Florida

Feature Descriptors - HOG

- Let I(i,j) represent the pixel in the i'th row and j'th column.
- Gradient of an image
  - Gradient in x direction: dx = I(i-1,j) - I(i+1,j)
  - Gradient in y direction: dy = I(i, j-1) - I(i,j+1)
- Gradient Magnitude = Euclidean distance(dx,dw)
- Gradient Orientation = arctan(dy/dx)
- Concatenate the histograms = 9*4*4 = 144
- Each image is represented with 144 feature vector
- Feature vector got reduced from 1x3072 to 1x144.



Feature Descriptors

Histogram of Color Bins

Feature Descriptors - Color Histogram

Color Histogram

- Use the color distribution of images as a feature
- Convert RGB colors to HSV(Hue, Saturation, Value) and divided them into 10 range bins
- HSV separates luma, or the image intensity, from chroma or the color information.
- Image is represented by a 1x10 vector.

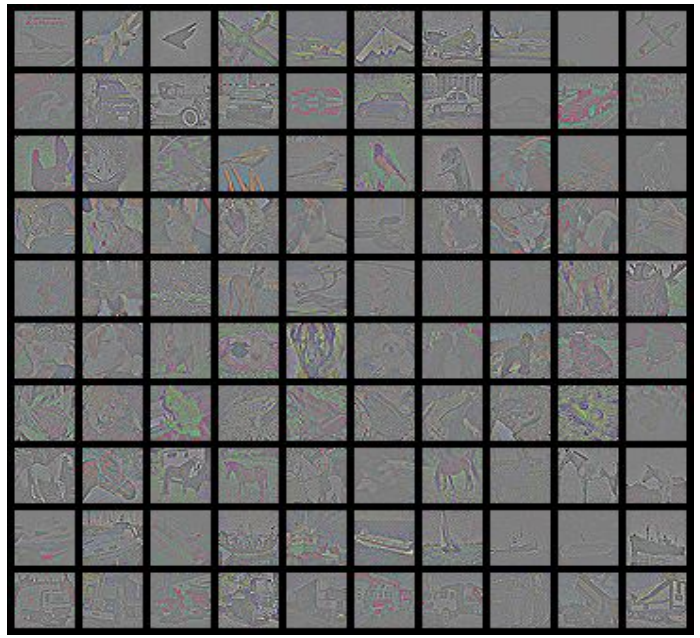
Feature Descriptors

ZCA Whitening

Feature Descriptors - ZCA

ZCA whitening

- Tiny images exhibit strong correlation between nearby pixels
- Make model to learn irregularities rather than merely learn similarity
- Also a rough model of how biological eyes process the images



Classification Algorithms

K Nearest Neighbors

K Nearest Neighbors

- Measure the euclidean distance between images.
- Compute the K Nearest Neighbors for Images based on their euclidean distance.
- Finding the top k closest images vote on the label in the test image.

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Method	KNN
Original(1x3072)	15.87%
ZCA(1x3072)	12.34
Hog(1x144)	10.12%
Histogram(1x10)	7.78%
Hog and Histogram(1x154)	6.66%

Classification Algorithms

Linear Classifier with SVM Loss function

SVM loss (Hinge Loss)

Linear Classifier

- Image vector(X) = 1x3072
- Weight Vector(W) = 3072x10
- Bias vector(b) = 1x10
- Score

$$f(x_i, W, b) = Wx_i + b$$

- Loss Function

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

- Mini Batch Stochastic Gradient Descent

Method	SVM
Original	33.84%
ZCA	31.60%
Hog	21.16%
Histogram	17.48%
Hog and Histogram	16.20%

Classification Algorithms

Softmax Regression

Sofmax loss

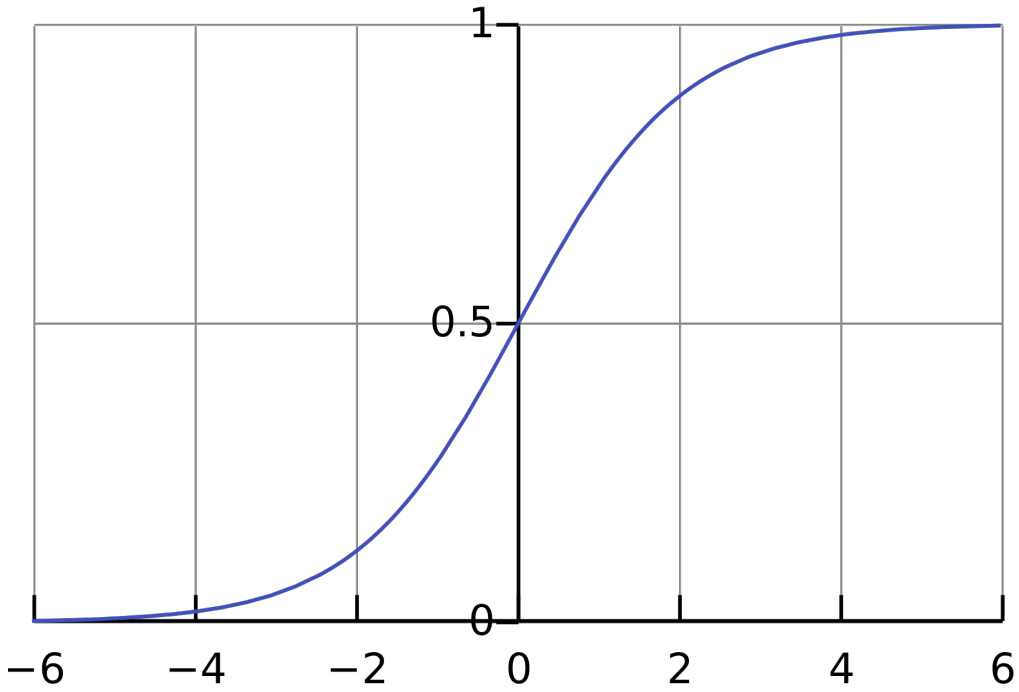
Linear Classifier

- Image vector(X) = 1x3072
- Weight Vector(W) = 3072x10
- Bias vector(b) = 1x10
- Score

$$f(x_i, W, b) = Wx_i + b$$

- Loss Function $$L_i = -log(\frac{e^{w_{y_j}^T x^{(i)}}}{\sum_{j=1}^{k} e^{w_j^T x^{(i)}}})$$

- Mini Batch Stochastic Gradient Descent



Results

Method	Softmax	KNN	SVM
Original	39.22%	15.87%	33.84%
ZCA	10.65%	12.34	31.60%
Hog	10.77%	10.12%	21.16%
Histogram	11.46%	7.78%	17.48%
Hog and Histogram	10.11%	6.66%	16.20%

Classification Algorithm

Convolutional Neural Network

What is Convolution?

A convolution is an integral that expresses the amount of overlap of one function **‘g’** as it is shifted over another function **‘f’**. It therefore "blends" one function with another.

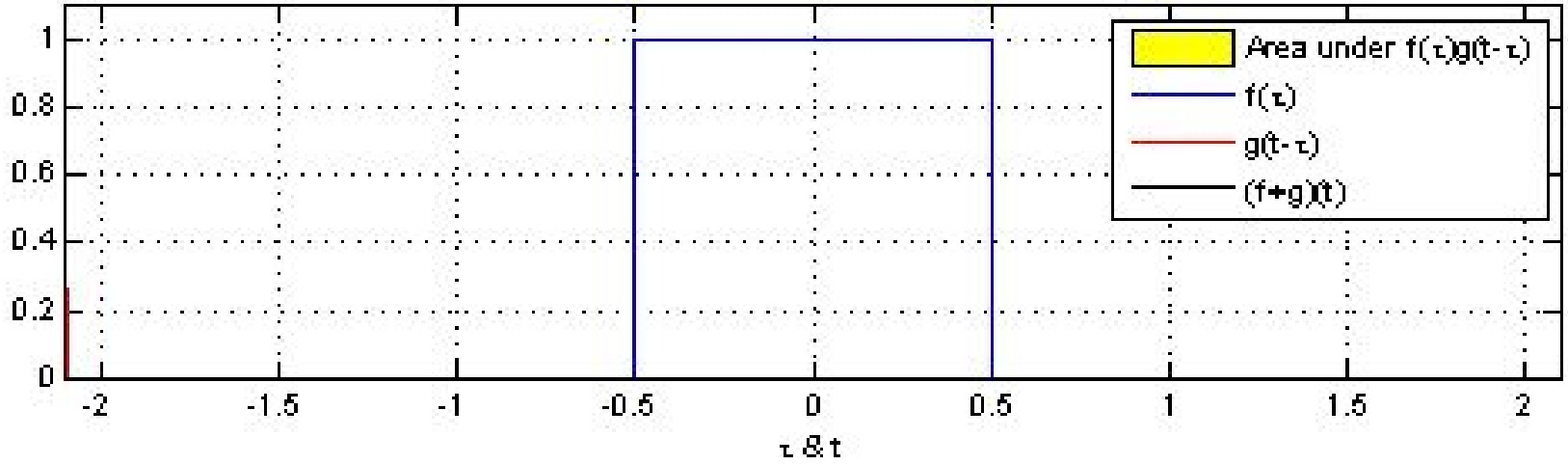
$$(f * g)(t) = \int f(\tau)\, g(t - \tau)\, d\tau$$

f - Original Function

g - Kernel or Filter Function

Convolution Example

$$(f * g)(t) = \int f(\tau)\, g(t - \tau)\, d\tau$$



Source: Wikipedia

Convolution 1D Example

Original Function $f = \boxed{10 \mid 50 \mid 60 \mid 10 \mid 20 \mid 40 \mid 30}$

Kernel Function $g = \boxed{1/3 \mid 1/3 \mid 1/3}$

Source: CS1114 Cornell University

Convolution 1D Example

Original Function

$$f = \begin{array}{|c|c|c|c|c|c|c|} \hline 10 & 50 & 60 & 10 & 20 & 40 & 30 \\ \hline \end{array}$$

Kernel Function

$$g = \begin{array}{|c|c|c|} \hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

Convolving from left to right

10	50	60	10	20	30	40
0	1/3	1/3	1/3	0	0	0

Source: CS1114 Cornell University

Convolution 1D Example

Original Function

$$f = \begin{array}{|c|c|c|c|c|c|c|} \hline 10 & 50 & 60 & 10 & 20 & 40 & 30 \\ \hline \end{array}$$

Kernel Function

$$g = \begin{array}{|c|c|c|} \hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

Convolving from left to right

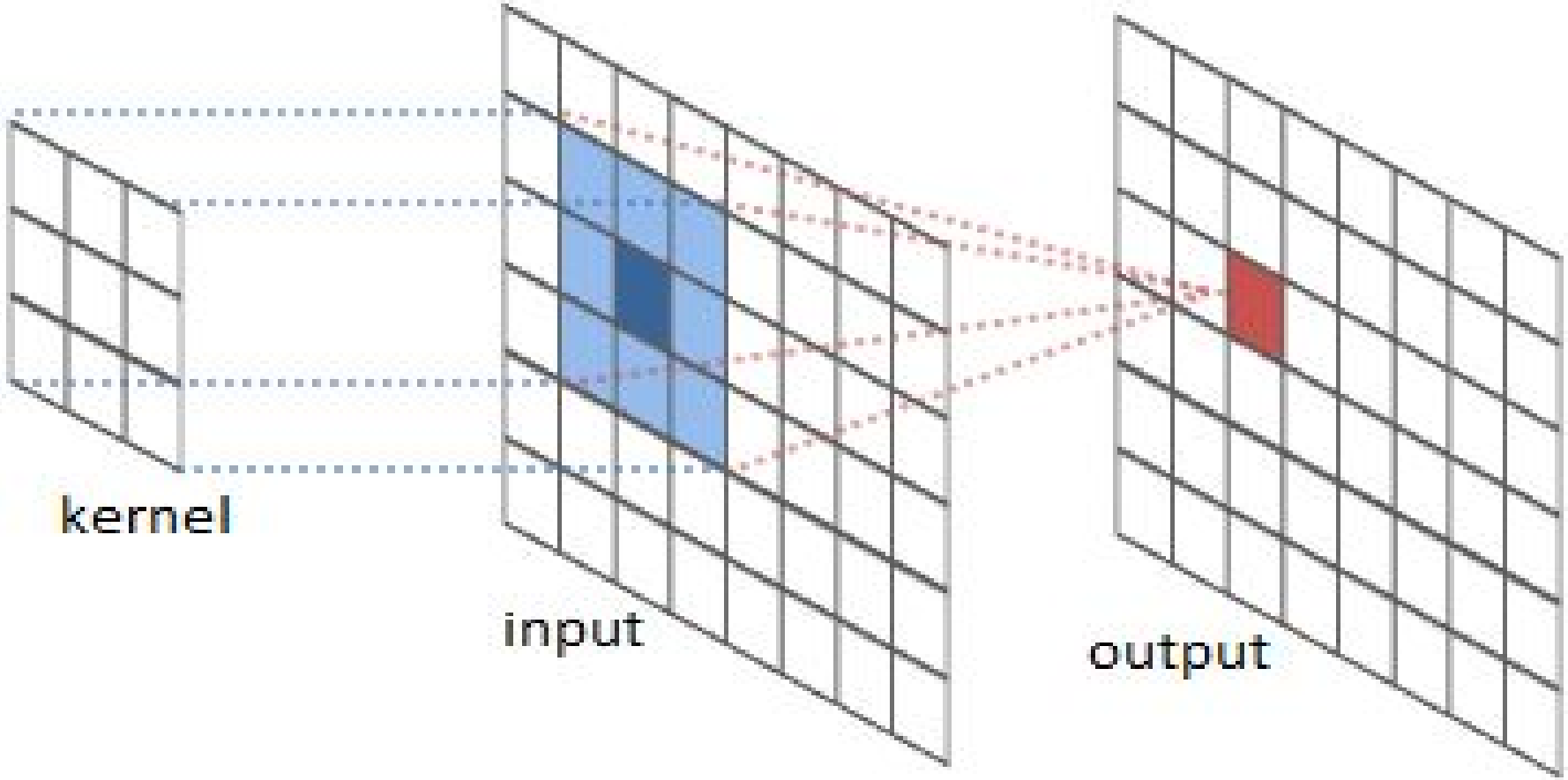
10	50	60	10	20	30	40
0	1/3	1/3	1/3	0	0	0

Result of Convolution

$$h = \begin{array}{|c|c|c|c|c|c|c|} \hline 20 & 40 & 40 & 30 & 20 & 30 & 23.333 \\ \hline \end{array}$$

Source: CS1114 Cornell University

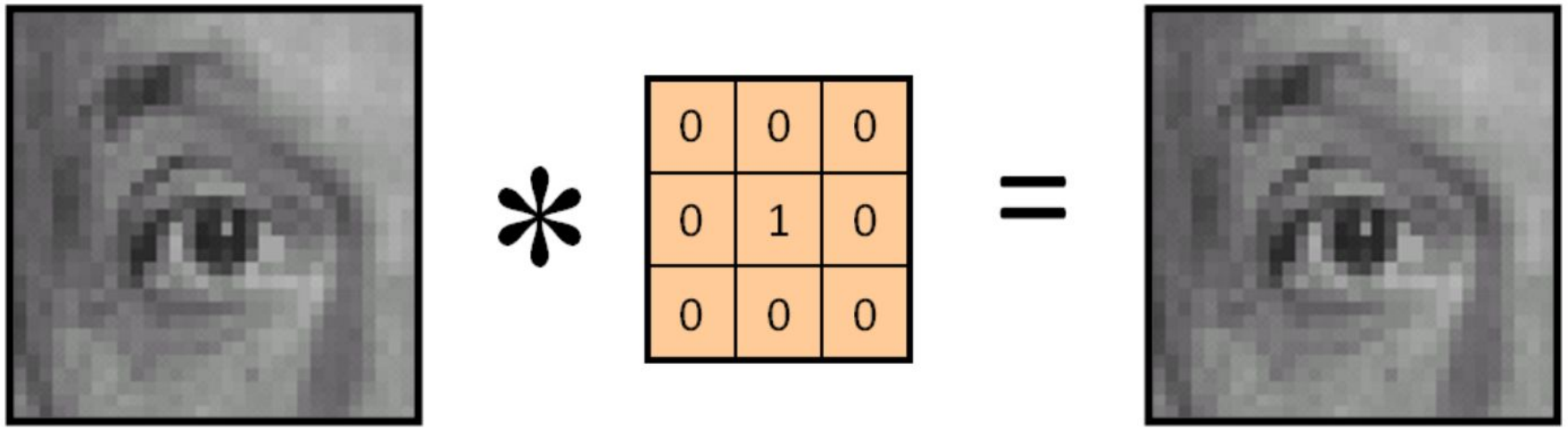
Convolution 2D Example



Source: http://colah.github.io/posts/2014-07-Understanding-Convolutions/

Kernel Functions

- A kernel, convolution matrix, or mask is a small matrix useful for blurring, sharpening, embossing, edge detection, and more.
- By convolving Kernel functions over the image, we can trigger specific features of the image.



- Demo: Image Kernels

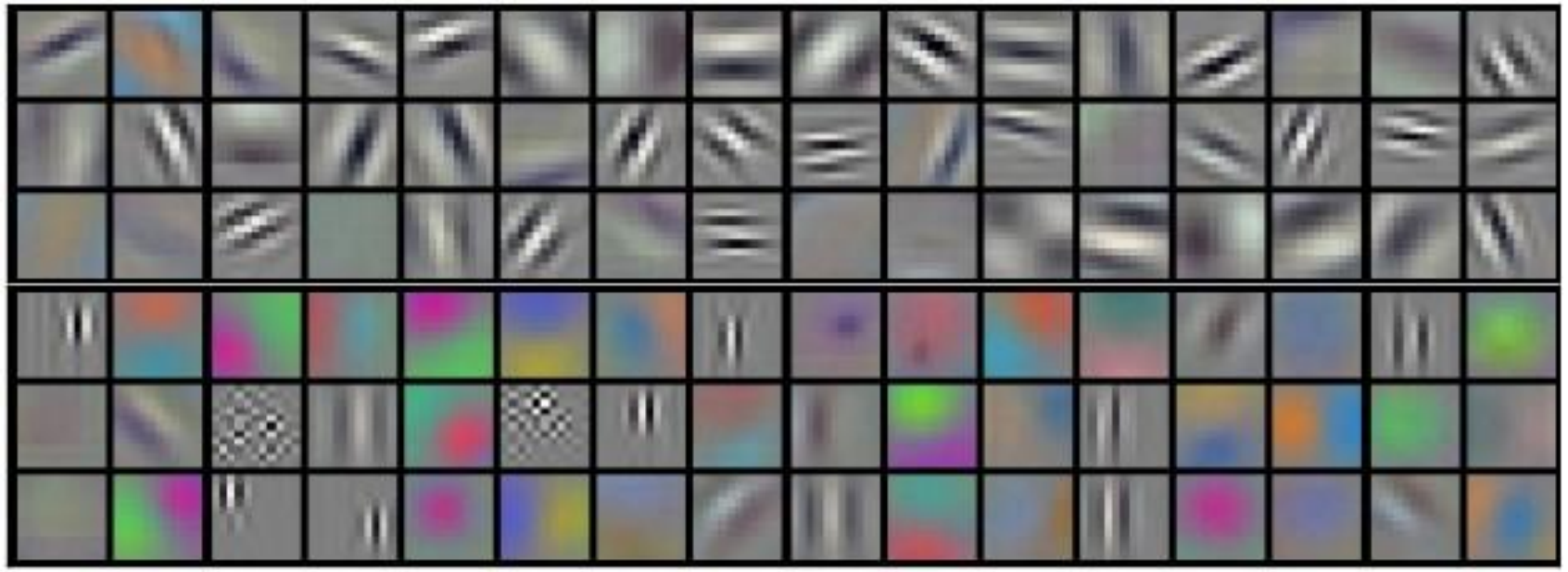
Source: CS1114 Cornell University

Convolutional Neural Network

- Core Idea: Learn a set of kernel functions which when applied over the input image activates specific features of the image that describes the type of the image.

Convolutional Neural Network

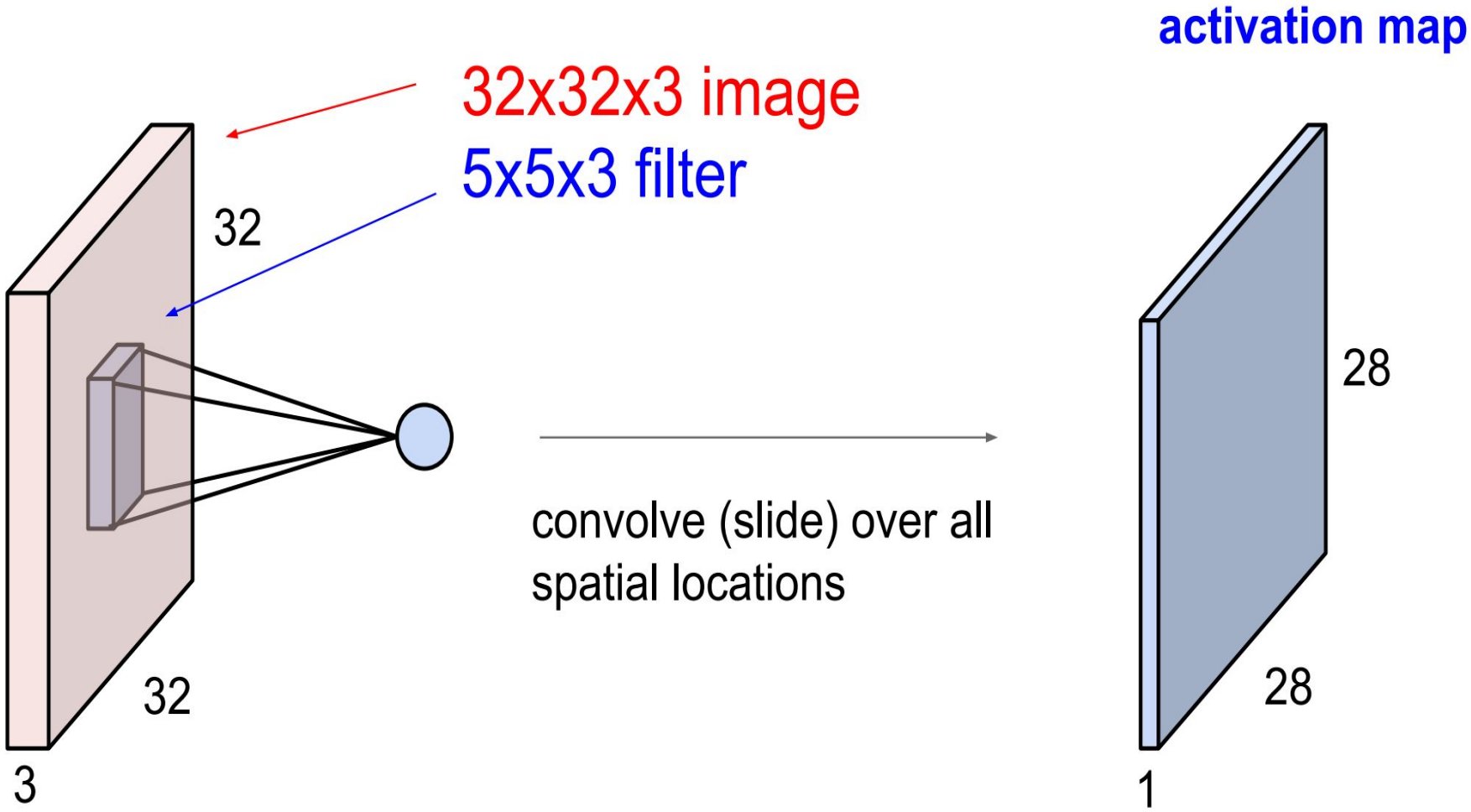
- Core Idea: Learn a set of kernel functions which when applied over the input image activates specific features of the image that describes the type of the image.
- **Kernel Visualization:**



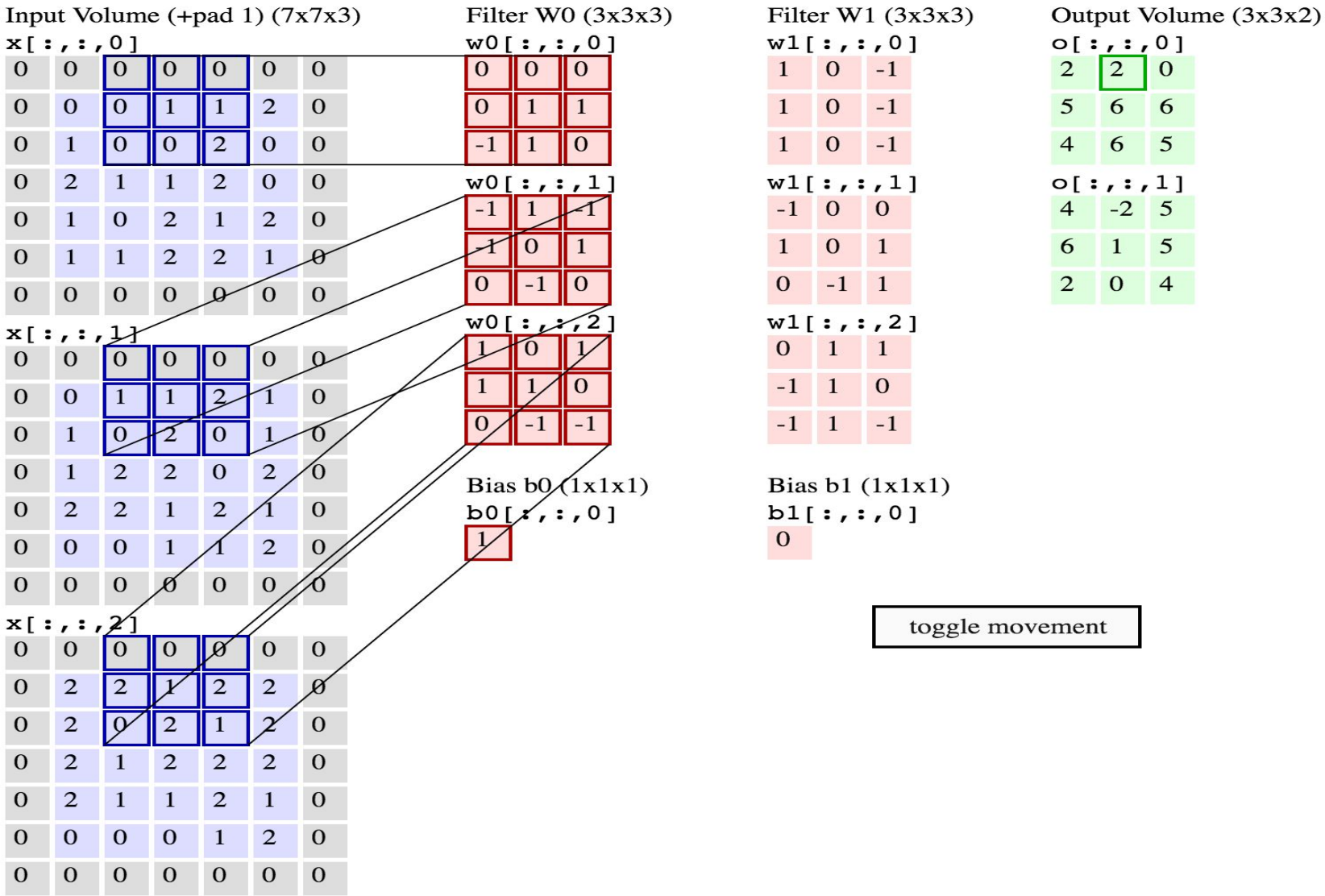
Source: AlexNet paper

ConvNets - Implementation

Convolution on Images - 3D



Convolution Layer - Example



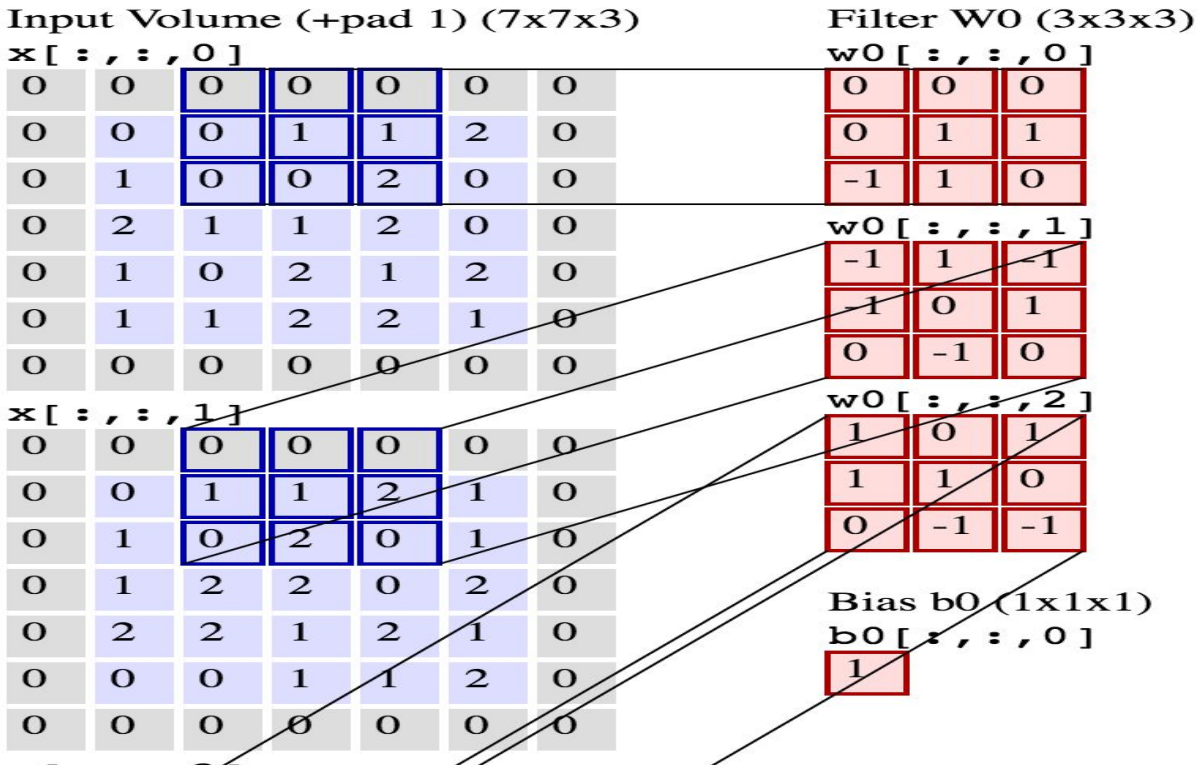
Our Implementation(~25 lines)

```
def forward(self, x):
    """..."""
    N, C, img_height, img_width = x.shape
    F, C, HH, WW = self.W.shape
    stride = self.stride
    pad = self.pad
    H_out = 1 + (img_height + 2 * pad - HH) // stride
    W_out = 1 + (img_width + 2 * pad - WW) // stride

    x_col = np.zeros((N, F, H_out, W_out))
    for img in range(N):
        x_img = x[img]
        npad = ((0, 0), (pad, pad), (pad, pad))
        x_img_padded = np.pad(x_img, npad, mode='constant', constant_values=0.0)
        for filterId in range(F):
            kernel_3d = self.W[filterId]
            bias = self.b[filterId]
            w_start = 0
            for width in range(W_out):
                w_end = w_start + WW
                h_start = 0
                for height in range(H_out):
                    h_end = h_start + HH
                    xin = x_img_padded[:, w_start:w_end, h_start:h_end]
                    x_col[img][filterId][width][height] = np.sum(xin * kernel_3d) + bias
                    h_start += stride
                w_start += stride
    self.cache = x
    return x_col
```

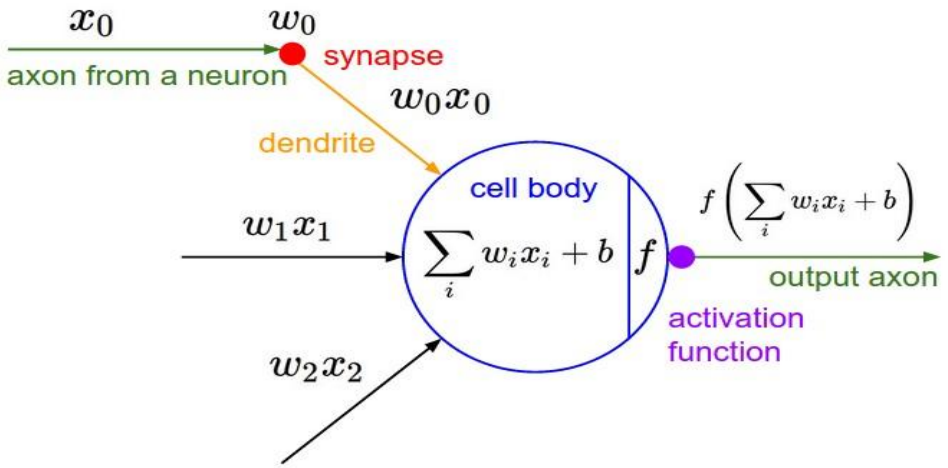
Convolution Layer - Challenges

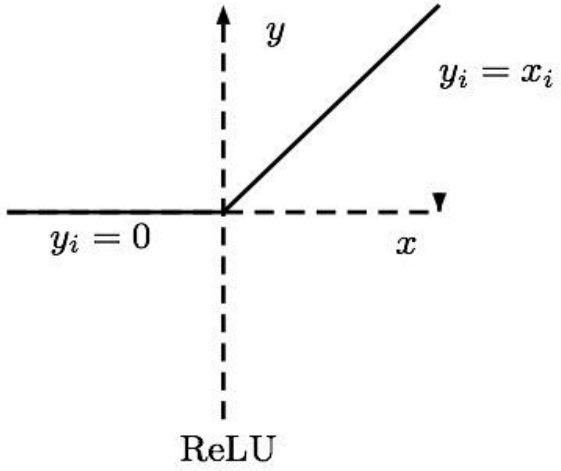
- Implementing a naive convolution layer is easy.
- Efficient Convolution implementation is tough.
- One trick is to flatten the strides into column vectors and perform a dot product.
- Popular solution: GPU implementation

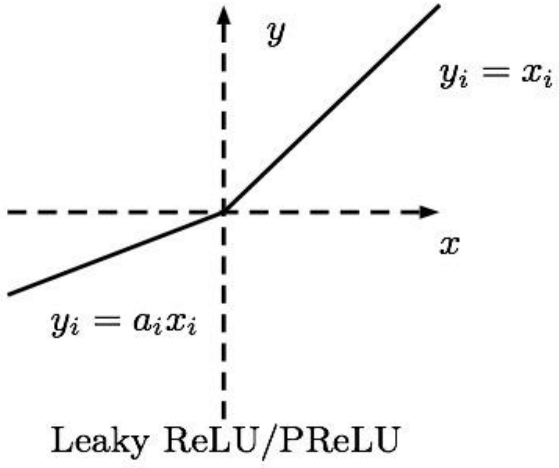


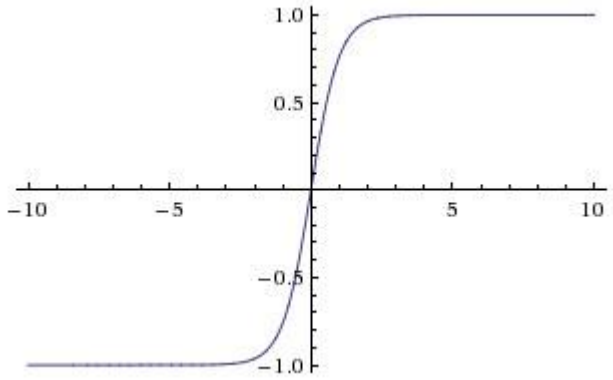
Activation Functions

- Rectifier Linear Unit
- Leaky ReLU
- Tan H activation



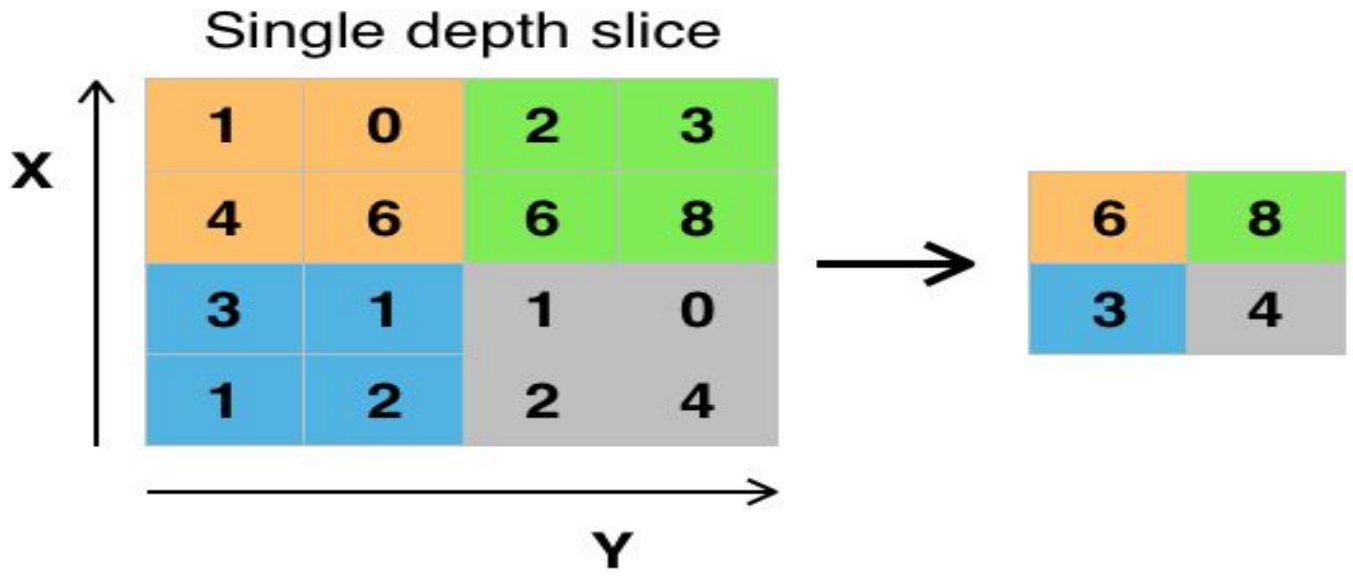






Pooling Layer - Max Pooling

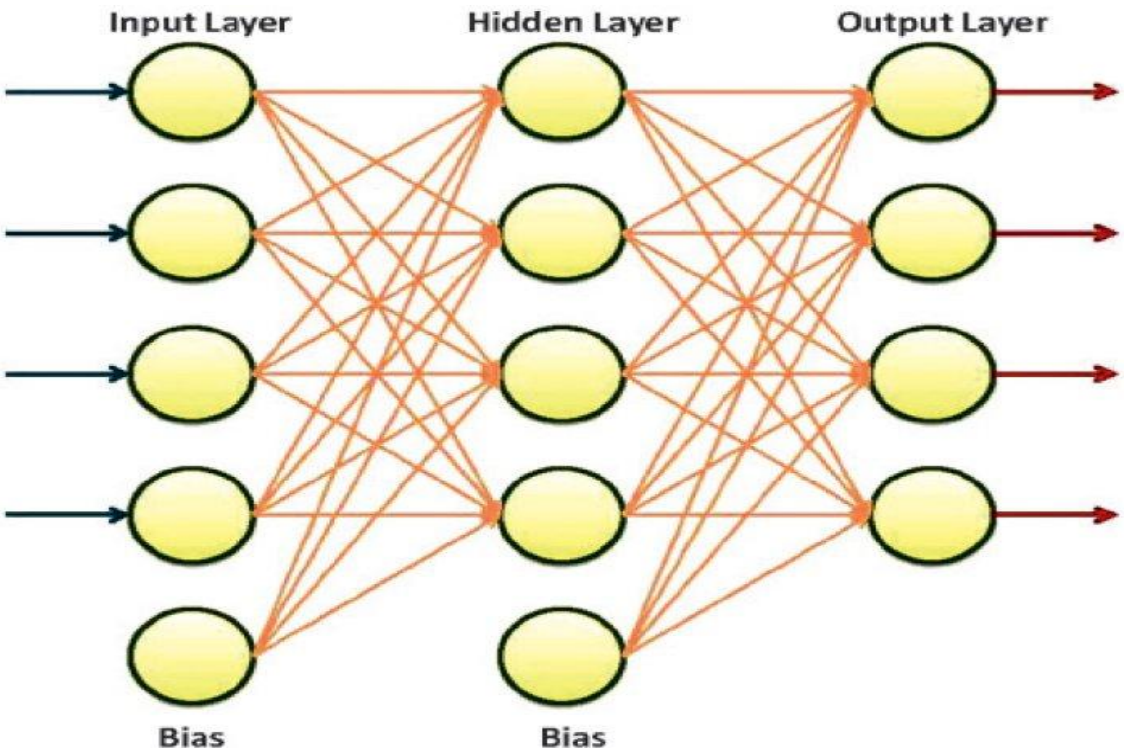
- Reduce the spatial size of the representation
- Reduce the amount of parameters and computation in the network
- Control Overfitting.



Fully Connected Layer

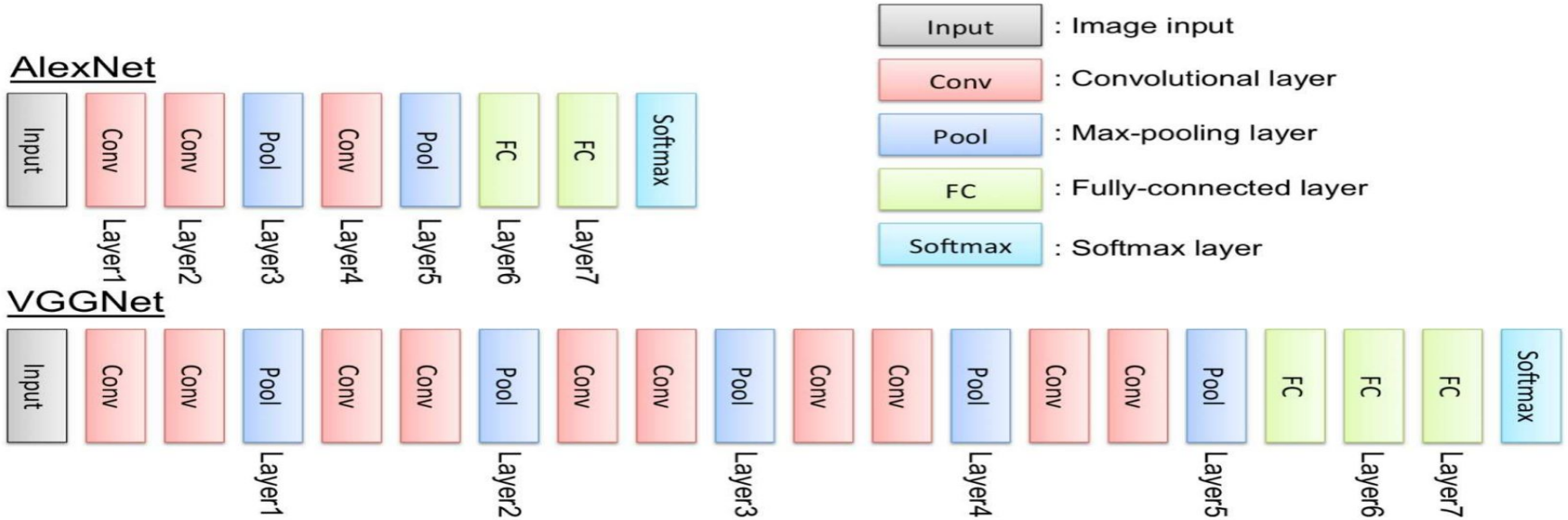
- Vanilla Neural Network
- Each neuron is connected to each neuron in the previous layer.
- Neurons in a single layer function independently and do not share any connections.
- Used as the final output layers of a Convolutional Neural Network

Fully Connected Layer



CNN Architectures

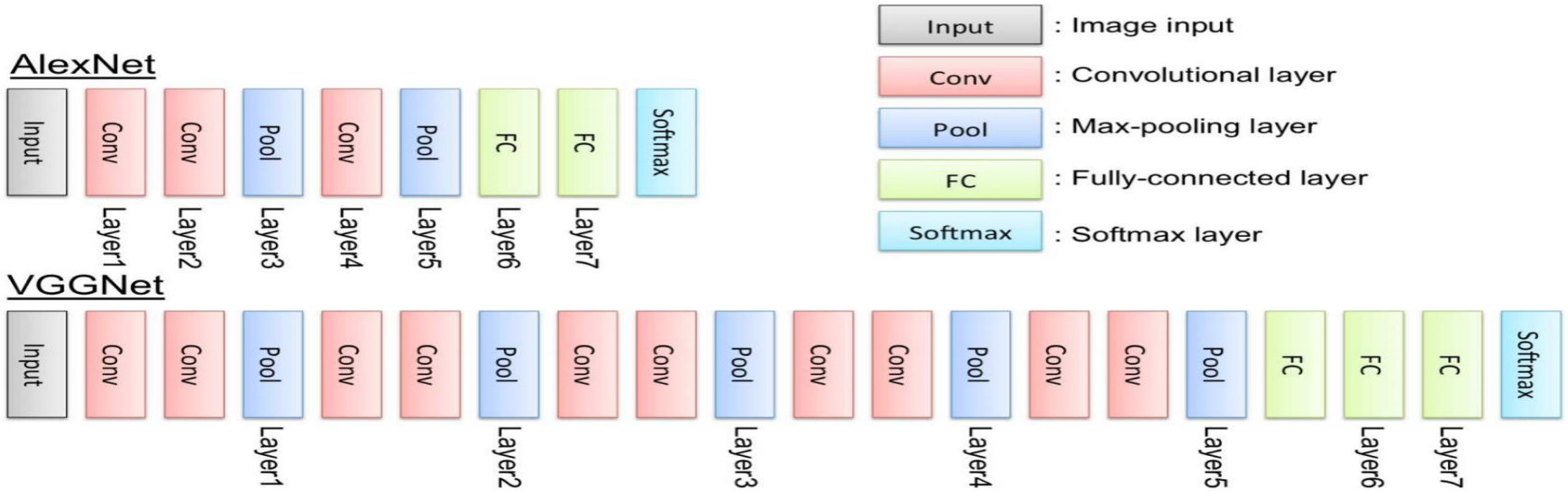
- Popular CNNs - AlexNet and VGGNet



Source: Paper on feature evaluation in AlexNet and VGGNet - http://arxiv.org/pdf/1509.07627.pdf

Our Architecture - Simplified VGG Net

- Popular CNNs - AlexNet and VGGNet



- Our Architecture

Gradient Descent

- Mini Batch Stochastic Gradient descent.
- Momentum based Gradient update function - Adam

Gradient Descent

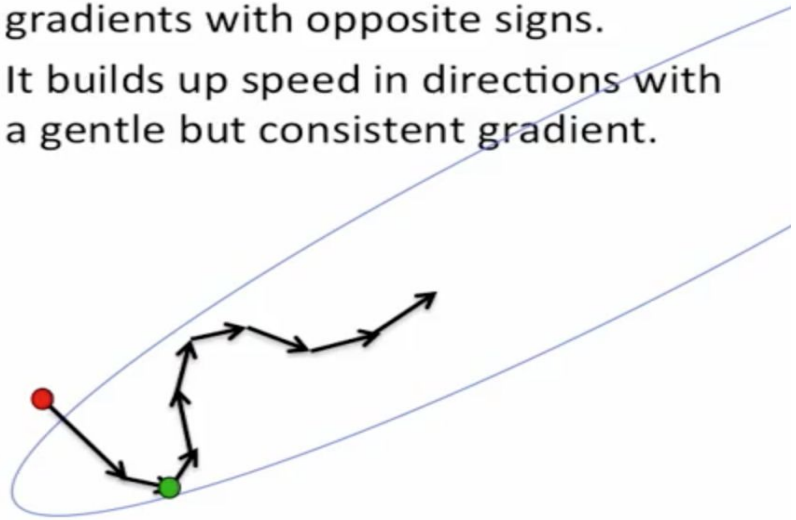
- Mini Batch Stochastic Gradient descent.
- Momentum based Gradient update function - Adam

The intuition behind the momentum method

Imagine a ball on the error surface. The location of the ball in the horizontal plane represents the weight vector.

- The ball starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
- Its momentum makes it keep going in the previous direction.

- It damps oscillations in directions of high curvature by combining gradients with opposite signs.
- It builds up speed in directions with a gentle but consistent gradient.



Source: Geoff Hinton's Coursera class on Optimization algorithms

Adam - Update Algorithm

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Source: Adam Paper - http://arxiv.org/pdf/1412.6980.pdf

Evaluation - Setup

- Training Data: 49000 images
- Validation Data: 1000 images
- Test Data: 10000 images
- Mini-batch gradient descent with batch size(SGD,Adam): 50
- Learning Rate: 0.001(Decayed after each epoch)
- L2 Regularization Strength: 0.05
- Epochs Count: 20
- Calculate Validation accuracy after each epoch. The weights that give the best accuracy is selected for the test dataset.
- The weights of all the convolutional layers and fully connected layers are initialized with unit normal distribution.
- The bias used by the convolution and fully connected layers are initialized with zeros.
- AWS instances: m3.2xlarge and m4.4xlarge

Evaluation - Results - 3Layer CNN

Configurations				Accuracy		
Convolutional Layer1		FC Layer 2	FC Layer 3			
Filter Size	Filter Count	Hidden Layer Size	Hidden Layer Size	Training Set	Validation Set	Test Set
3	16	500	500	60.00%	58.10%	57.37%
	32	500	500	62.23%	57.2%	57.09%
	64	500	500	63.90%	55.80%	56.71%
5	16	500	500	61.34%	57.10%	55.29%
	32	500	500	59.4%	54.37%	53.35%
	64	500	500	58.3%	53.38%	52.44%
7	16	500	500	56.12%	54.23%	51.33%
	32	500	500	55.31%	52.12%	49.2%
	64	500	500	54.32%	48.03%	47.77%

Evaluation - Results - 4 and 5 Layer CNN

Configurations					Accuracy		
Filter Size	Conv Layer1	Conv Layer2	FC Layer3	FC Layer4			
	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	16	32	500	500	81.8%	68.8%	69.71%
	32	64	500	500	69.70%	61.70%	63.24%
5	16	32	500	500	75.3%	66.34%	66.8%

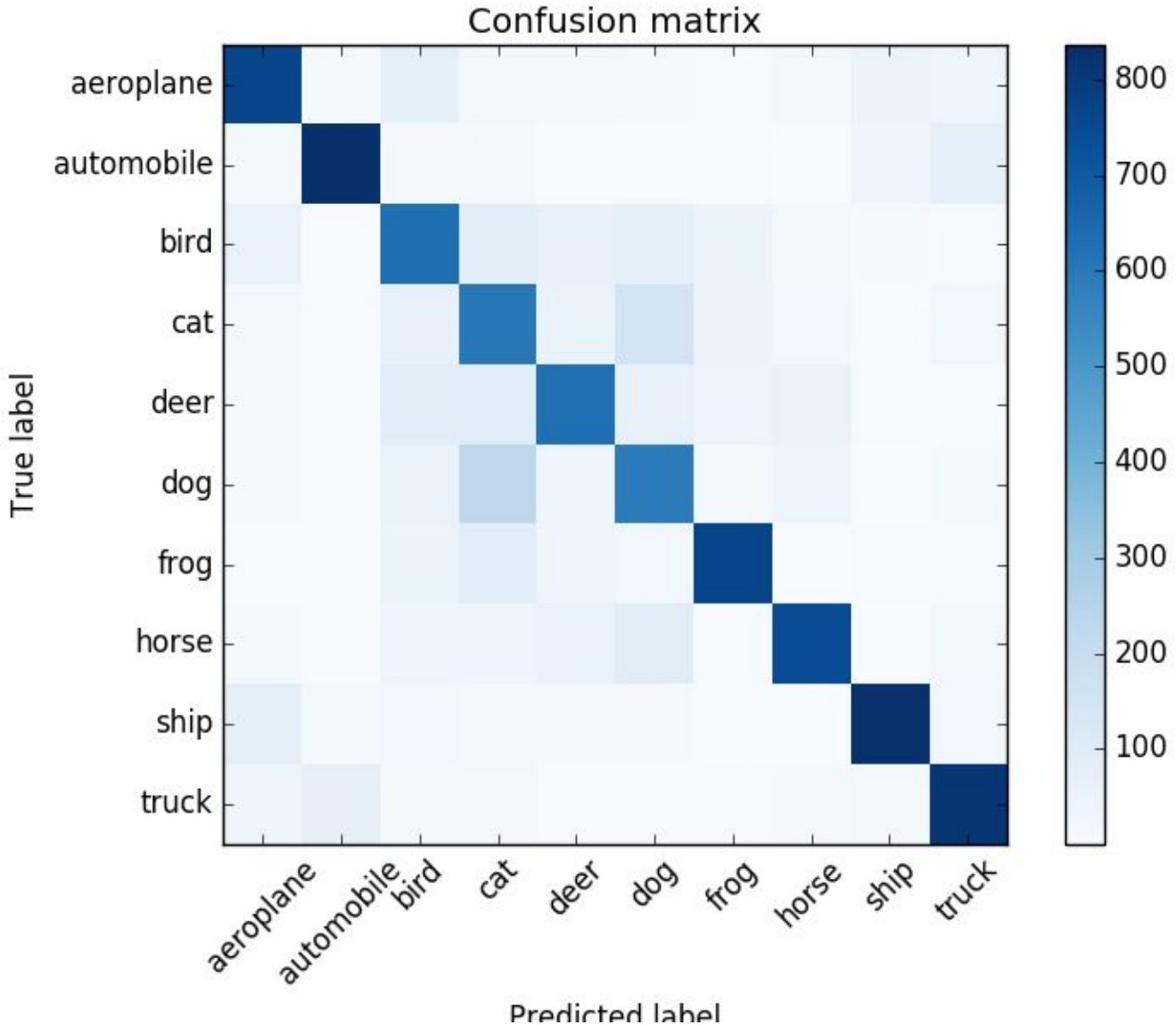
Configurations						Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	FC Layer4	FC Layer5			
	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	16	32	64	500	500	88.5%	74.6%	73.96%
	32	64	128	500	500	87.2%	73.22%	70.31%
5	16	32	64	500	500	86.31%	70.21	68.32%

Evaluation - Results - 6 & 7 Layer CNNs

Configurations							Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	Conv Layer4	FC Layer5	FC Layer6			
	Filter Count	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	8	16	32	64	500	500	80.5	69.1	68.82
	16	32	64	128	500	500	65.3	70.8	69.23
	32	32	64	128	500	500	98.6	72.00	68.89
5	16	32	64	128	500	500	98.3	72.96	72.1

Configurations								Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	Conv Layer4	Conv Layer5	FC Layer5	FC Layer6			
	Filter Count	Filter Count	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	8	16	32	64	128	500	500	92.4%	63.9%	67.12%
	16	32	64	128	256	500	500	98.6%	73.9%	71.69%

Confusion Matrix



Future Work

- Introduce batch normalization after each convolution layer.
- Replace the simplified version with the full VGG net architecture.
- Currently our code targets a CPU, we could improve to support GPU.
- Currently, we preserve the dimensions of the input image using zero padding. We could allow more flexibility in the strides and pooling window size.
- Write tools to visualize the kernels. Existing tools are targeted at deep learning frameworks like caffe,torch etc.,

Questions

Thank You

Softmax Regression

- Transfer multi-class classification into binary classification problem
- Construct several logistic classifier for each class.
- Set the value of y (label) of one class to 1, and 0 for other classes. Thus, if we have K classes, we build K logistic classifiers and use it for prediction.
- Classify the sample to class having highest value