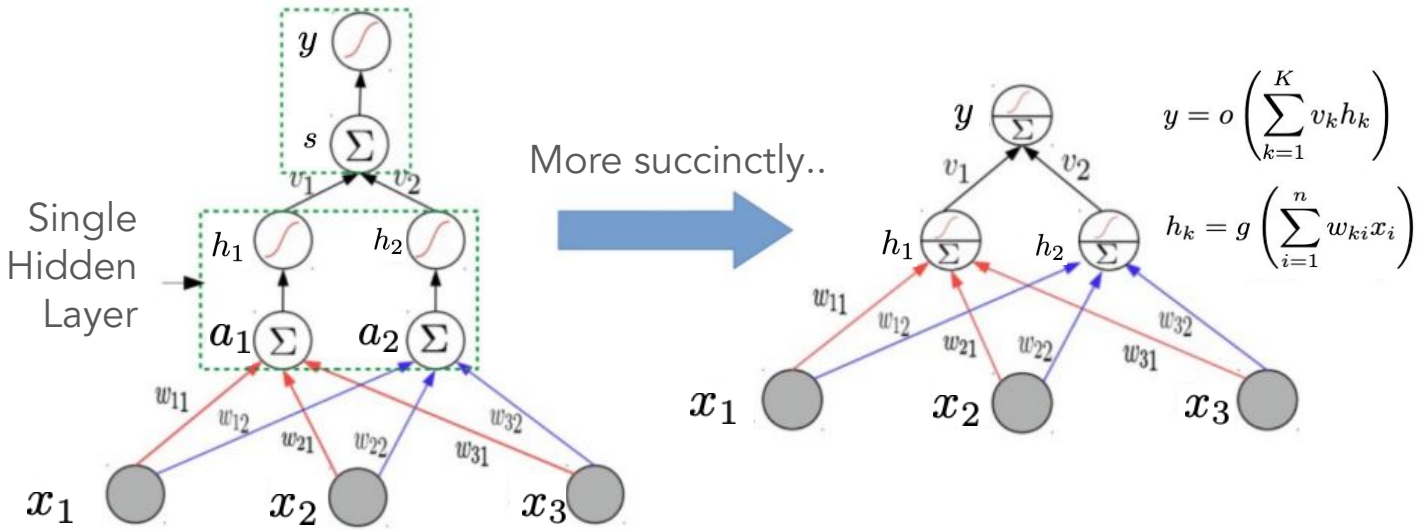
Georgia Tech
College of Computing

# CS4641 Spring 2025
# Convolution Neural Networks

Bo Dai
School of CSE, Georgia Tech
bodai@cc.gatech.edu

Based on Fei-Fei Li & Ehsan Adeli's Slides

# MLP Revisit


Single Hidden Layer
More succinctly..
$y$
$s$
$v_1$
$v_2$
$h_1$
$h_2$
$a_1$
$a_2$
$w_{11}$
$w_{12}$
$w_{21}$
$w_{22}$
$w_{31}$
$w_{32}$
$x_1$
$x_2$
$x_3$
$y = o\left(\sum_{k=1}^{K} v_k h_k\right)$
$h_k = g\left(\sum_{i=1}^{n} w_{ki} x_i\right)$

# Vector Formation

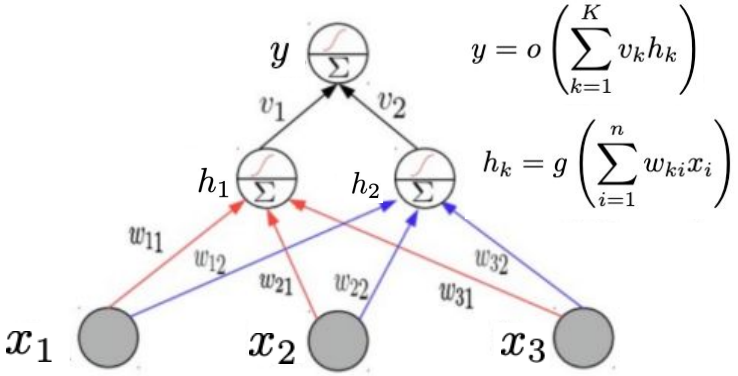
$$y = o(V g(W x))$$

$$V = [v_1, v_2]$$

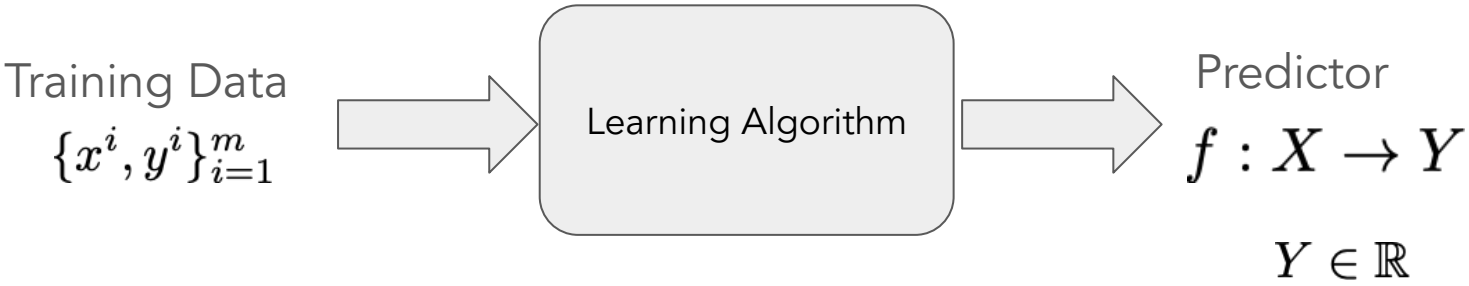
$$h = [h_1, h_2]^\top = g(Wx)$$

$$W = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix}$$

$$x = [x_1, x_2, x_3]^\top$$



# Regression Algorithms

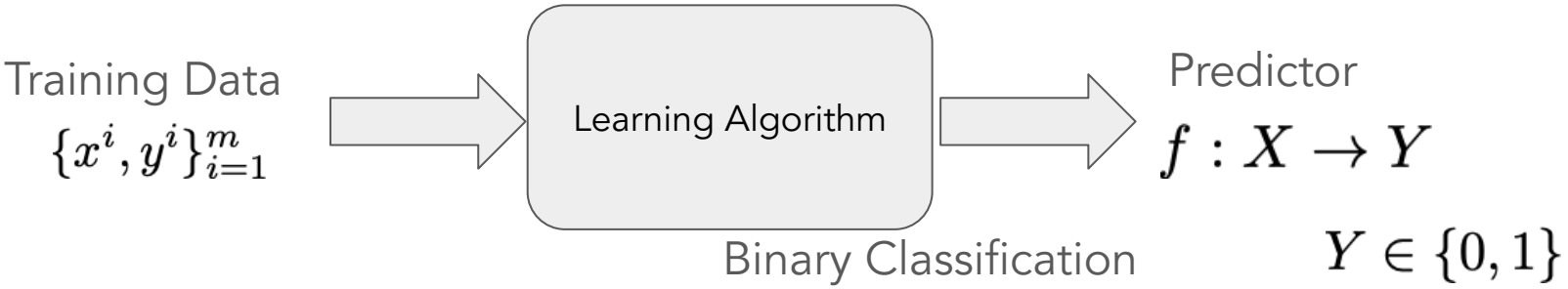



## Linear Regression Pipeline

1. Build probabilistic models:
   Gaussian Distribution + Neural Network
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) GD

$$y = o(V g(W x))$$

# Binary Classification Algorithms




## Binary Logistic Regression Pipeline

1. Build probabilistic models:
   Bernoulli Distribution + Neural Network $y = o(Vg(Wx))$
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

# Multiclass Logistic Regression Algorithms


Multiclass Classification $Y \in \{0, 1, \ldots, k\}$

## Multiclass Logistic Regression Pipeline

1. Build probabilistic models:
   Categorical Distribution + Neural Network $y = o(Vg(Wx))$
2. Derive loss function: MLE and MAP
3. Select optimizer: (Stochastic) Gradient Descent

# Select Optimizer

$$L(\theta) = \sum_{i=1}^{m} \ell(x^i, y^i, \theta) + \lambda\Omega(\theta)$$

$$\theta = [V, W]$$

$$\ell(x^i, y^i, \theta) = (o(Vg(Wx^i)) - y^i)^2$$

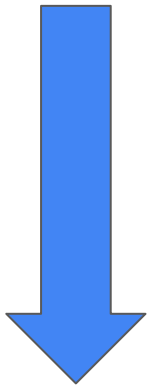
$$\ell(x^i, y^i, \theta) = -y^i \log \sigma(o(Vg(Wx^i)))$$
$$-(1 - y^i) \log(1 - \sigma(o(V_j g(Wx^i))))$$

$$\ell(x^i, y^i, \theta) = -\sum_{j=1}^{k} y^i \log \frac{\exp(o(V_j g(Wx^i)))}{\sum_{c=1}^{k} \exp(o(V_c g(Wx^i)))}$$

- (Stochastic) Gradient Descent

# Backpropagation: Chain Rule on Neural Network

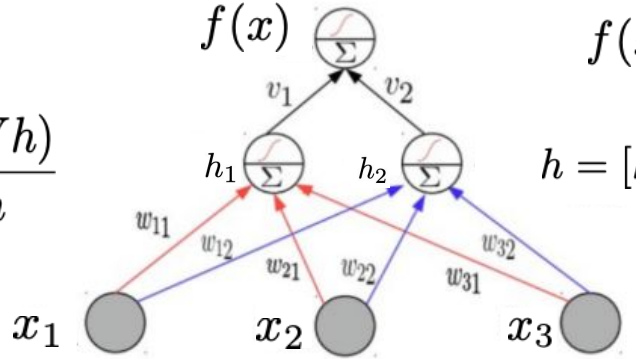
$$\ell(x^i, y^i, \theta) = (f(x^i, V, W) - y^i)^2$$



$$\frac{\partial \ell(x^i, y^i, \theta)}{\partial f}$$

$$\frac{\partial o(Vh)}{\partial V} \quad \frac{\partial o(Vh)}{\partial h}$$

$$\frac{\partial h}{\partial W}$$




$$f(x) = o(Vh)$$

$$h = [h_1, h_2]^\top = g(Wx)$$

Backward Pass

# (Stochastic) Gradient Descent

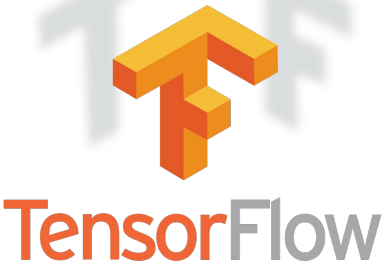
- Initialize parameter $\theta^0$
- Sample $\{x^i, y^i\}_{i=1}^B$
- Do $$\theta^{t+1} \leftarrow \theta^t - \eta \sum_{i=1}^{B} \nabla_\theta \ell(x^i, y^i, \theta^t) - (\lambda \nabla \Omega(\theta^t))$$

# Auto-differentiation Packages

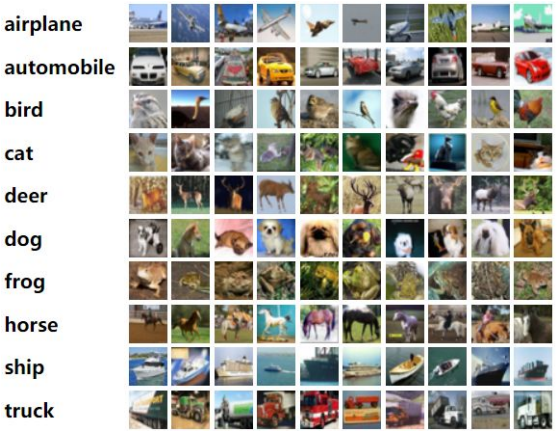
PyTorch


JAX

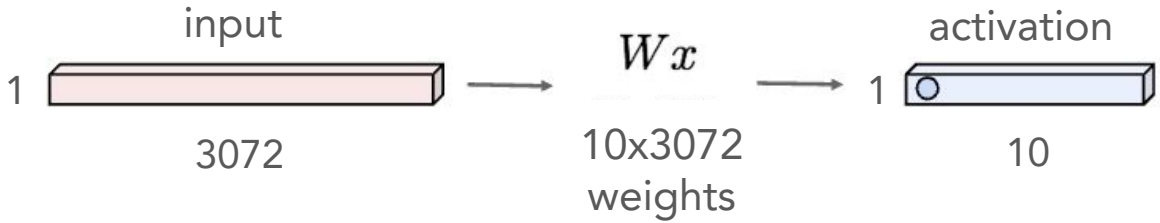
Tensorflow



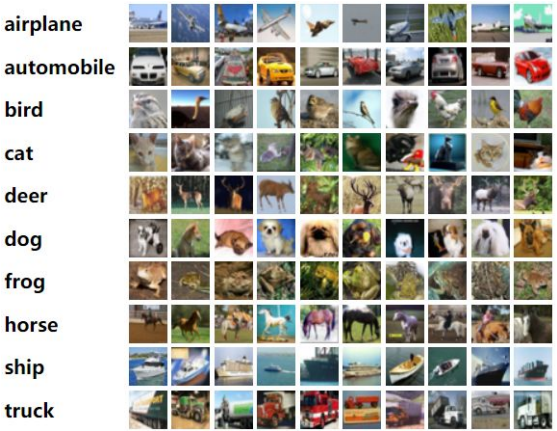
# Fully Connected Layer



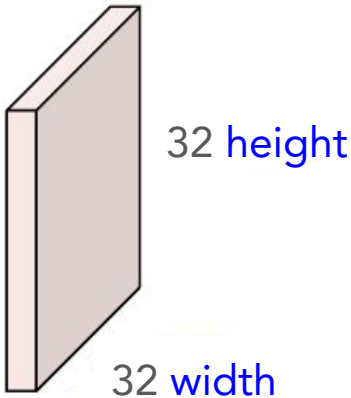

32x32x3 image → stretch to 3072x1



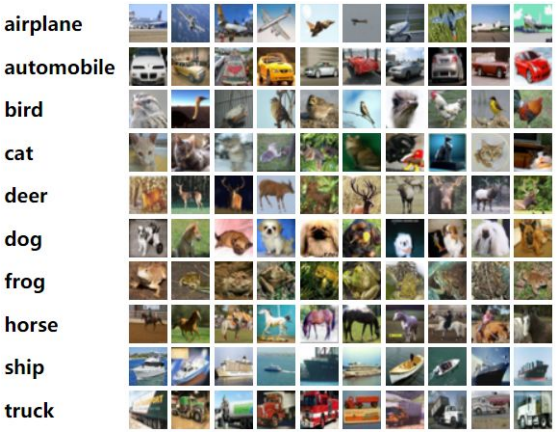
# Convolution Layer



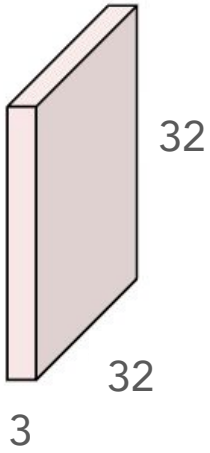

32x32x3 image → preserve spatial structure



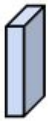
# Convolution Layer




32x32x3 image

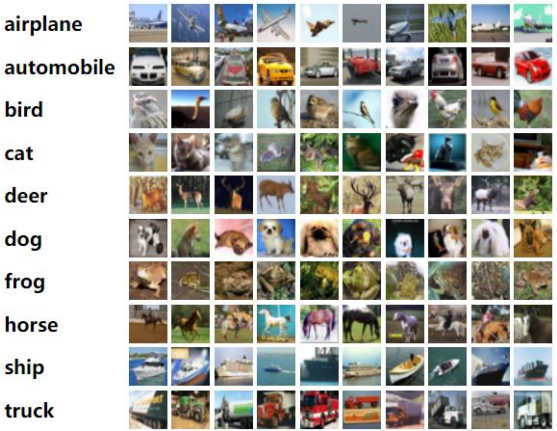


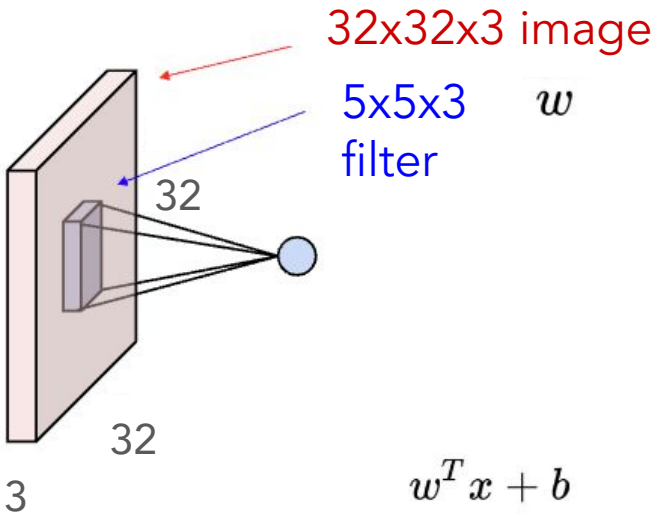

5x5x3 filter



Convolve the filter with the image, i.e., “slide over the image spatially, computing dot product”

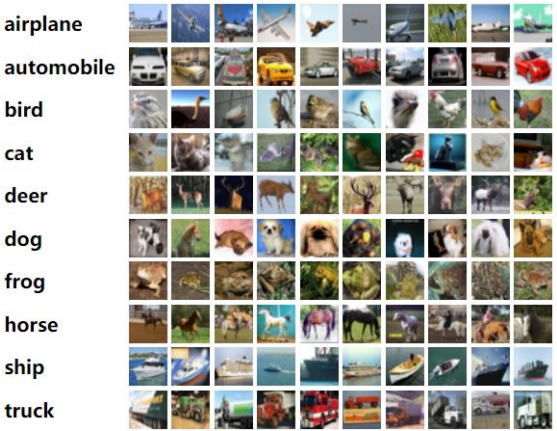
# Convolution Layer

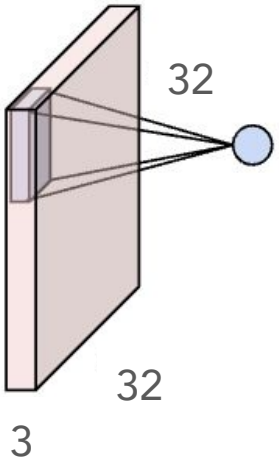


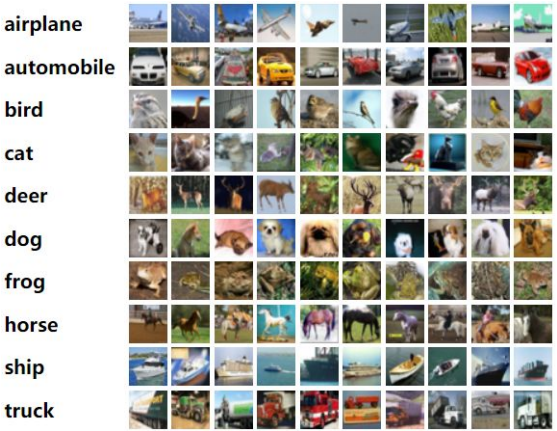

$$w^T x + b$$

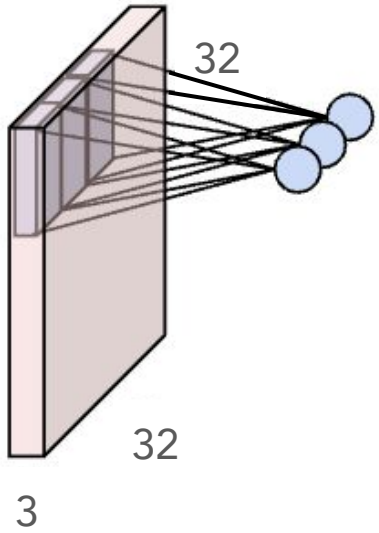
# Convolution Layer



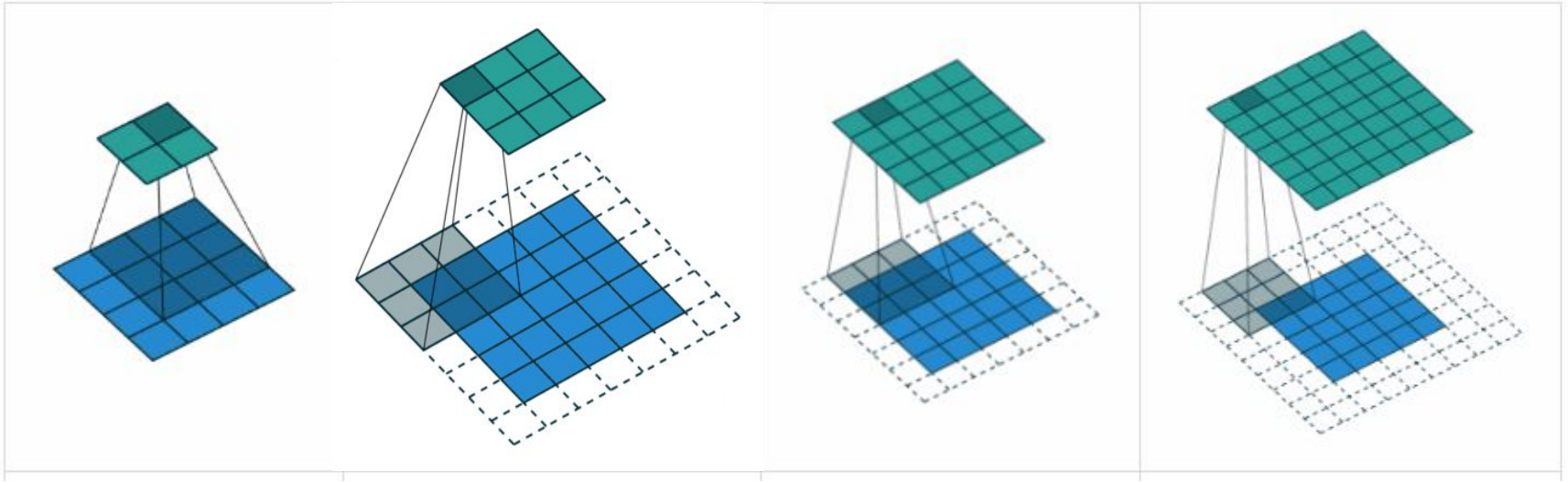



# Convolution Layer



# Convolution Layer



padding = 0, stride = 1

padding = 1, stride = 2

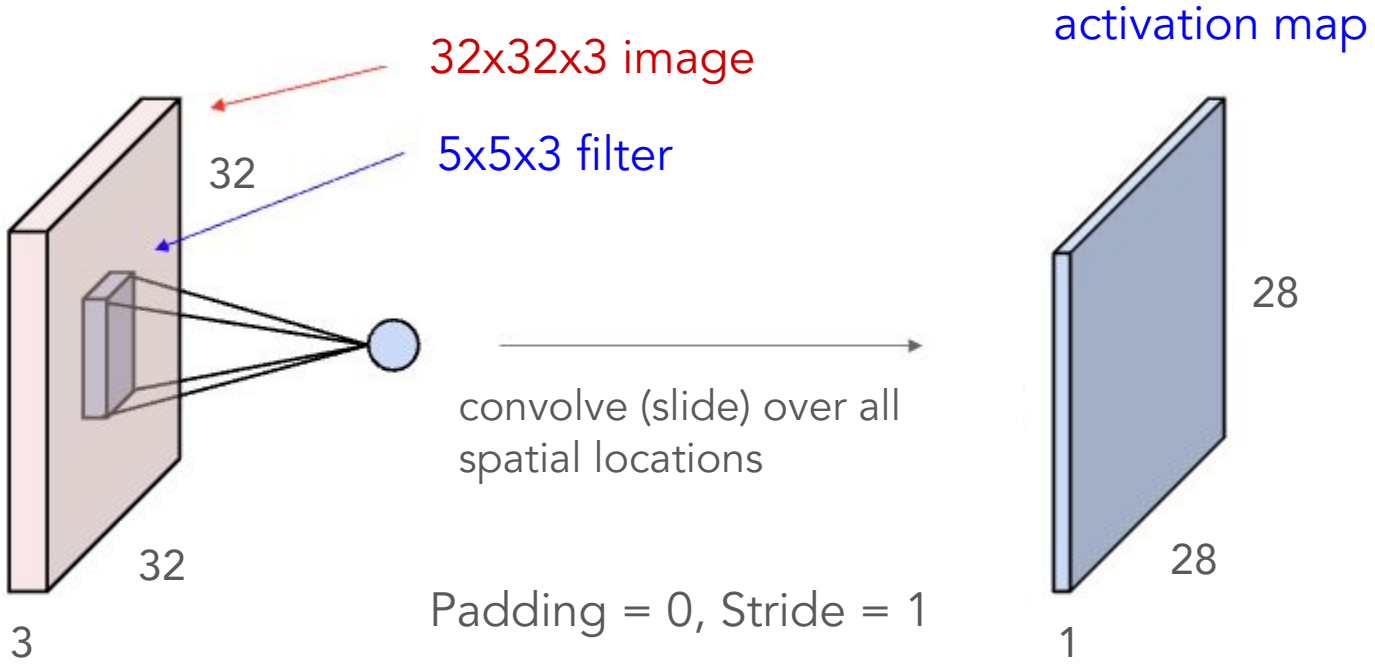
padding = 1, stride = 1

padding = 2, stride = 1

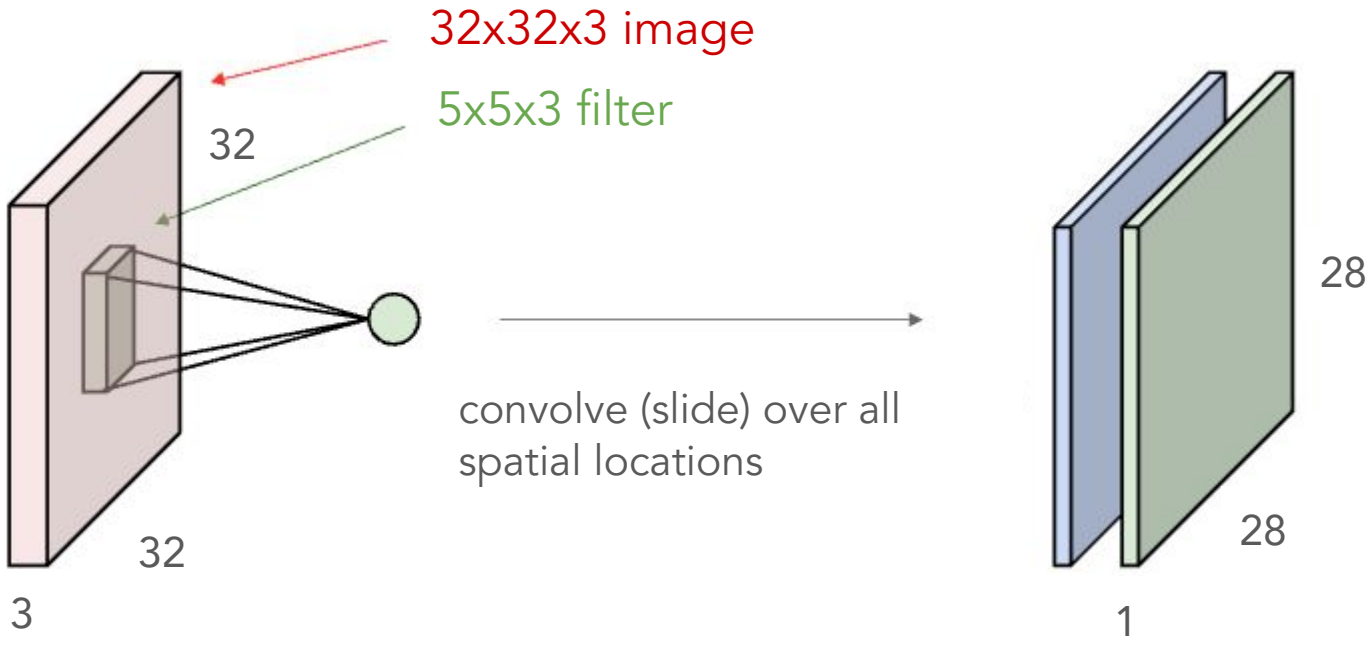
$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Animation from [Hochschule der Medien]

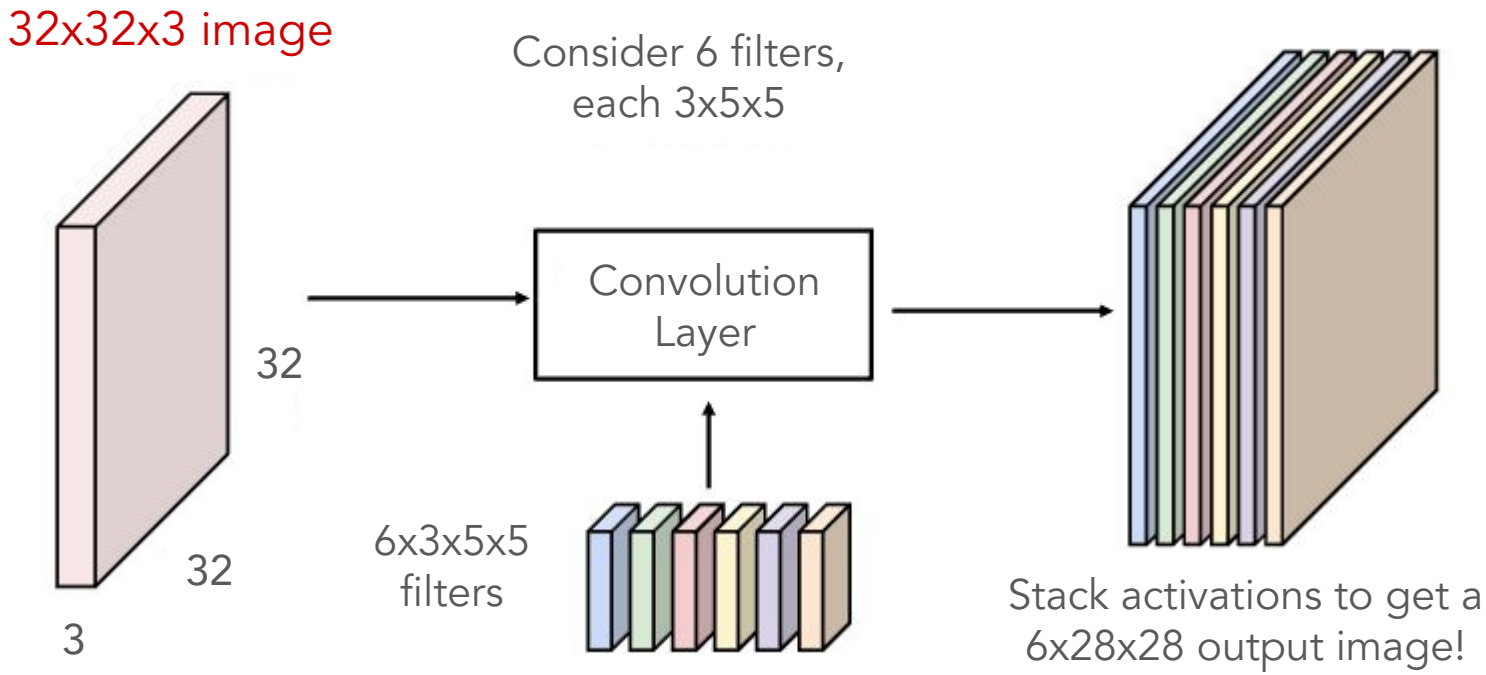
# Convolution Layer



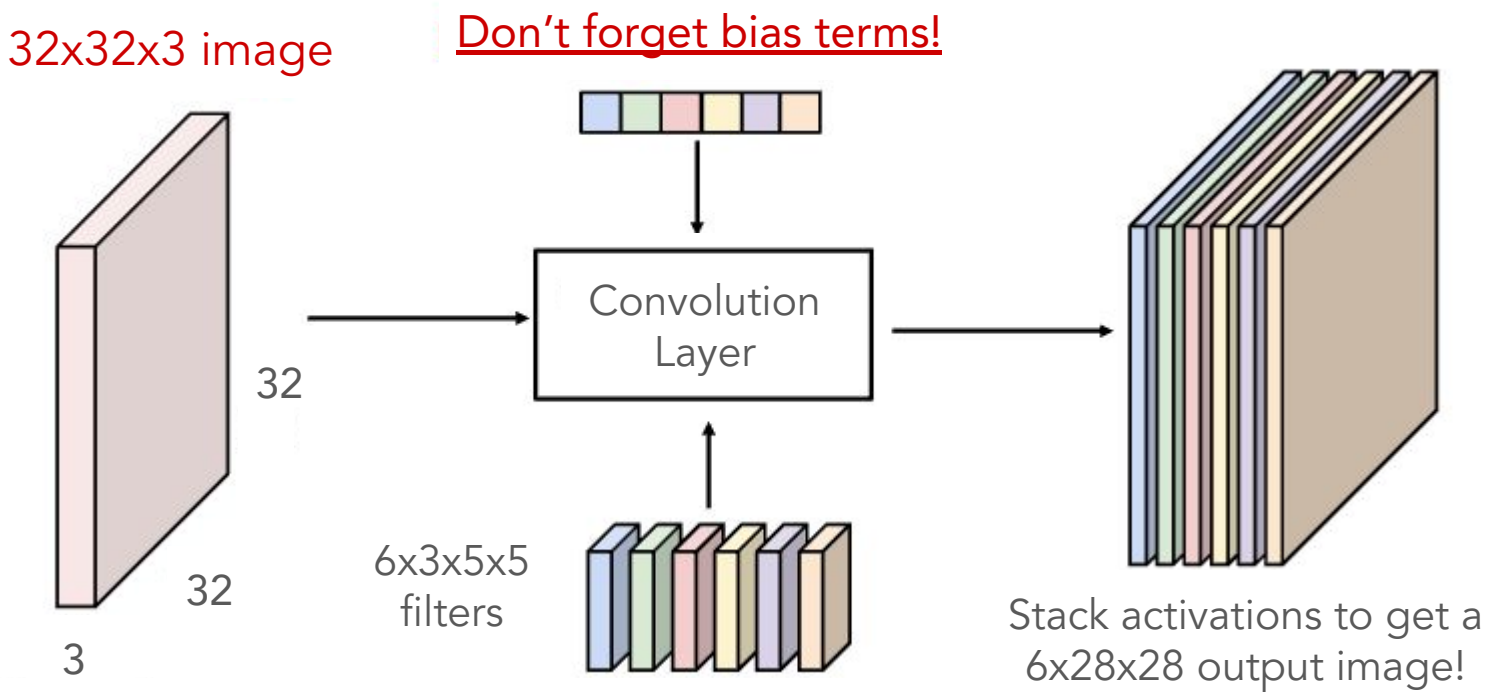
# Convolution Layer



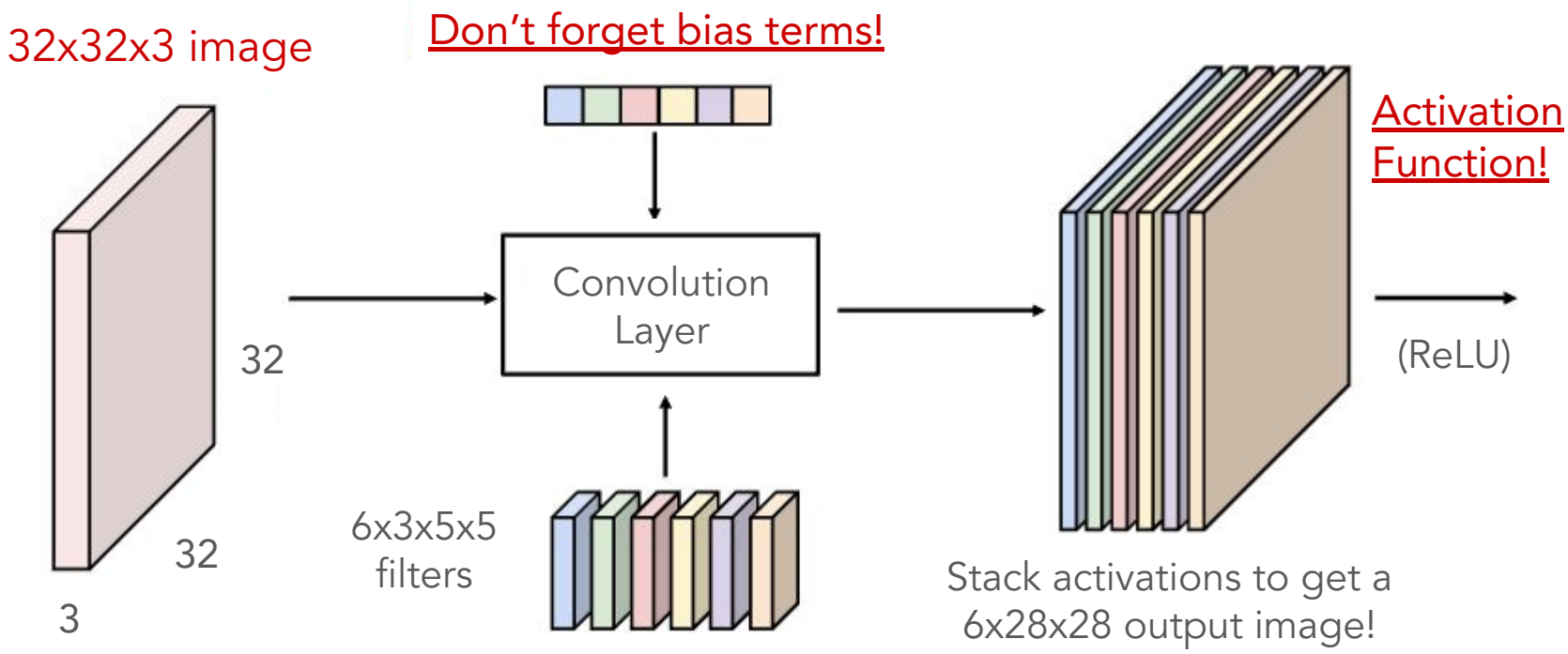
# Convolution Layer



# Convolution Layer



# Convolution Layer

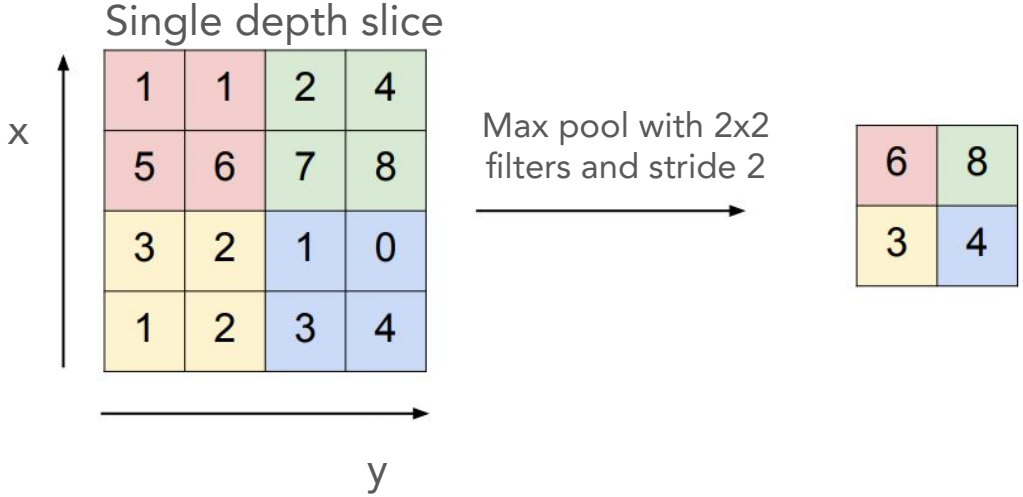


# Pooling (Subsampling)

MAX POOLING

(Wx - F)/Stride + 1

(Wy - F)/Stride + 1

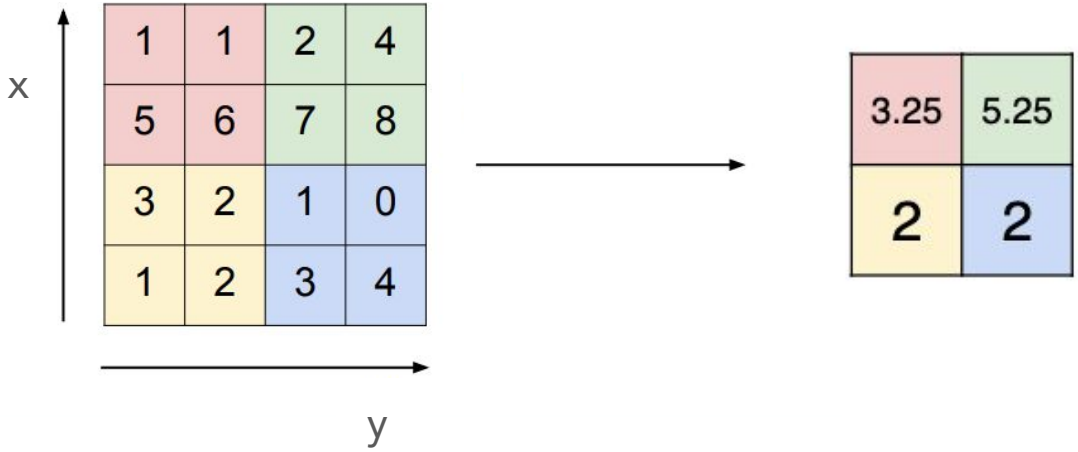


# Pooling (Subsampling)

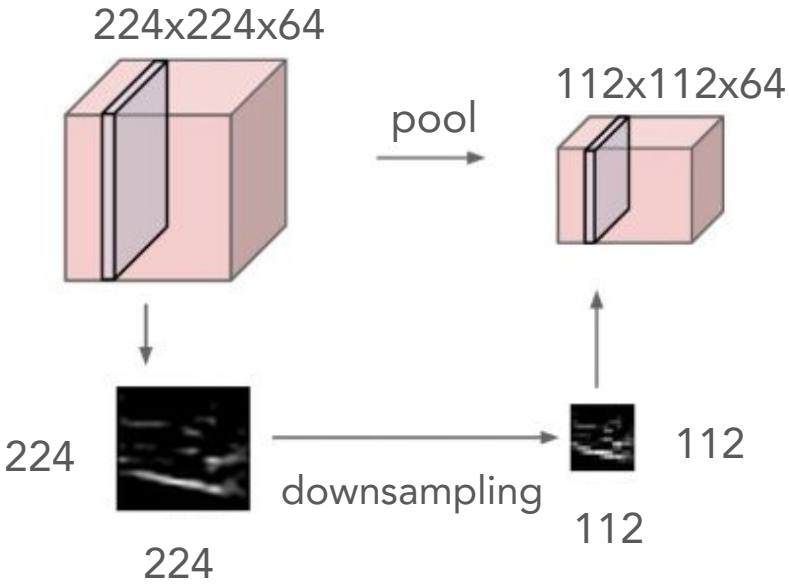
MEAN POOLING

(Wx - F)/Stride + 1
(Wy - F)/Stride + 1

Single depth slice

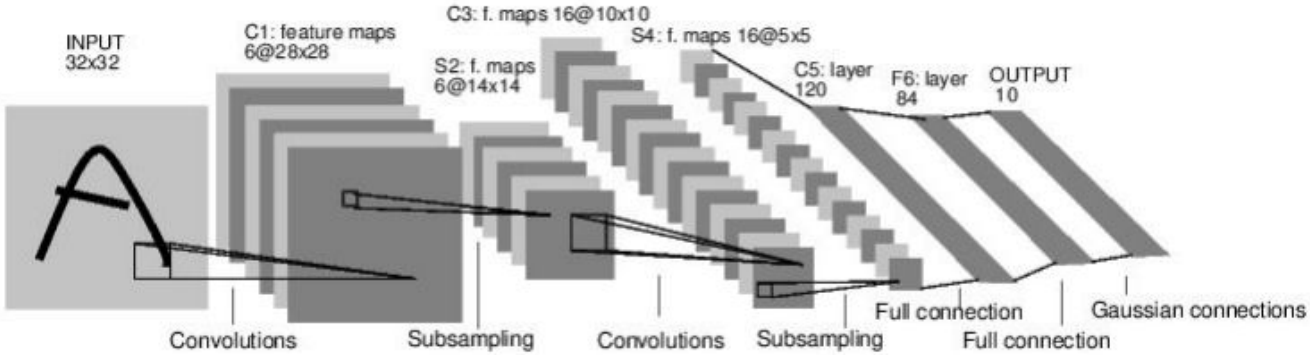


# Pooling (Subsampling)




- Pooling layers simplify / subsample / compress the information in the output from the convolutional layer
- Reduce parameters

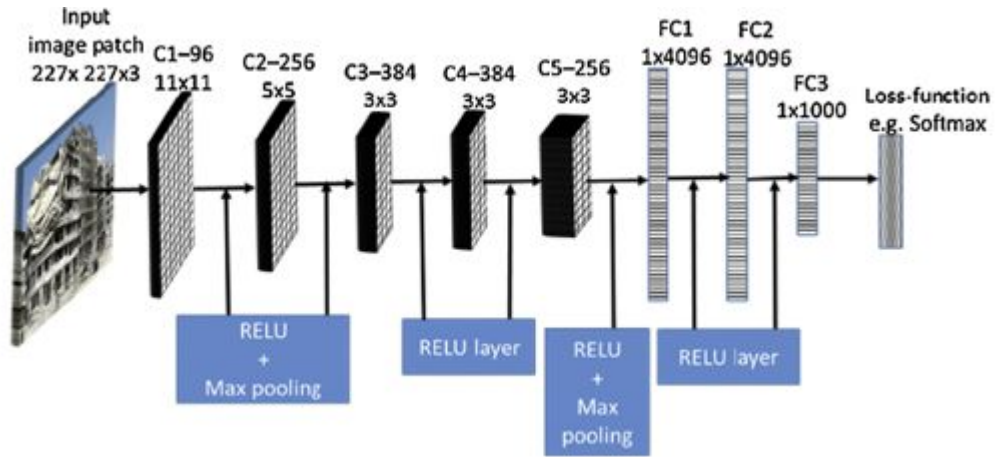
# Put Everything Together



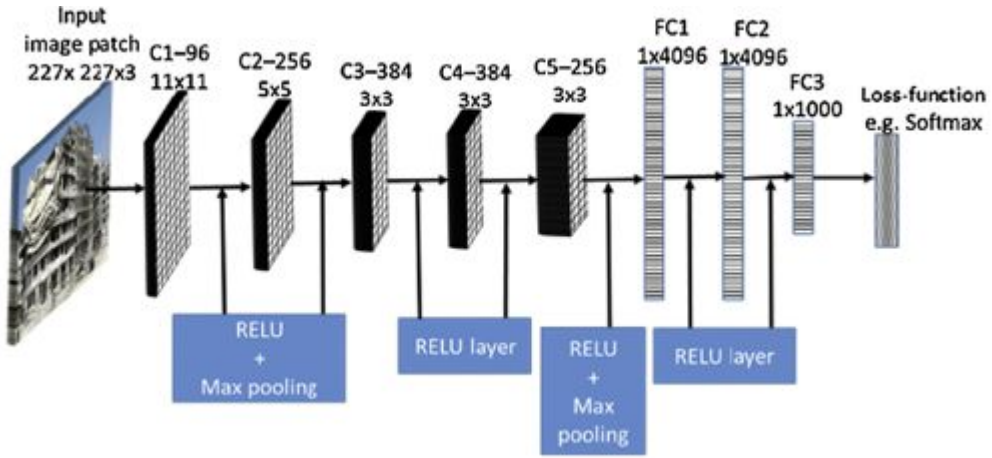

*[LeNet-5, LeCun 1980]*

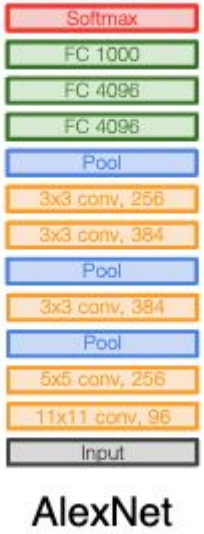
# AlexNet (2012)

- AlexNet was one of the first deep convolutional on ImageNet competition with an accuracy of 84.7% as compared to the second-best with an accuracy of 73.8%.
- The activation used is the Rectified Linear Unit (ReLU).
- The training set had 1.2 million images. It was trained for 90 epochs, which took five to six days on two NVIDIA GTX 580 3GB GPUs.



# AlexNet (2012)

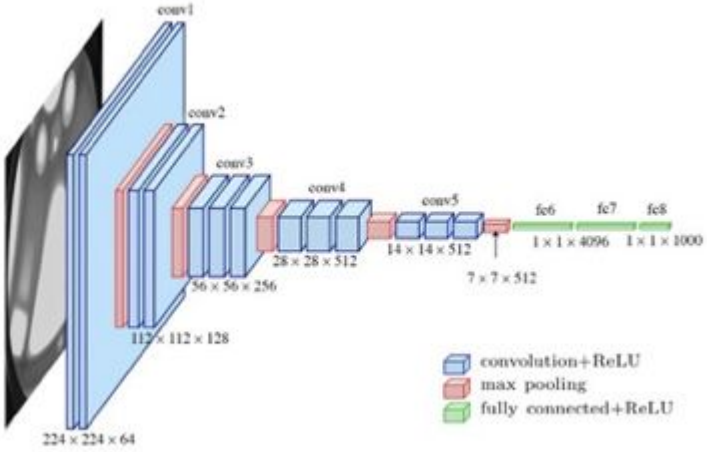



AlexNet

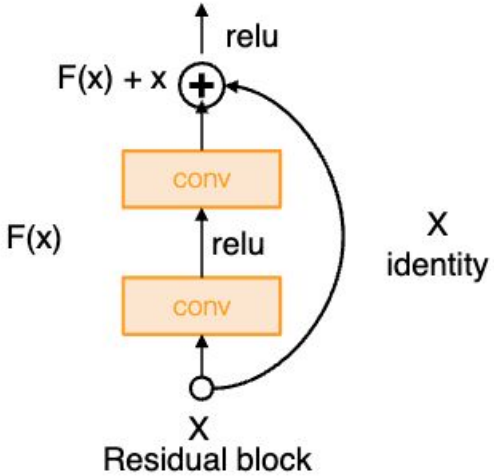
# VGGNet (2014)

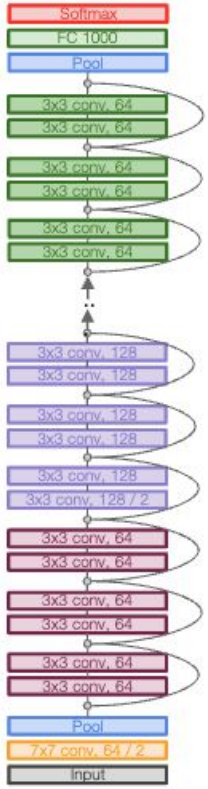
- Very Deep CNN
- With only 3*3 conv filters
  - Fewer parameters, deeper nonlinear layers



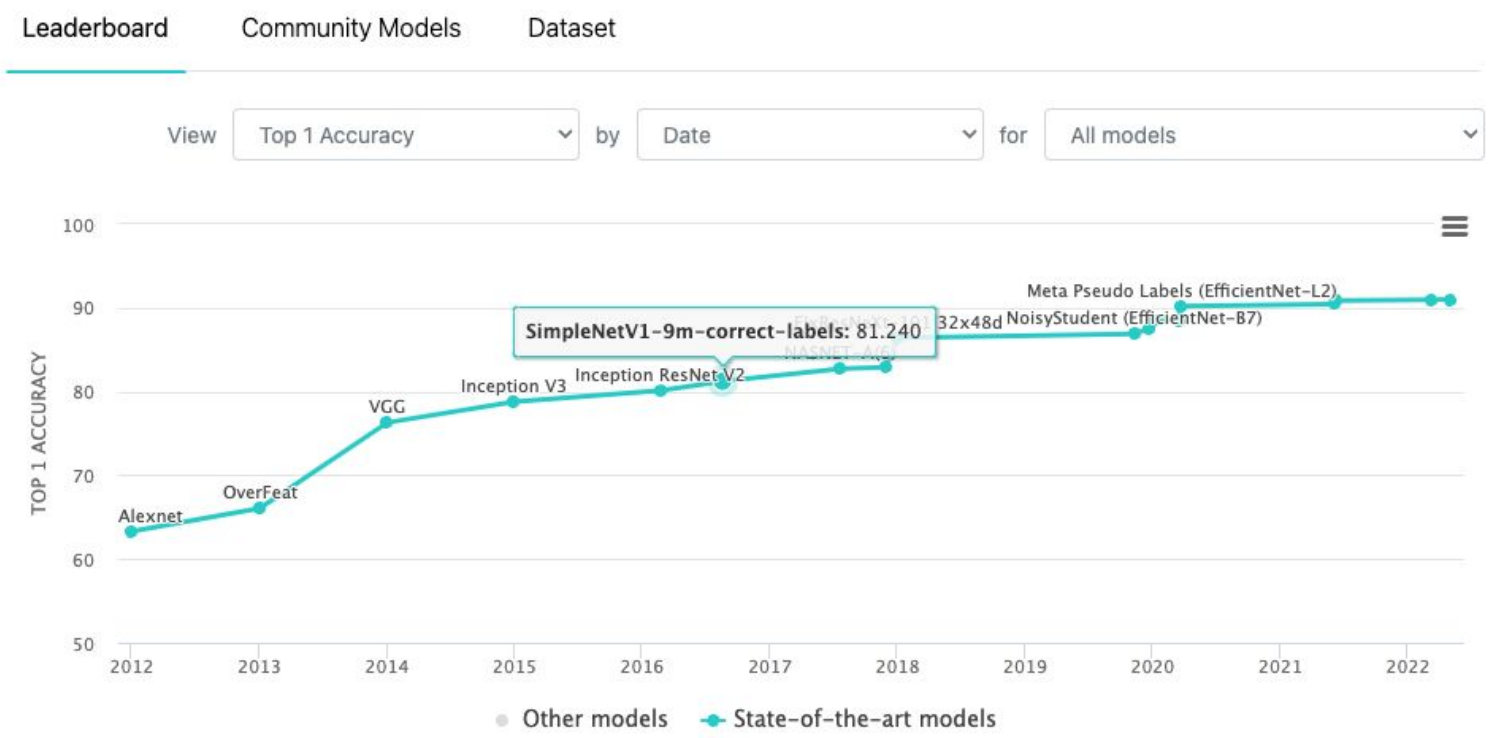

# ResNet (2015)

- Very Deep CNN with residual connections
  - 152-layer model for ImageNet
  - ILSVRC'15 classification winner (3.57% top 5 error)
  - Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

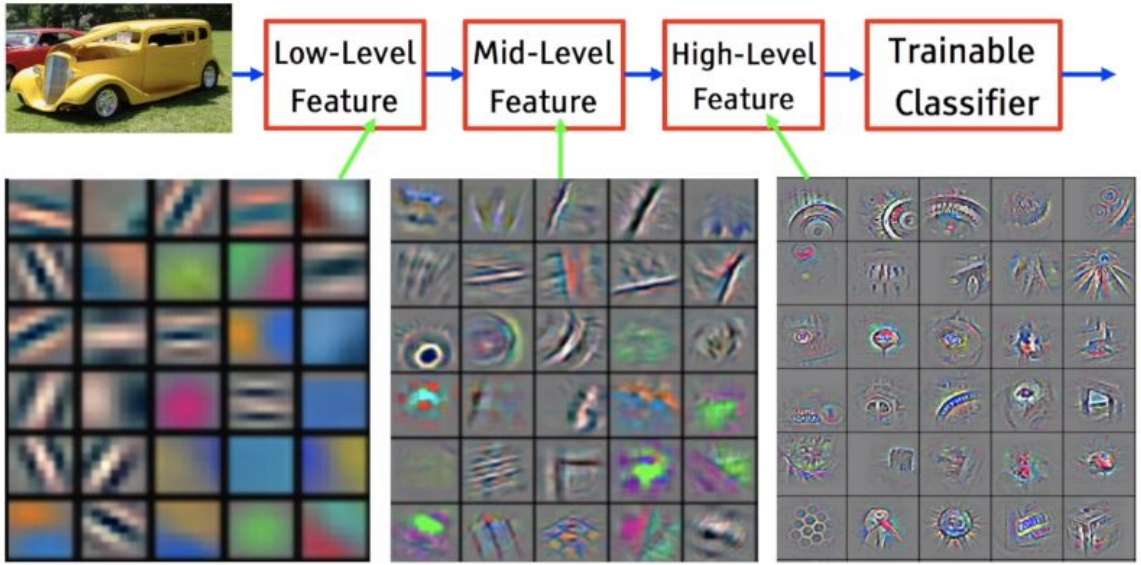


# Image Classification on ImageNet



# Convolution Features

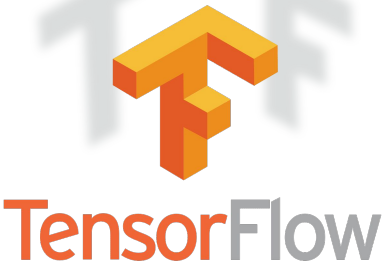



# Auto-differentiation Packages

PyTorch

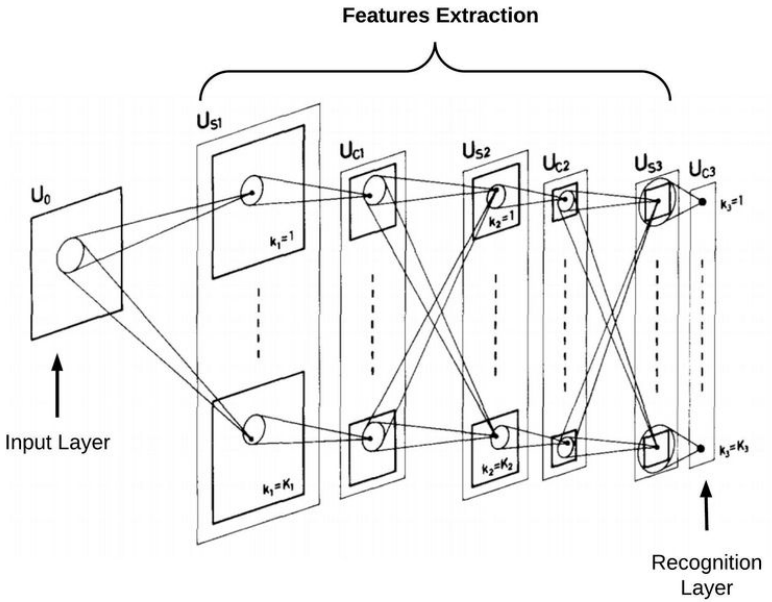

JAX

Tensorflow



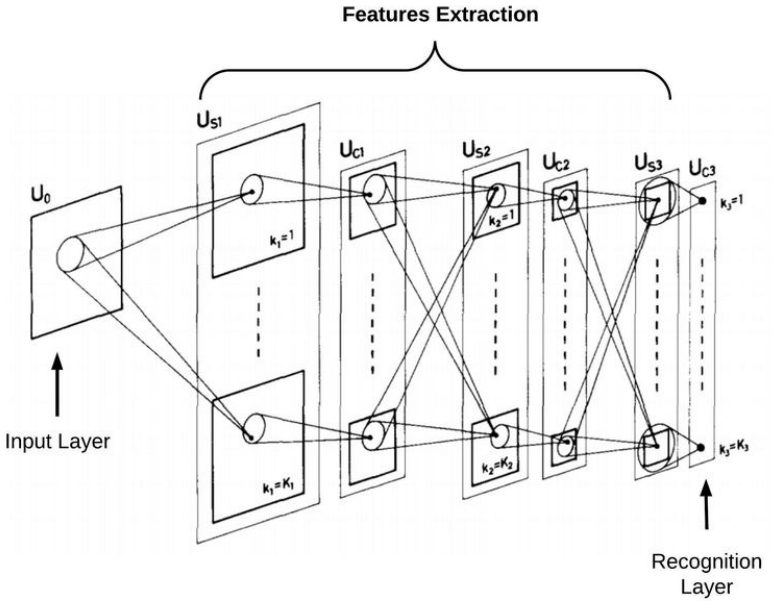
# A Bit of History

- Neocognitron [Fukushima 1980]



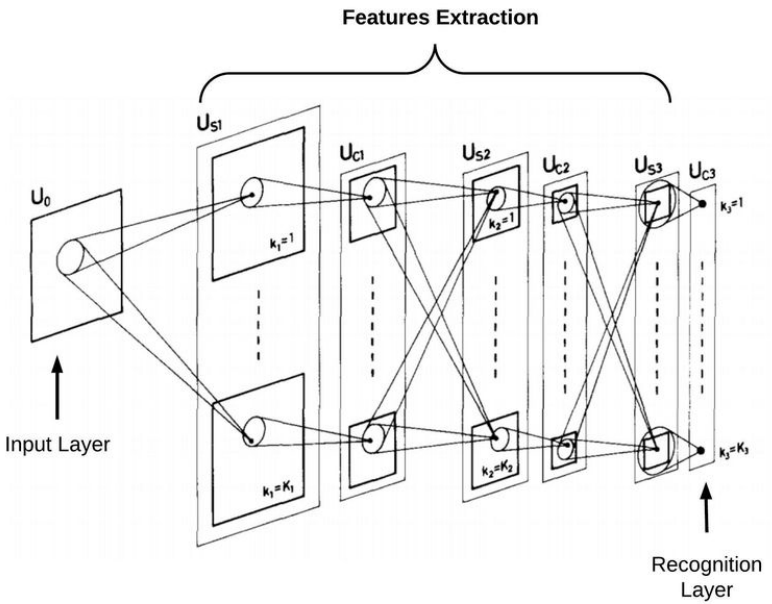
# A Bit of History

- Neocognitron [Fukushima 1980]
- Backpropagation [Rumelhart et al., 1986, Werbos 1974]



# A Bit of History

- Neocognitron [Fukushima 1980]
- Backpropagation [Rumelhart et al., 1986, Werbos 1974]
- BP on CNN [LeCun et al., 1989]



# Q&A

HW2 Due Today!

No HW until Mar. 05th
Use the time wisely for project