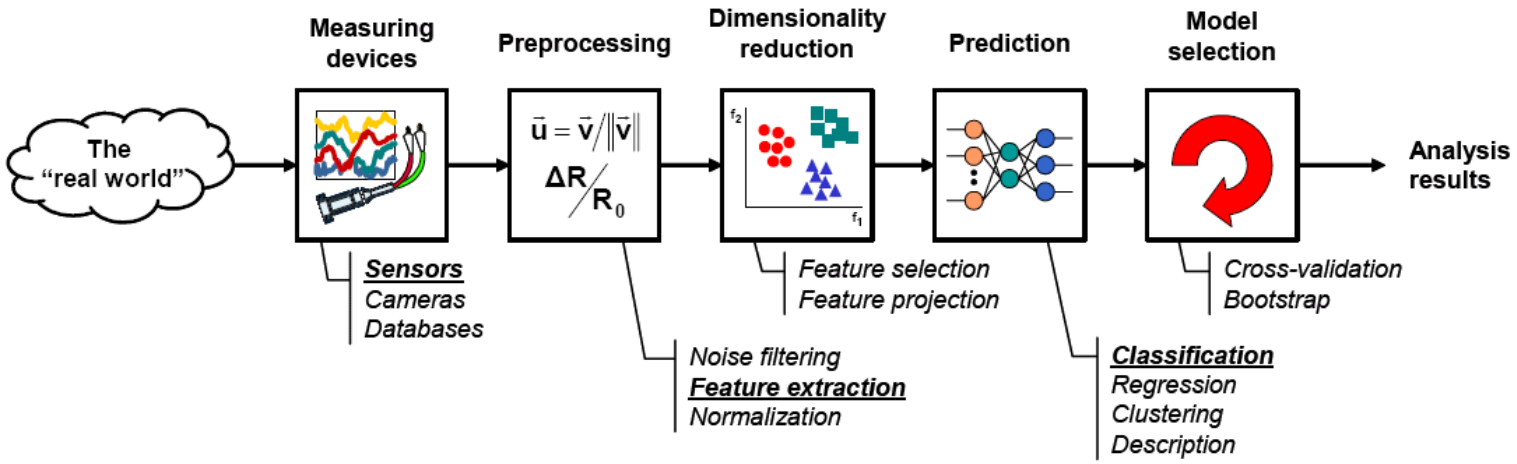
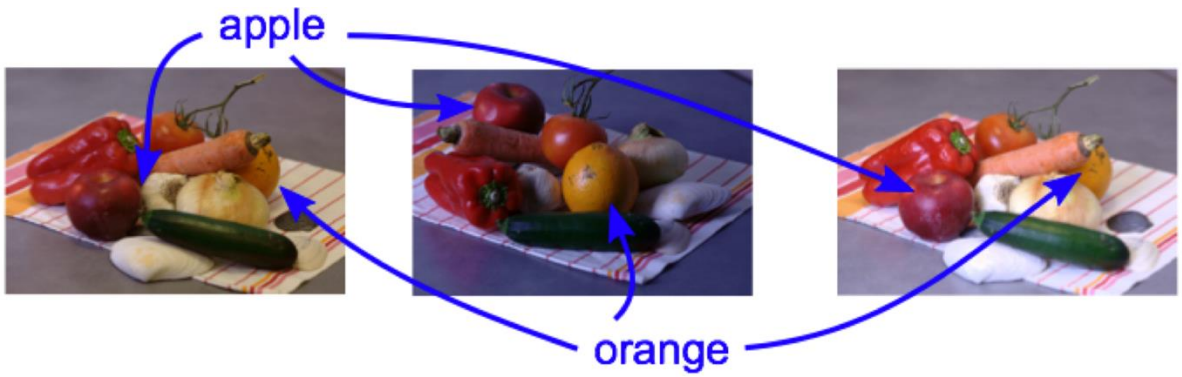
Convolutional Neural Networks

Prof. Dr. M. Elif Karslıgil

Stages of a machine learning system



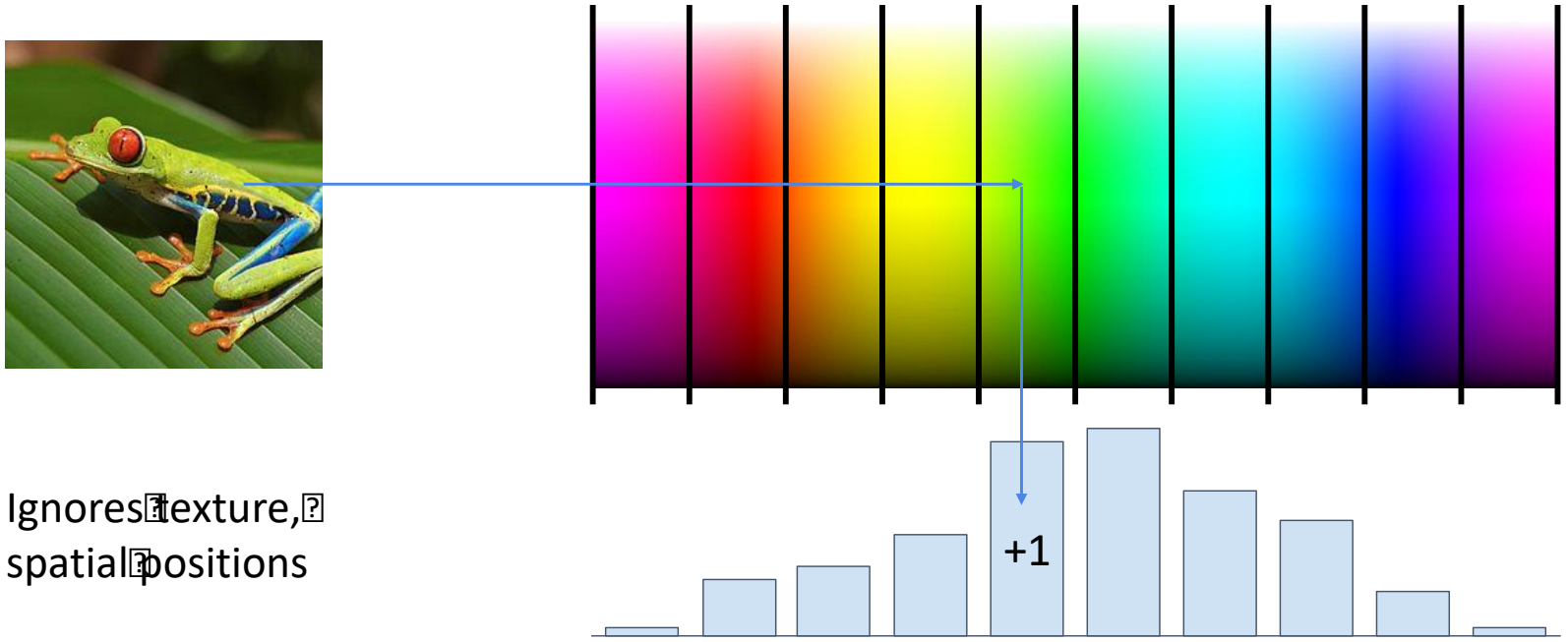
Why is Object Recognition Difficult ?



- Multiple objects
- Different viewpoints
- Changes lightning
- Deformations
- Many correlated features

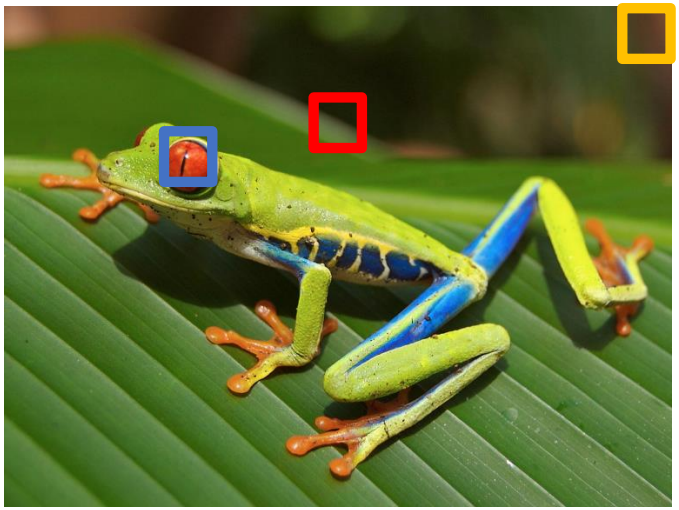
Source: Richard Turner 2014

Handcrafted Image Features: Color Histogram



Ignores texture, spatial positions

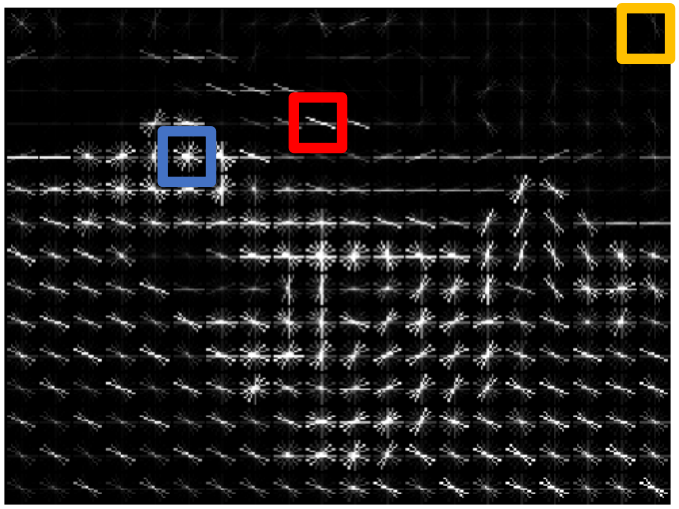
Handcrafted Image Features:
Feature Description: Histogram of Oriented Gradient



Weak edges

Strong diagonal edges

Edges in all directions



1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

Handcrafted Image Features: SIFT

(The **S**cale-**I**nvariant **F**eature **T**ransform)

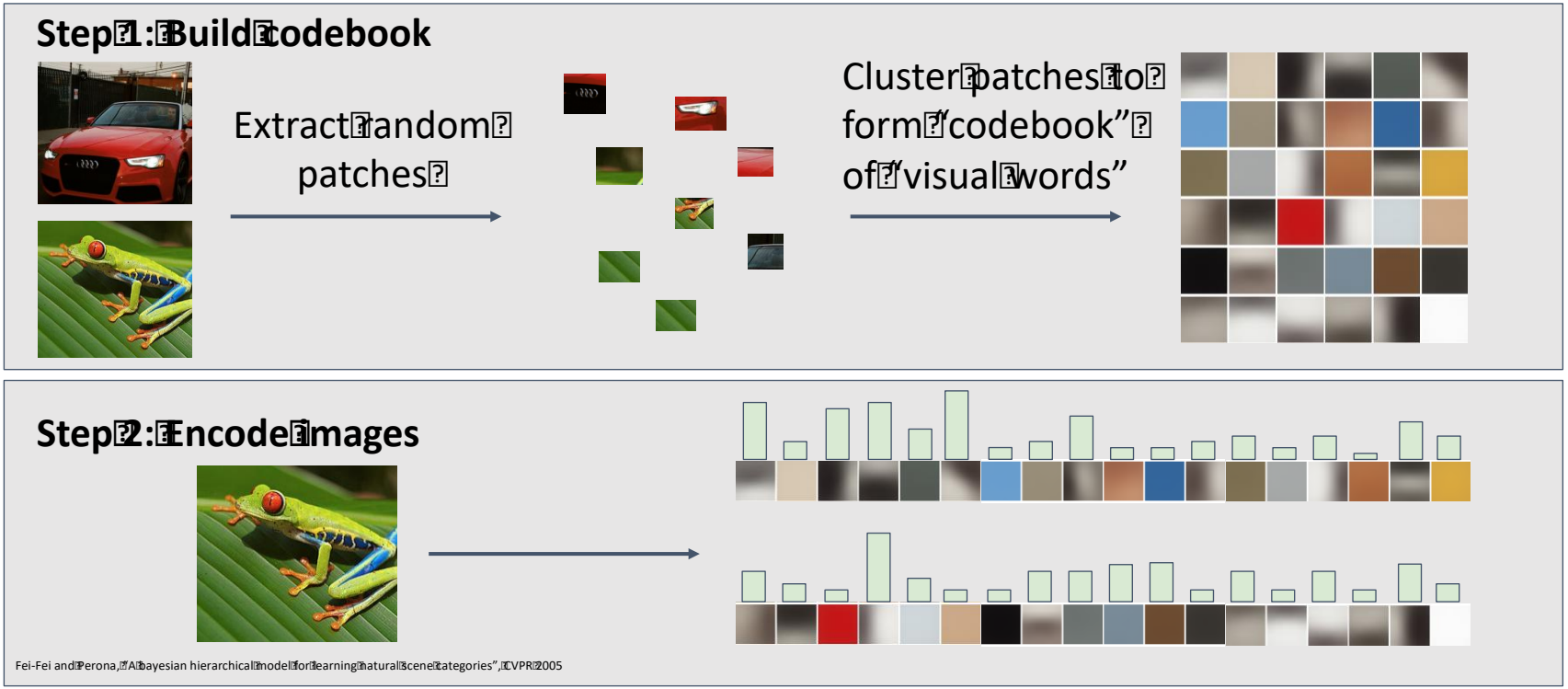
Keypoints between two images are matched by identifying their nearest neighbors

Image is public domain

Image is public domain

Handcrafted Image Features: Bag of Words

- **First, take a bunch of images, extract features**
- **Build up a "dictionary" or "visual vocabulary" – a list of common features**

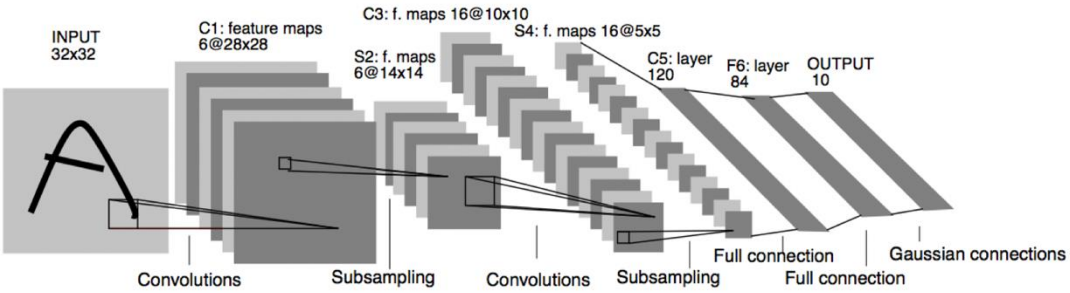


http://vision.stanford.edu/teaching/cs131_fall1718/files/14_BoW_bayes.pdf

LeNet 5, LeCun 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

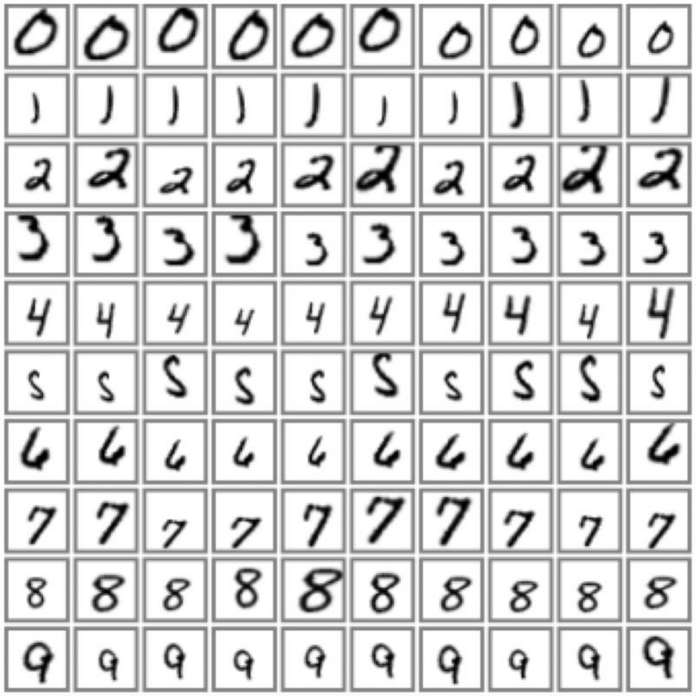


LeCun et al., 1998

MNIST Dataset

60,000 original datasets

Test error: 0.95%



540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

ImageNET

- AI researcher Fei Fei Li began working on the idea for ImageNet in 2006. Li wanted to expand and improve the data available to train AI algorithms.
- In 2007, Li met with Princeton professor Christiane Fellbaum, one of the creators of WordNet, to discuss the project.
- Li went on to build ImageNet starting from the word database of WordNet and using many of its features.
- As an assistant professor at Princeton, Li assembled a team of researchers to work on the ImageNet project.

ImageNET Challenge

IMAGENET **Large Scale Visual Recognition Challenge**

The Image Classification Challenge:
1,000 object classes
1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

Deng et al, 2009
Russakovsky et al. IJCV 2015

ImageNet Challenge

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images

- RGB images
- Variable-resolution, but this architecture scales them to 256x256 size

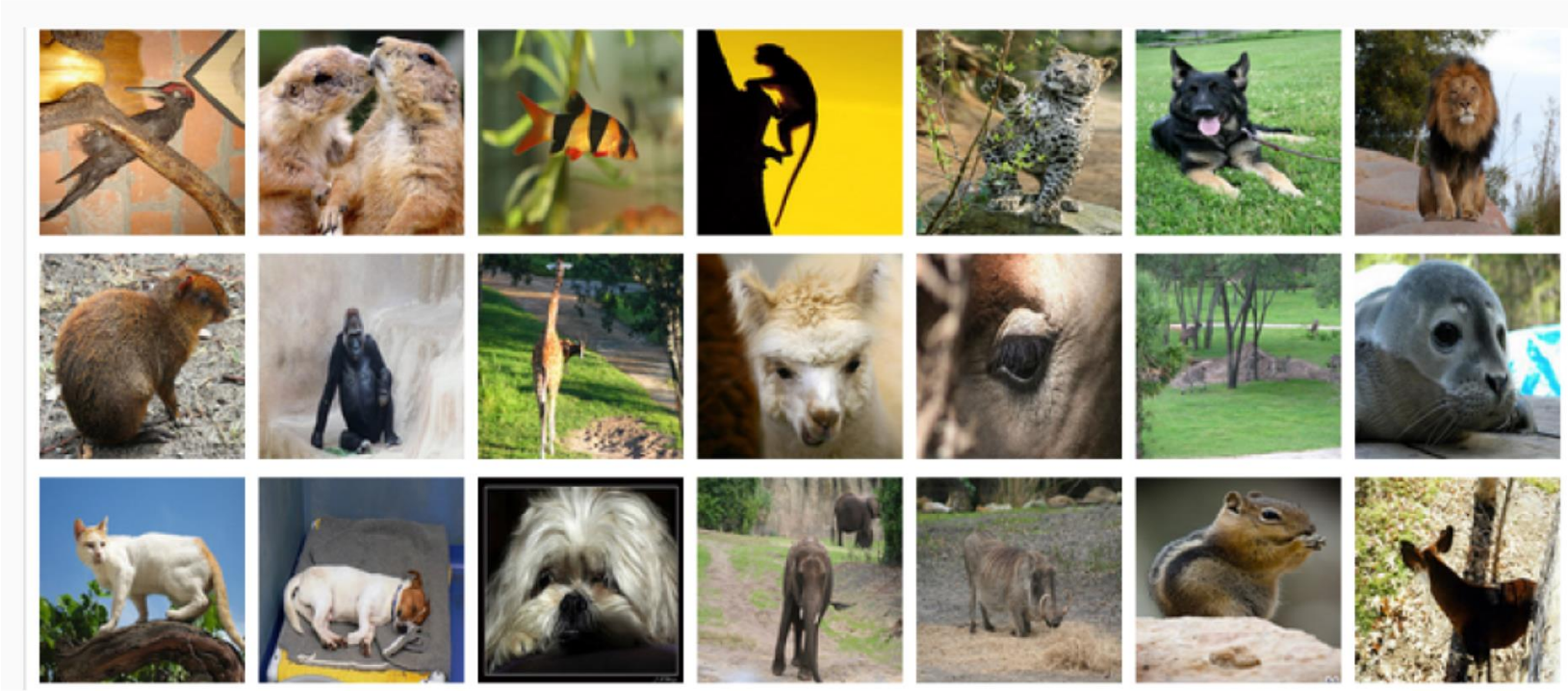
ImageNET Challenge

- Competitors are given
  - more than 1.2 million images
  - from 1,000 different object categories,
  - with the objective of developing the most accurate object detection and classification approach.

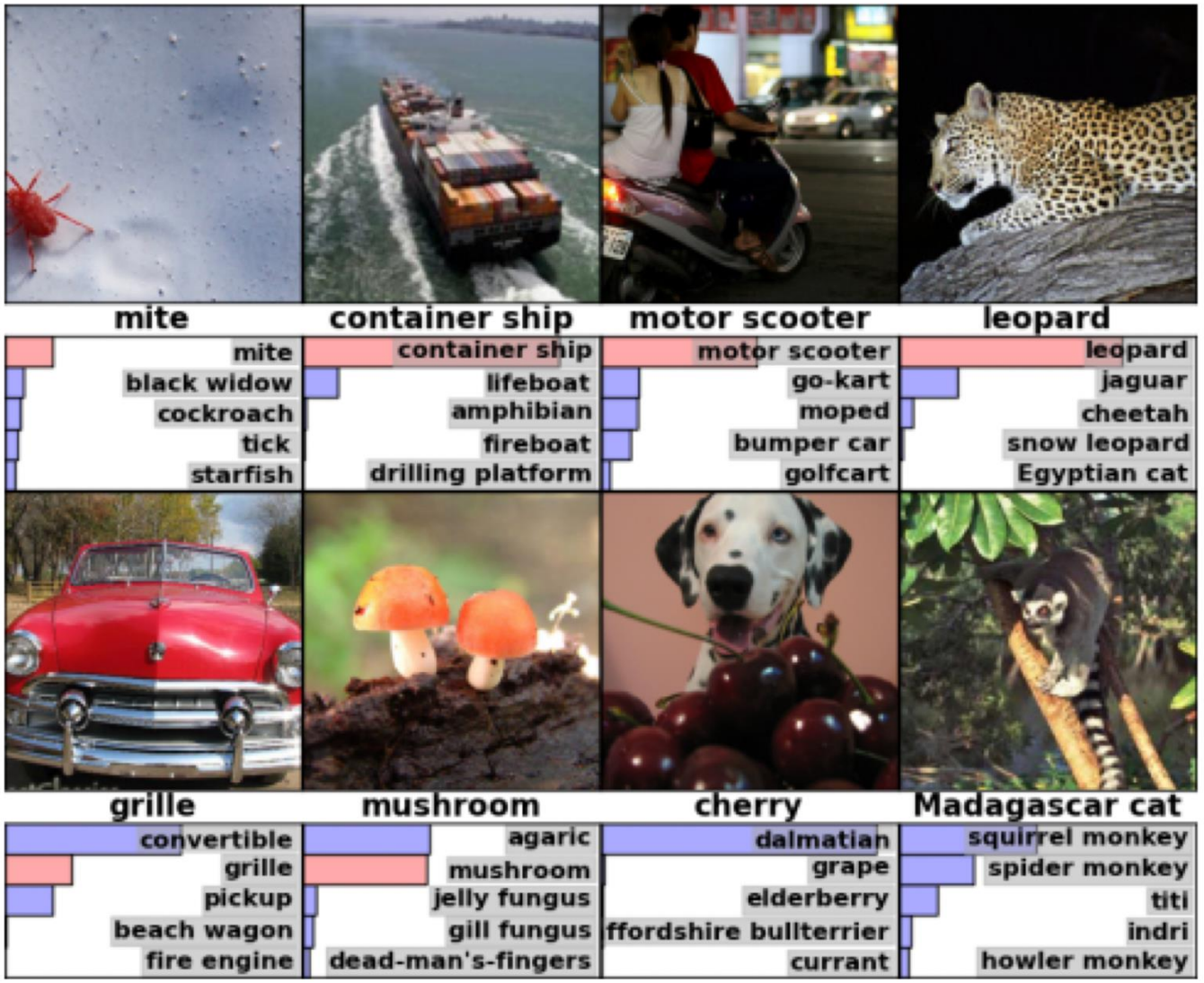
ImageNET Challenge

Classification goals:

- Make 1 guess about the label (Top-1 error)
- make 5 guesses about the label (Top-5 error)



Results


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

ImageNET Challenge

Example: Winner of 2011 ImageNet challenge

Low-level feature extraction ≈ 10k patches per image

- SIFT: 128-dim
- color: 96-dim

} reduced to 64-dim with PCA

FV extraction and compression:

- N=1,024 Gaussians, R=4 regions ⇨ 520K dim x 2
- compression: G=8, b=1 bit per dimension

One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

F. Perronnin, J. Sánchez, "Compressed Fisher vectors for LSVRC", PASCAL VOC / ImageNet workshop, ICCV, 2011.

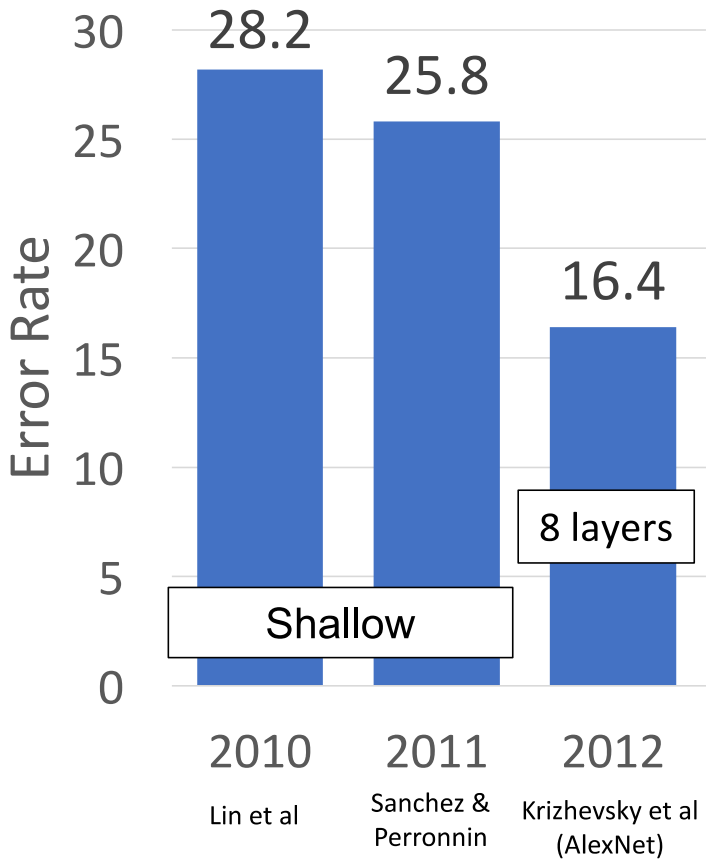
ImageNET Challenge

- In 2012, the winning team University of Toronto achieved error rate of 0.16.

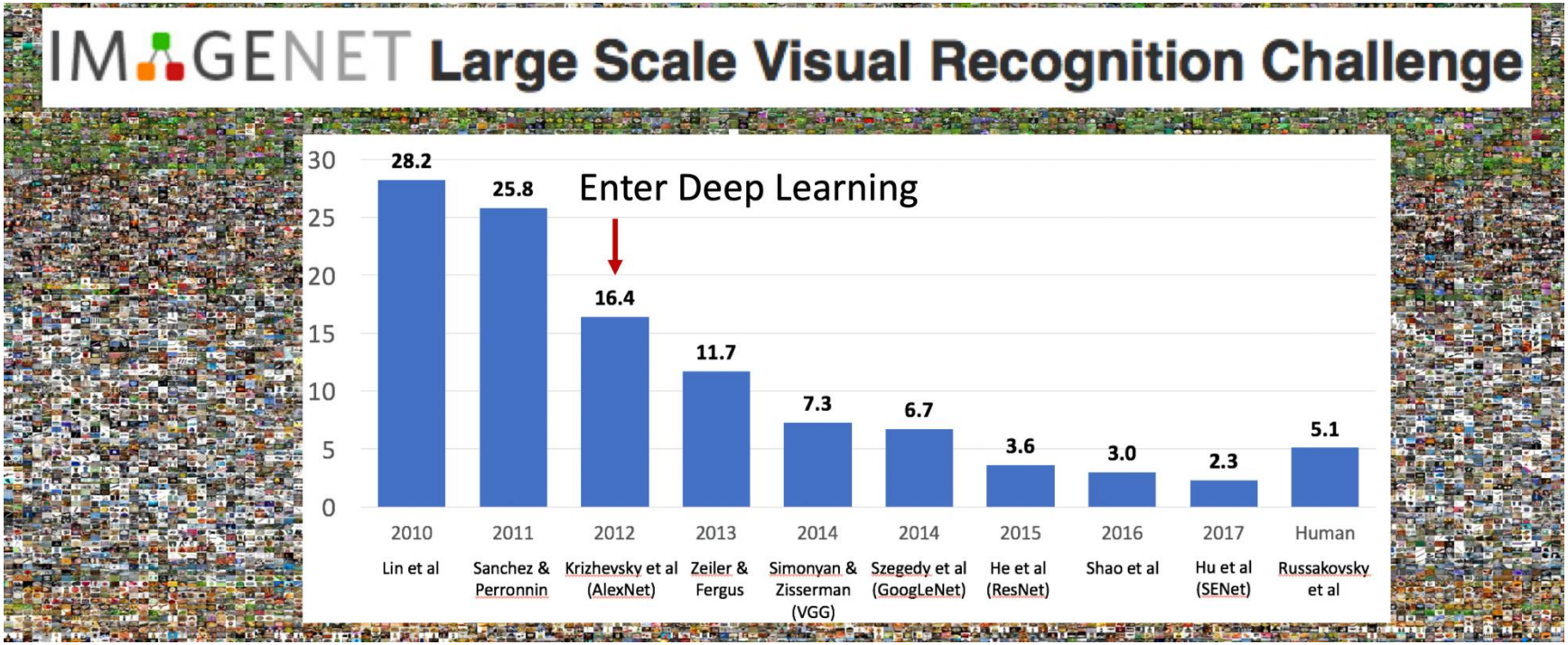
- A convolutional neural network (CNN) designed by Alex Krizhevsky, published with Ilya Sutskever and Krizhevsky's PhD advisor Geoffrey E. Hinton

- They trained the neural network model by using two GPUs, developing the most accurate classifier to date.

ImageNET Challenge



ImageNET Challenge



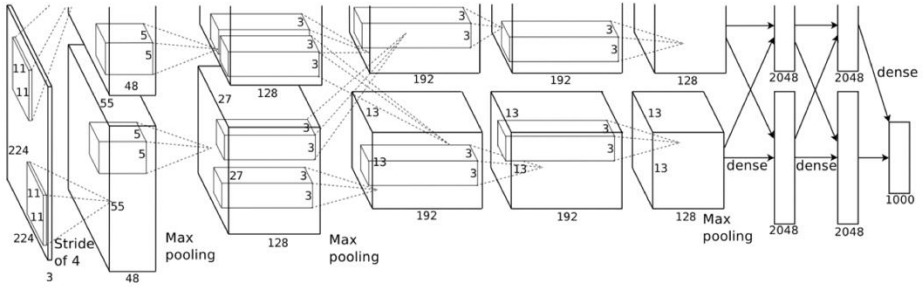
ImageNET Challenge

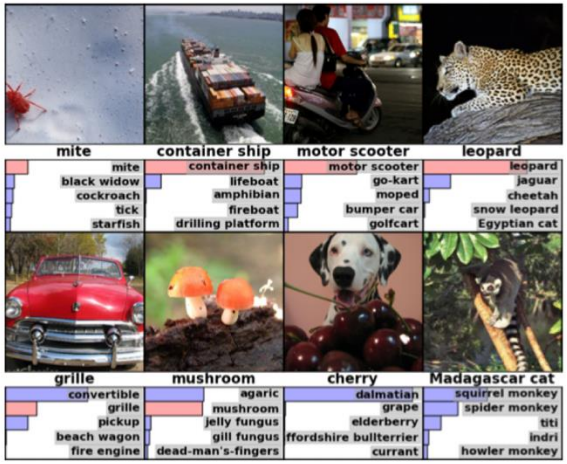
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



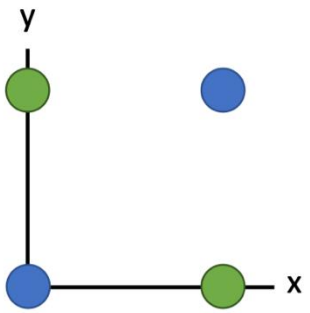


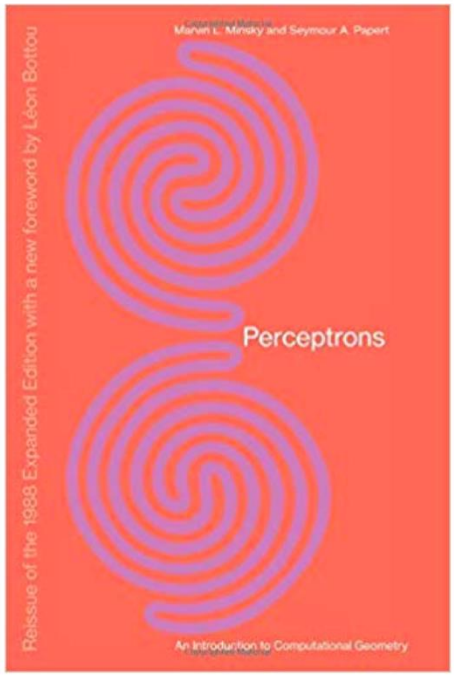
Krizhevsky et al., 2012

Artificial Neural Networks

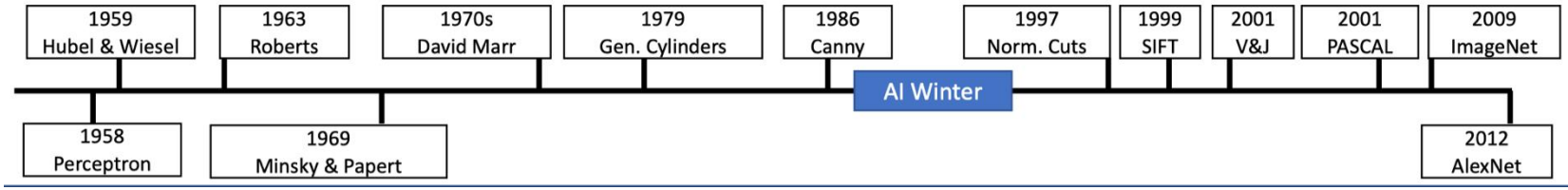
Minsky and Papert, 1969

X	Y	F(x,y)
0	0	0
0	1	1
1	0	1
1	1	0





Showed that Perceptrons could not learn the XOR function
Caused a lot of disillusionment in the field



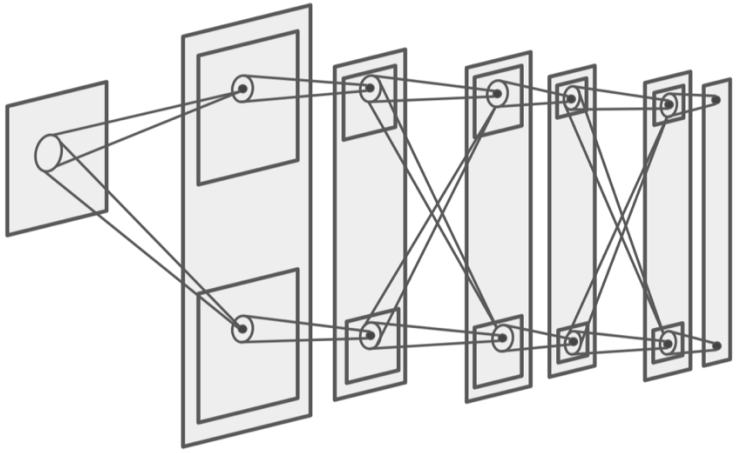
Artificial Neural Networks

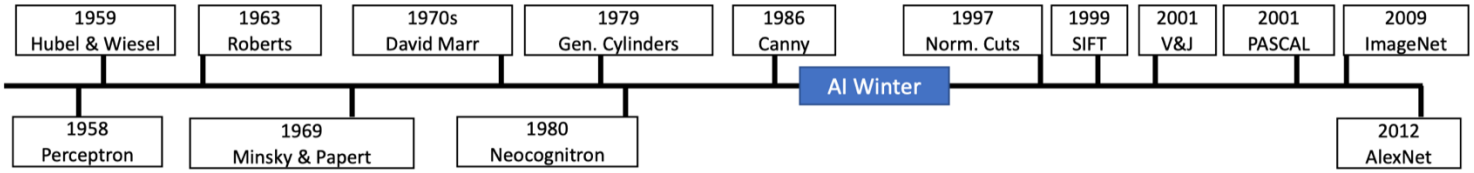
Neocognitron: Fukushima, 1980

Computational model the visual system, directly inspired by Hubel and Wiesel's hierarchy of complex and simple cells

Interleaved simple cells (convolution) and complex cells (pooling)

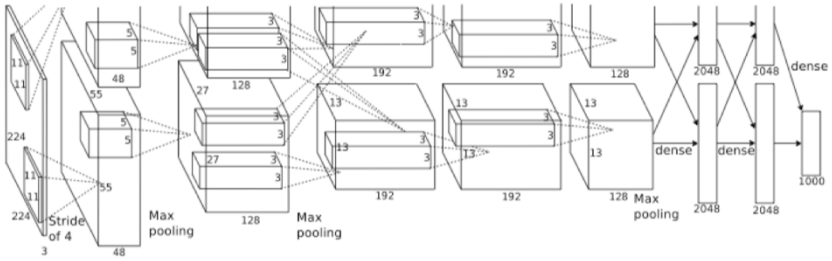
No practical training algorithm

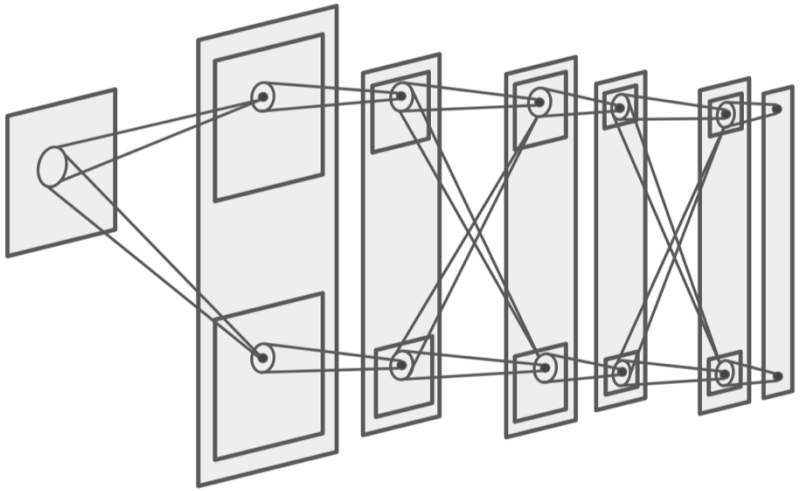




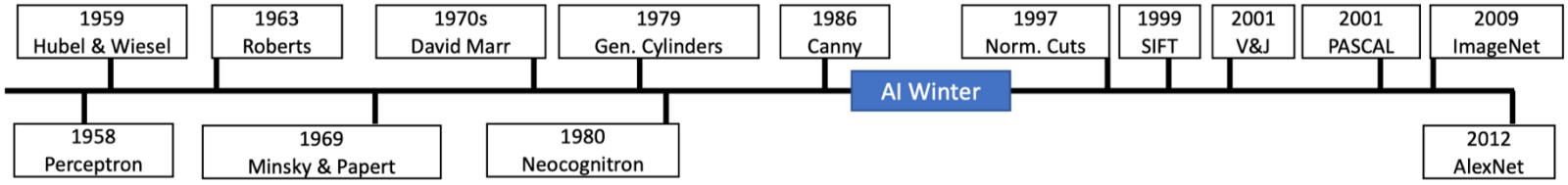
Artificial Neural Networks

Neocognitron: Fukushima, 1980



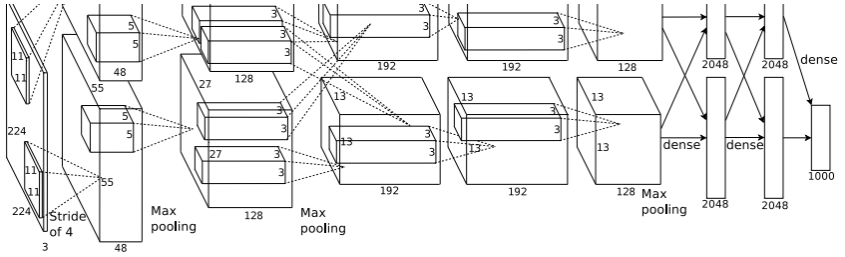


Looks a lot like AlexNet
more than 32 years later!



AlexNet

AlexNet

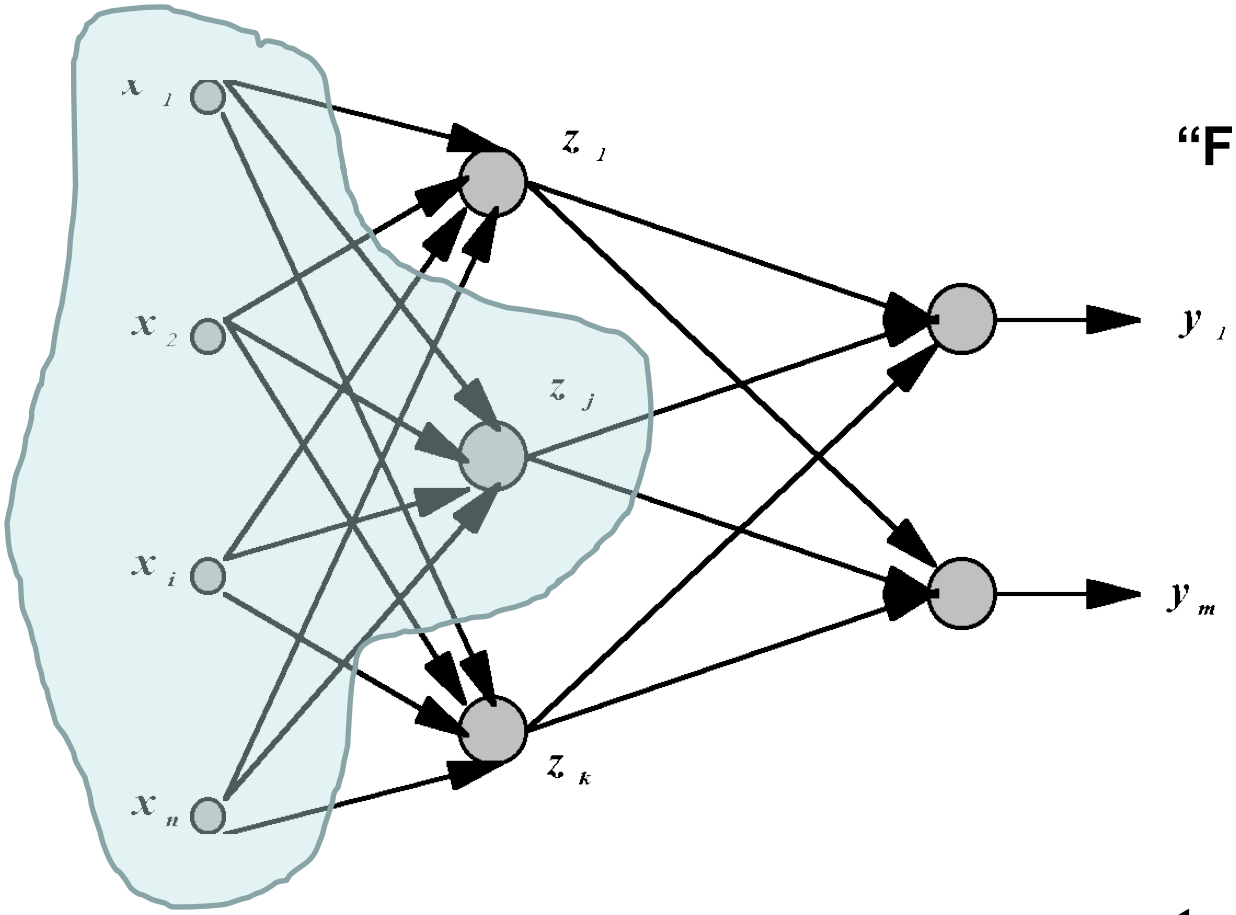


227x227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used "Local response normalization"; Not used anymore

Trained on two GTX 580 GPUs - only 3GB of memory each! Model split over two GPUs

Standard Neural Networks



"Fully connected"

$$\boldsymbol{x} = (x_1, \ldots, x_{784})^T \qquad z_j = g(\boldsymbol{w}_j^T \boldsymbol{x}) \qquad g(t) = \frac{1}{1 + e^{-t}}$$

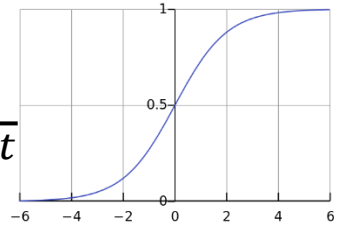
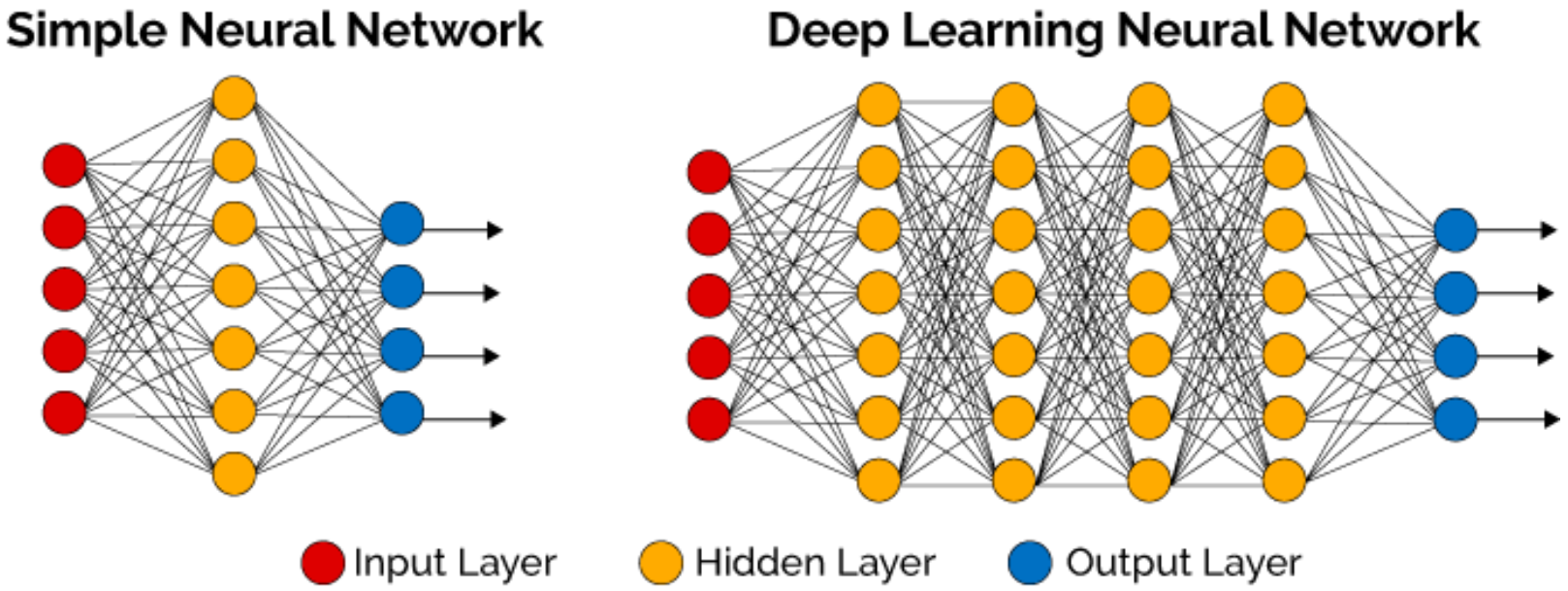


Image Size : 28x28

Deep Neural Network

- A neural network that consists of more than **three layers**—*which would be inclusive of the inputs and the output*—can be considered a deep learning algorithm



Are NNs likely to overfit? **YES**.

- Consequence of being able to fit any function!

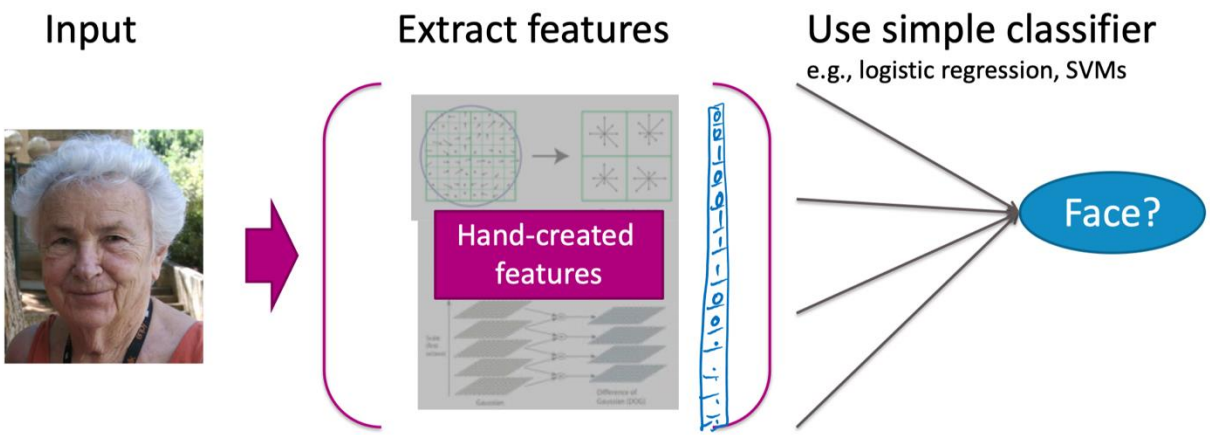
How to avoid overfitting?

- Get more training data
- Few hidden nodes / better topology
  - **Rule of thumb:** 3-layer NNs outperform 2-layer NNs, but going deeper rarely helps (different story next time with convolutional neural networks)
- Regularization
  - Dropout
- Early stopping

- Have risen in popularity due to huge datasets, GPUs, and improvements to

Object Detection

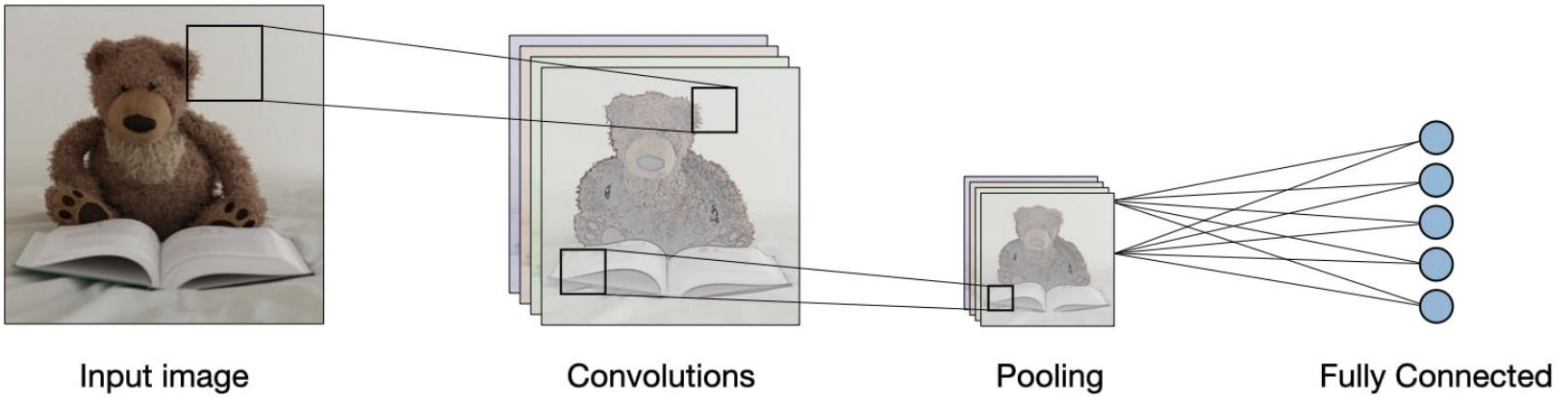
A popular approach to computer vision was to make hand-crafted features for object detection



Relies on coming up with these features by hand (yuck!)

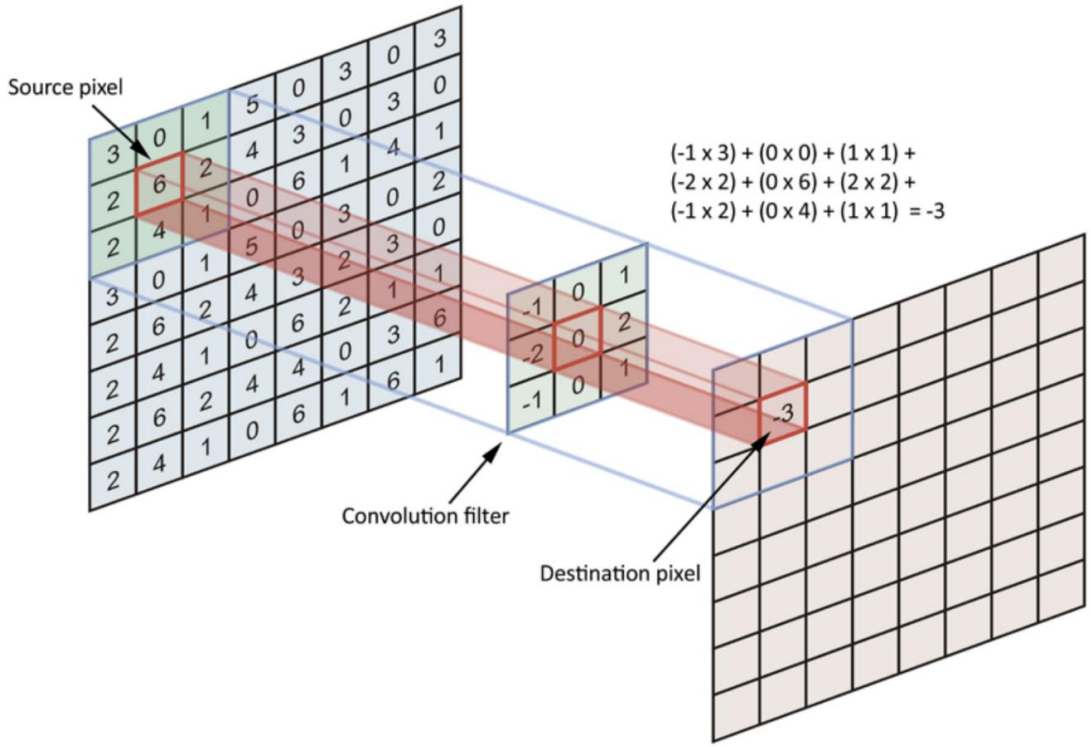
Convolutional Neural Network

A convolutional neural network (CNN, or ConvNet) is most commonly applied to analyze *visual imagery*

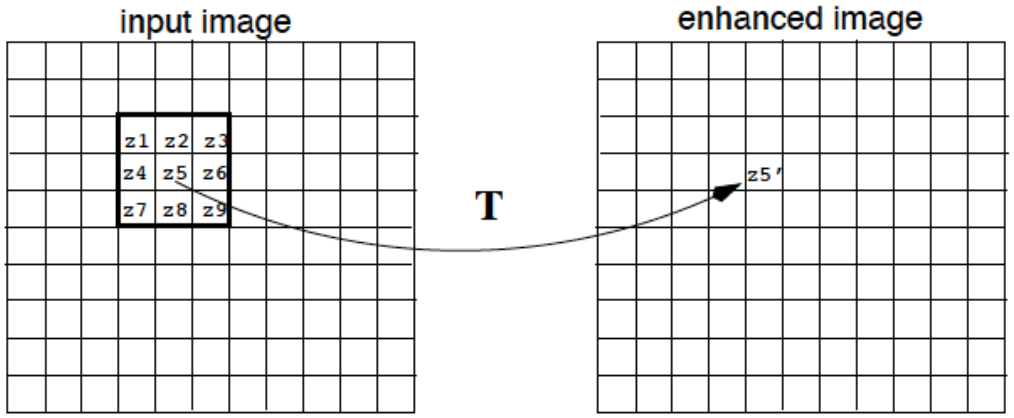


Convolution Layer

- The convolution layer (CONV) has filters that perform convolution operations



What is convolution?



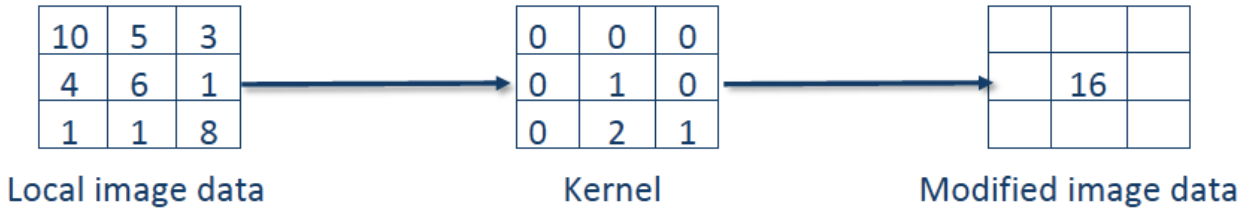
$$g(x,y) = T[f(x,y)]$$

T operates on a neighborhood of pixels

w1	w2	w3
w4	w5	w6
w7	w8	w9

$$z5' = R = w1z1 + w2z2 + \ldots + z9w9$$

Applying a kernel to an image



$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i z_i$$

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Where the w's are mask coefficients, the z's are the value of the image gray levels corresponding to those coefficients

2-D Convolution

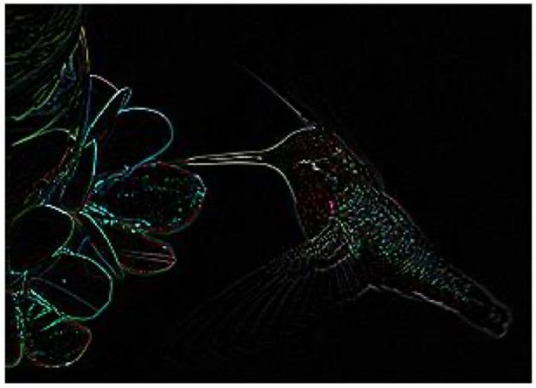
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

https://graphics.stanford.edu/courses/cs178/applets/convolution.html

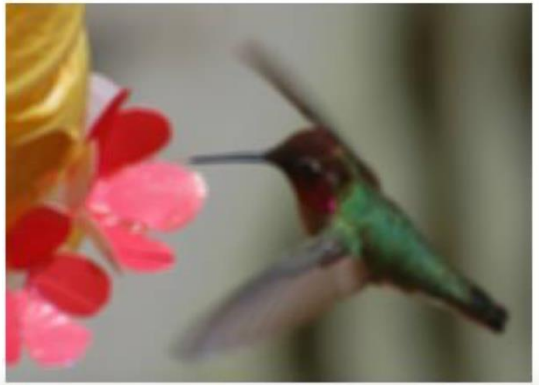
Original

Filter (=kernel)

0.00	0.00	0.00	0.00	0.00
0.00	0.00	-2.00	0.00	0.00
0.00	-2.00	8.00	-2.00	0.00
0.00	0.00	-2.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00



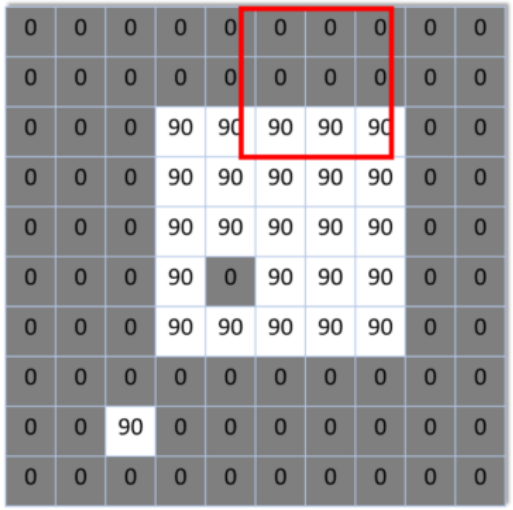
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04



Applying a kernel to an image

Averaging filter

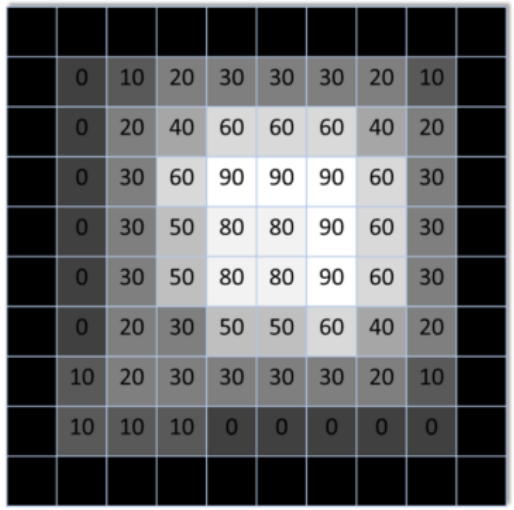
$$F(x,y) \quad * \quad H(u,v) \quad = \quad G(x,y)$$



$$* \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

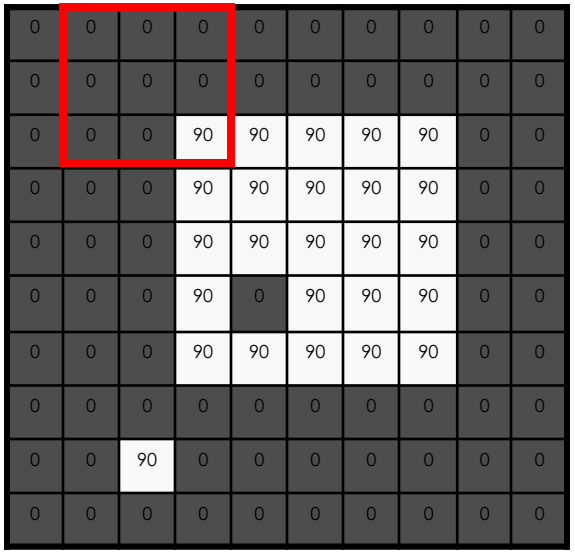
"box filter"



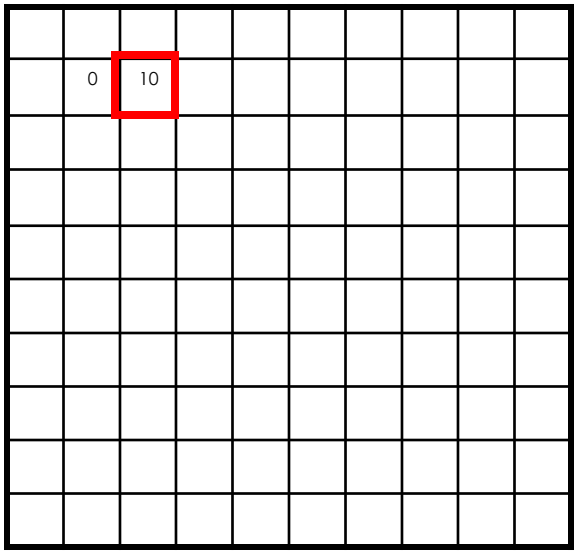
$$G = F * H$$

Filter example #1: Moving Average

$\mathsf{f}[n,m]$

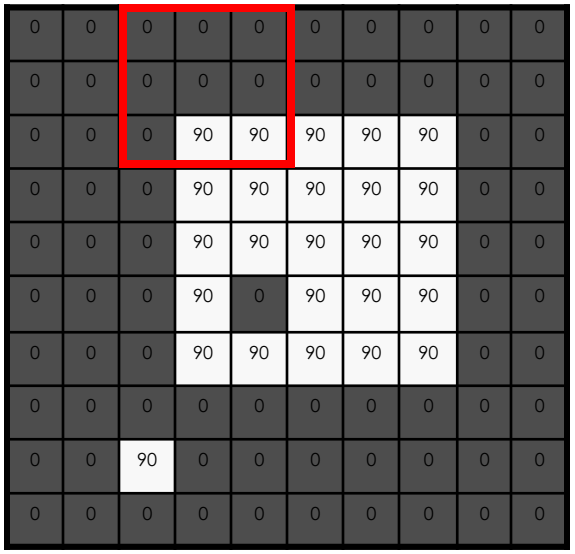


$\mathbf{g}[n,m]$

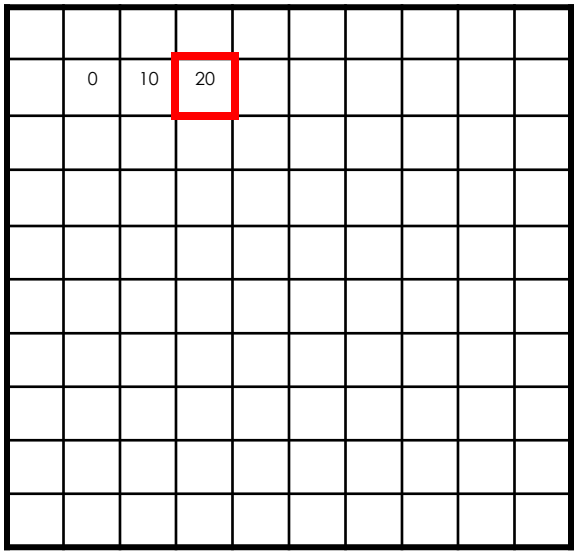


Filter example #1: Moving Average

$f[n,m]$

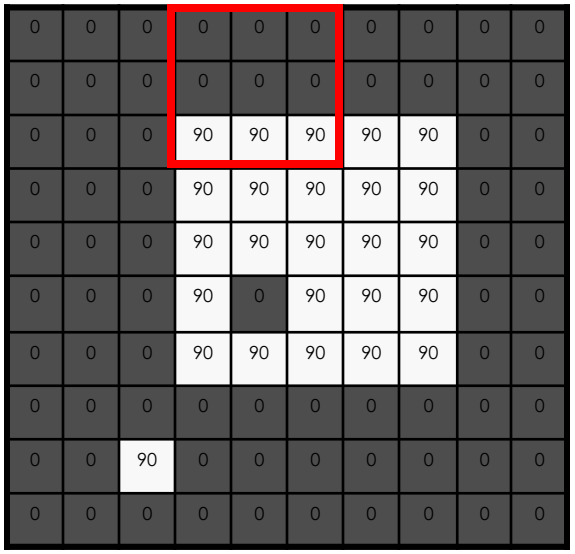


$\mathbf{g}[n,m]$

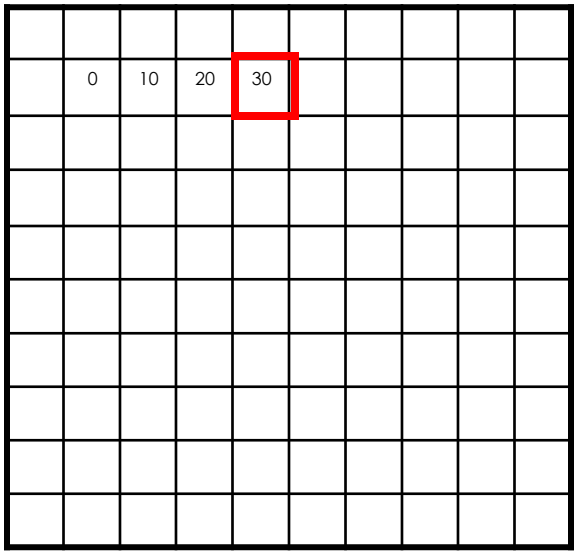


Filter example #1: Moving Average

$\mathsf{f}[n,m]$

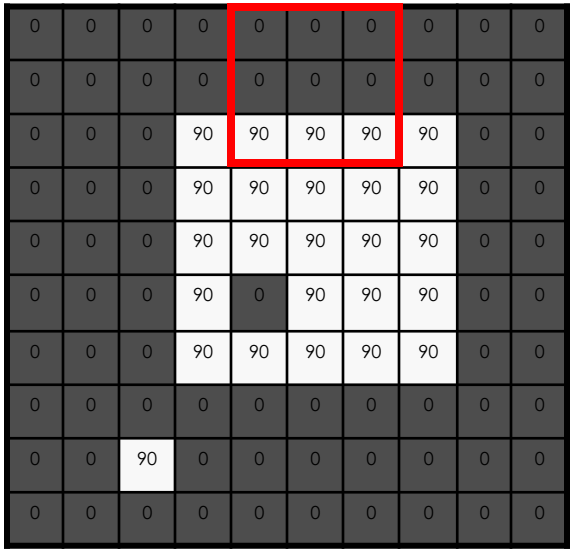


$\mathbf{g}[n,m]$

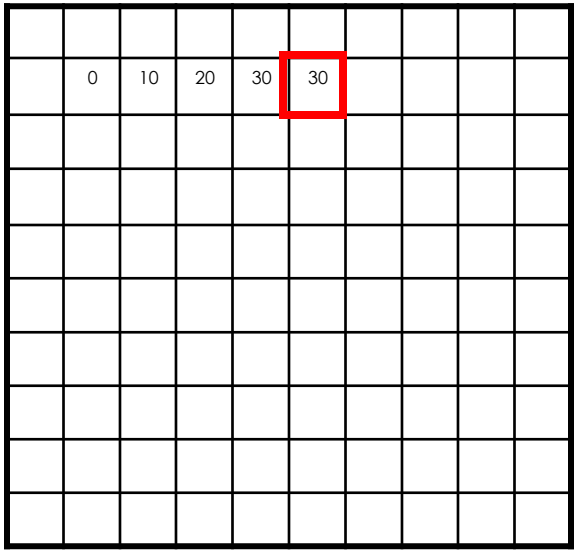


Filter example #1: Moving Average

$\mathsf{f}[n,m]$



$\mathbf{g}[n,m]$



Filter example #1: Moving Average

$\mathsf{f}[n,m]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

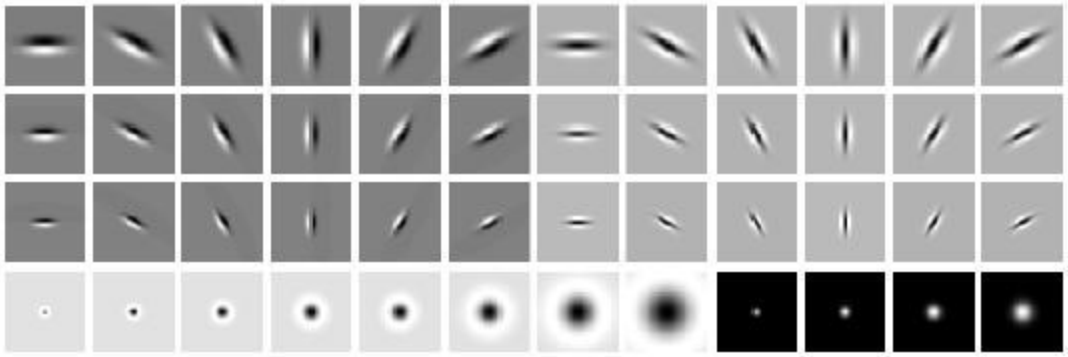
$\mathbf{g}[n,m]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

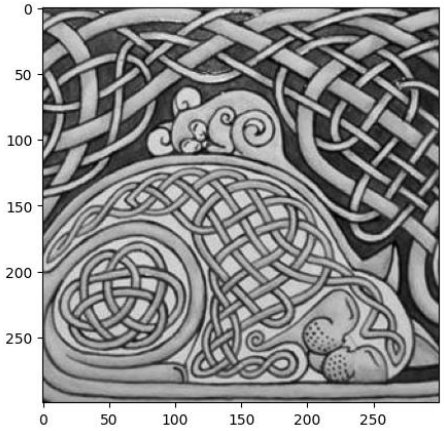
Averaging Filter Resuluts for Different Sizes

Texture Analysis

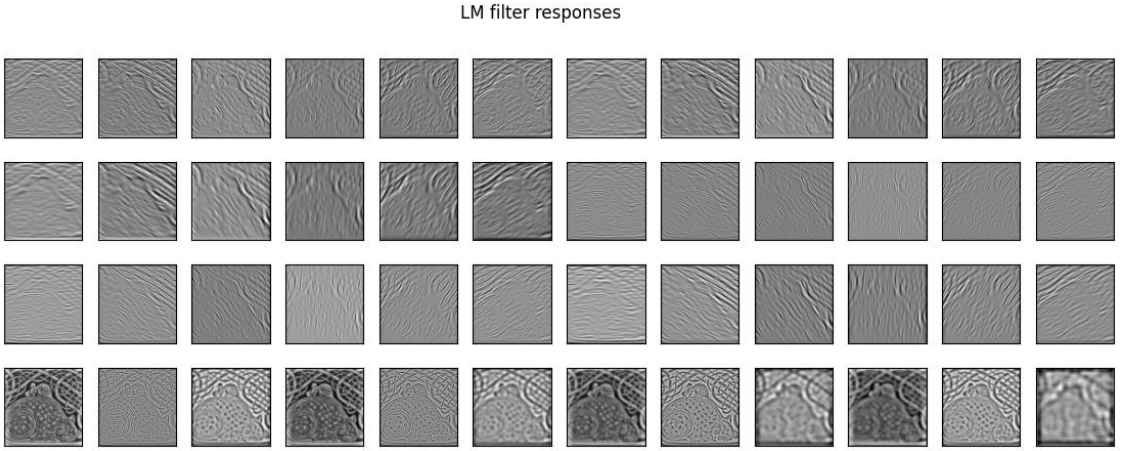
- The Leung-Malik (LM) Texture Filter Bank



Texture Analysis

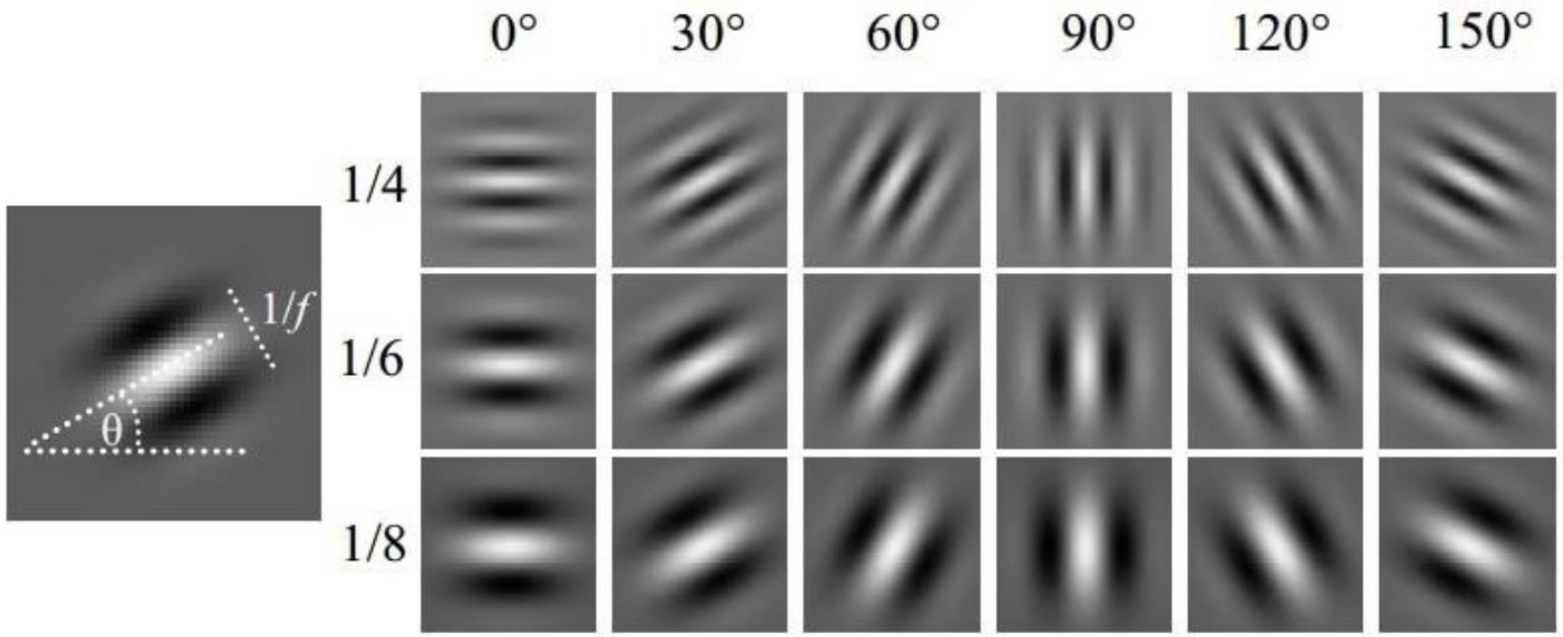


Input Image



LM Filter response

Gabor Filters



A Gabor filter-bank with six orientations θ (columns) and three frequencies f (rows) [8].

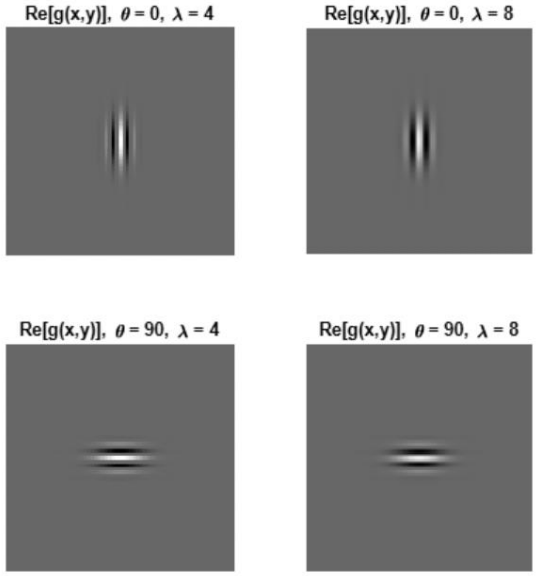
$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left[i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right]$$

where $x' = x\cos\theta + y\sin\theta$ and $y' = -x\sin\theta + y\cos\theta$.

Gabor Filters

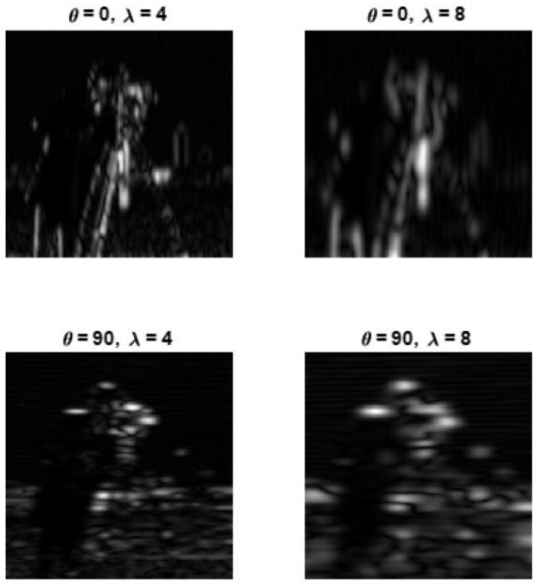
Original image

*



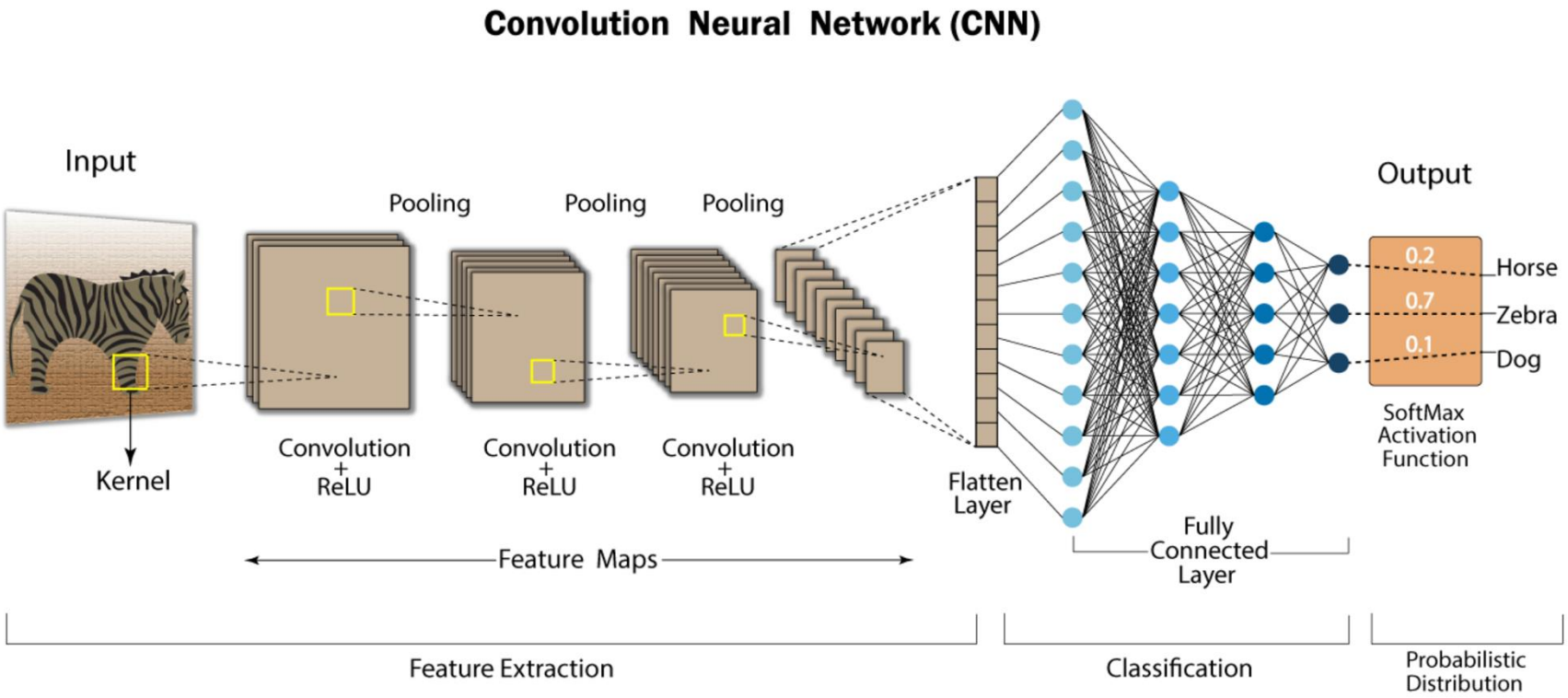
Gabor filters

=



Filtered images

Convolutional Neural Network



https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Convolution Layer

Description:

- Learnable filters (e.g. edge detector) organized in feature maps
- Each filter scans the image and detects a specific pattern
- Convolution refers to the spatial dimensions
- Input and outputs channels are still fully connected

Convolution Layer- Some Filters

Edge detection

Kernel

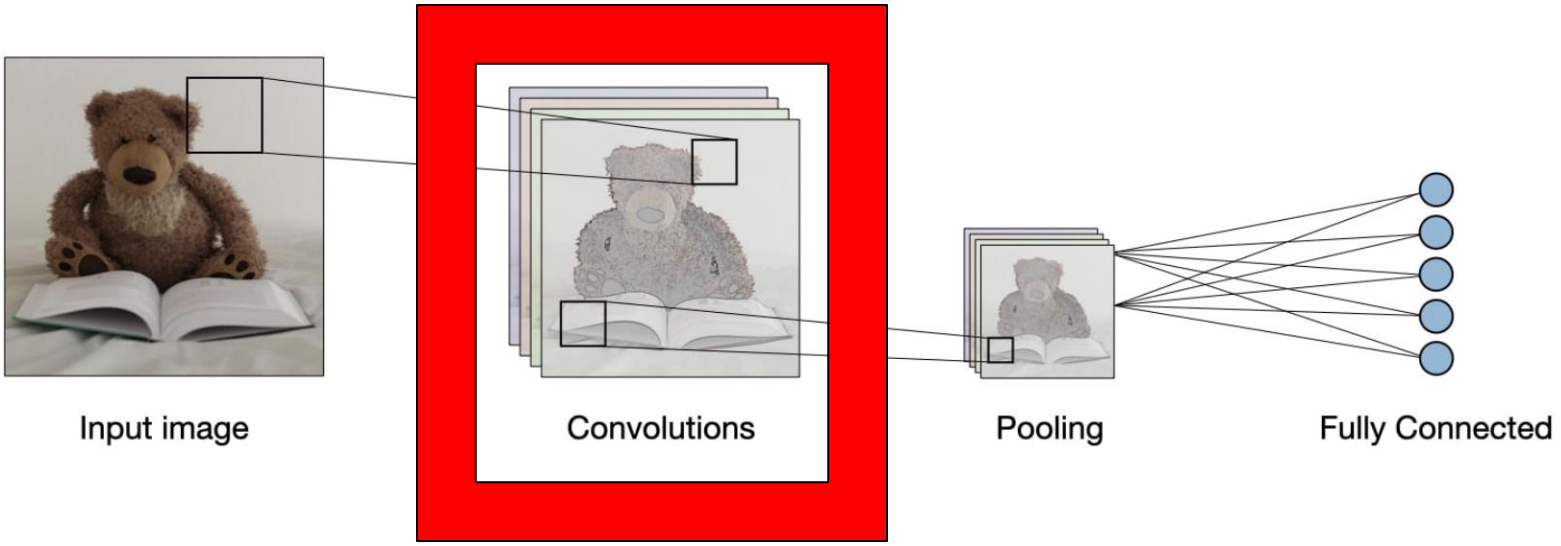
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Sharpen

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

Convolutional Neural Network

A convolutional neural network (CNN, or ConvNet) is most commonly applied to analyze *visual imagery*

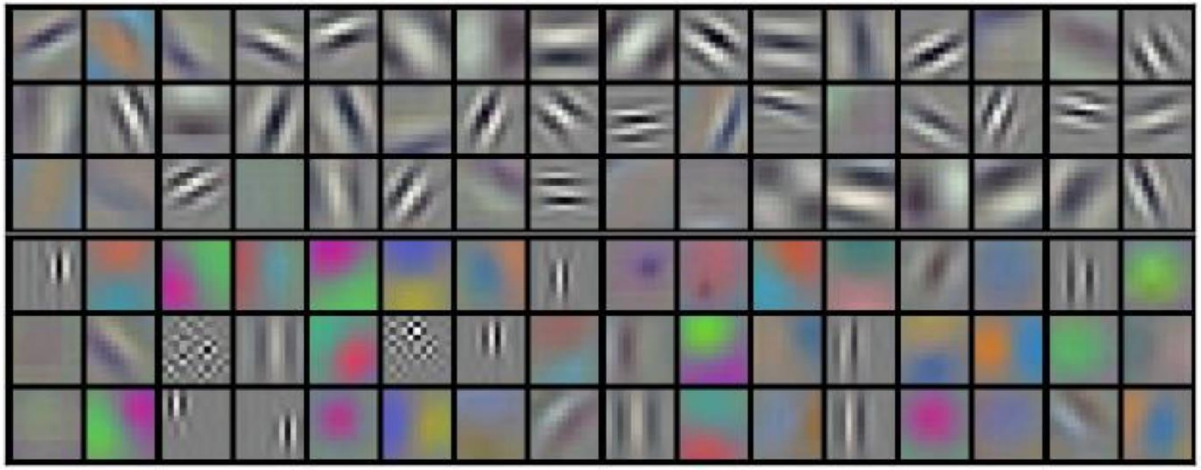


Convolution Layer

Hyper-Parameters:

- depth – number of filters (also known as kernels)
- size – dimension of the filter e.g. 3×3 or $3 \times 3 \times 4$
- stride – step size while sliding the filter through the input
- padding – behavior of the convolution near the borders

The first Convolution Layer



96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images.

The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU2

Looks like Gabor wavelets, ICA filters…

Convolution Layer

If the input image size is **n x n** and filter size is **f x f :**
after convolution, the size of the output image is:
(Size of input image – filter size + 1)
x
(Size of input image – filter size + 1)

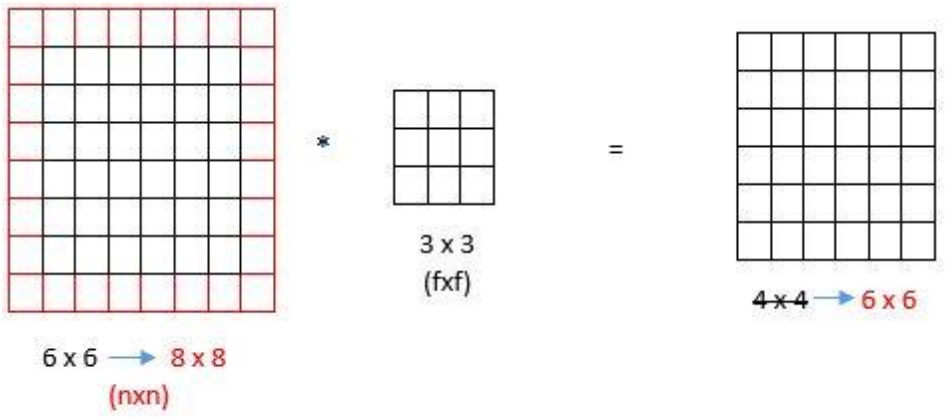


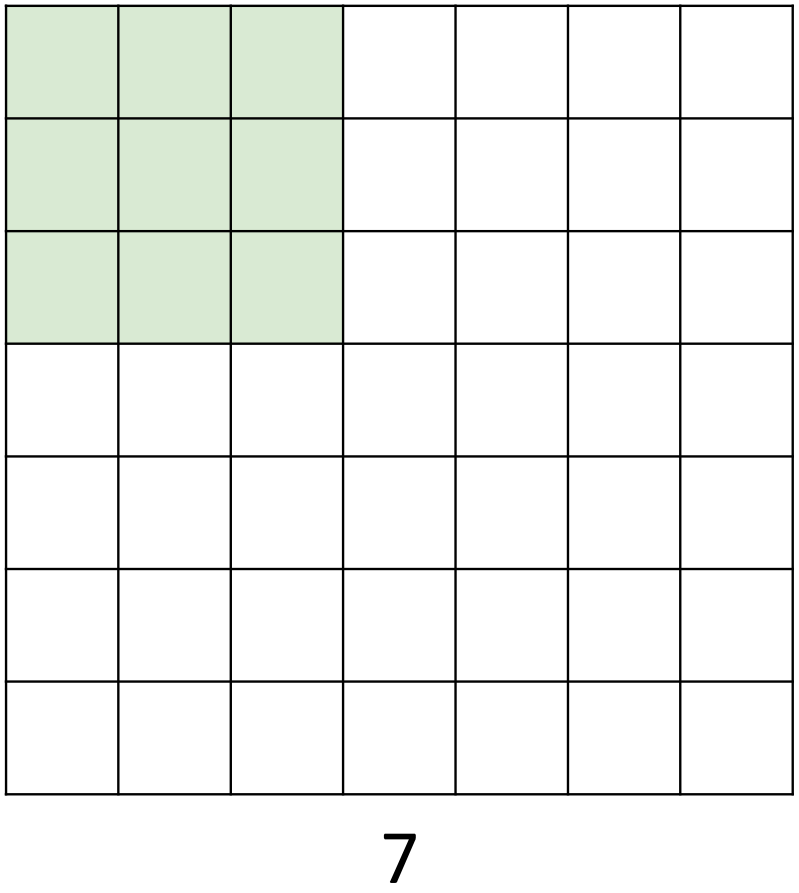
Fig: Padding the input image

Convolution Layer - Padding

- There are two common choices for padding:
  1. **Valid convolutions:** This Means no padding. Thus, in this case, we might have (nxn) image convolve with (fxf) filter & this would give us :

     (n-f+1) x (n-f+1) dimensional output.
  2. **Same convolutions:** In this case, padding is such that the output size is the same as the input image size. When we do padding by 'p' pixels then, size of the input image changes from (nxn) to (n + 2p –f +1) x (n + 2p –f +1).

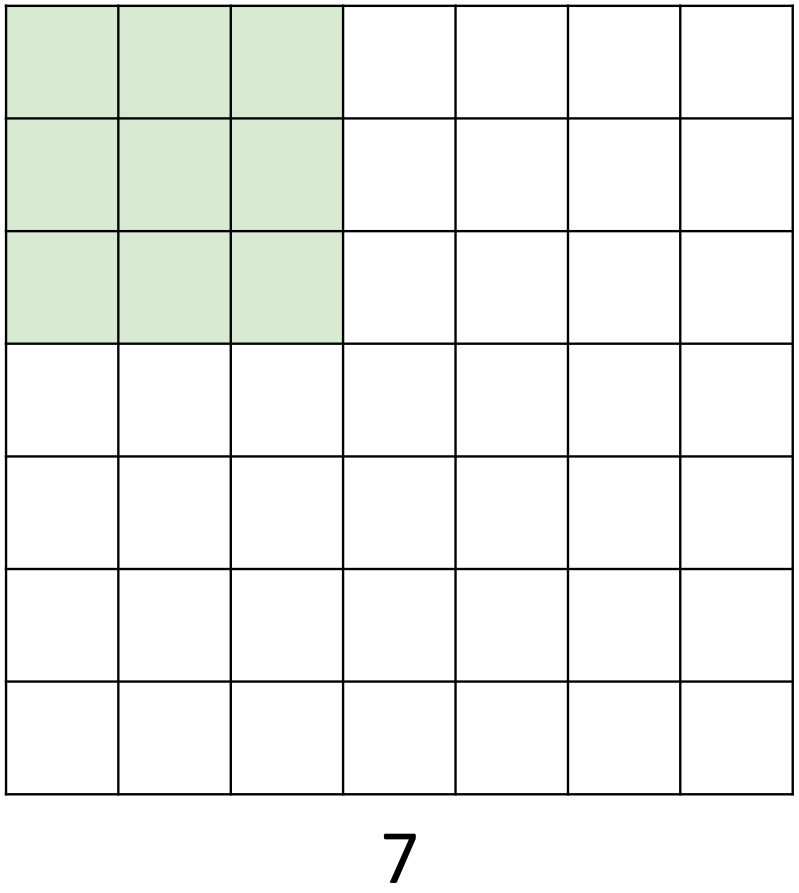
Convolution Layer



Input: 7x7

Filter: 3x3

Convolution Layer

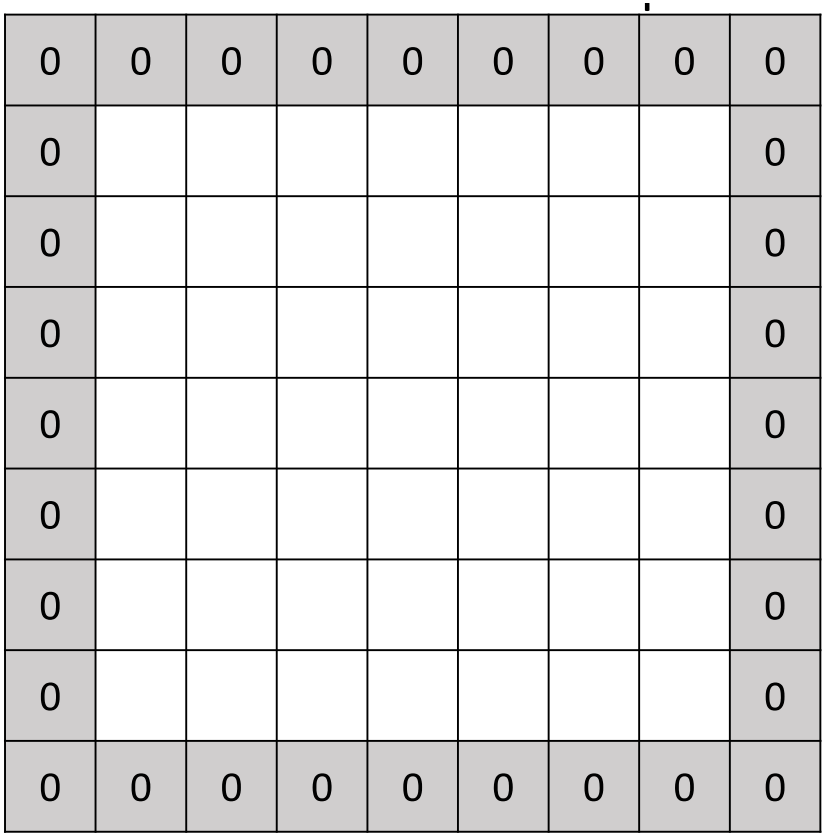


Input: 7x7
Filter: 3x3

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Convolution Layer - Padding



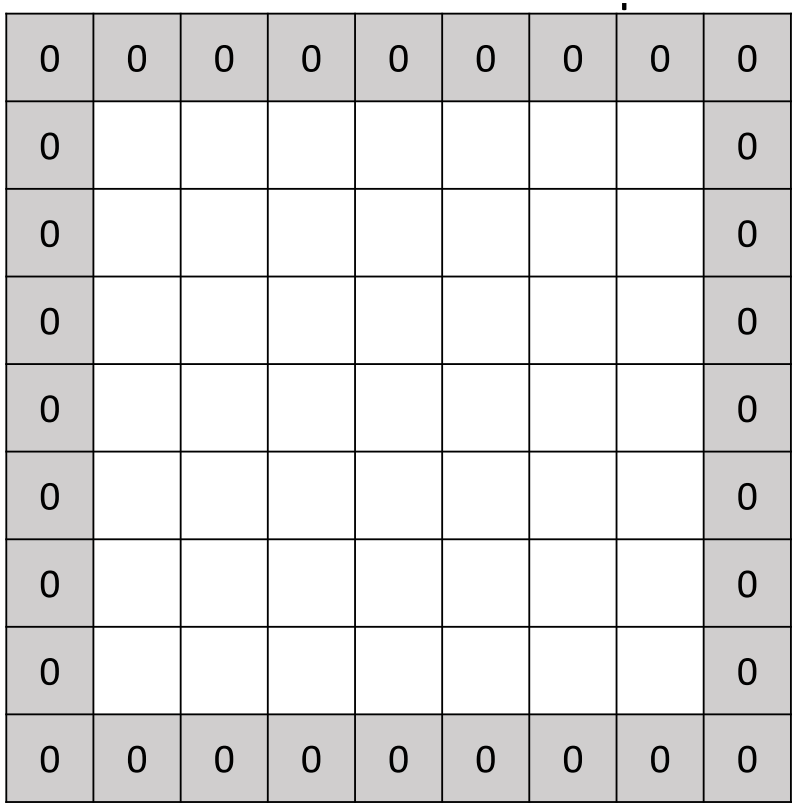
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

Convolution Layer - Padding



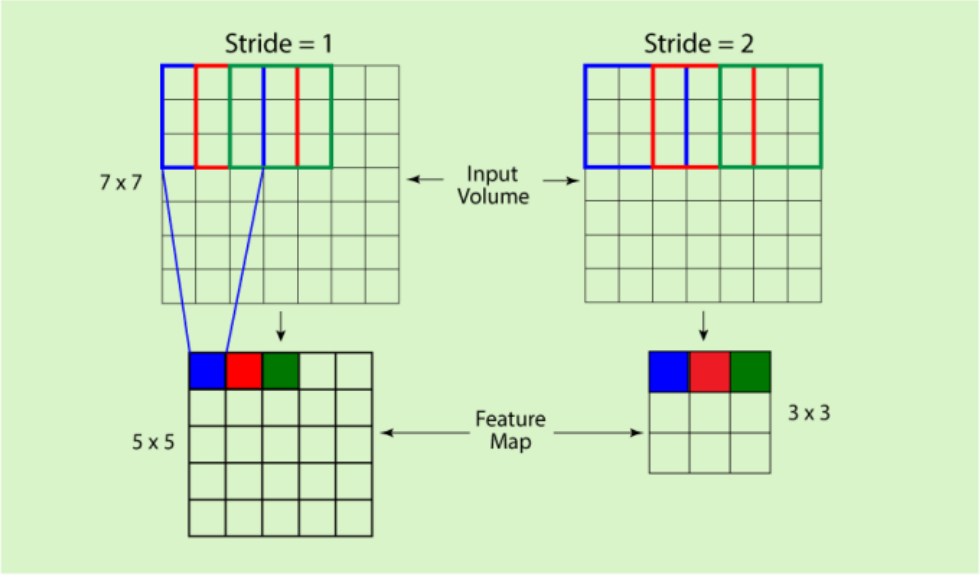
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common:
Set P = (K – 1) / 2 to make output have same size as input!

Convolution Layer - Stride

- Stride: the number of pixels by which the window moves after each operation.
- If stride is set to 1, filter moves across 1 pixel at a time and if stride is 2, filter moves 2 pixels at a time.
- the smaller the steps, the more details will be reflected in the resulting feature map



Convolution Layer - Stride

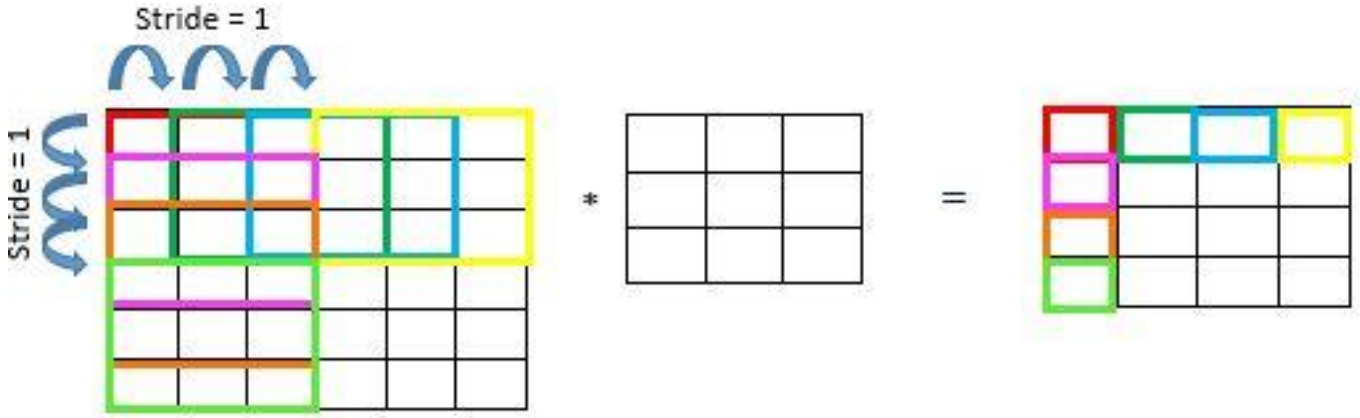


Fig: Stride during convolution

- We prefer a smaller stride size if we expect several fine-grained features to reflect in our output.
- On the other hand, if we are only interested in the macro-level of features, we choose a larger stride size.

Convolution over RGB images

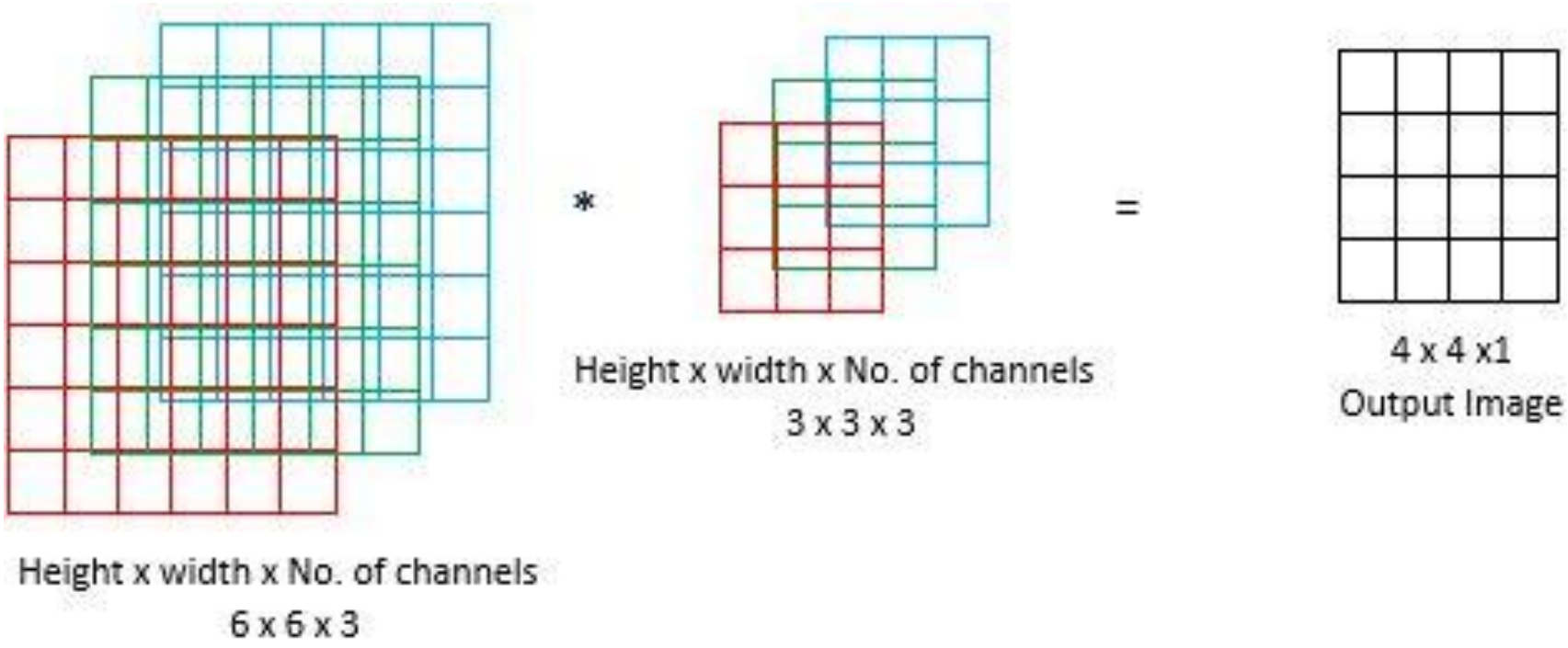
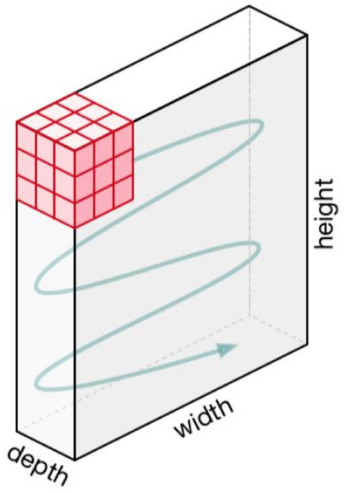


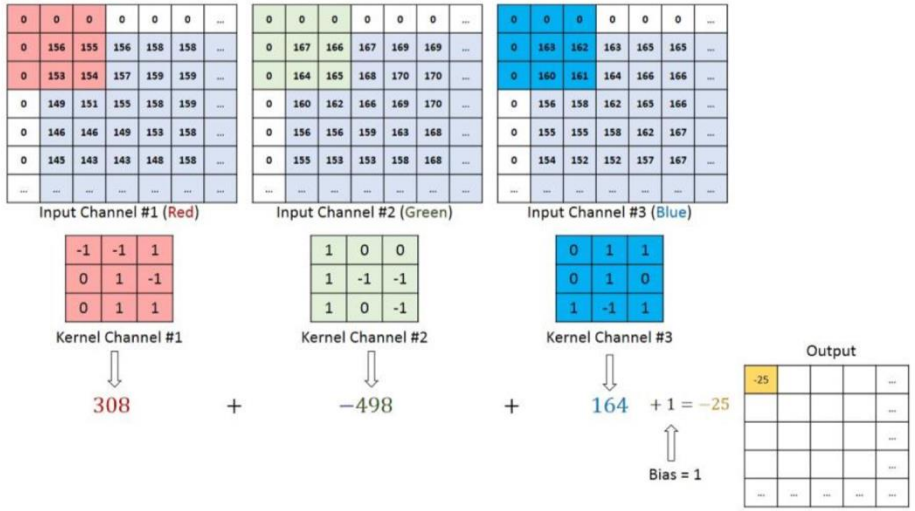
Fig: Convolution over volume

CNN with Color Images

How does this work if there is more than one input channel?

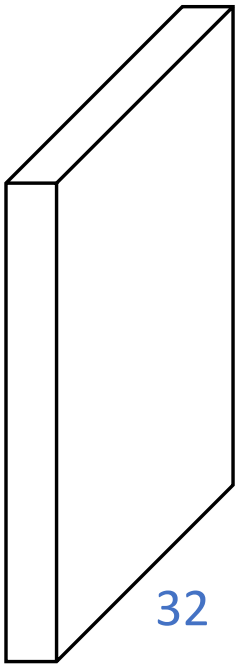
- Usually, use a 3 dimensional **tensor** as the kernel to combine information from each input channel





Convolution Layer

3x32x32 image: preserve spatial structure

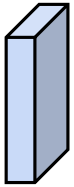


32 height

32 width

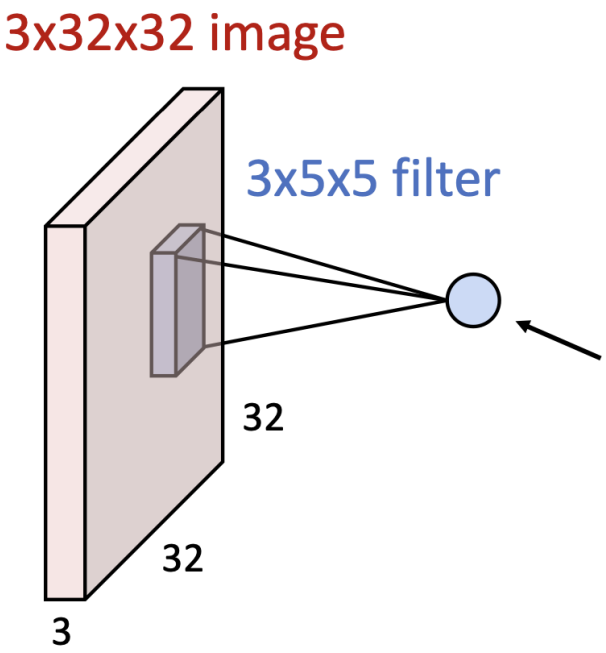
3 depth / channels

3x5x5 filter



Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"

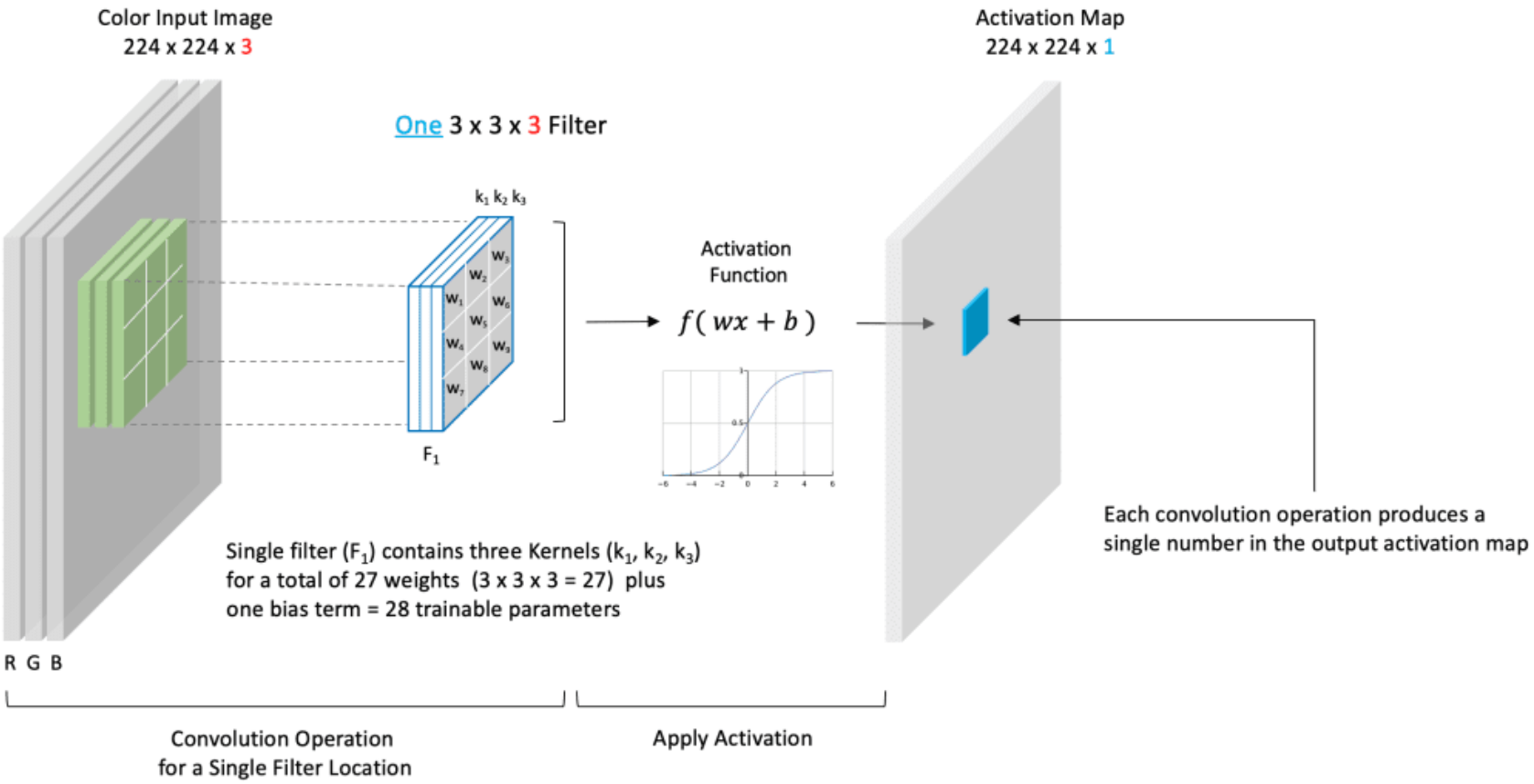
Convolution Layer



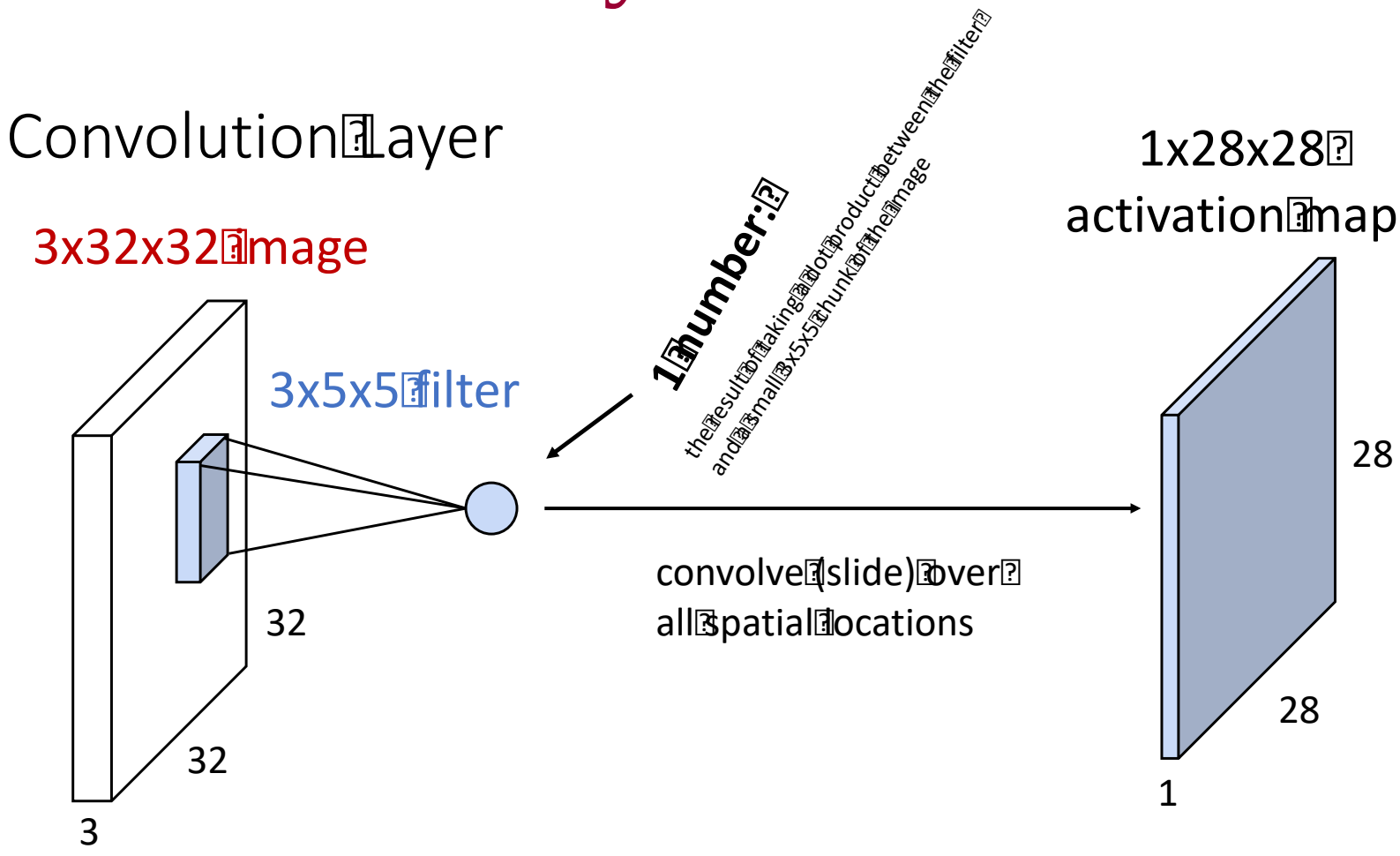
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

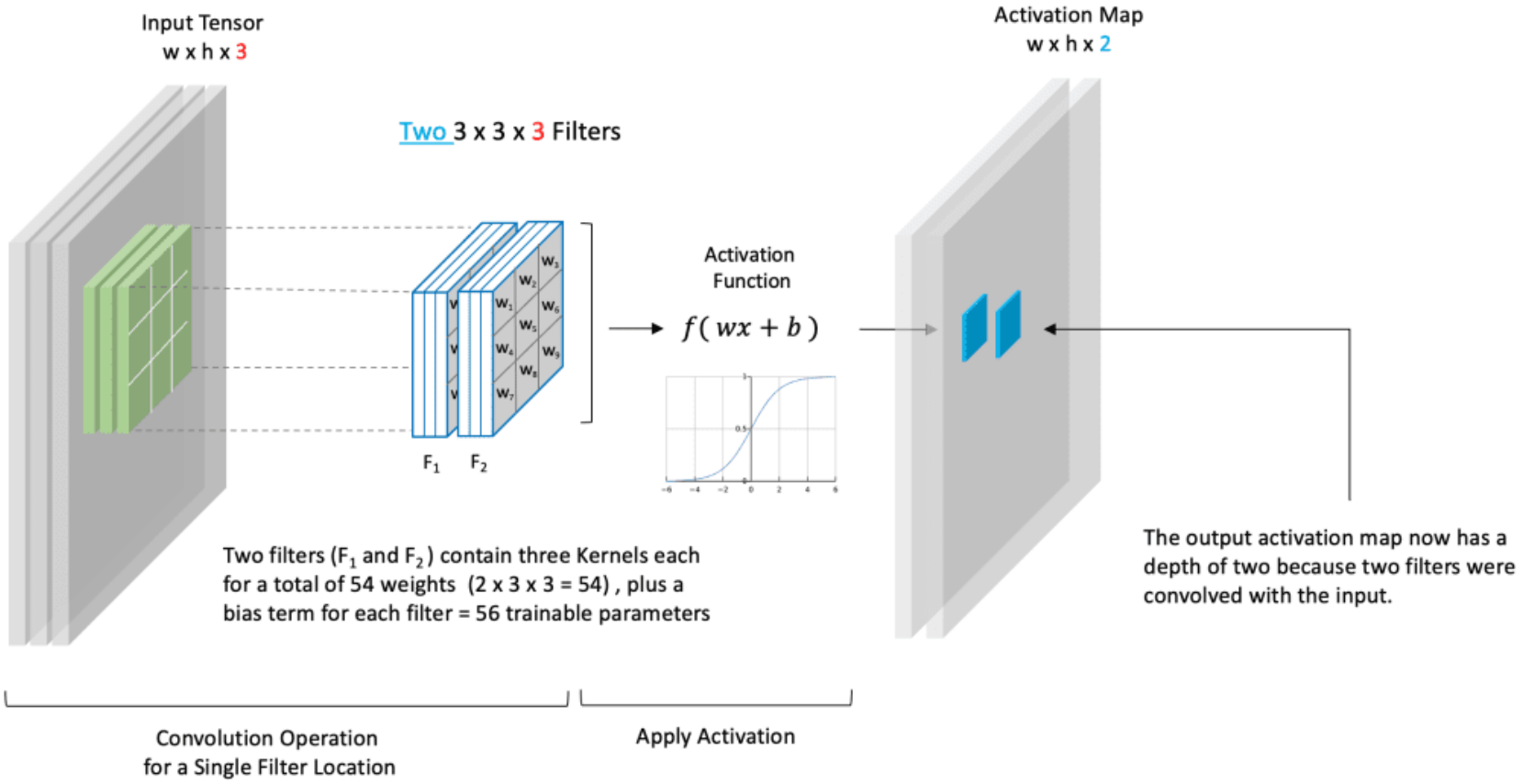
Convolution Layer – One Filter



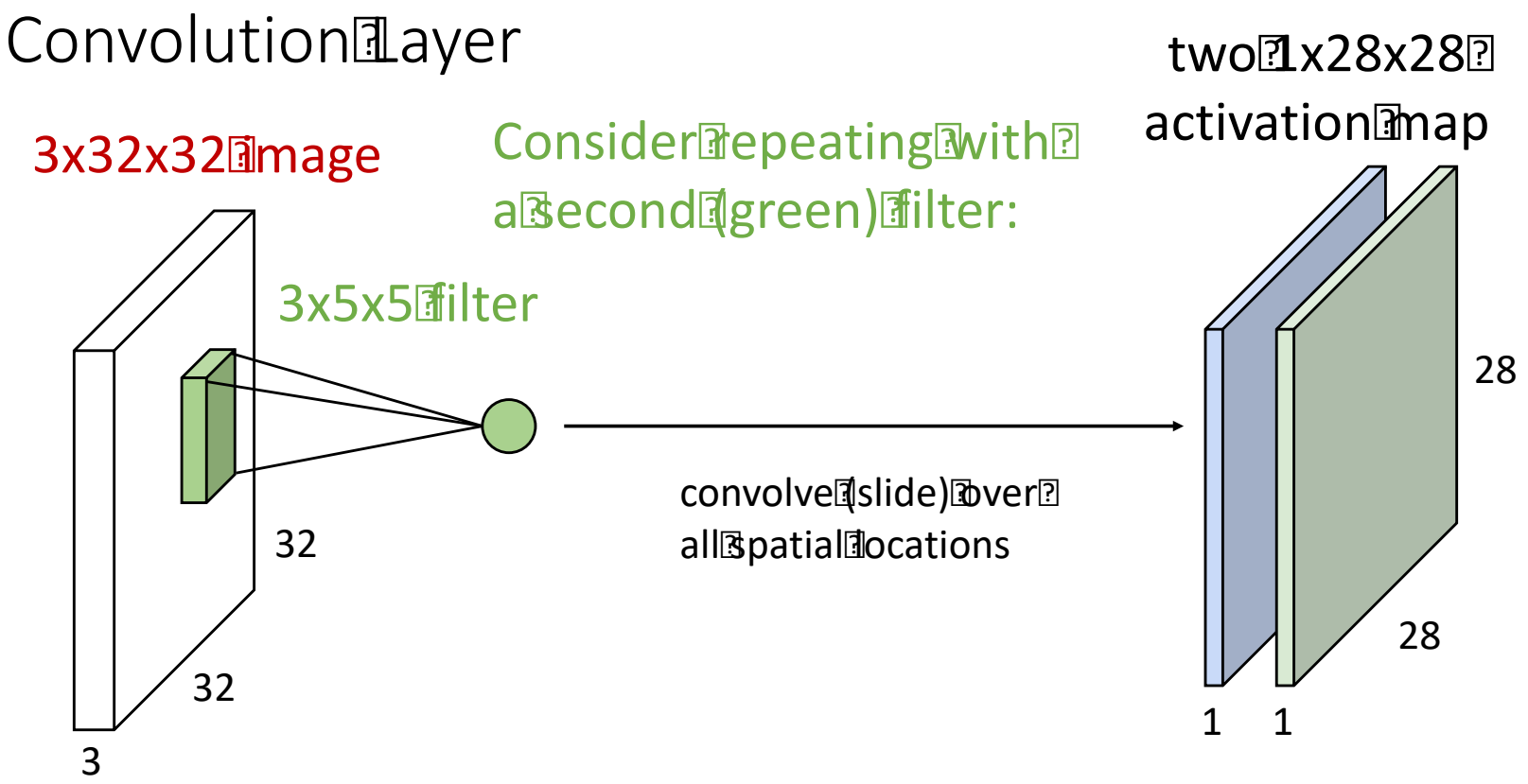
Convolution Layer – One Filter



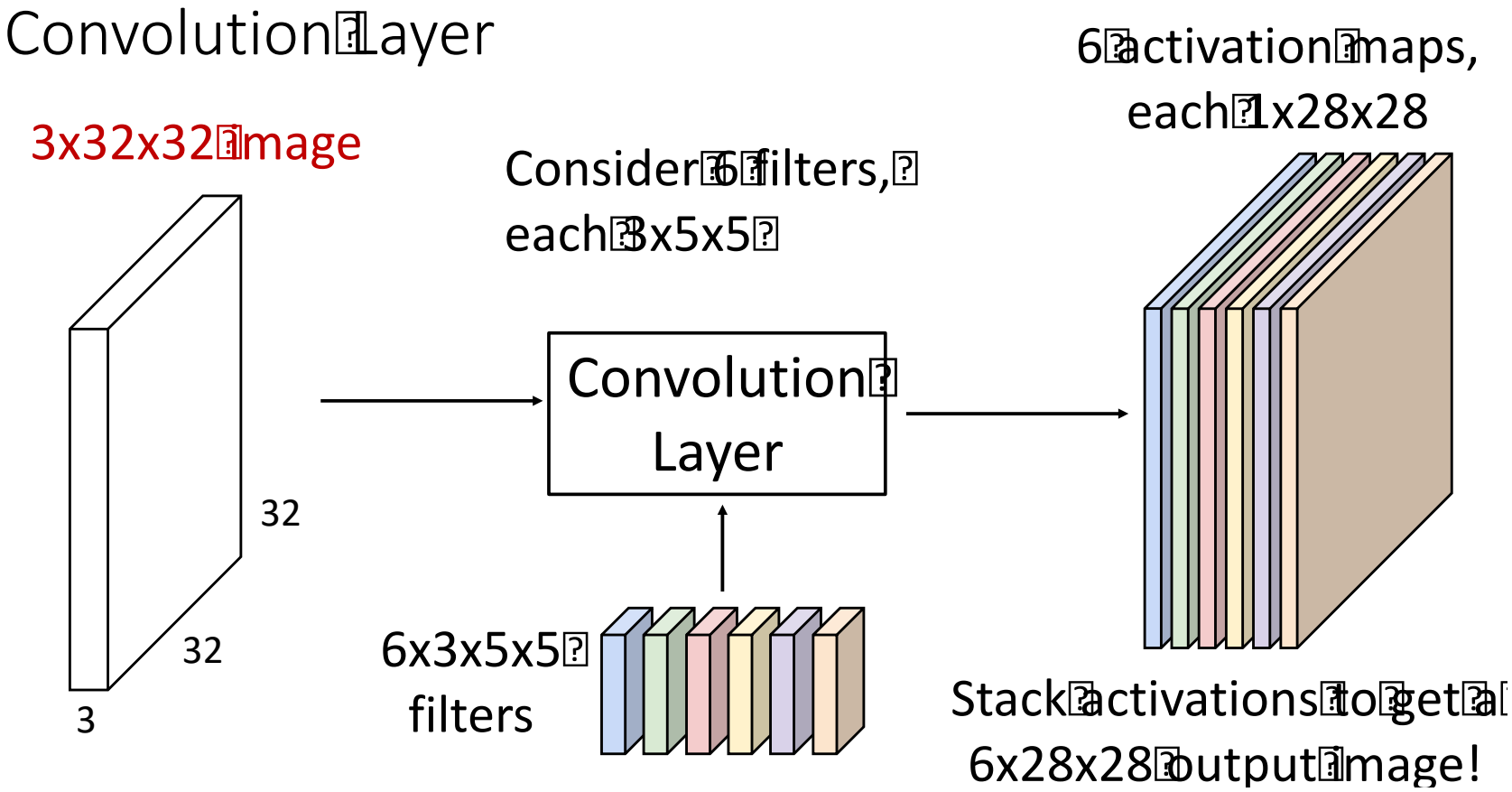
Convolution Layer – Two Filters



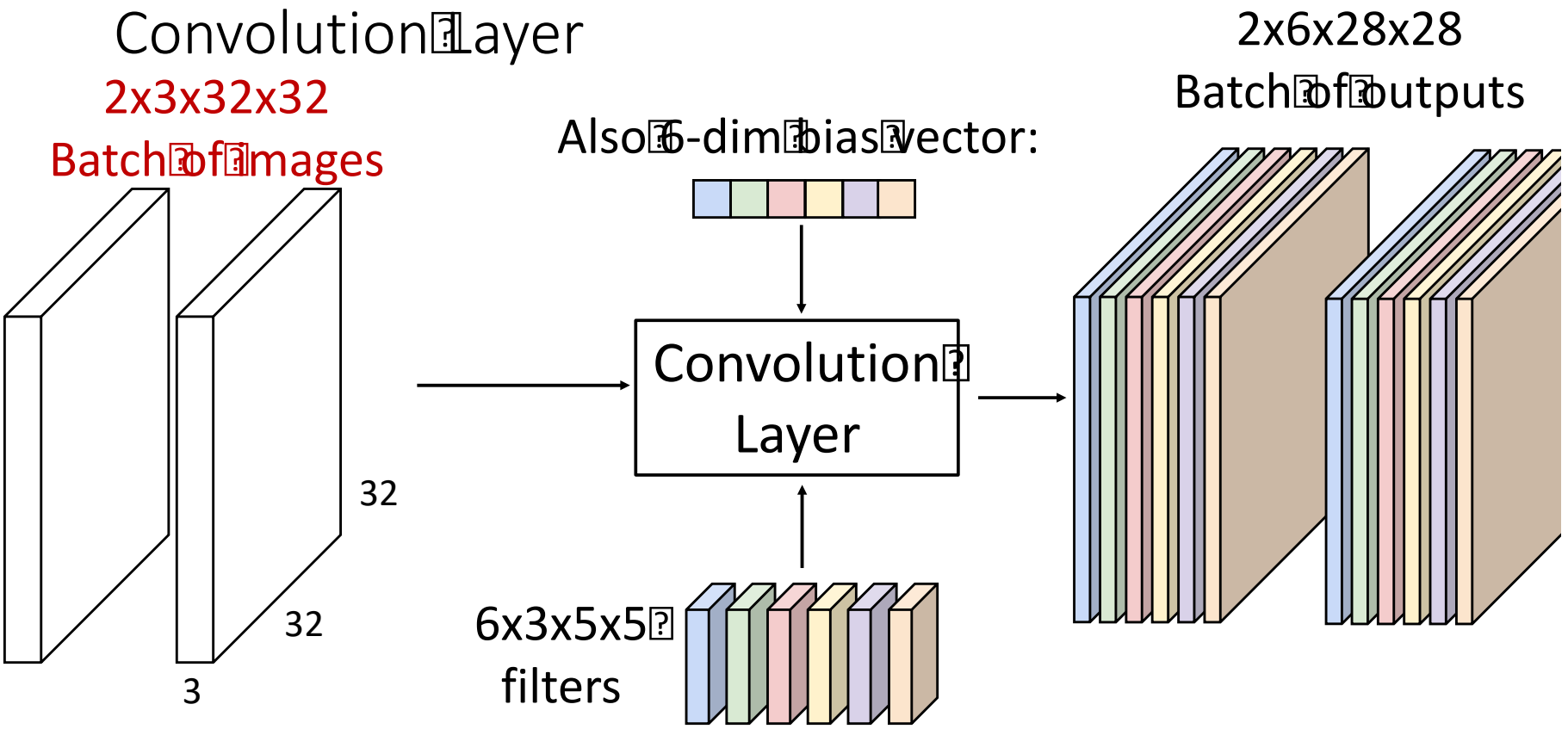
Convolution Layer – Two Filters



Convolution Layer

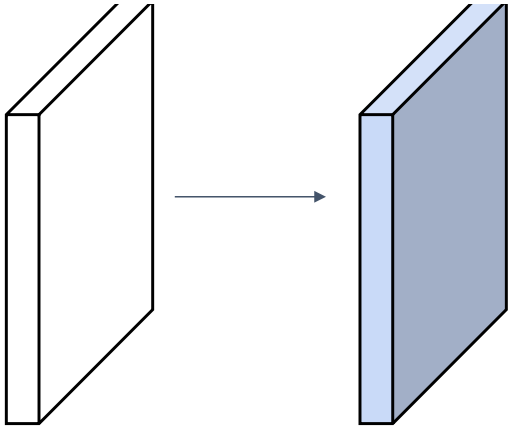


Convolution Layer



Example

Convolution Example

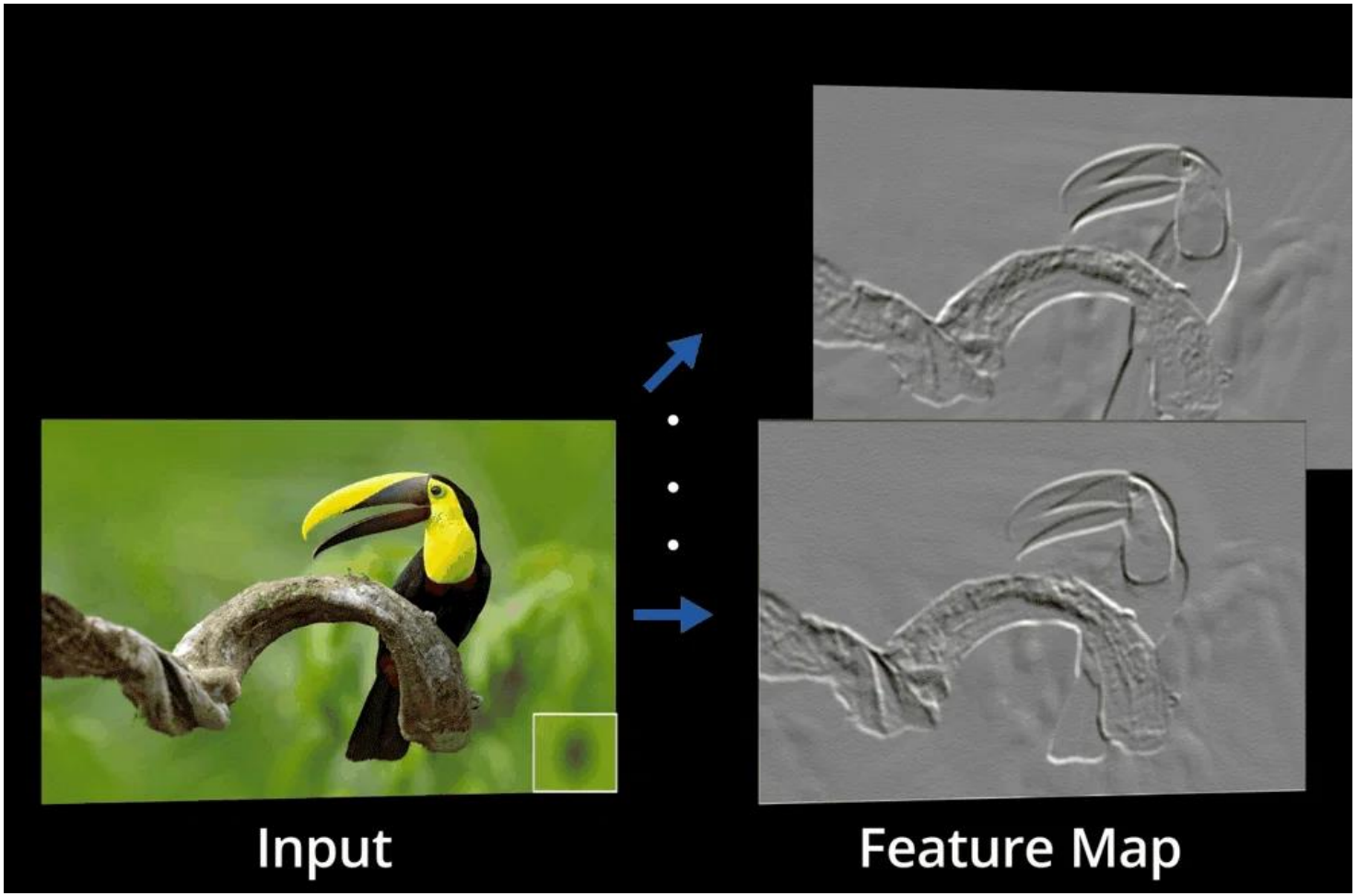


Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
10 x 32 x 32

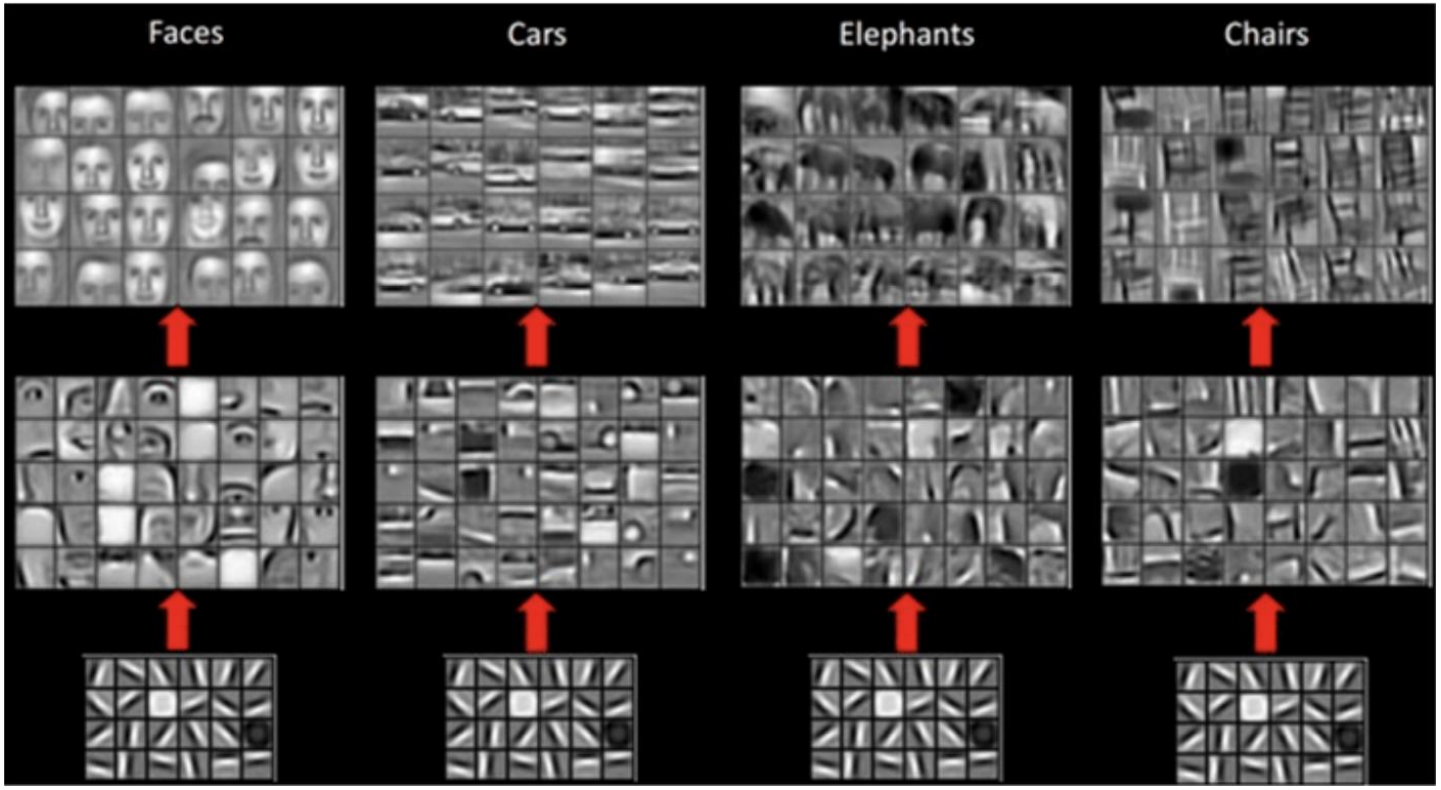
In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: (W – K + 2P) / S + 1

Convolution Layer



Example of feature maps

- Examples of CNN's trained to recognize specific objects and their generated feature maps.



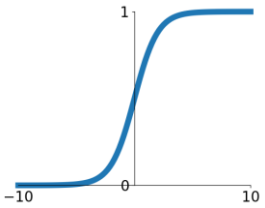
Activation Functions

Activation Functions

ReLU is a good default choice for most problems

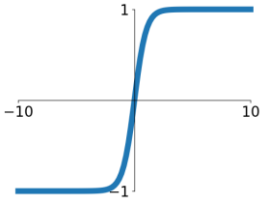
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



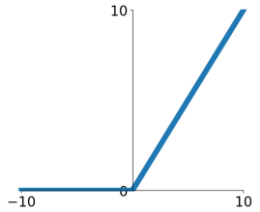
tanh

$$\tanh(x)$$



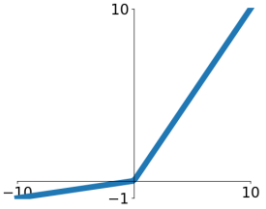
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

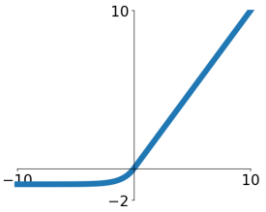


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Vanishing Gradient Problem

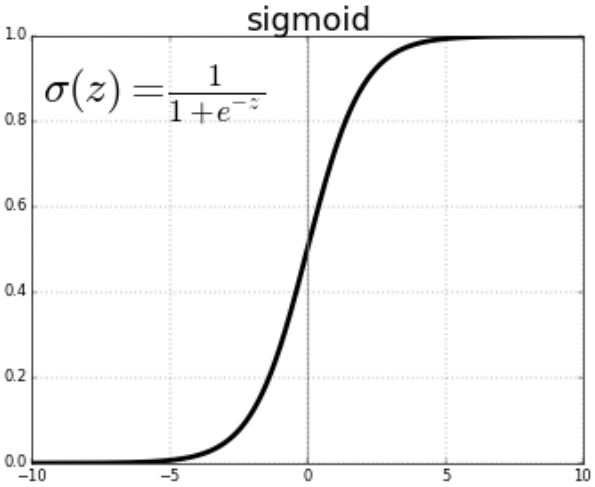
- In gradient-based learning algorithms, we use gradients to learn the weights of a neural network.
- It works like a chain reaction as the gradients closer to the output layers are multiplied with the gradients of the layers closer to the input layers.
- If the gradients are small, the multiplication of these gradients will become so small that it will be close to zero.
- This results in the model being unable to learn, and its behavior becomes unstable.
- This problem is called the vanishing gradient problem.

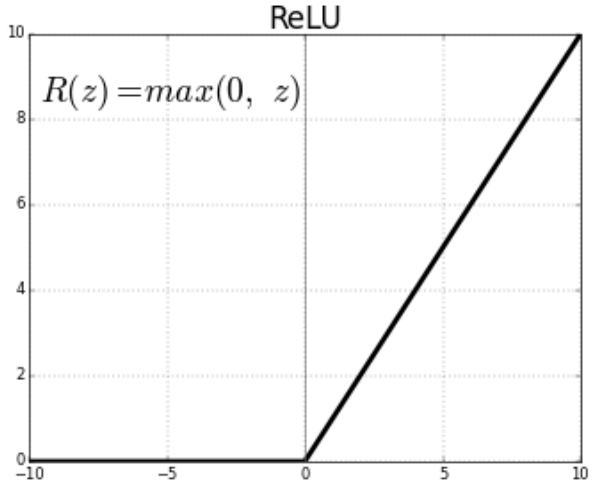
Vanishing Gradient Problem

- The effect of the small gradient value is smaller and smaller changes in weights, the neural network stops training at all.
- The vanishing gradient is a very common problem caused by the use of the sigmoid function.
- Some common ways to counter the vanishing gradient:
  - **Use a different activation function**
  - Use careful weight initialization
  - Use residual blocks: ResNets

ReLU Layer (Rectified Linear Unit)

- Y = Max(0,x)
- ReLU is computed after convolution
- Makes all negative values to zero
- RELU overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

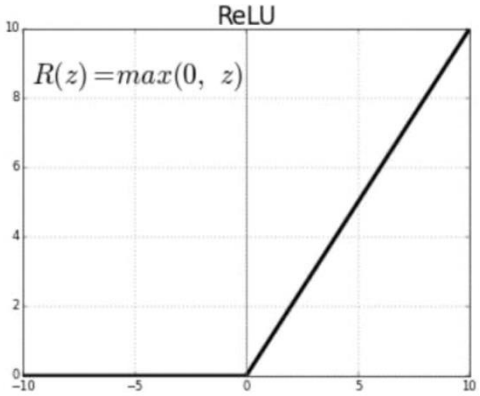


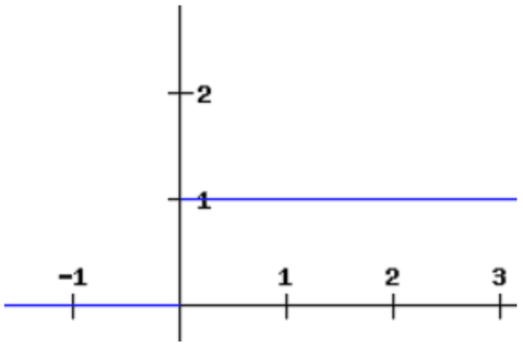


ReLU Layer (Rectified Linear Unit)

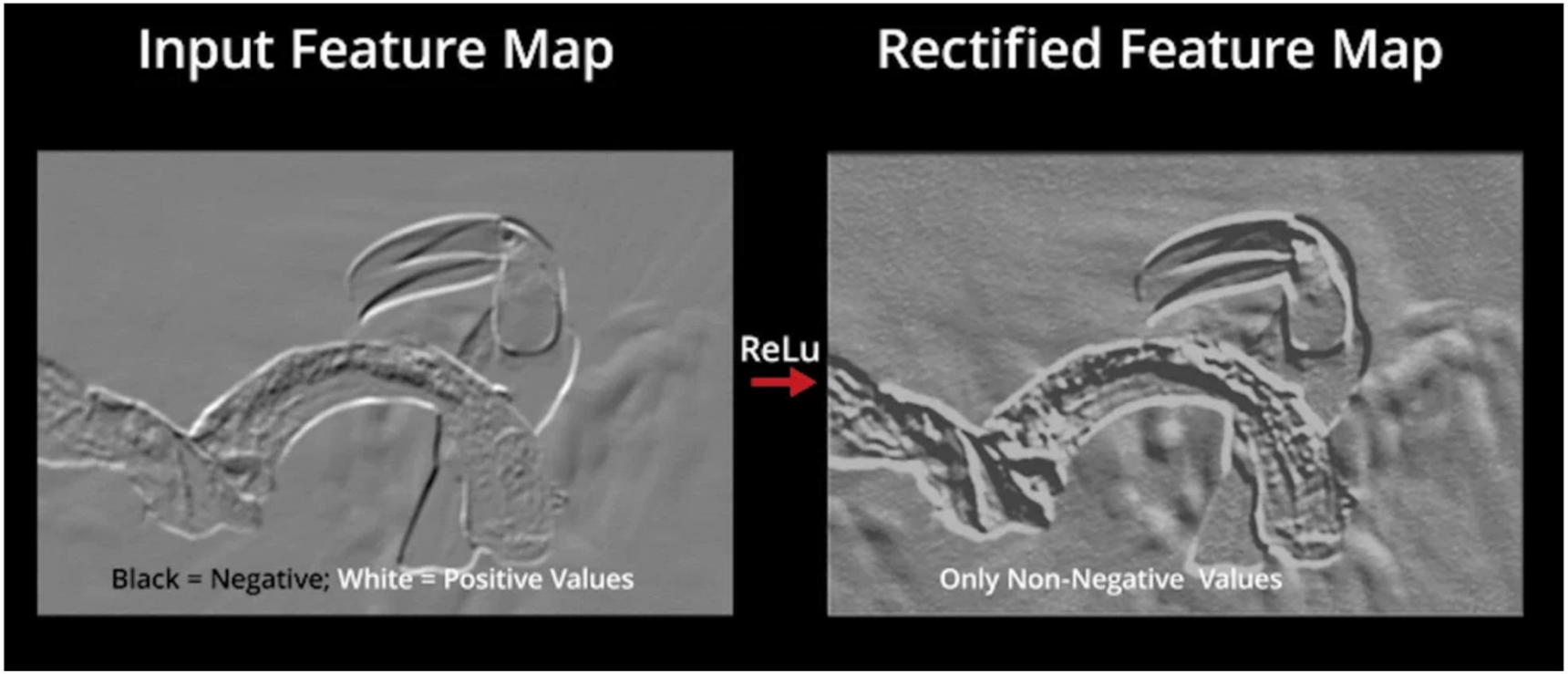
- The value of the partial derivative of the loss function will be having values of 0 or 1 which prevents the gradient from vanishing.

- If the gradient is 0, the old and new weight values are the same, so the node is considered a dead node.

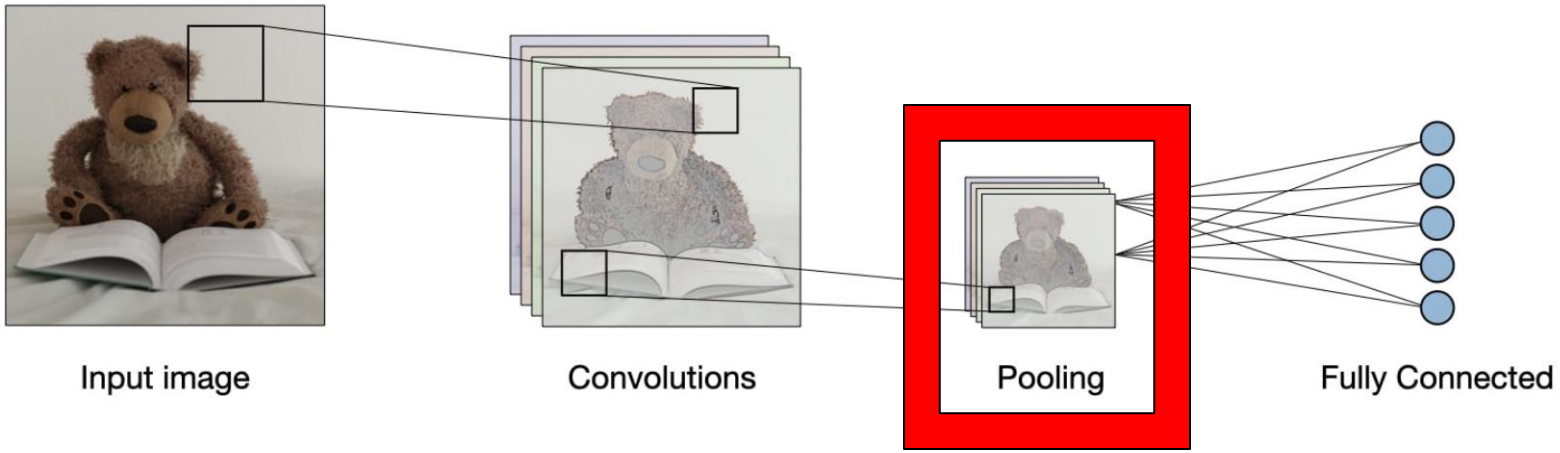




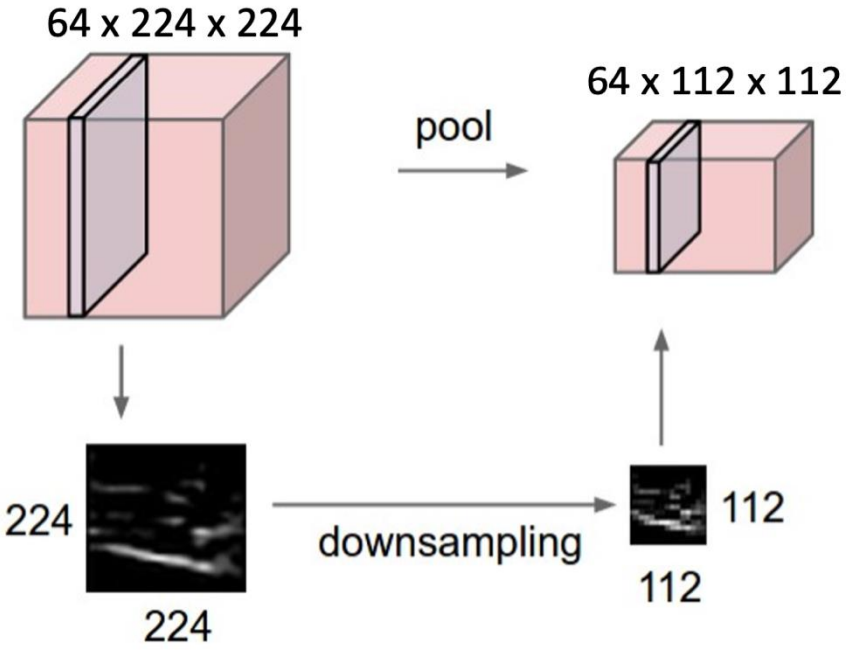
ReLU Layer


Input Feature Map
Rectified Feature Map
ReLu
Black = Negative; White = Positive Values
Only Non-Negative Values

CNN – Pooling Layer



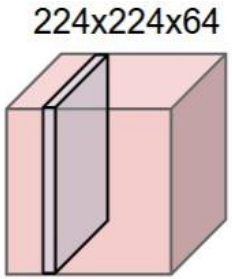
CNN – Pooling Layer - Downsampling



Hyperparameters:
Kernel Size
Stride
Pooling function

Pooling – Downsampling layer

Max Pooling



Single depth slice

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

x

y

Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Introduces **invariance** to small spatial shifts

No learnable parameters!

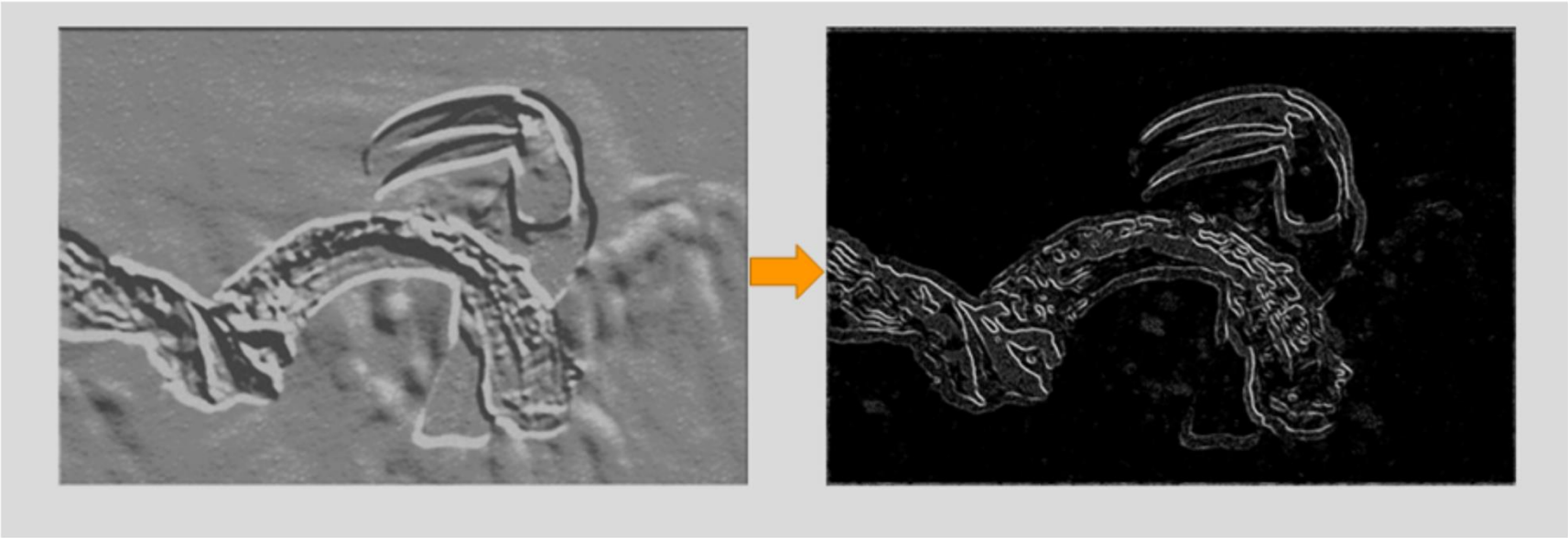
Pooling – Downsampling layer

- Help to reduce complexity, improve efficiency, and limit risk of overfitting.

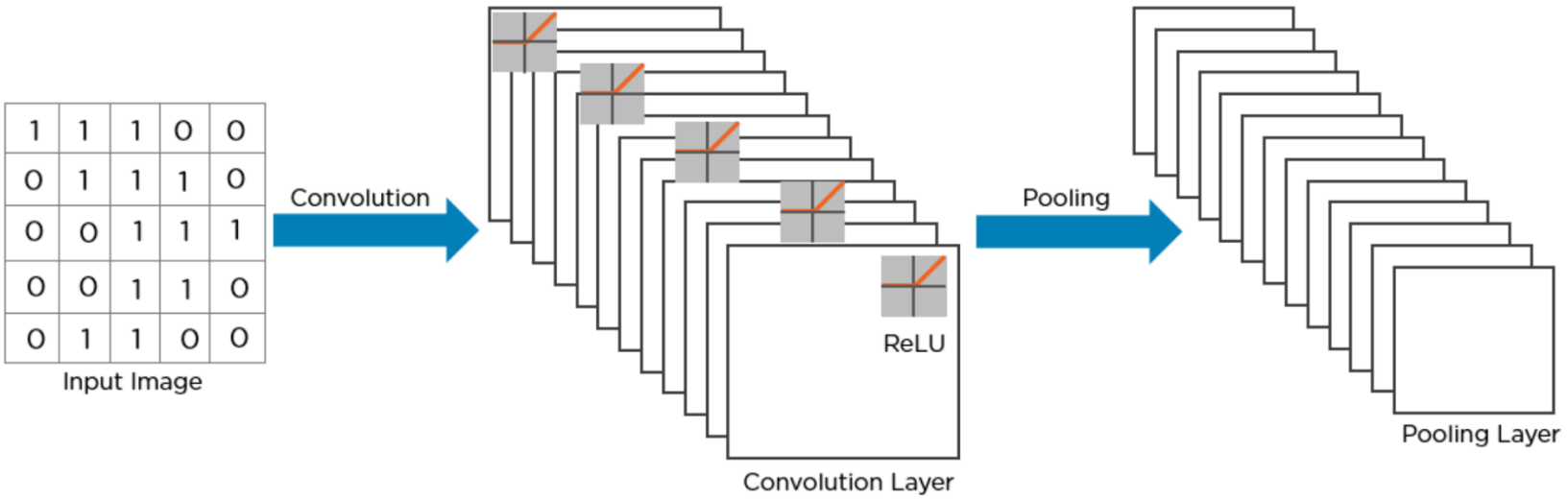
- Two main types of pooling:
  - Max pooling: Each pooling operation selects the maximum value of the current view.
  - Average pooling: Each pooling operation averages the values of the current view.

Pooling Layer

The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes etc.



CNN Architecture – Part I



Residual Networks

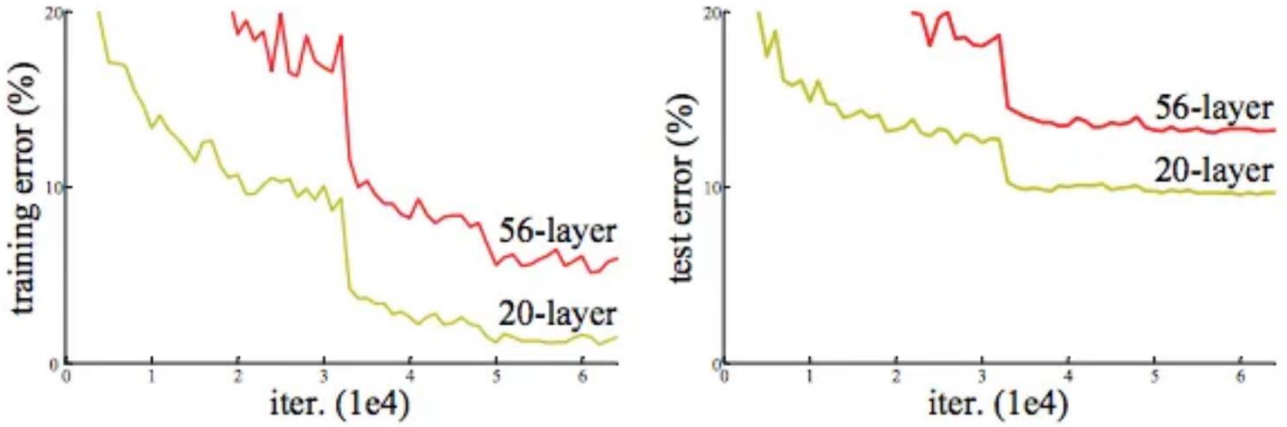
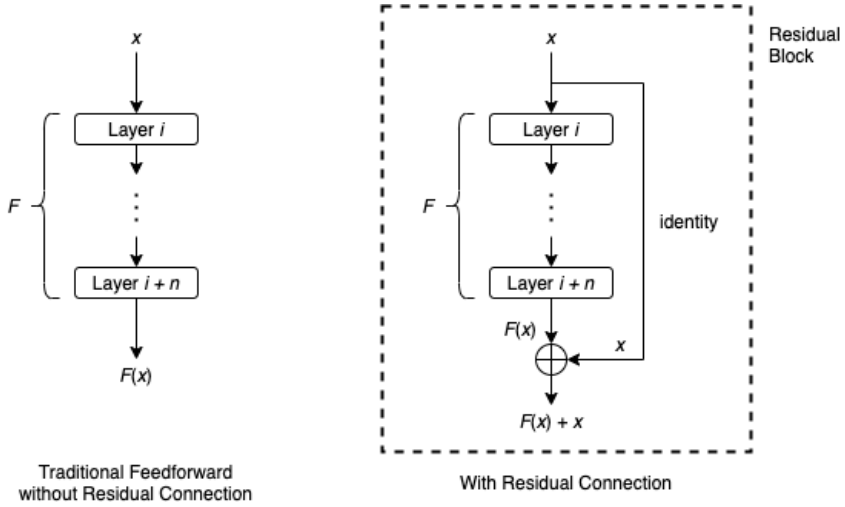


Figure 4. The graph shows that the error rate in a 20-layer is lesser than that of 56-layered network, and theoretically, it should be the opposite. (Image Source: (Original Citation)Deep Residual Learning for Image Recognition)

Residual Networks

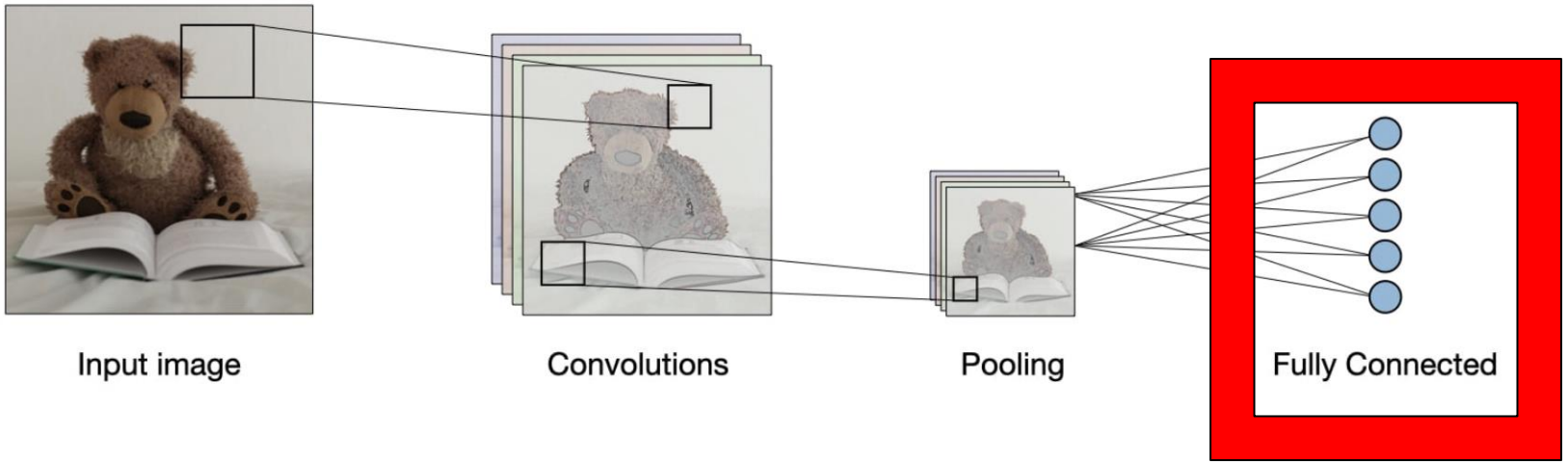
Skip the training of few layers using skip connections or residual connections



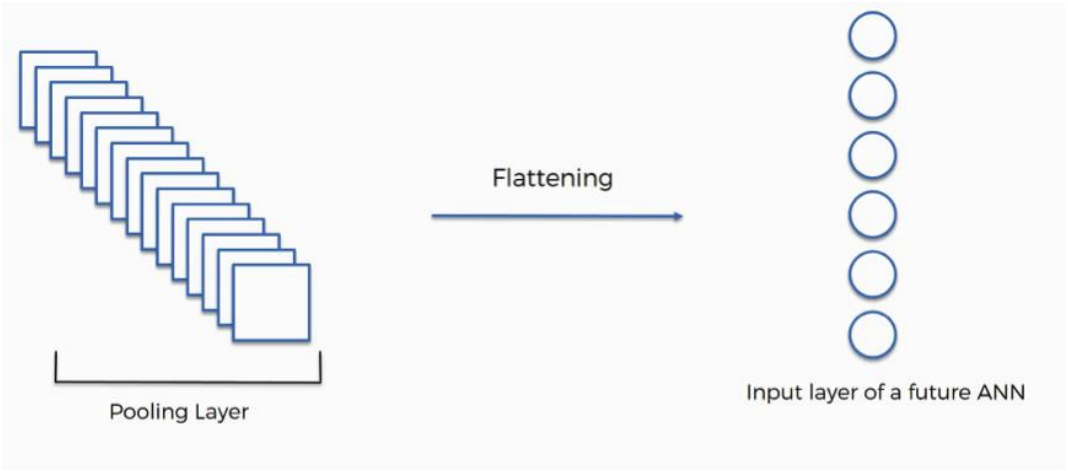
Residual connection does not resolve the exploding or vanishing gradient problems.

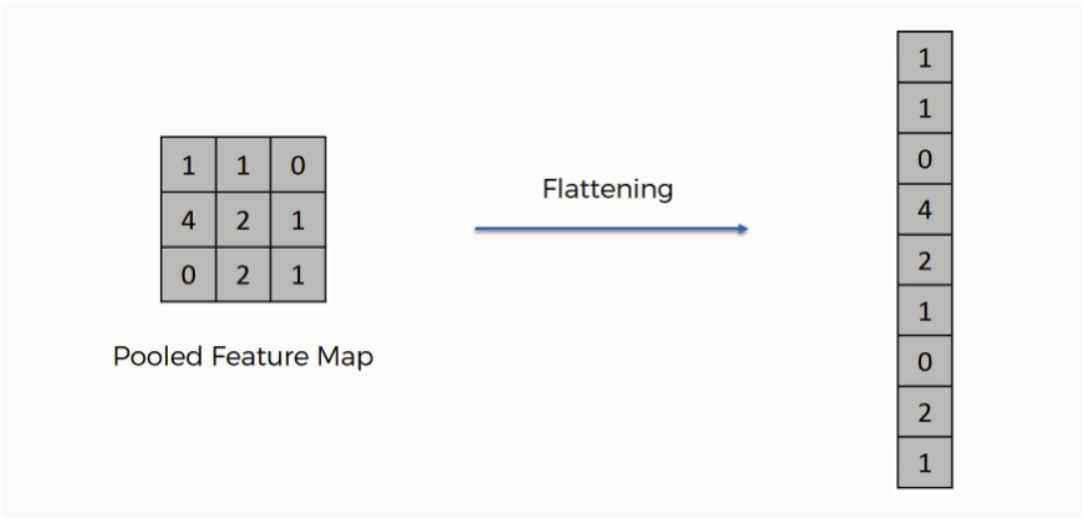
Rather, it avoids those problems by having shallow networks in the “ensembles”.

CNN – Fully Connected Layer

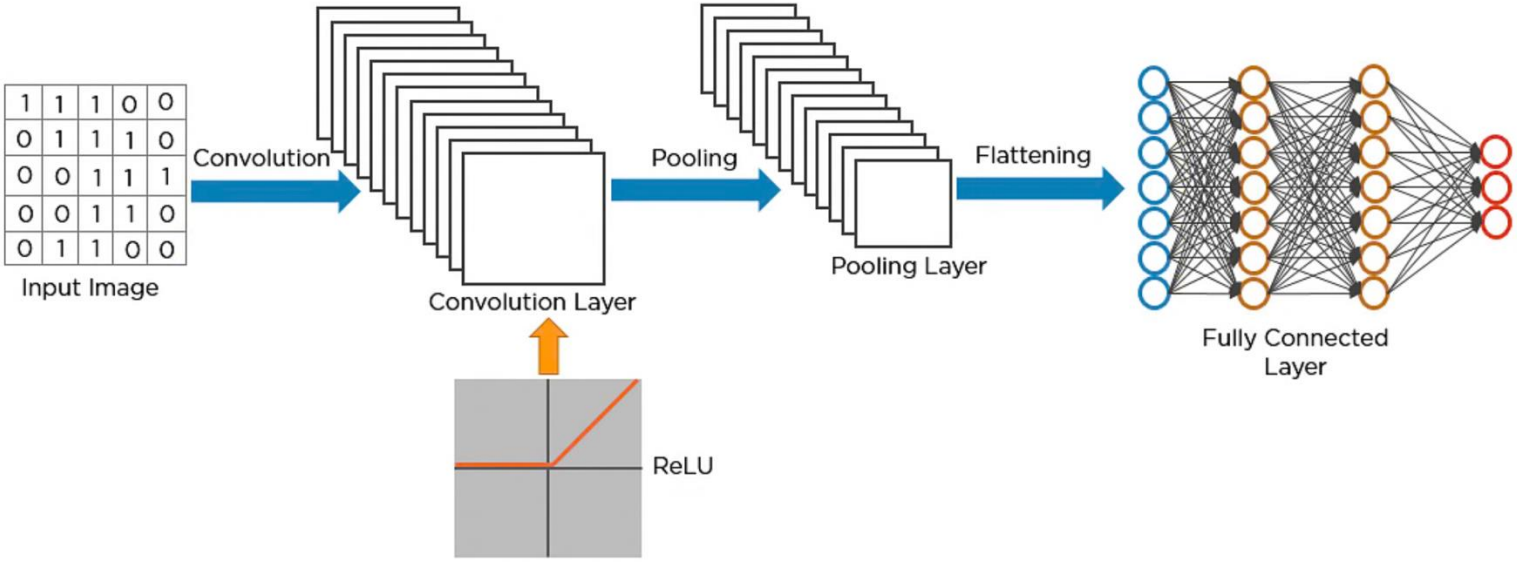


Flatten Layer

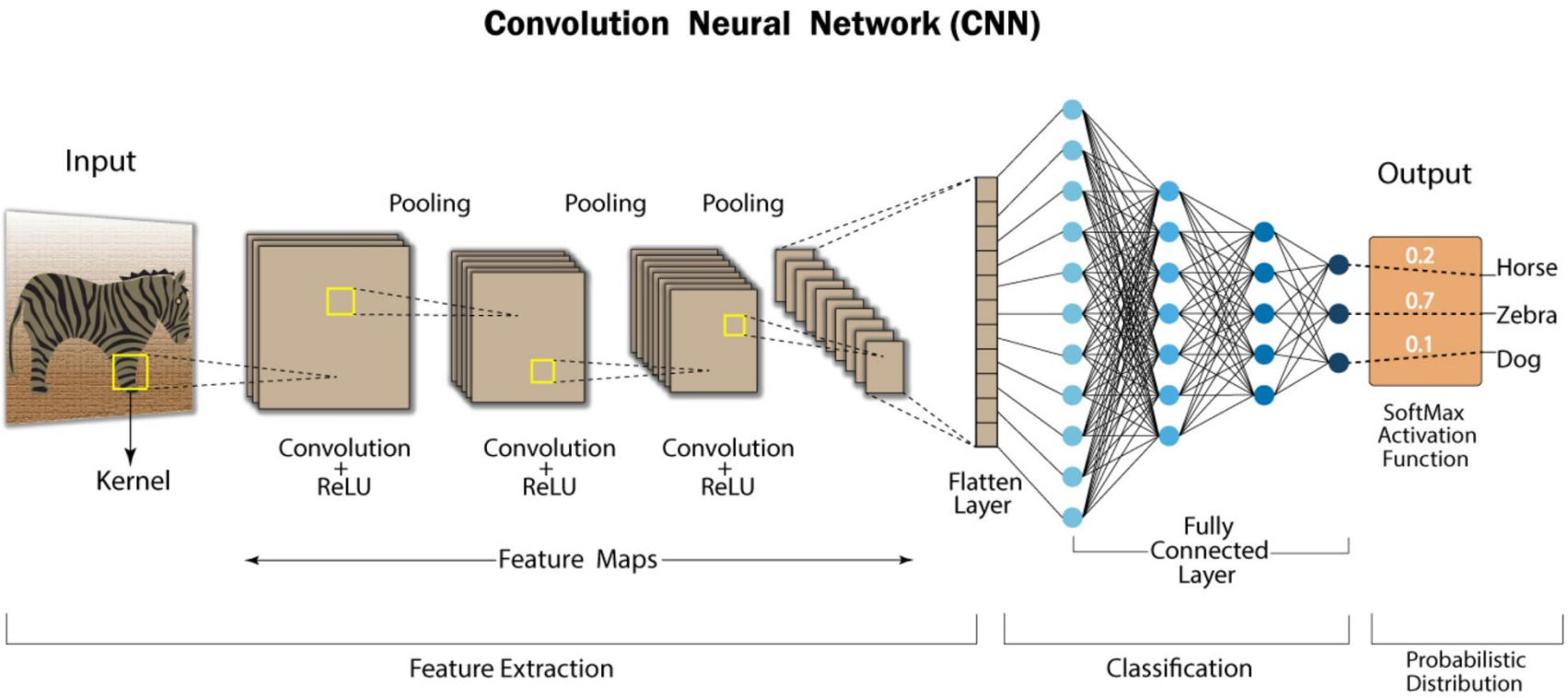




CNN Architecture – Part II



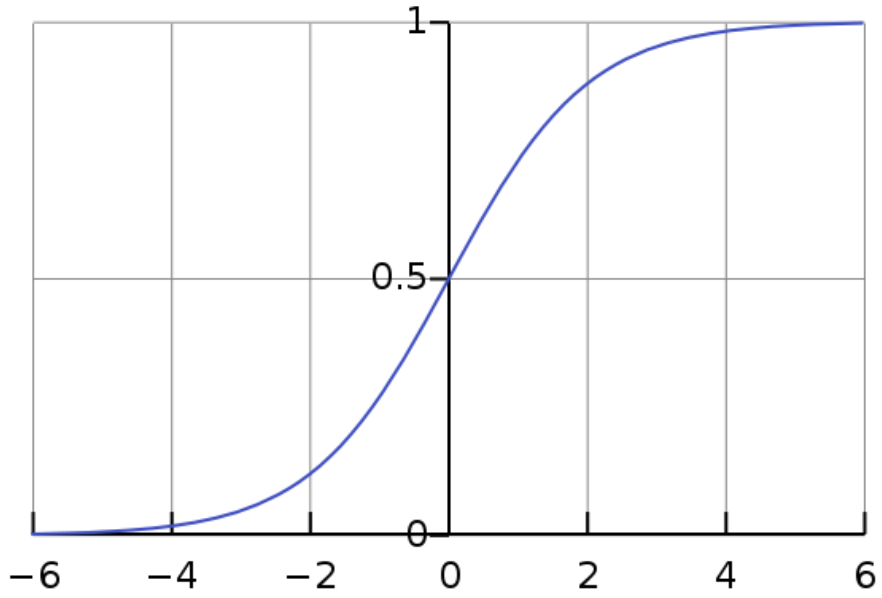
CNN Architecture



https://poloclub.github.io/cnn-explainer/

Sigmoid Activation Function

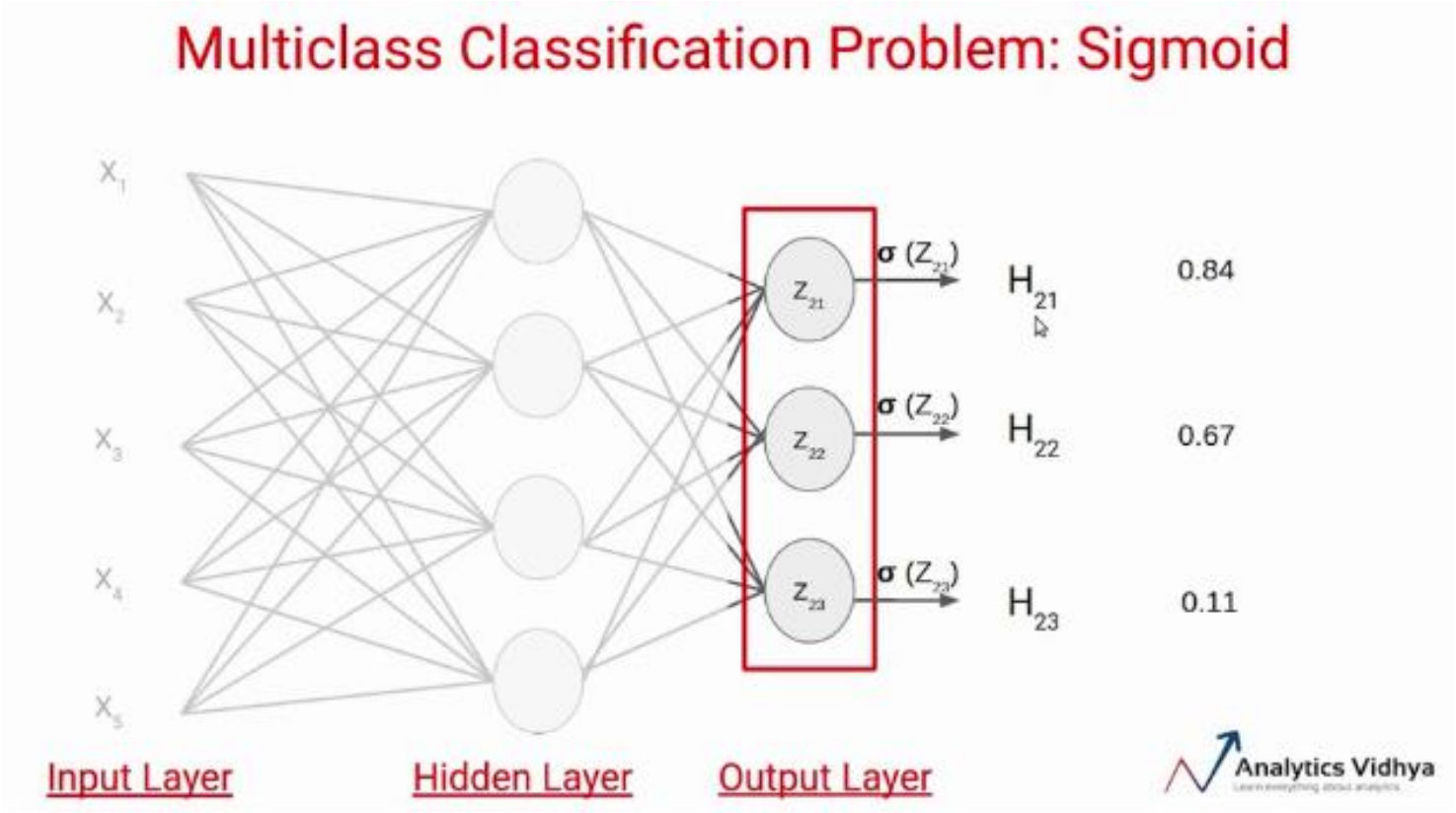
- It squashes a vector in the range (0, 1).
- It's also called **logistic function**.



$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

Sigmoid Activation Function

- The sigmoid activation function gives the value between 0 and 1.



Sigmoid vs SoftMax

Sigmoid

2 classes

$$out = P(Y=class1|X)$$

SoftMax

k>2 classes

$$out = \begin{bmatrix} P(Y=class1|X) \\ P(Y=class2|X) \\ P(Y=class3|X) \\ \vdots \\ P(Y=classk|X) \end{bmatrix}$$

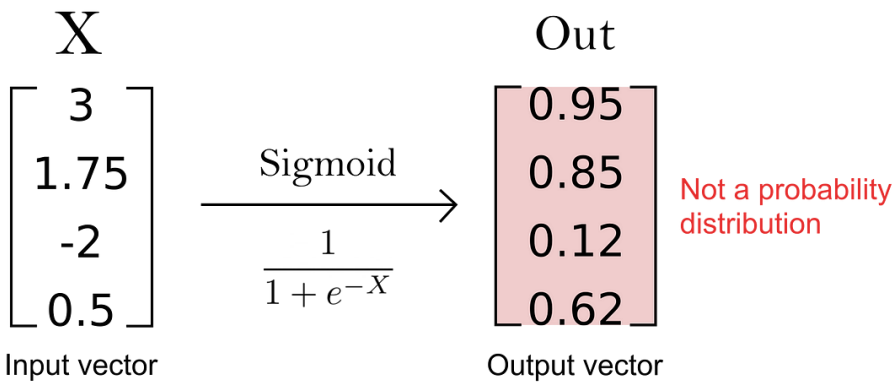
Softmax Activation Function

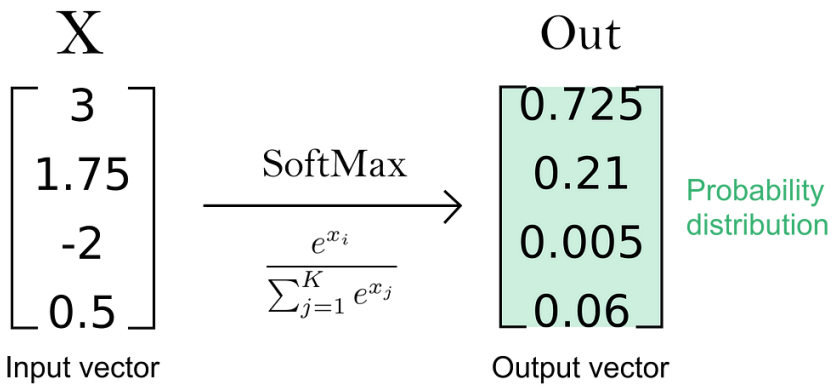
$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e$-24 (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Sigmoid vs SoftMax

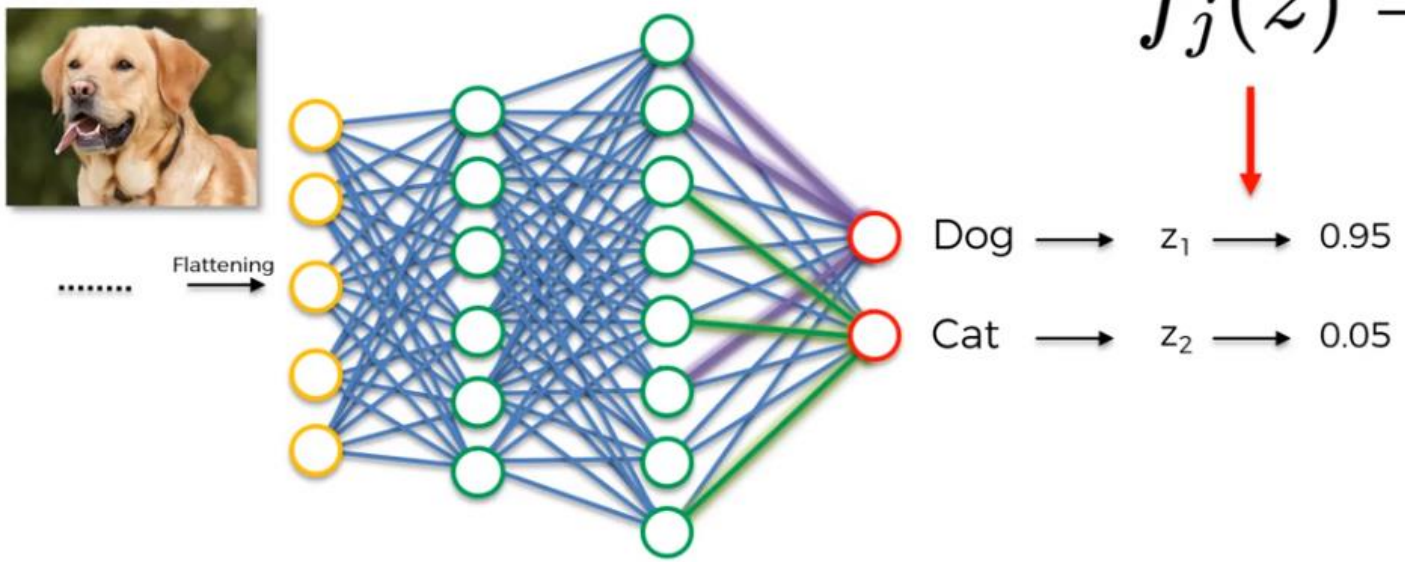




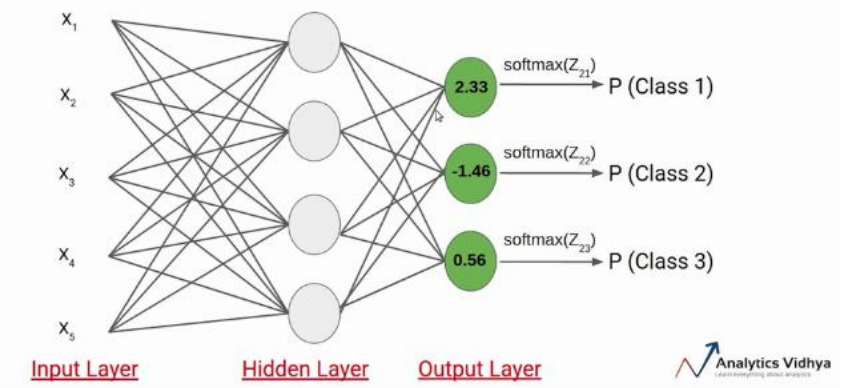
Softmax Activation Function

- **Softmax function is usually applied to the output layer only**
- **Assign decimal probabilities to each class**
- **Must add-up to 1**
- **softmax helps us quantify how sure we are of our prediction**

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



Softmax Activation Function



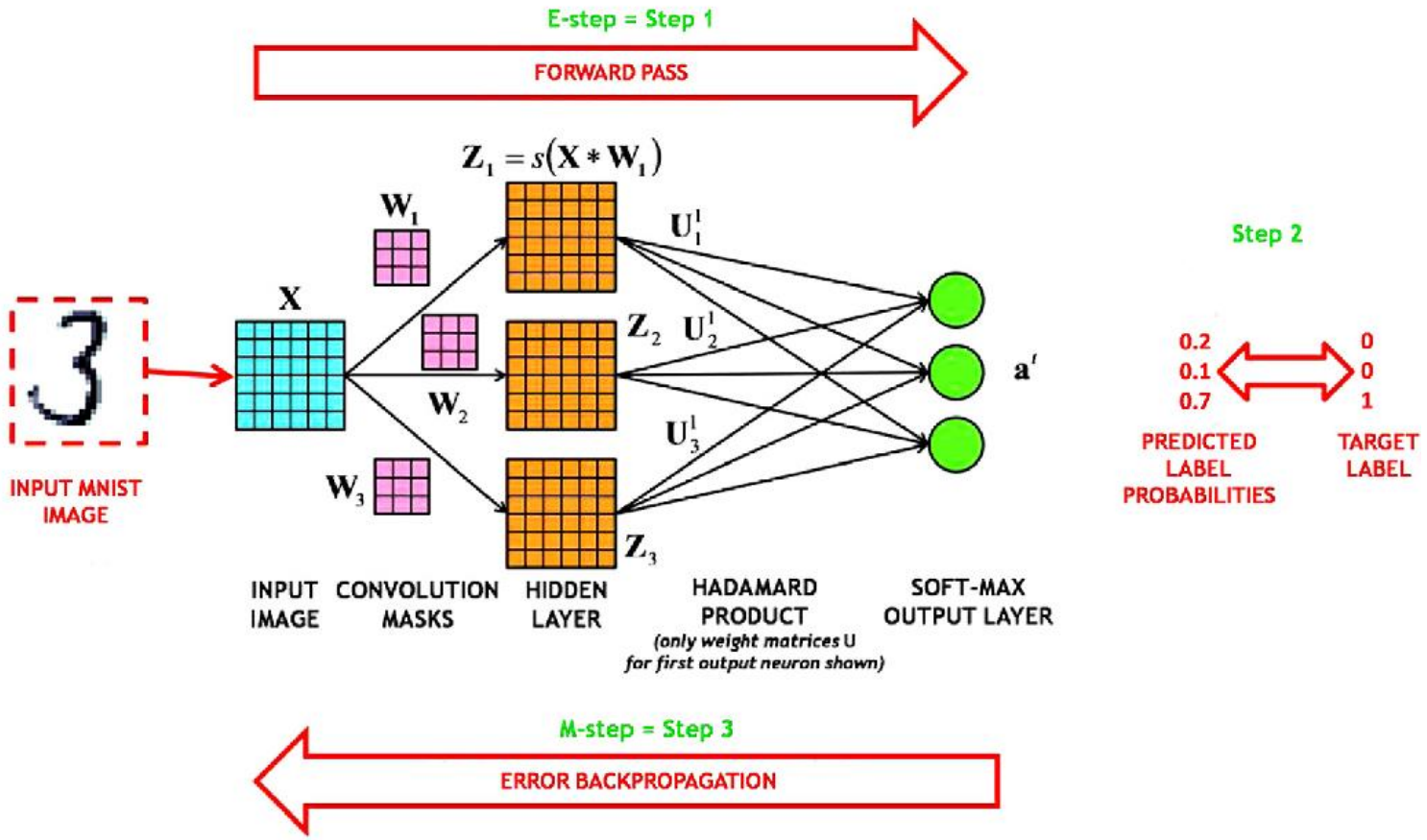
Example :

$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

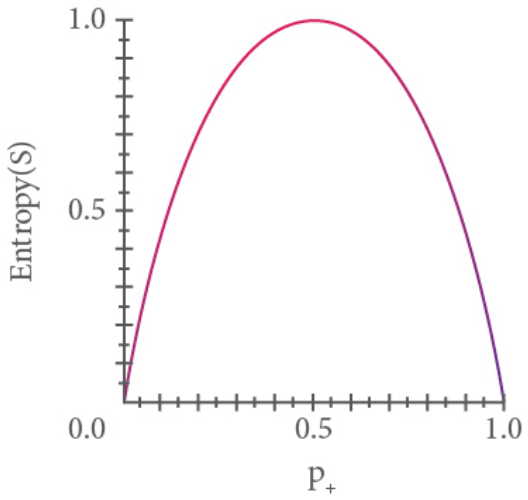
$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

Forward and Back Propagation



Entropy



$$\text{Entropy} = \sum_i -p_i \log_2 p_i$$

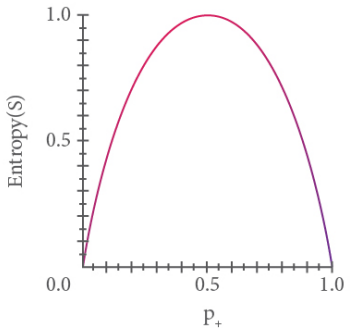
- **Entropy measures the impurity of S**

For binary classification:

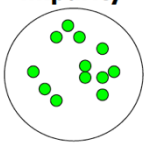
- **S is a sample of training examples**
- **p_+ is the proportion of positive examples**
- **p_- is the proportion of negative examples**
- **Entropy is expected number of bits needed to encode class + or class -**

Entropy

Entropy = $\sum_i - p_i \log_2 p_i$



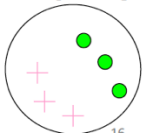
Minimum impurity



- **If all samples are in one class then Entropy = 0**

  entropy = - 1 $\log_2 1 = 0$

Maximum impurity

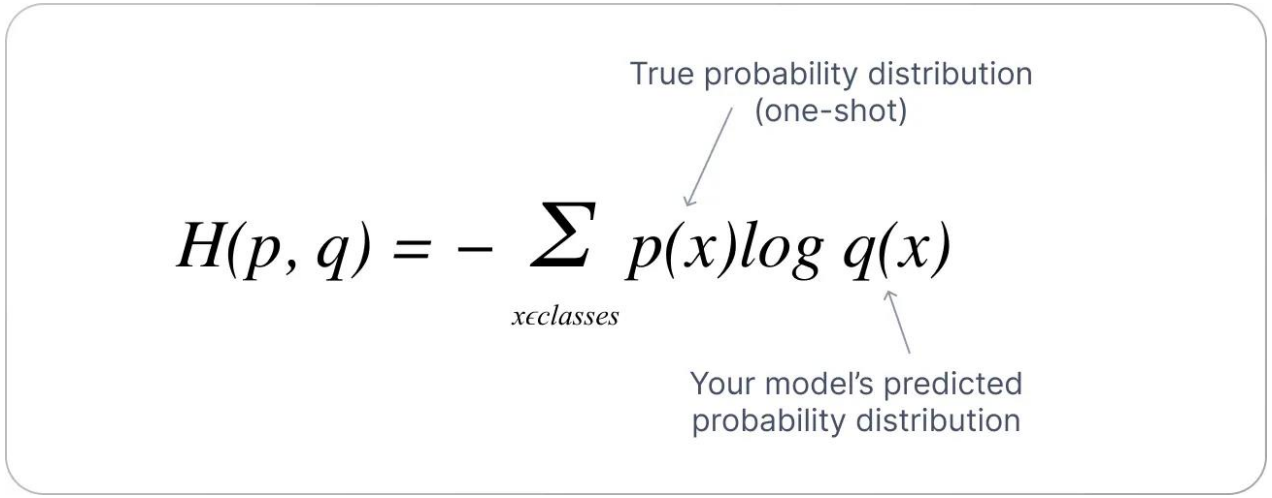


- If 50/50 positive and negative then Entropy = 1

  entropy = -0.5 $\log_2 0.5 - 0.5 \log_2 0.5 = 1$

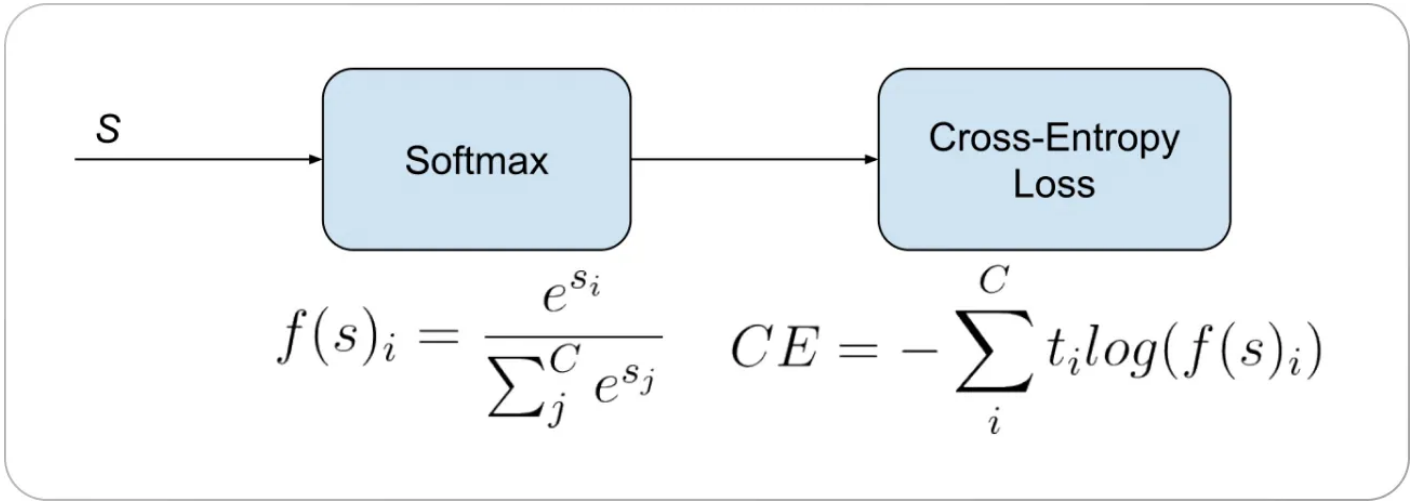
Cross Entropy

- the Cross Entropy between two discrete probability distributions is a metric that captures how similar the two distributions are.
- **Cross Entropy** between real output and predicted output



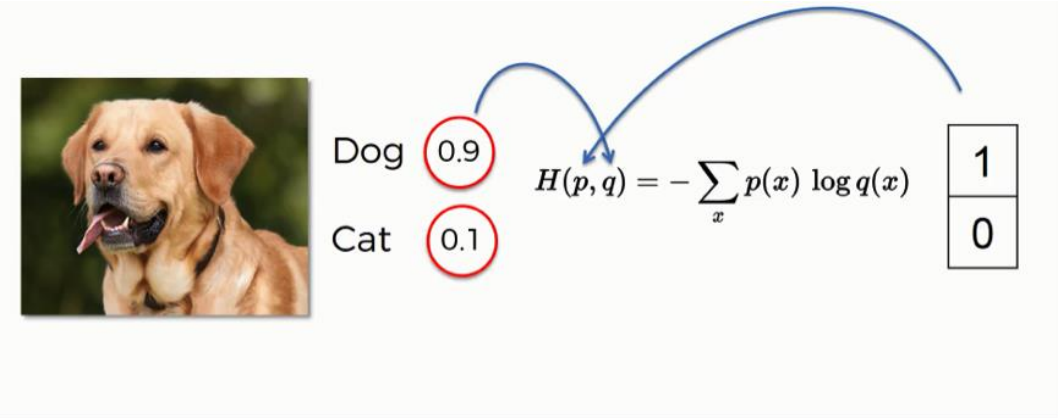
Cross Entropy Loss Function

- Cross-entropy loss is used when adjusting model weights during training
- It takes the output probabilities (P) and measure the distance from the truth values
- A perfect model has a cross-entropy loss of 0.



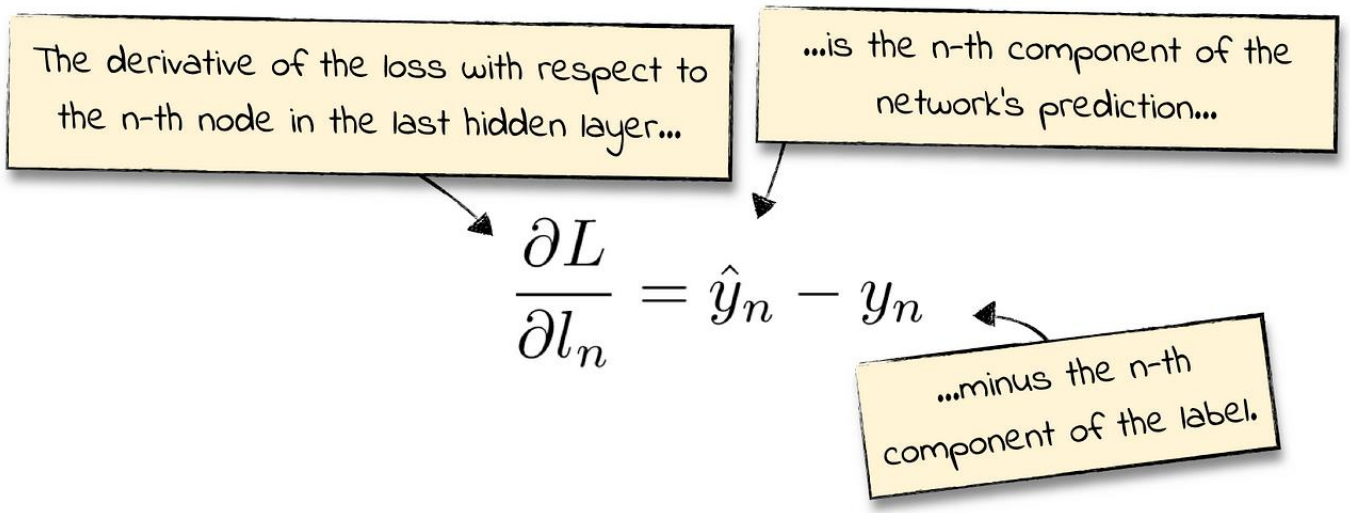
Cross Entropy Loss Function

- Cross-entropy loss is used when adjusting model weights during training
- It takes the output probabilities (P) and measure the distance from the truth values
- A perfect model has a cross-entropy loss of 0.



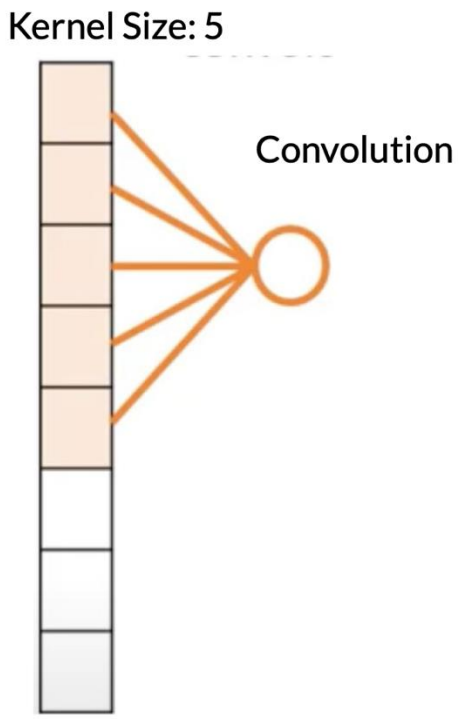
Softmax and Cross Entropy

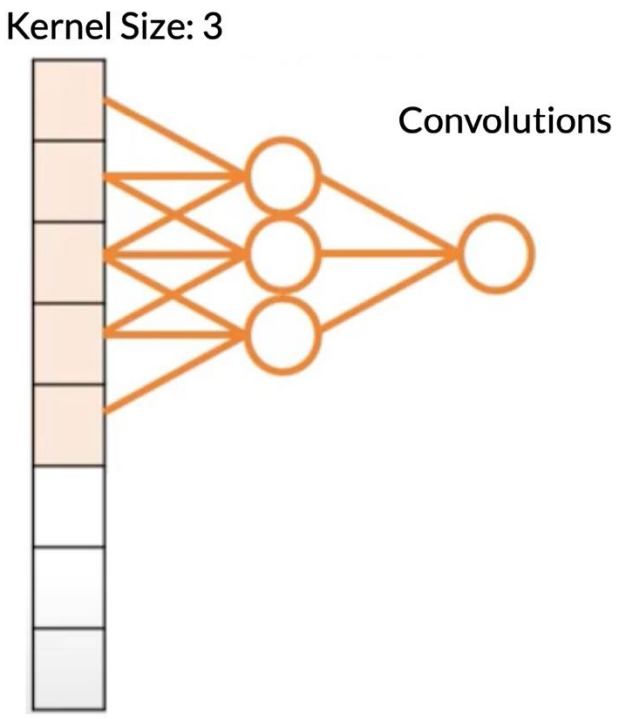
- to train the network with backpropagation, we calculate the derivative of the loss :



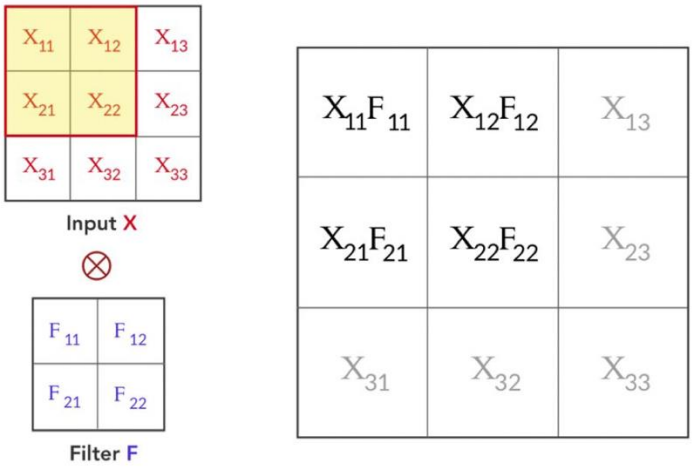
https://levelup.gitconnected.com/killer-combo-softmax-and-cross-entropy-5907442f60ba

BackPropagation in Convolution Layer

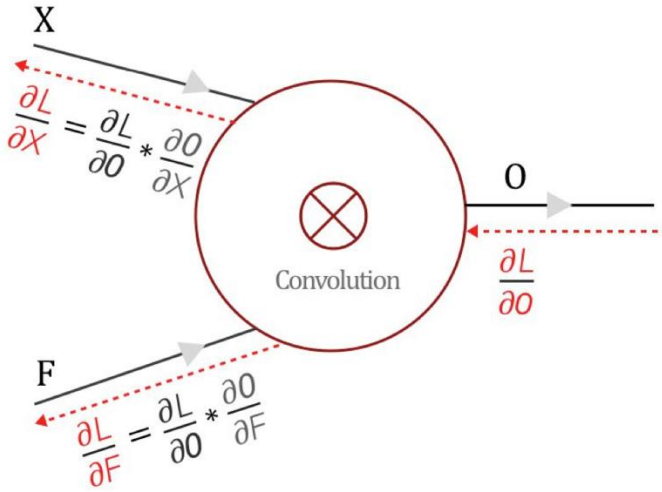




BackPropagation in Convolution Layer



$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$



$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

BackPropagation in Convolution Layer

Finding the gradients:

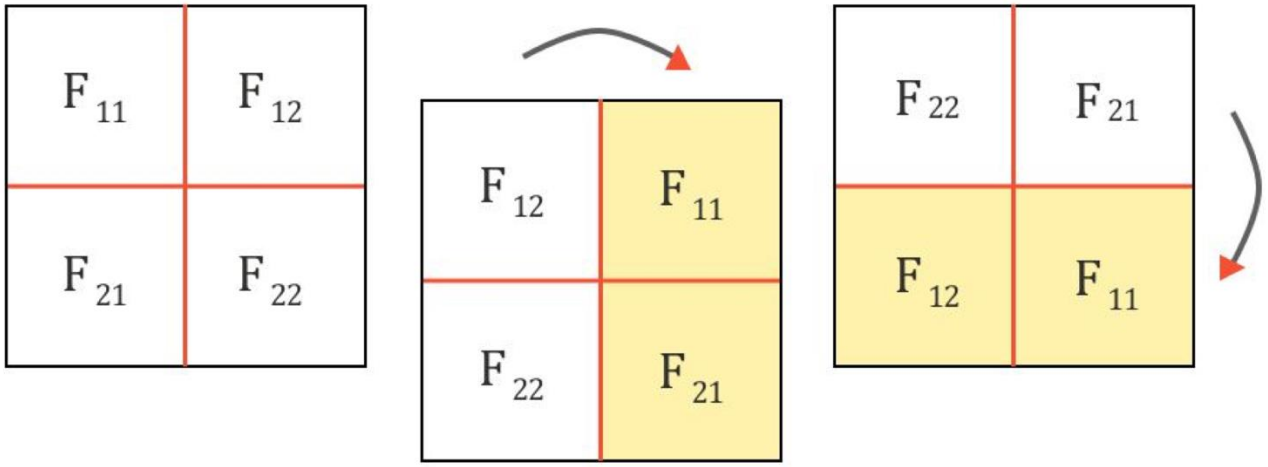
$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X\,,\ \text{Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^{\circ}\text{ rotated Filter } F\,,\ \text{Loss Gradient } \frac{\partial L}{\partial O}\right)$$

https://deeplearning.cs.cmu.edu/S23/document/recitation/Recitation5/S23_IDL_%20Recitation%205.pdf

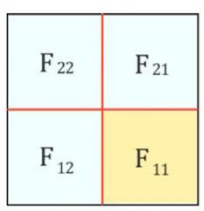
BackPropagation in Convolution Layer

Step 1: Rotate the Filter **F** by 180 degrees - flipping it first vertically and then horizontally

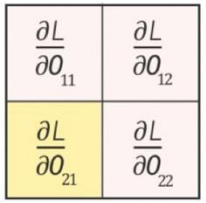


BackPropagation in Convolution Layer

Step 2: Full convolution between flipped filter F and $\partial L/\partial O$

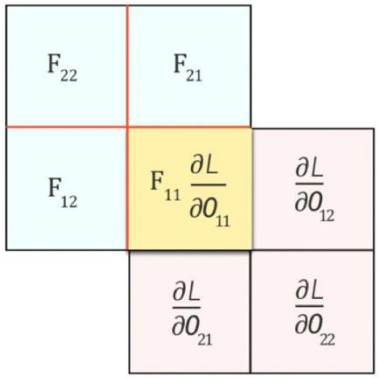


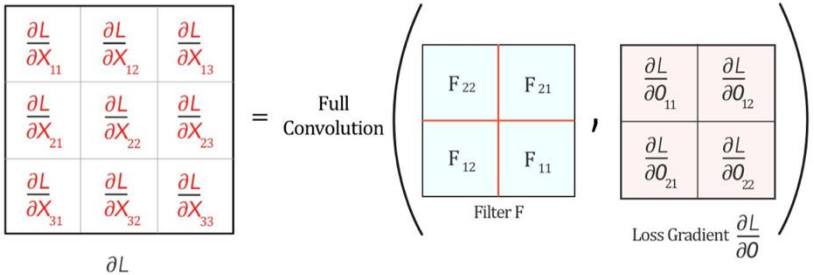
Filter F



Loss Gradient $\frac{\partial L}{\partial O}$

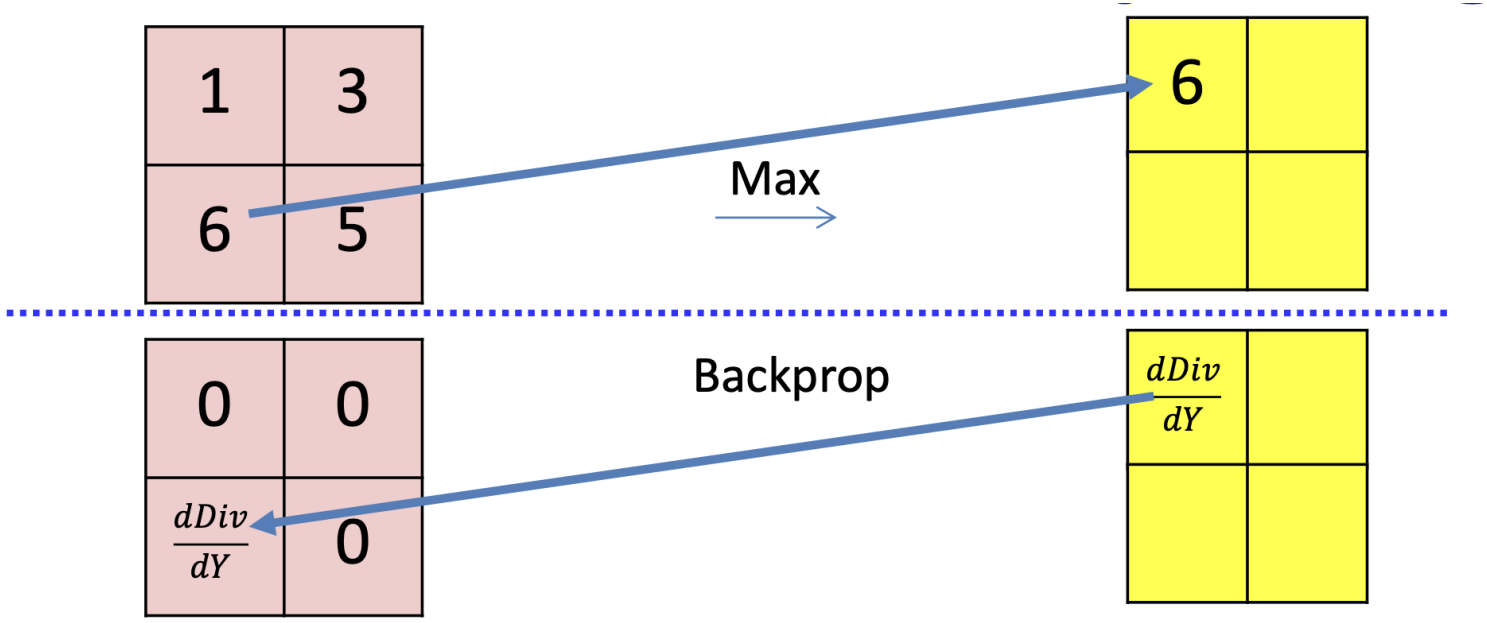
$$\frac{\partial L}{\partial X_{11}} = F_{11} * \frac{\partial L}{\partial O_{11}}$$





$\frac{\partial L}{\partial X}$

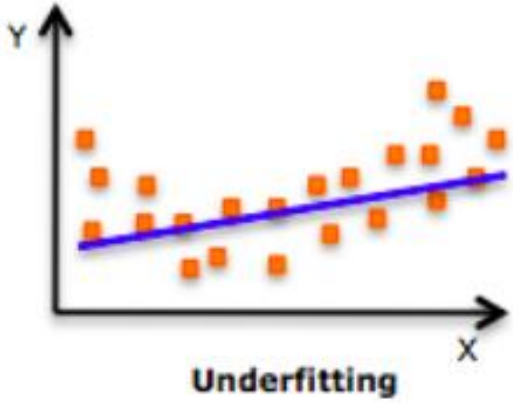
Derivative of Max Pooling

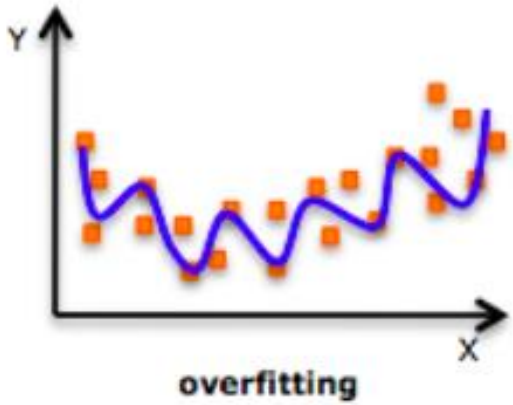


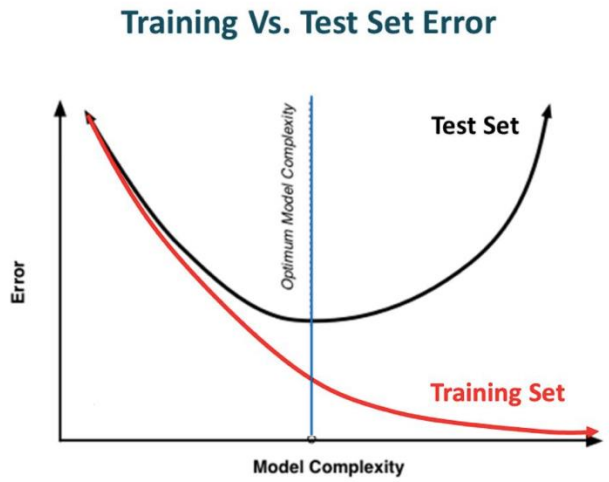
Batch Size

- Neural networks are trained using the stochastic gradient descent optimization algorithm.
- The number of training examples used in the estimate of the error gradient is the **batch size.**
  - **Stochastic Gradient Descent:** Batch size is set to one.
  - **Batch Gradient Descent:** Batch size is set to the total number of examples in the training dataset.
  - **Minibatch Gradient Descent:** Batch size is set to more than one and less than the total number of examples in the training dataset.

Overfitting

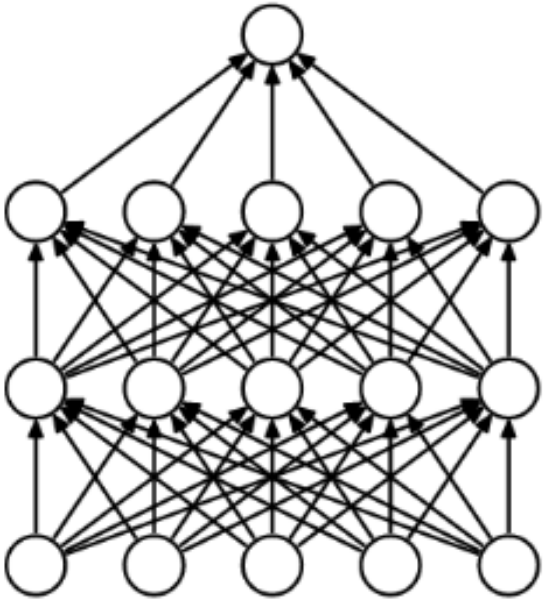




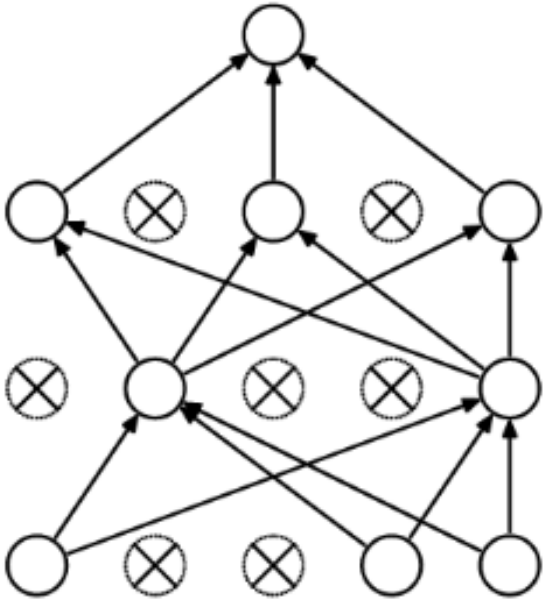


Regularization with Dropout

- The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network.



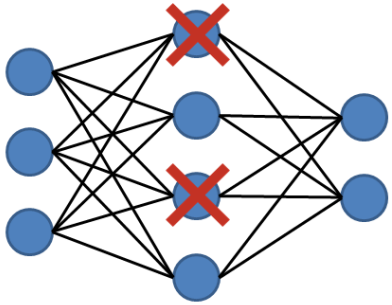
(a) Standard Neural Net



(b) After applying dropout.

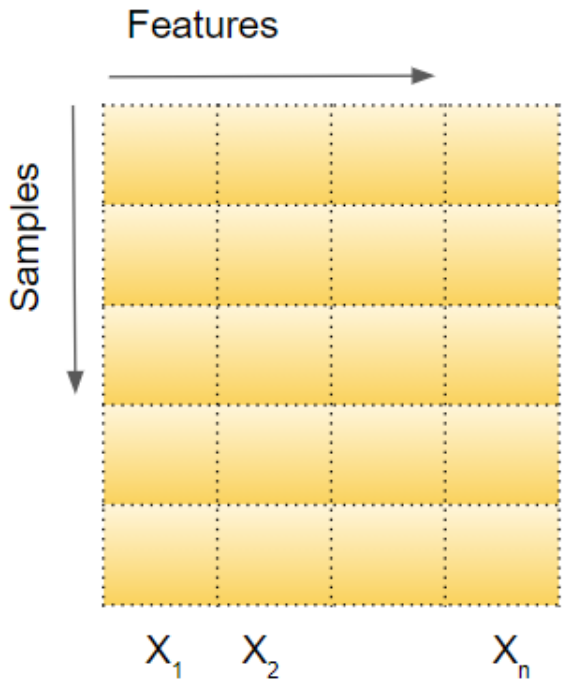
Regularization with Dropout

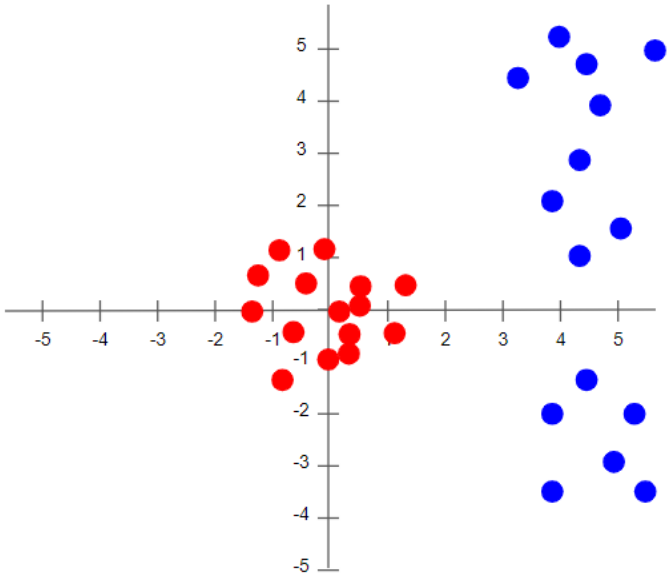
- On each iteration, we randomly shut down some neurons (units) on each layer and don't use those neurons in both forward propagation and back-propagation
- All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network.
- The nodes are dropped by a dropout probability of p.
- Since we drop some units on each iteration, this will lead to smaller network which in turns means simpler network (regularization).
- Force the learning algorithm to spread out the weights and not focus on some specific feattures (units)



Normalization of the Input Data

When inputting data to a deep learning model, it is normalized to:
zero mean and unit variance

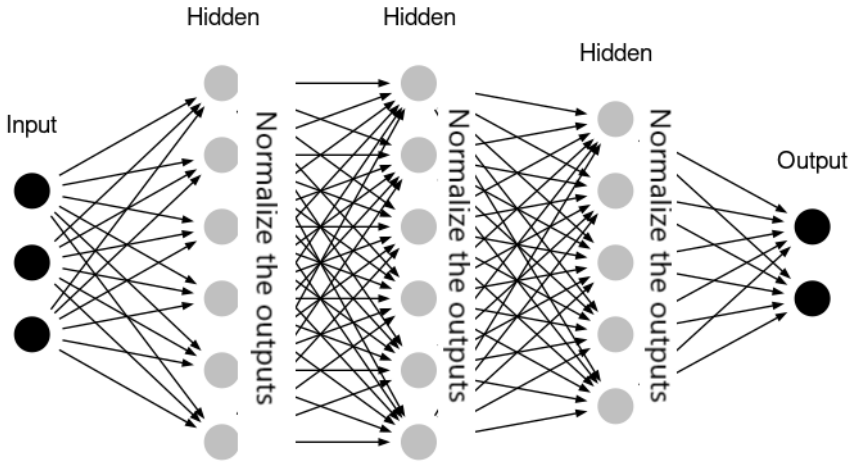




$$X_i = \frac{X_i - Mean_i}{StdDev_i}$$

Batch Normalization

- if we normalize the activations from each previous layer then the gradient descent will converge better during training.
- Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance



Batch Normalization

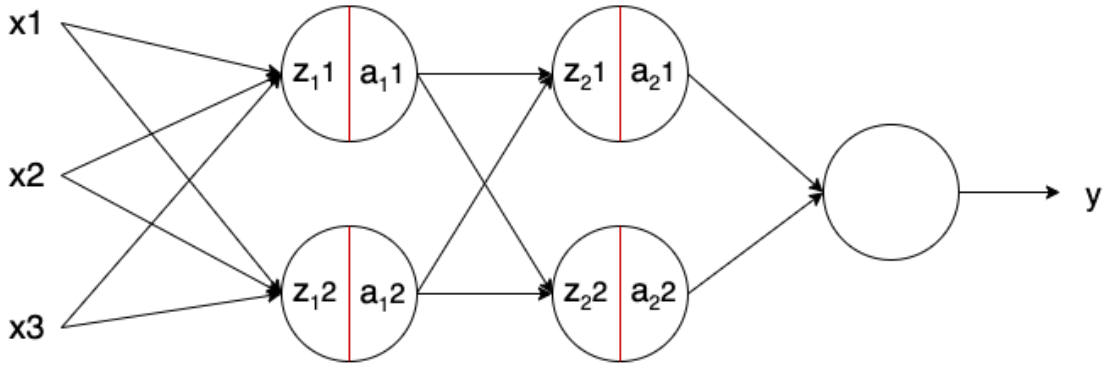
- Whenever we train a network, we provide a batch of input from the complete dataset and these batch keep changing until the epoch is completed.
- the inputs to each layer are affected by the parameters of all preceding layers.
- small changes in the network parameters amplify as the network becomes deeper.

Batch Normalization

- Each mini-batch has a different distribution from different parts of the dataset
- The network adapts continuously to new distribution.
- whenever this input distribution changes, it is called Covariate Shift
- To reduce this problem of internal covariate shift, Batch Normalization adds Normalization Layer between each layers

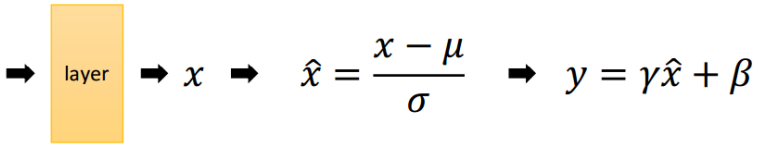
Batch Normalization

- It is done along mini-batches instead of the full data set.
- It is applied to the neurons' output just before applying the activation function
- in the image represented with a red line – is applied to the neurons' output just before applying the activation function.



Batch Normalization

- calculates mean & variance for each minibatch(x1,x2,…,xn)
- updates that mini-batch(x1`,x2`,…,xn`) and passes to next layer and so on.



- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

- Batch normalization helps prevent overfitting and speeds up the training of deep neural networks.
- It normalizes the activations of each layer by subtracting the mean and dividing by the standard deviation.
- Rescaling and offsetting are done using learnable parameters such as gamma and beta.
- It handles internal covariate shifts and smoothens the loss of landscape.

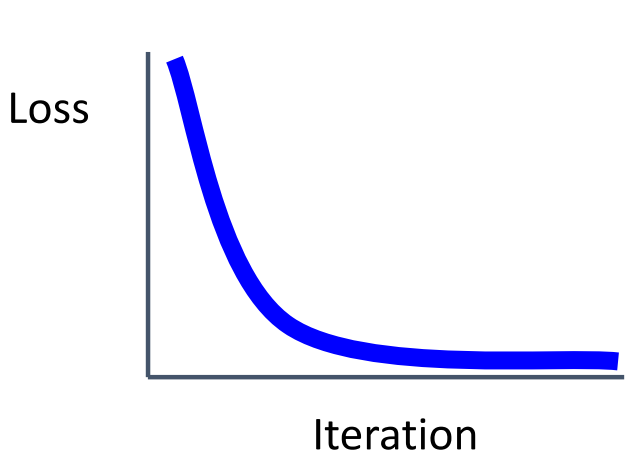
Batch Normalization

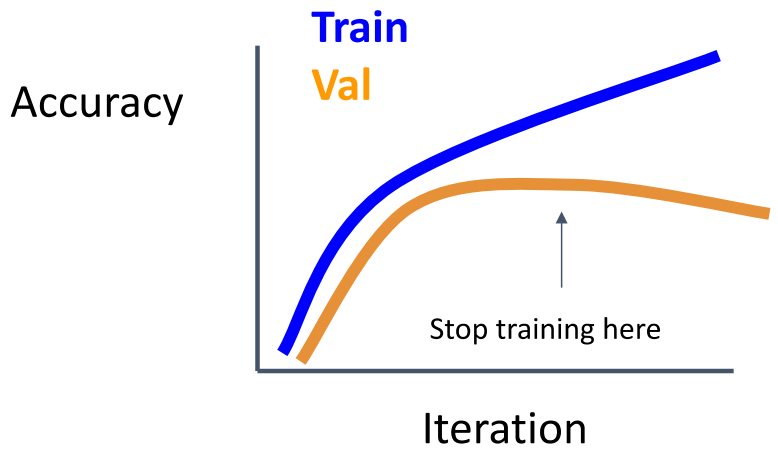
- Makes deep networks much easier to train
- Allows higher learning rates, faster convergence
- Acts as regularization during training
- Networks become more robust to initialization

https://medium.com/hitchhikers-guide-to-deep-learning/9-introduction-to-deep-learning-with-compute-vision-normalization-batch-normalization-ba5f60c77cf3

How long to train?

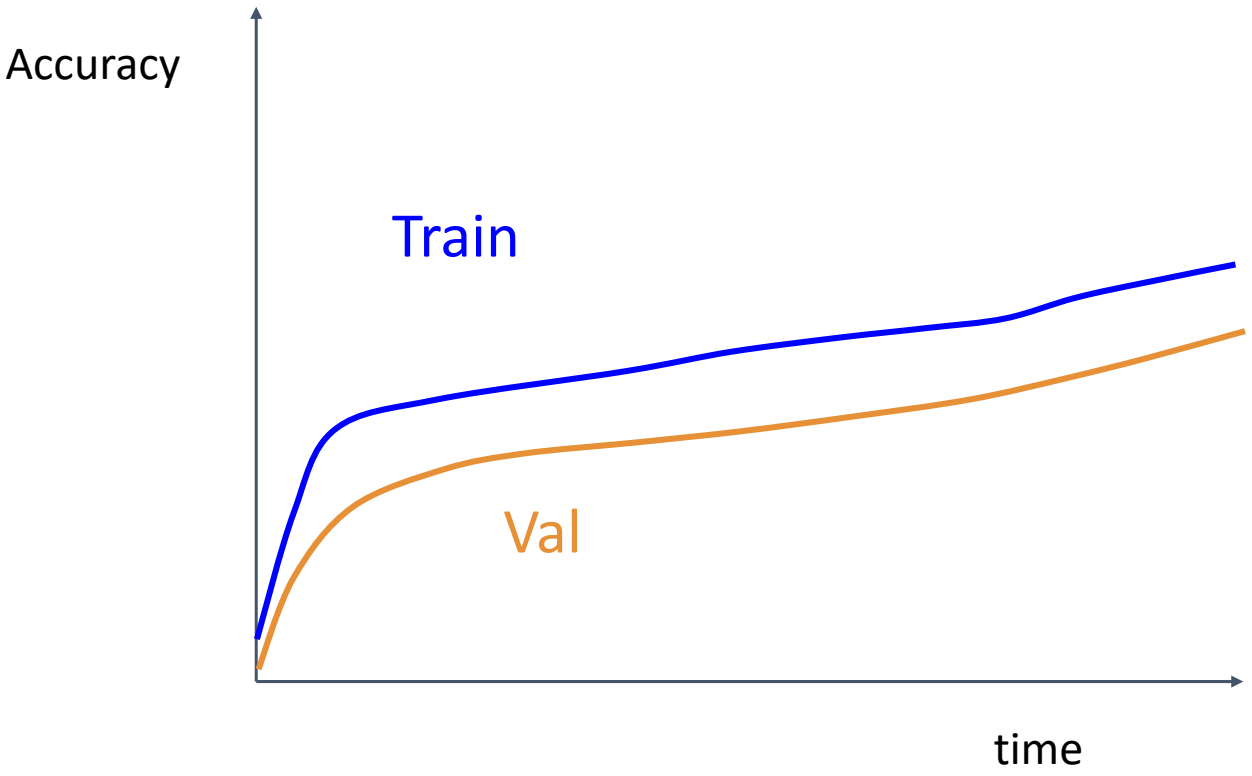
How long to train? **Early Stopping**



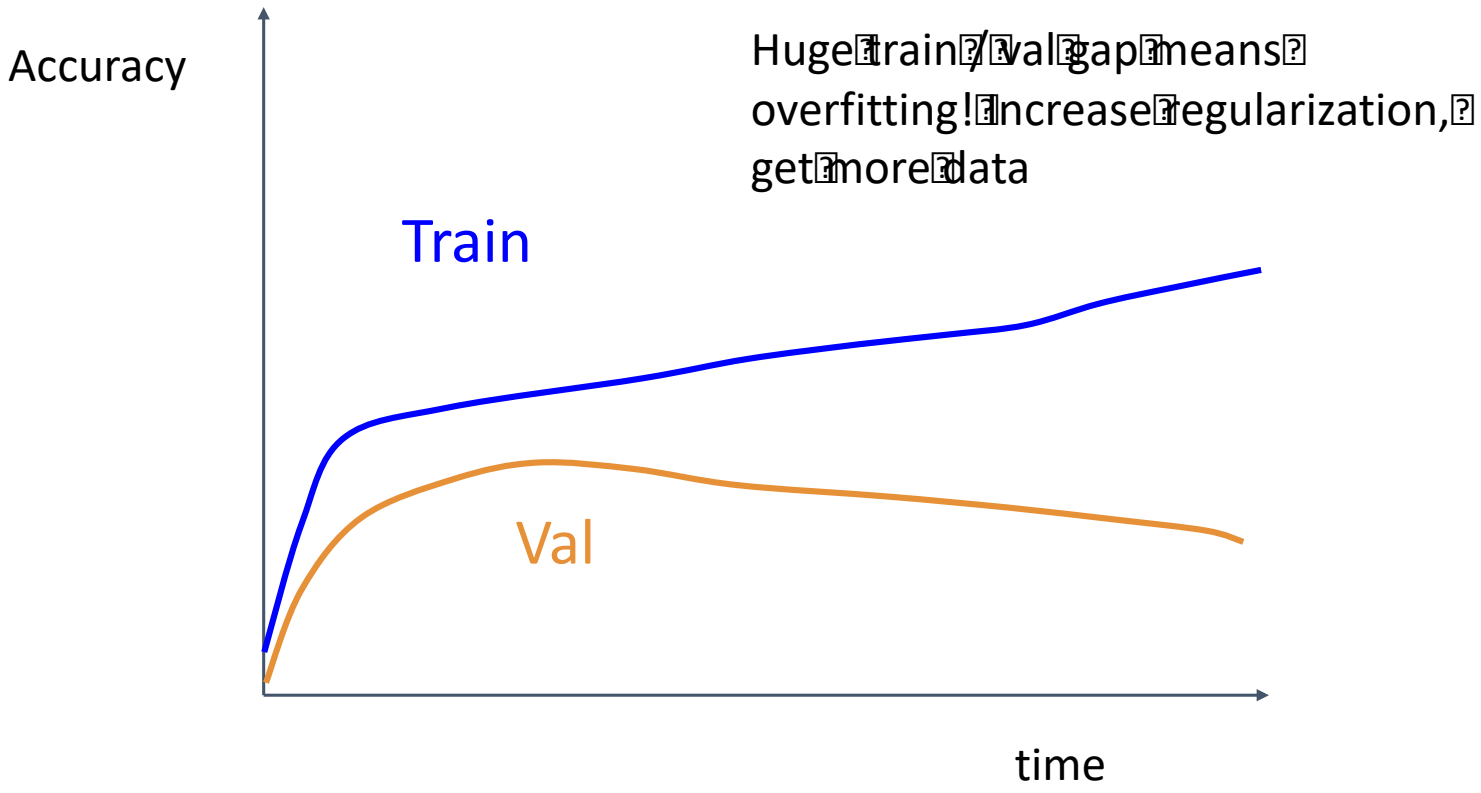


Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val. **Always a good idea to do this!**

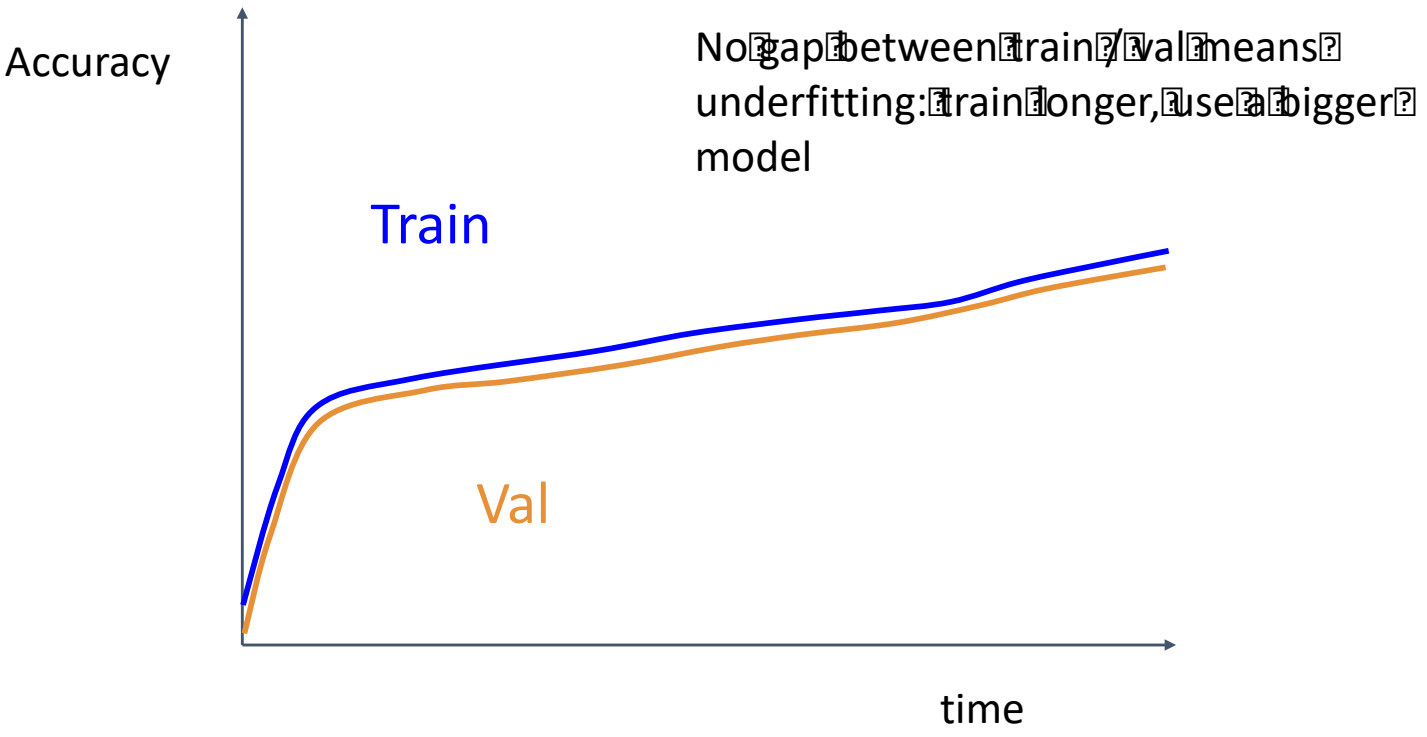
How long to train?



How long to train?



How long to train?



VGG16(16 layers), VGG19(19 layers) (VGG - Visiual Geometry Group)

VGG: Deeper Networks, Regular Design

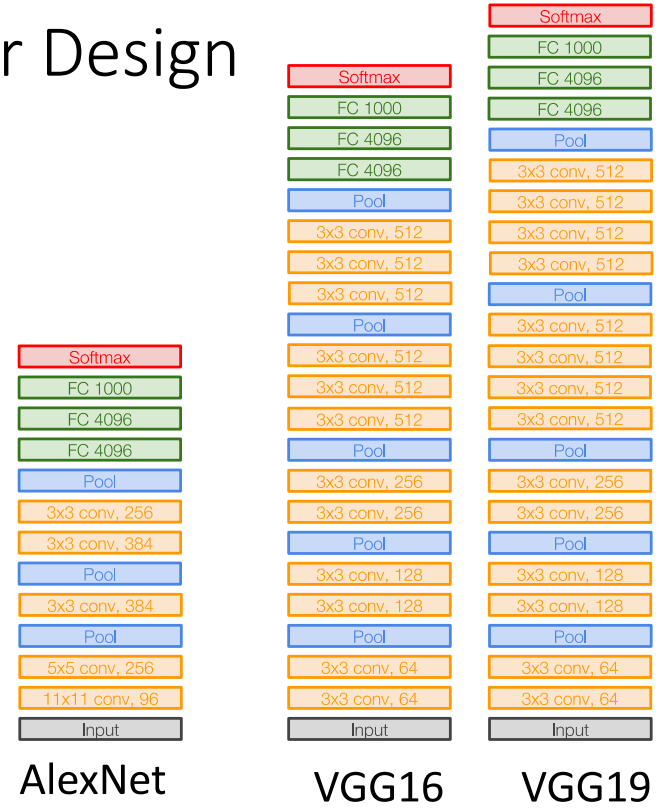
VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

This model achieves *92.7%* test accuracy on ImageNet dataset which contains *14* million images belonging to 1000 classes



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015