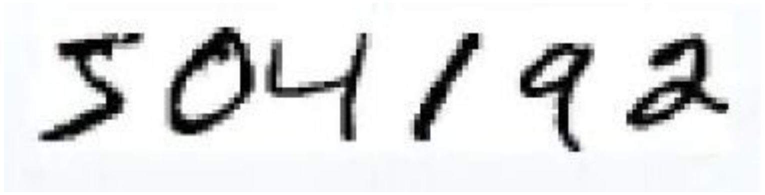
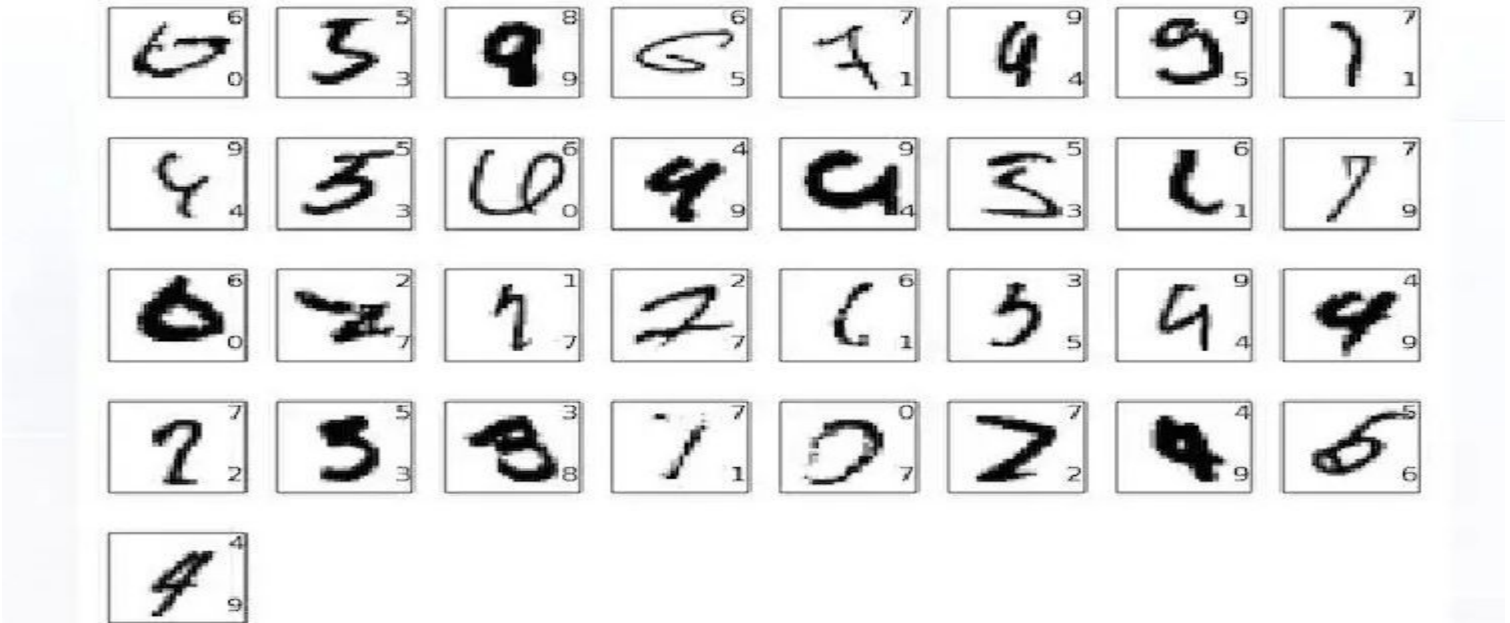
DEEP LEARNING NEURAL NETWORK

Handwritten Sequence



- Human being effortlessly can recognize these digits
- If we attempt to write computer program to recognize the digits, it is an Herculean task in the field of pattern

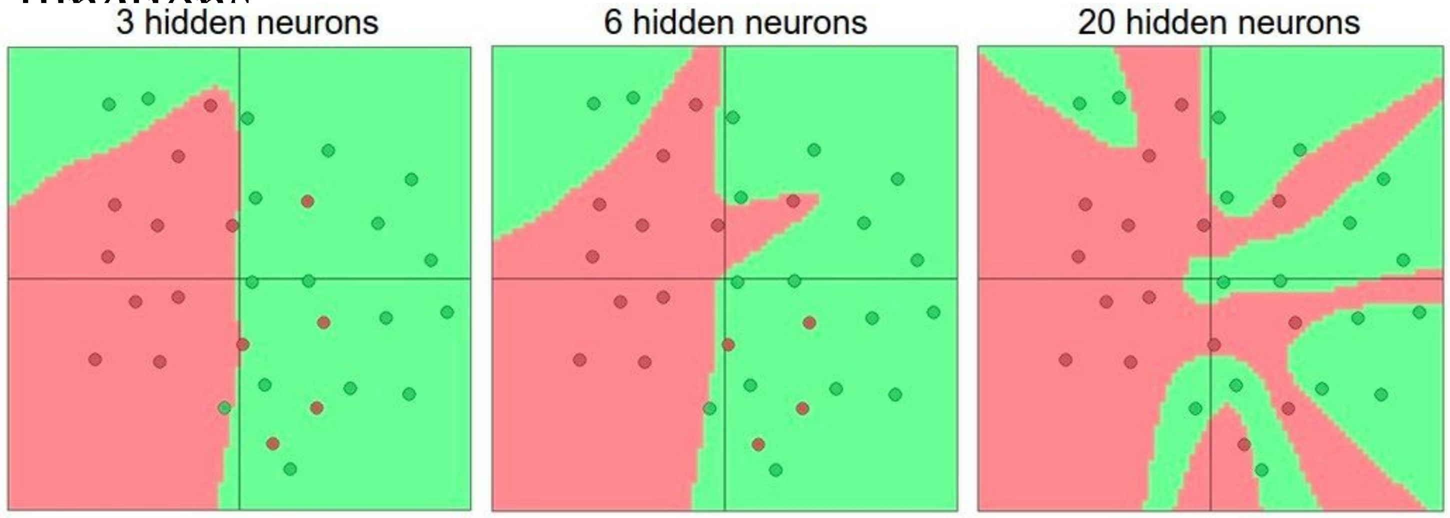


Why Deep Learning

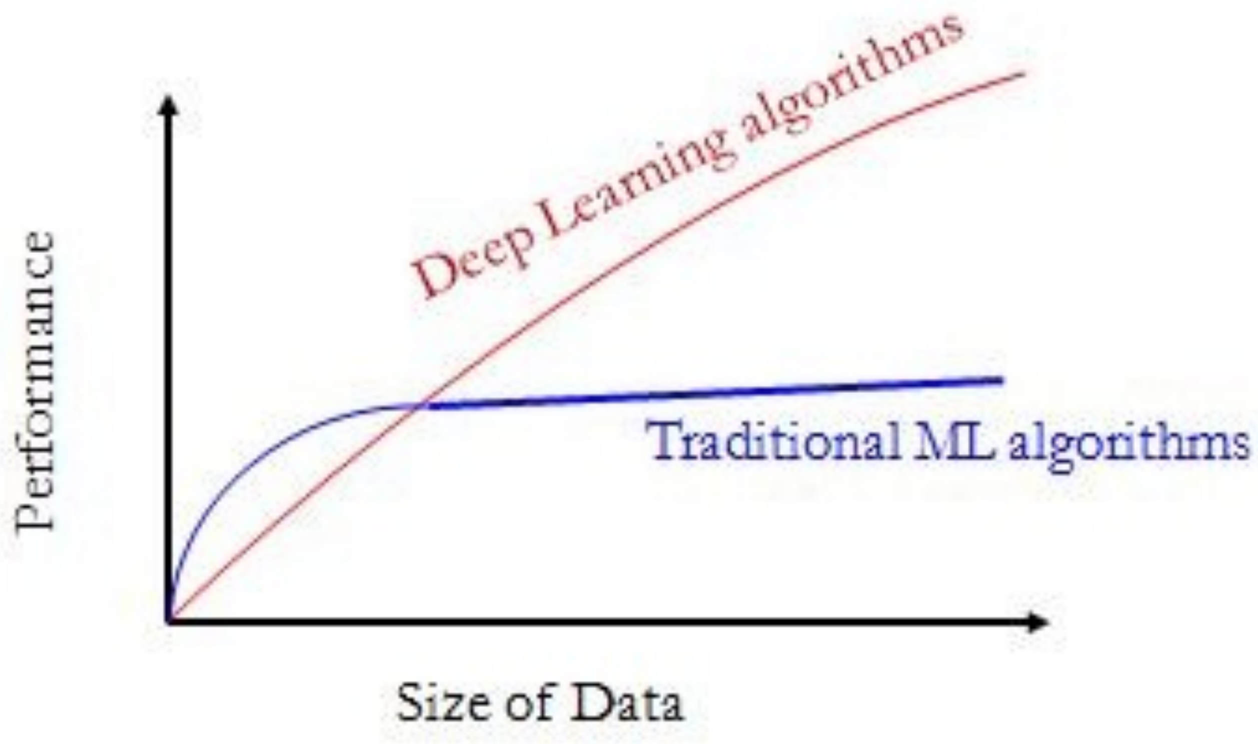
- Non-linearities needed to learn complex representations of data.

- Deep learning algorithms attempt to learn data representation by using a **hierarchy of multiple layers.**

- More layers and neurons approximate more complex functions



Performance Vs. Sample Size



Deep Learning

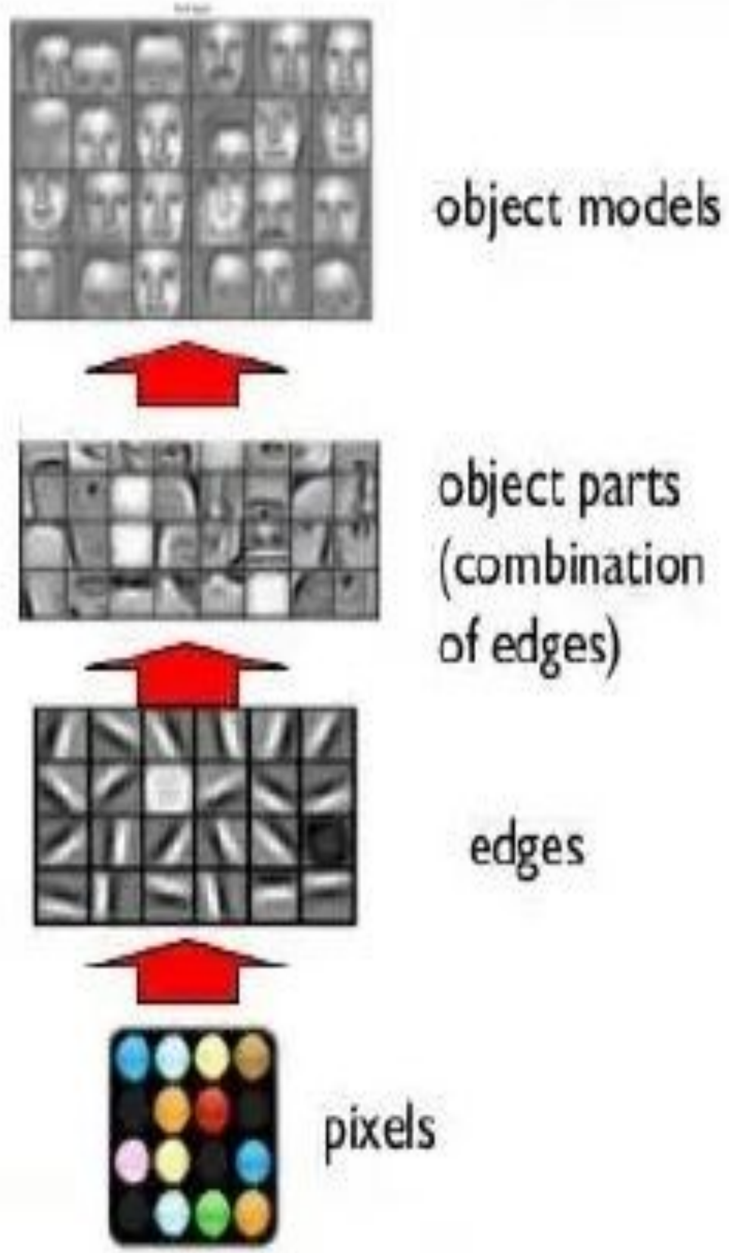
- Deep learning models high-level abstractions by using a deep network with multiple processing layers, composed of multiple linear and non-linear transformation.
- The term deep learning begins to gain popularity after a paper by Hinton on 2000
- Deep learning based models learn representations of large-scale unlabeled data.
- It replaces handcrafted features with efficient algorithms for feature learning and hierarchical feature extraction

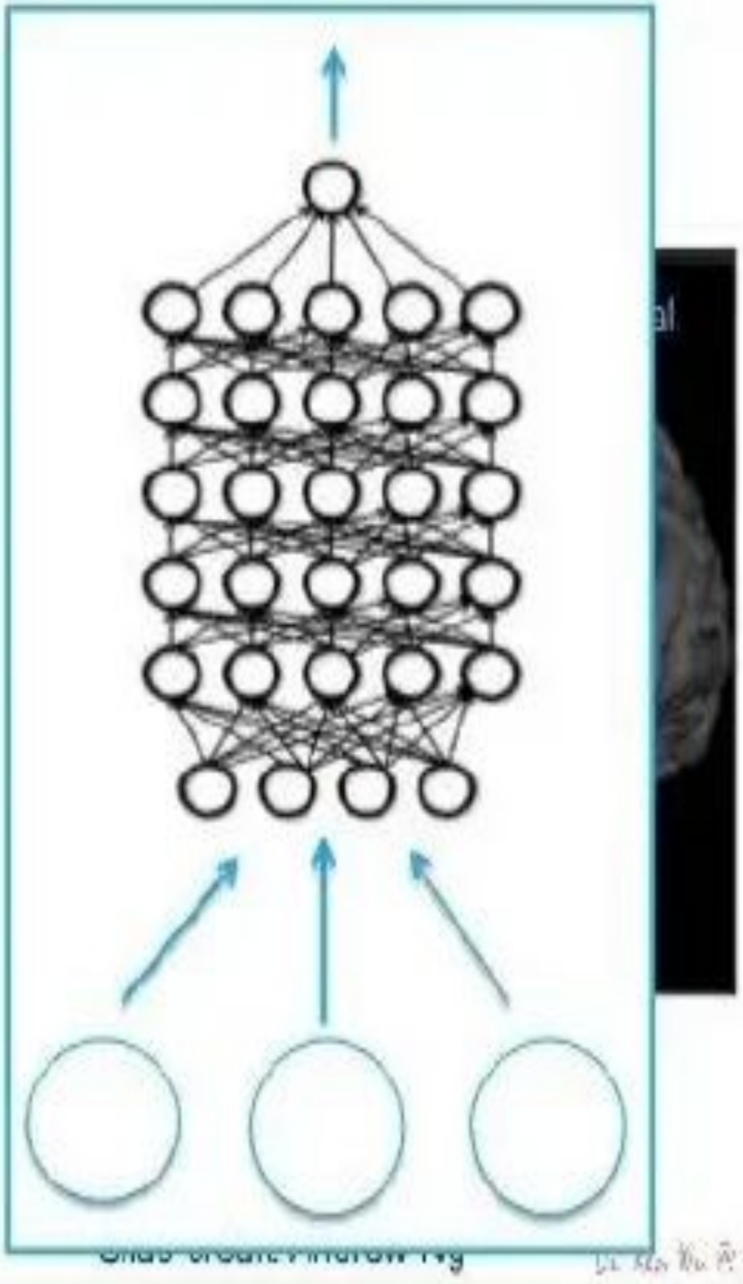
Deep Learning

- As we construct larger neural networks and train them with more and more data, their performance continues to increase.
- This is generally different to other machine learning techniques that reach a plateau in performance.
- Deep learning involves training a hierarchy of feature detectors in comparison to shallow learning models which rely on hand-crafted feature detectors.[6]

Deep Learning

- The first layer learns primitive features, occur more often, like an edge in an image or the tiniest unit of speech sound.
- Once that layer learns the features, they are fed to the next layer to learn complex features, like a corner or a combination of speech sounds.
- The process is repeated in successive layers until the system can reliably recognize phonemes or objects.
- Drivers behind deep learning: huge amount of unstructured complex data increases performance with the size of the neural network.


object models
object parts
(combination
of edges)
edges
pixels



Neural Network

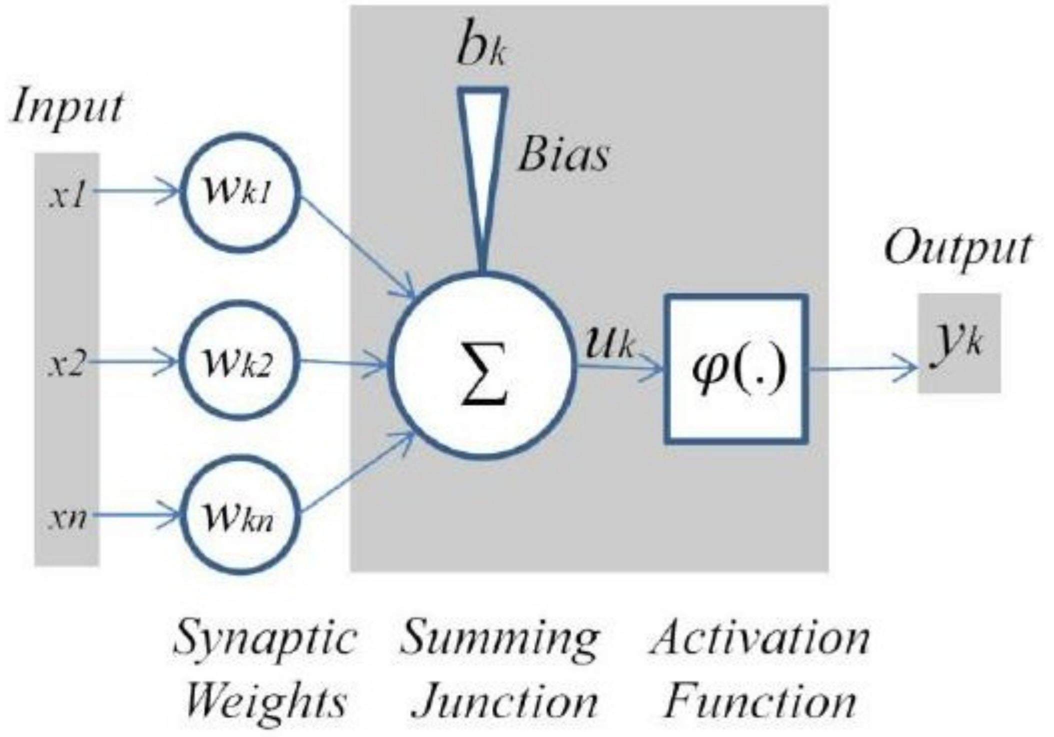
- A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. [2]

- It resembles the brain in two respects

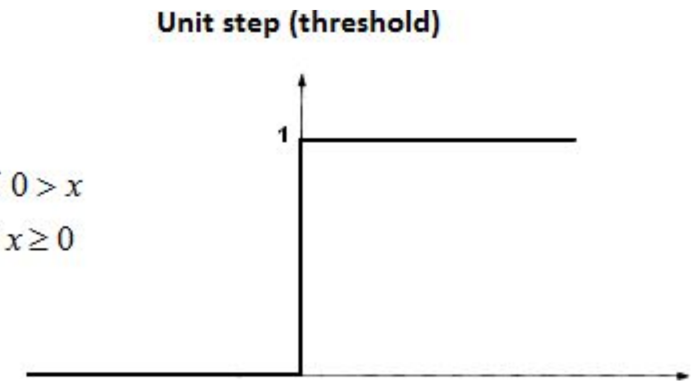
- 1. Knowledge is acquired by the network from its environment through a learning process.

- 2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Nonlinear Model of a Neuron

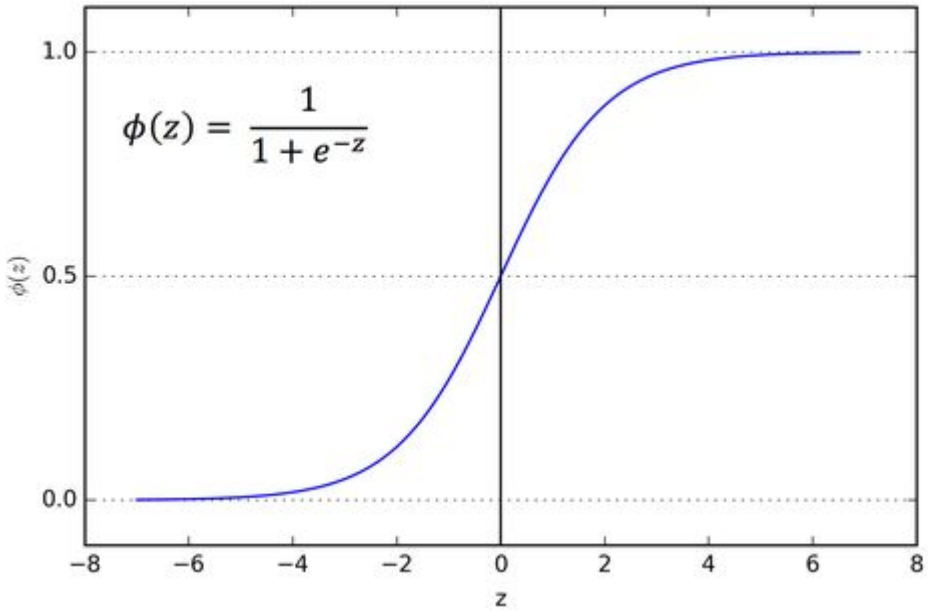


Activation Function (Threshold)



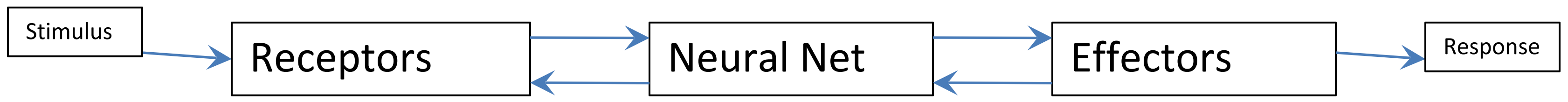
$$f(x)=\begin{cases}0 \text{ if } 0>x\\ 1 \text{ if } x\geq 0\end{cases}$$

Activation Function (Sigmoid)



The sigmoid function **saturates when its argument is very positive or very negative,** meaning that the function becomes very flat and insensitive to small changes in its input.

Human Nervous System [2]



Human Brain[2]

- Receptors – convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net (brain)
- Neural Net – continually receives information, perceives it, and makes appropriate decisions
- Effectors – convert electrical impulses generated by the neural net into discernible responses as system output
- Arrows pointing from left to right indicate the forward transmission of information-bearing signals through the system
- Arrows pointing from right to left signify the presence of feedback in the system

Feed forward Neural Networks

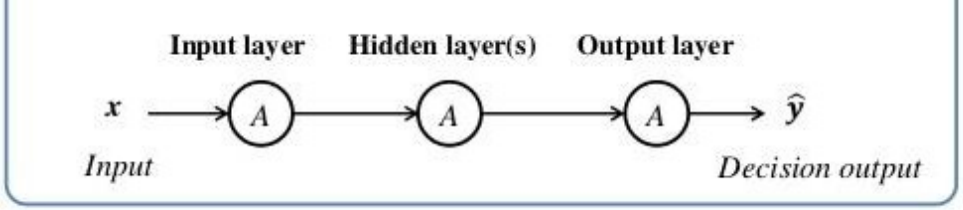
- Multilayer Perceptrons
- Deep Feedforward Networks
- A feedforward network defines a mapping $y = f(x; \vartheta)$ *and learns the value of the parameters* ϑ *that result* in the best function approximation.
- There are no **feedback connections in which** outputs of the model are fed back into itself.
- When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks.**[7]

Feed-forward Network

Feed-forward network

Decisions are based on current inputs:
- No memory about the past
- No future scope

Simplified representation:



Vector of input features: $\boldsymbol{x}$

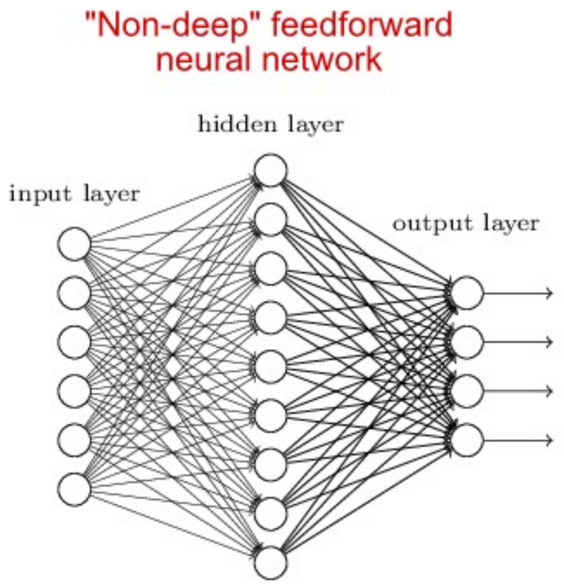
Vector of predicted values: $\hat{\boldsymbol{y}}$

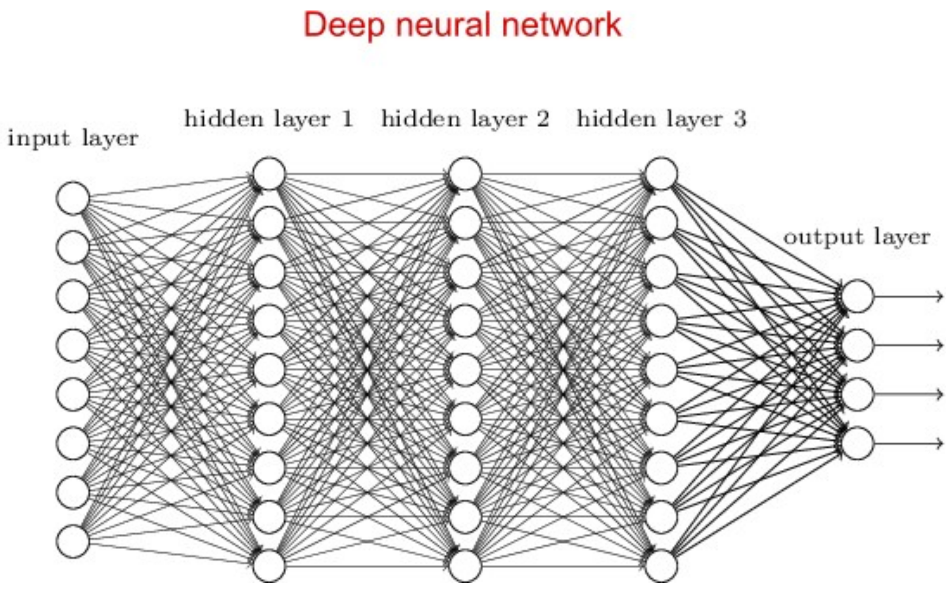
Neural activation: $$y = A\left(\sum_{i=1}^{n} x_i \cdot w_{iy} + b_y\right)$$

A – some activation function (*tanh* etc…)

w, b – network parameters

Feed Forward Neural Network



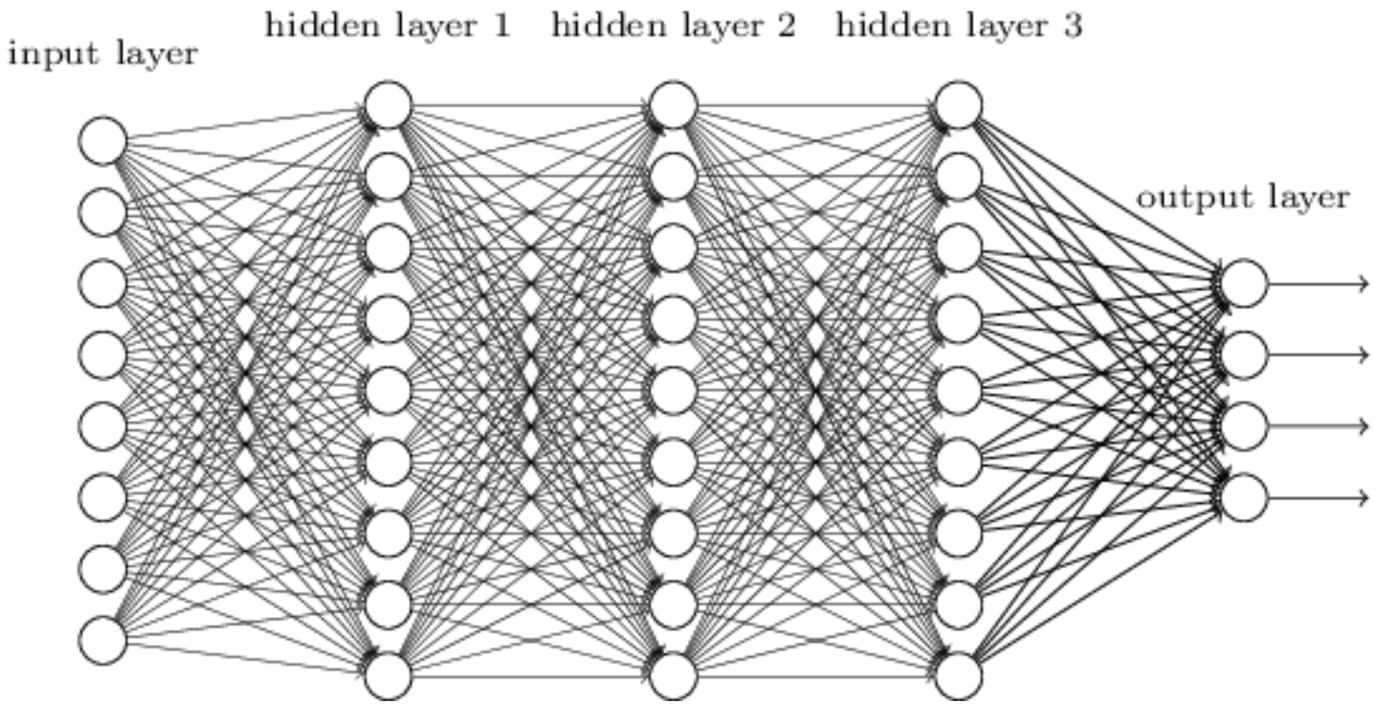


Convolutional Neural Networks

- Connectivity pattern between its neurons is inspired by the organization of the animal visual cortex
- Individual cortical neurons respond to stimuli(kernel function) in a restricted region of space known as the receptive field
- The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation

Convolution Neural Network

- Do we really need all the edges of the network?
- Can some of these be shared?



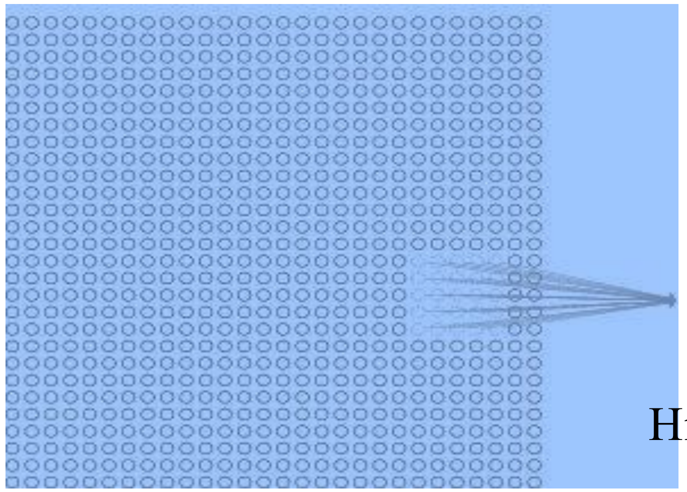
Convolutional Neural Networks

Features

- Local connectivity- influenced by spatial local correlations,i.e, objects are more related to nearby objects.
- Shared weights- Same weights or kernel function or filter is applied to different overlapping areas of the image to see response in those parts.

Local Receptive field

Each neuron in the first hidden layer will be connected to a small region of the input neurons

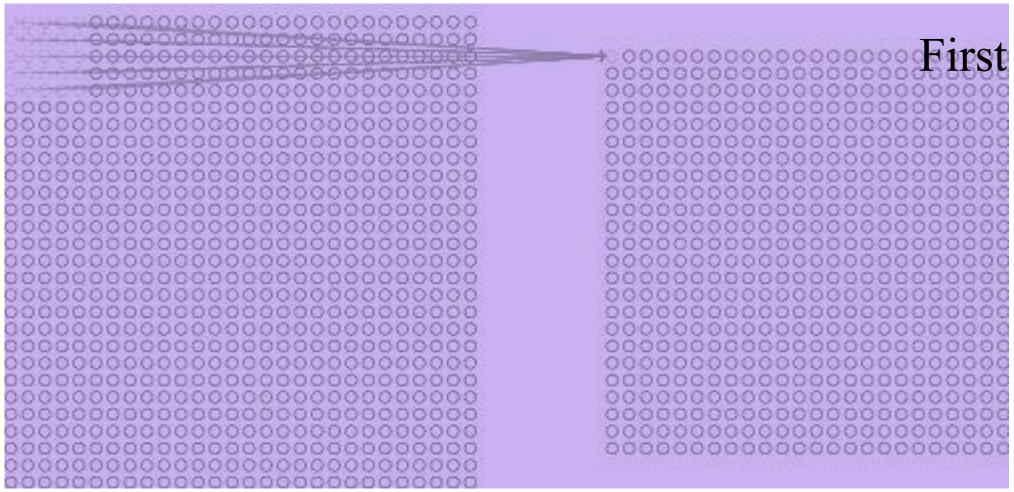


- The region in the input image is called the *local receptive field* for the hidden neuron.

- Each connection learns a weight.

Hidden Layer

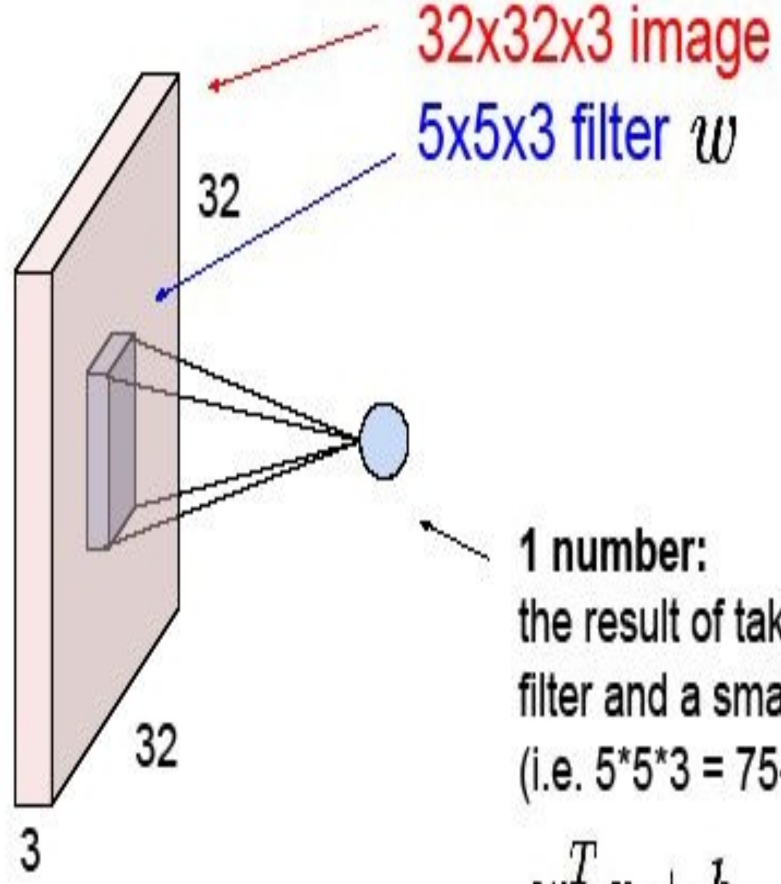
- Slide the local receptive field across the entire input image and by sliding one pixel to the right (i.e., by one neuron), connect to a second hidden neuron and so on building the first hidden layer.
- For each local receptive field, there is a different hidden neuron in the first hidden layer.



First Hidden layer

A different stride length is also used

Convolution Layer

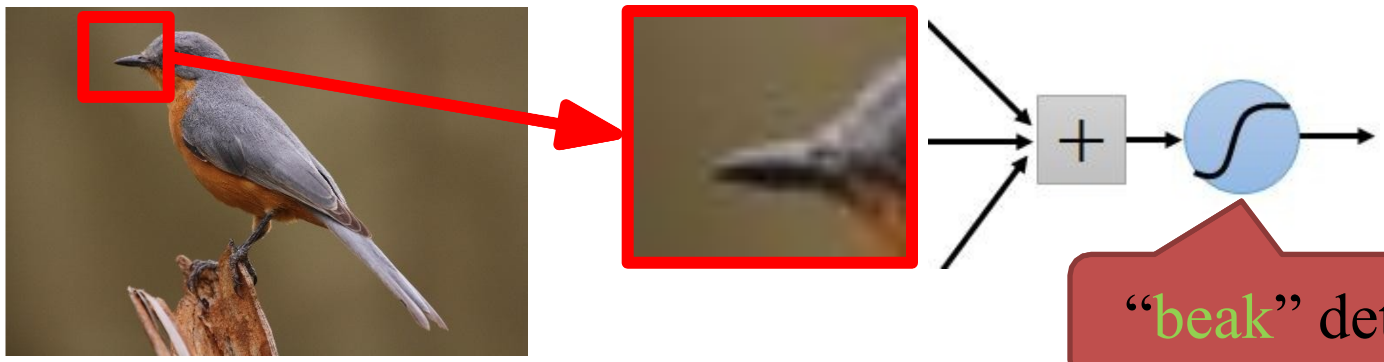


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

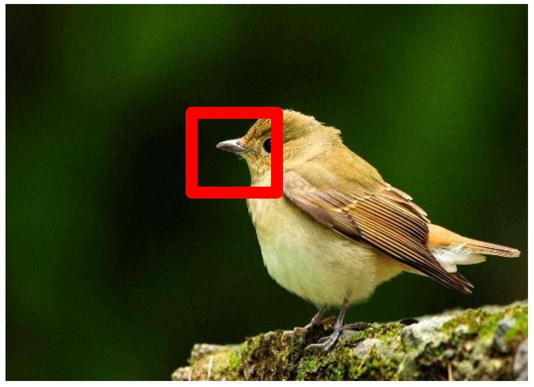
Learning an image:

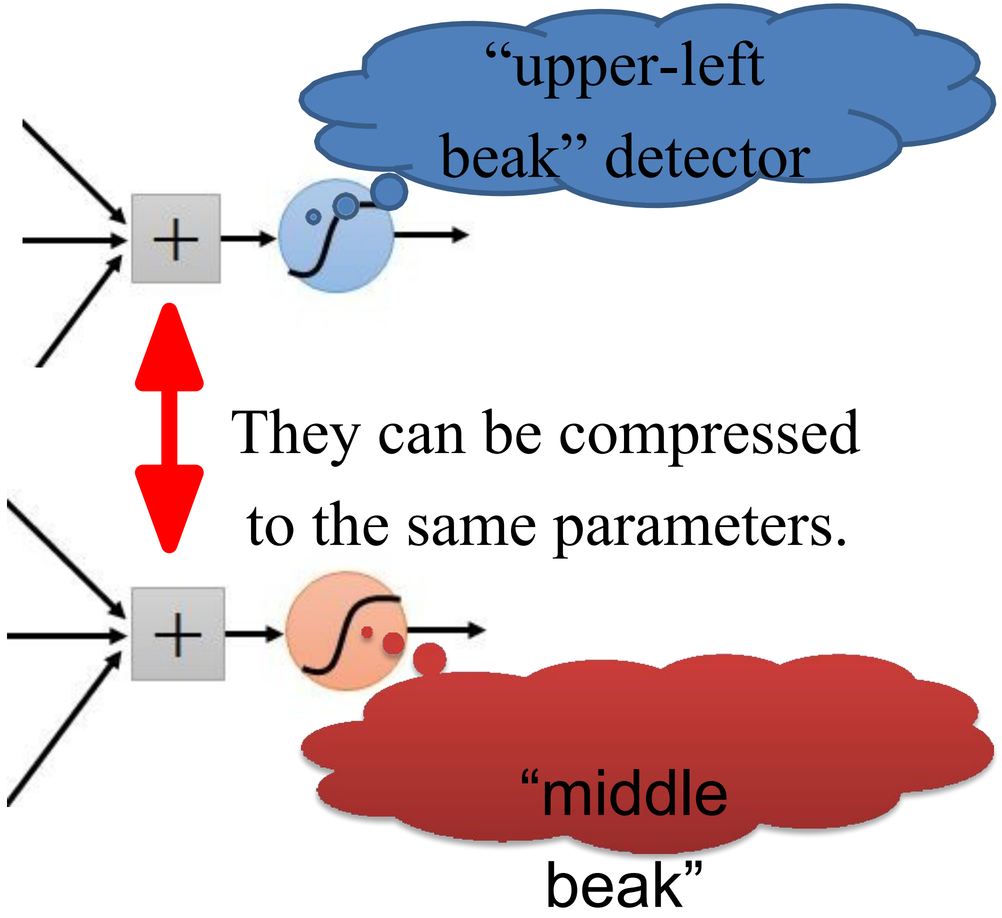
Can represent a small region with fewer parameters



"beak" detector

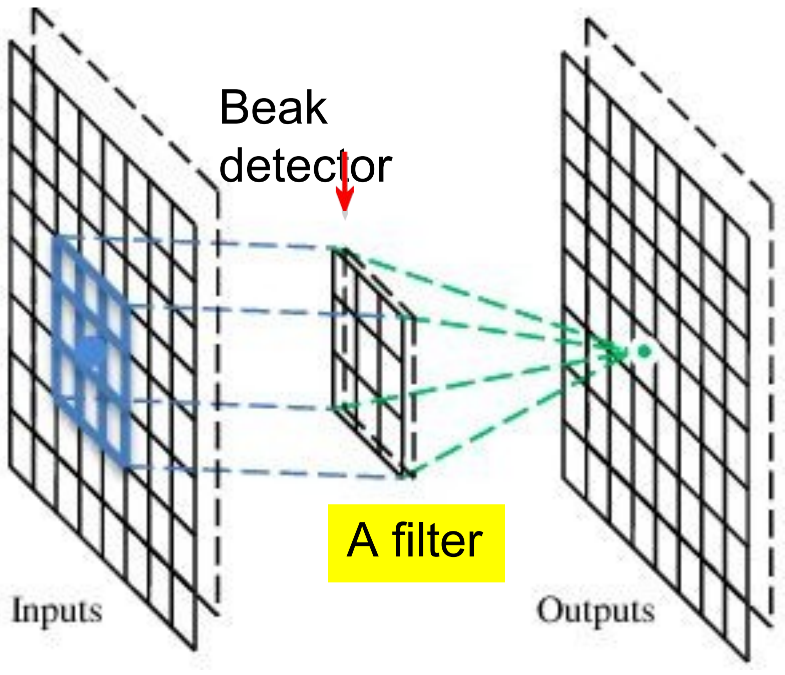
Same pattern appears in different places:
They can be compressed!
What about training a lot of such "small" detectors and each detector must "move around".




"upper-left beak" detector
They can be compressed to the same parameters.
"middle beak"

A convolutional layer

A convolutional layer has a number of filters that does convolutional operation.



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

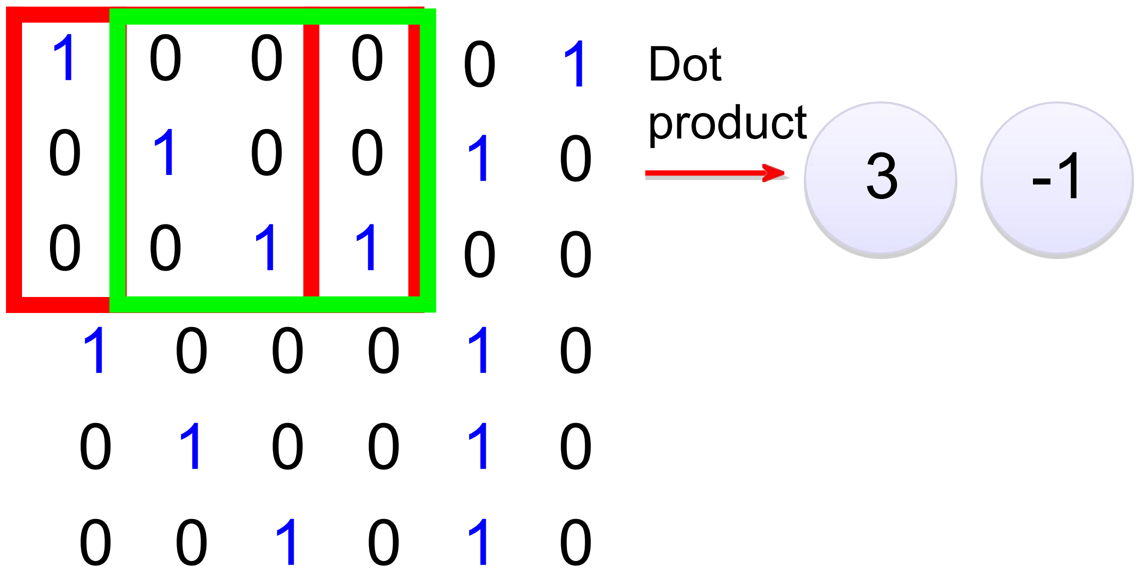
Each filter detects a small pattern (3 x 3).

Convolution

stride=1

1	-1	-1
-1	1	-1
-1	-1	1

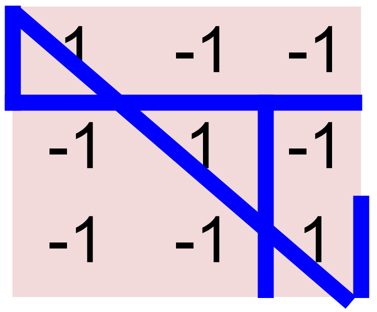
Filter 1



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

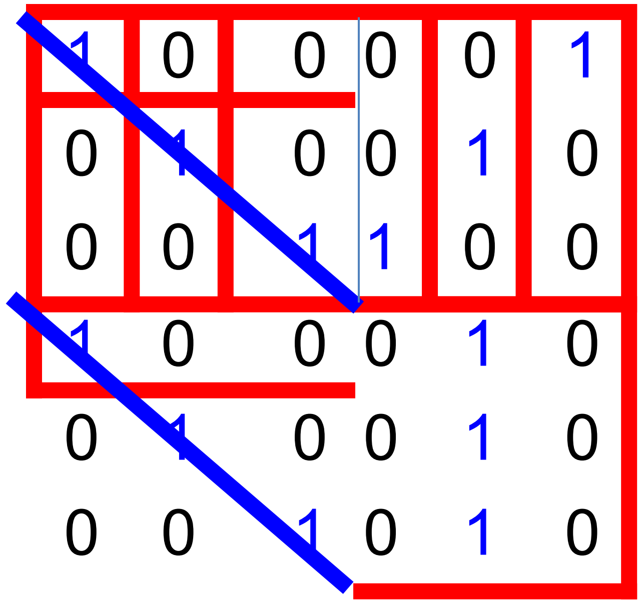
6 x 6 image

Convolution

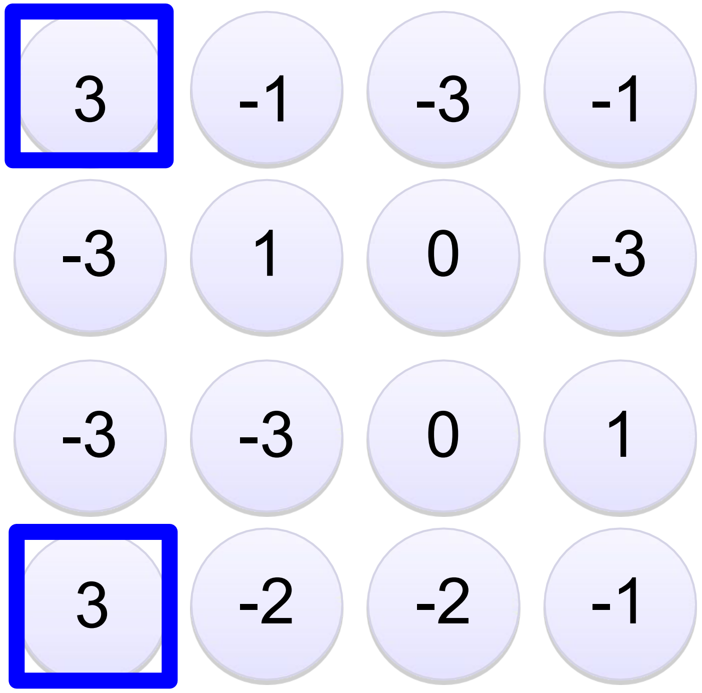


Filter 1

stride=1



6 x 6 image

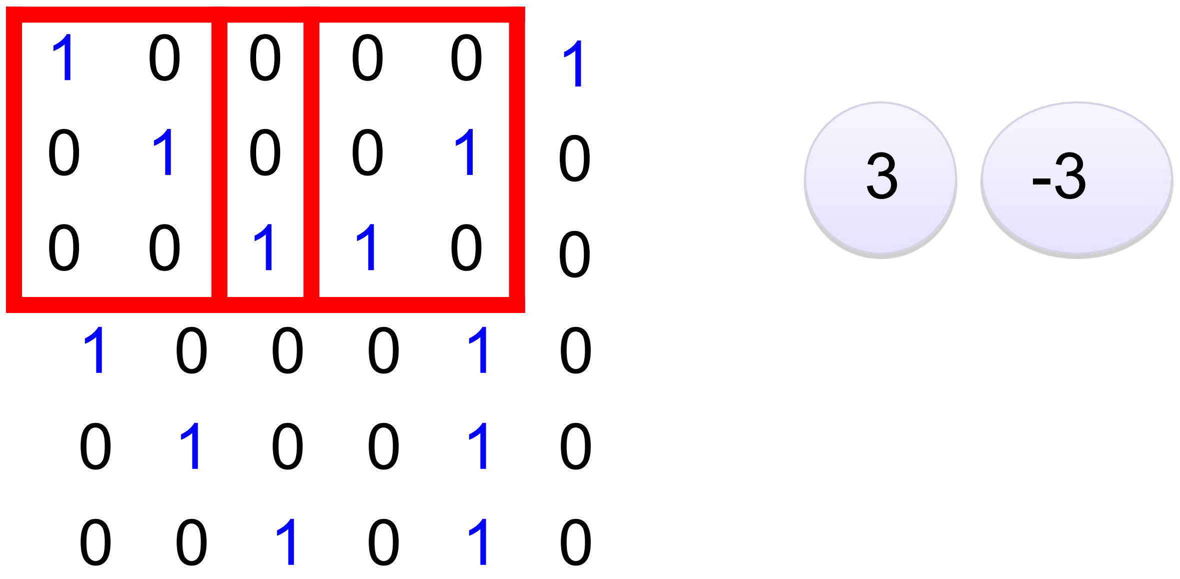


Convolution

If stride=2

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

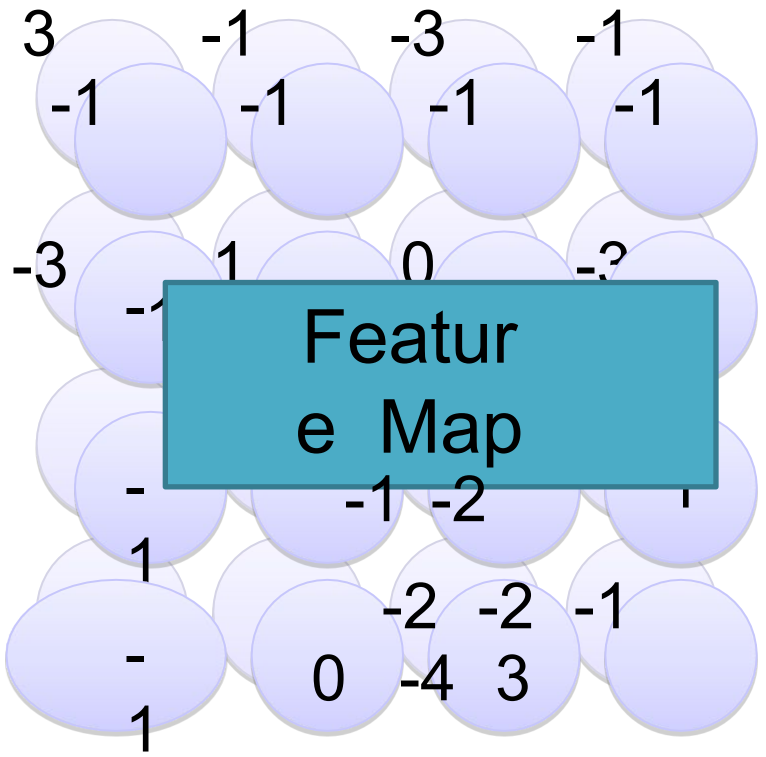
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

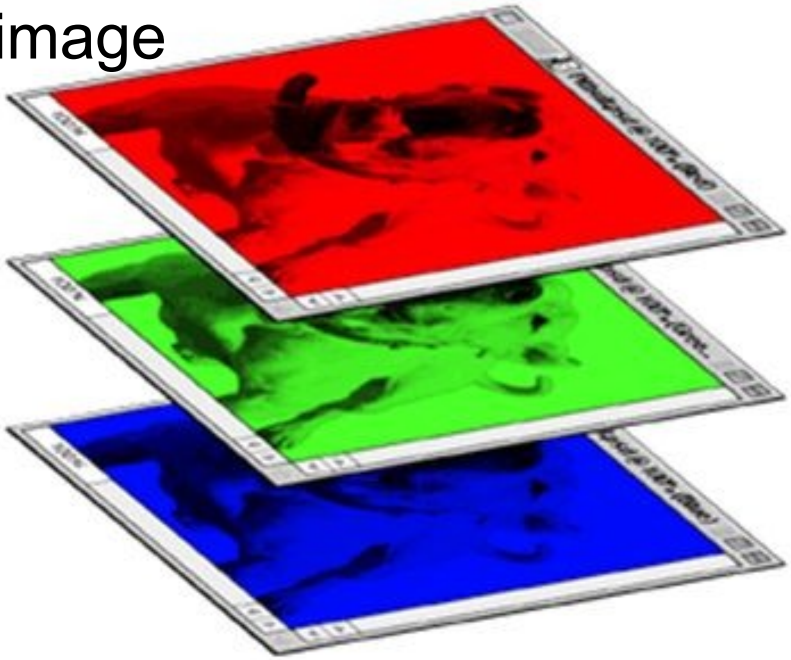
1	-1	-1
-1	1	-1
-1	-1	1

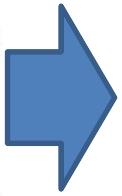
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Color image





1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

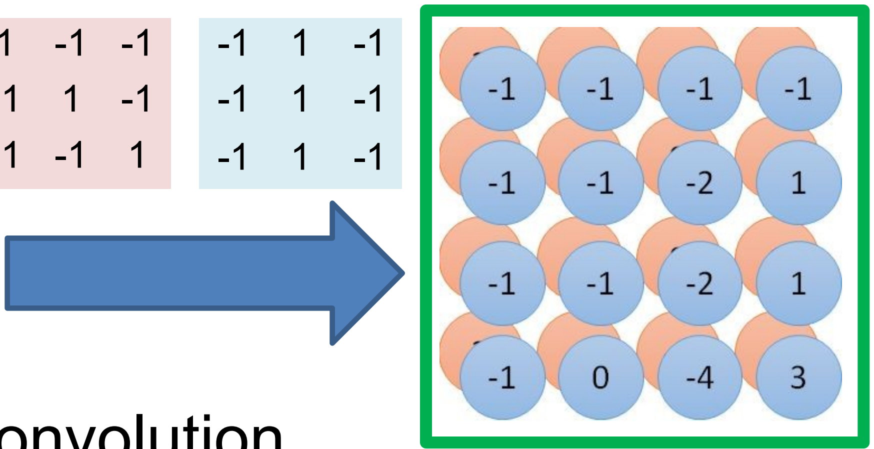
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolution v.s. Fully Connected

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{matrix}$$

$$\begin{matrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{matrix} \qquad \begin{matrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{matrix}$$



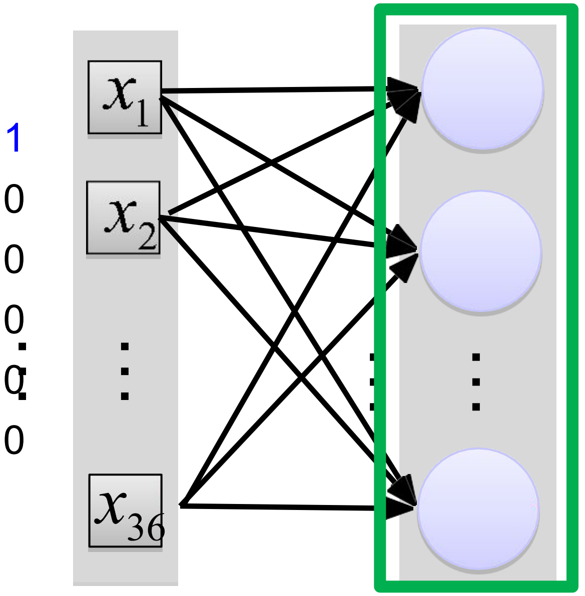
convolution

image

fewer parameters!

Fully-conn
e cted

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{matrix}$$



Convolution and Shallow NN

- Convolutional Neural Networks are very similar to regular Neural Networks.
- CNN are made up of neurons that have learnable weights and biases, updated using loss function.
- Each neuron receives inputs, performs a dot product and follows with a non-linearity.
- The CNN has an activation function (e.g. sigmoid/Softmax) on the last (fully-connected) layer and all steps of learning regular Neural Networks still apply.

So what does change?

- It makes the forward function more efficient to implement and vastly reduce the amount of parameters in the network.
- Regular Neural Nets don’t scale well to full images.
- For example, an image of size, 200×200×3 (200 wide, 200 high, 3 color channels), would lead to a single fully-connected neuron in a first hidden layer with 200×200×3 = 120,000 weights.
- Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Network

- A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network.

- Local connections and tied weights followed by some form of **pooling** results in translation invariant features.

- Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

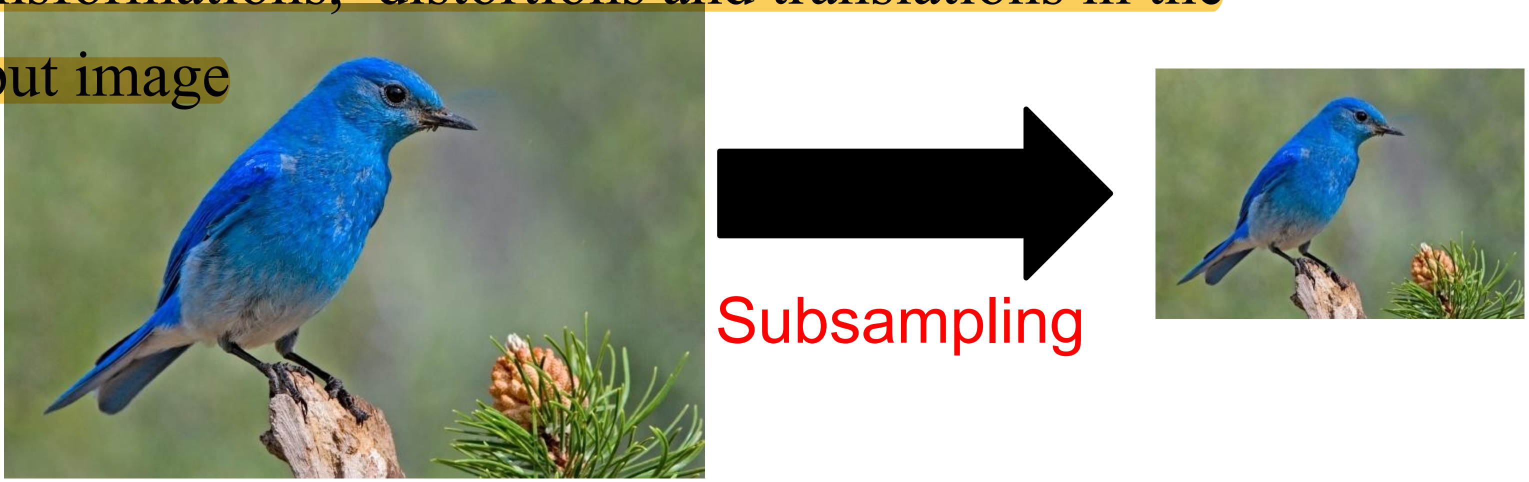
Convolutional Neural Networks- Architecture

- Pooling layer- Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. Max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.
- Fully connected layer- Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.

Why Pooling

- Subsampling pixels will not change the object

Makes the network invariant to small transformations, distortions and translations in the input image



We can subsample the pixels to make image smaller

Fewer parameters to characterize the image

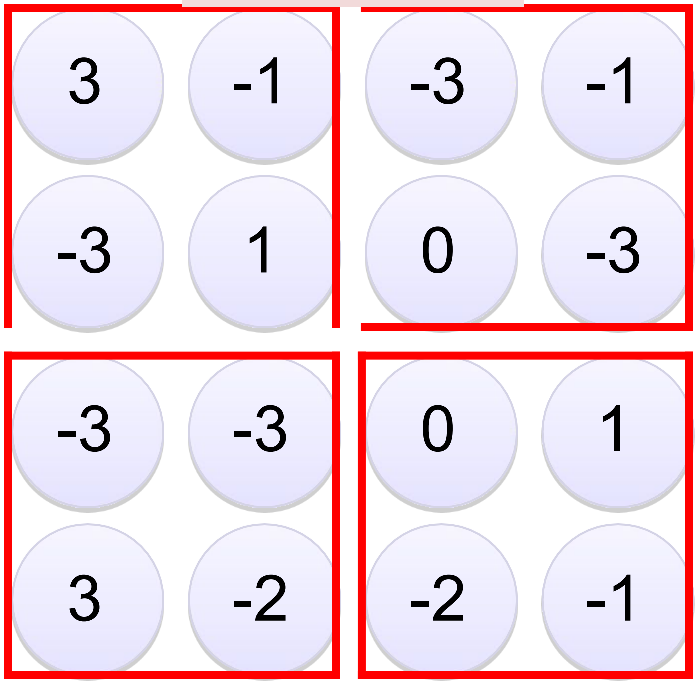
Max Pooling

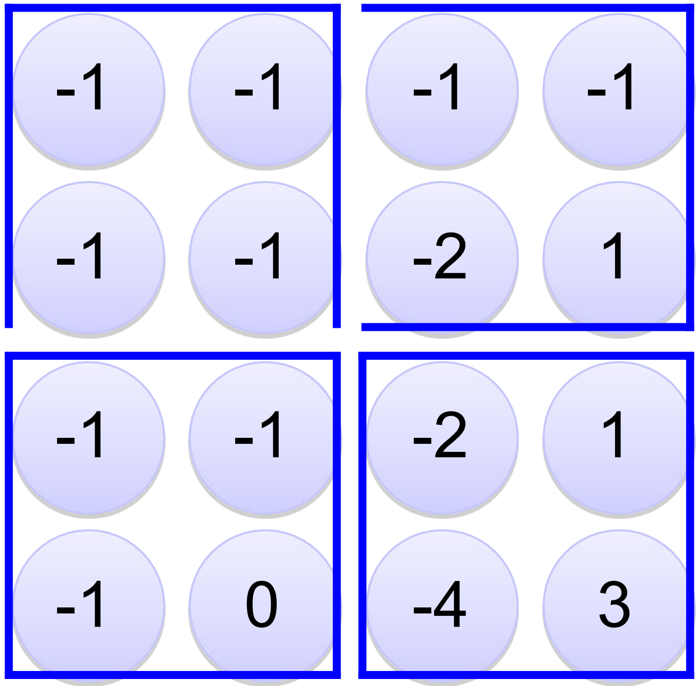
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

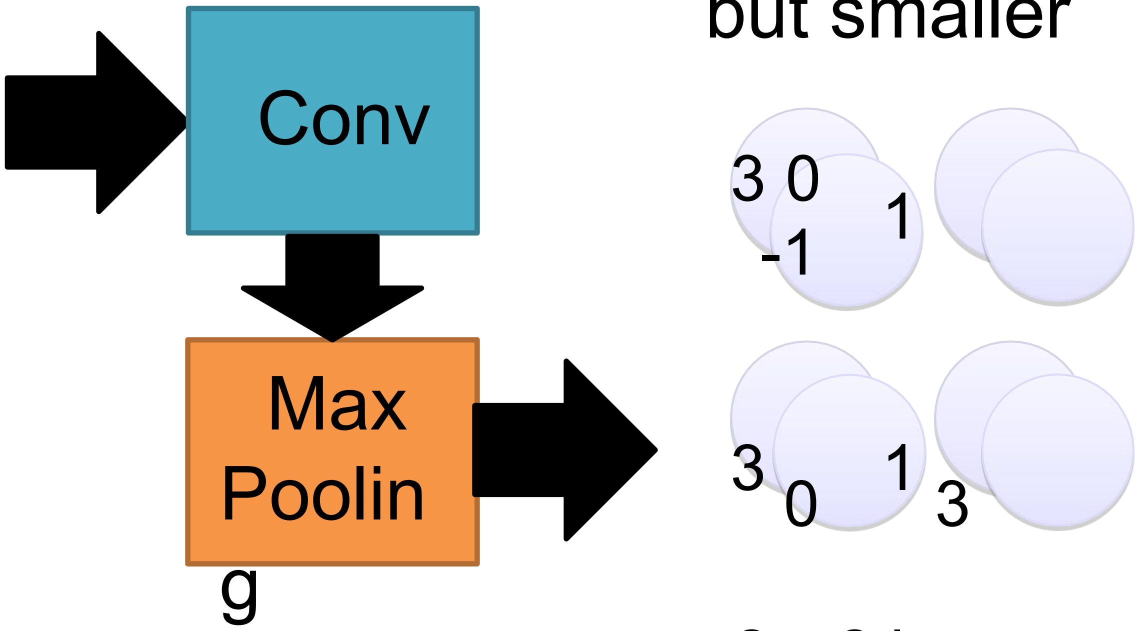




Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

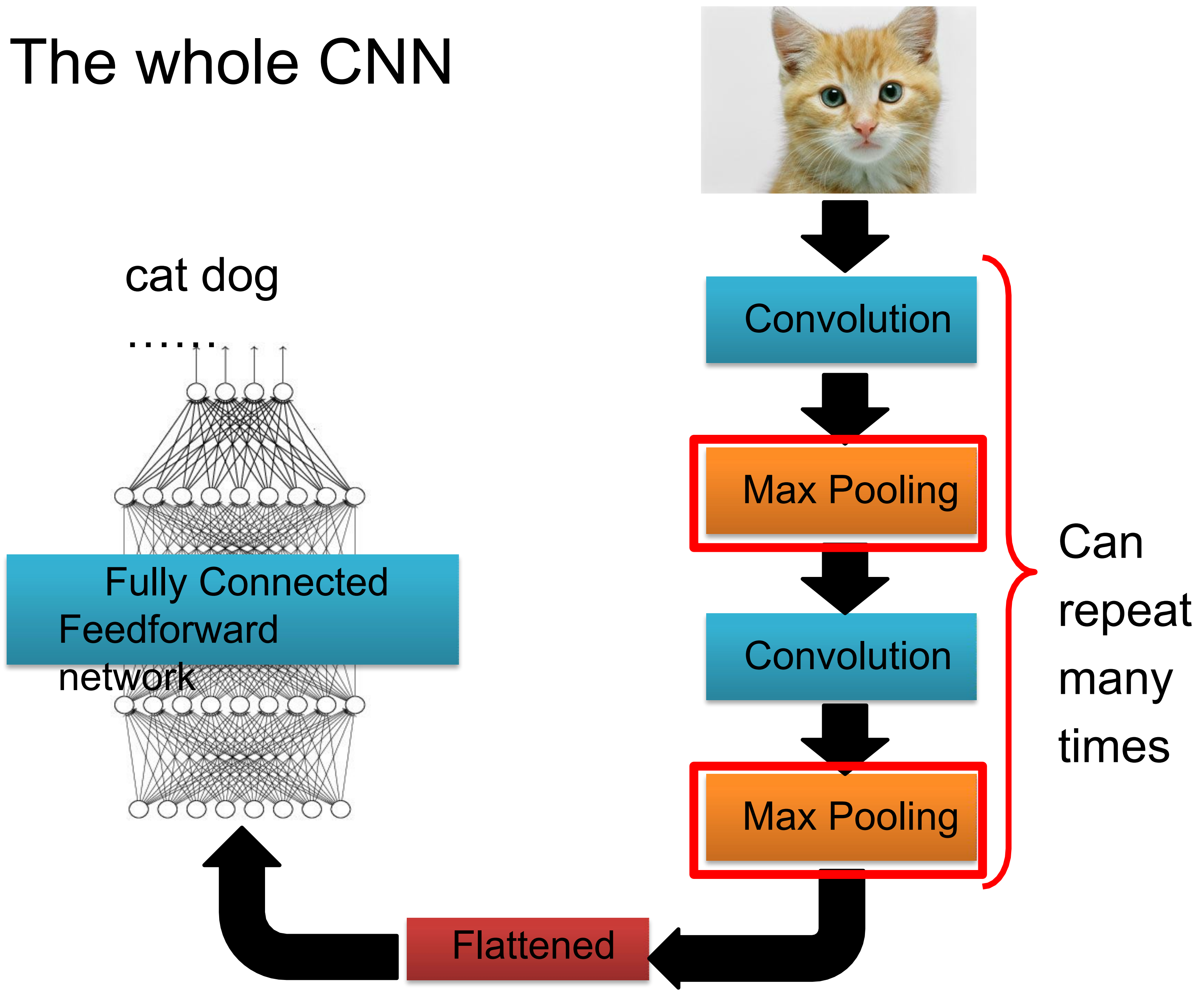


New image but smaller

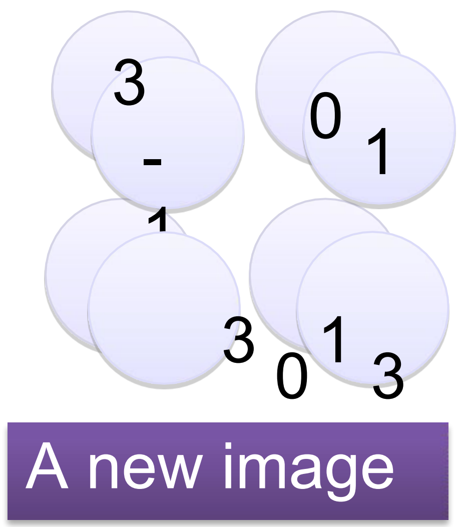
2 x 2 image

Each filter is a channel

The whole CNN

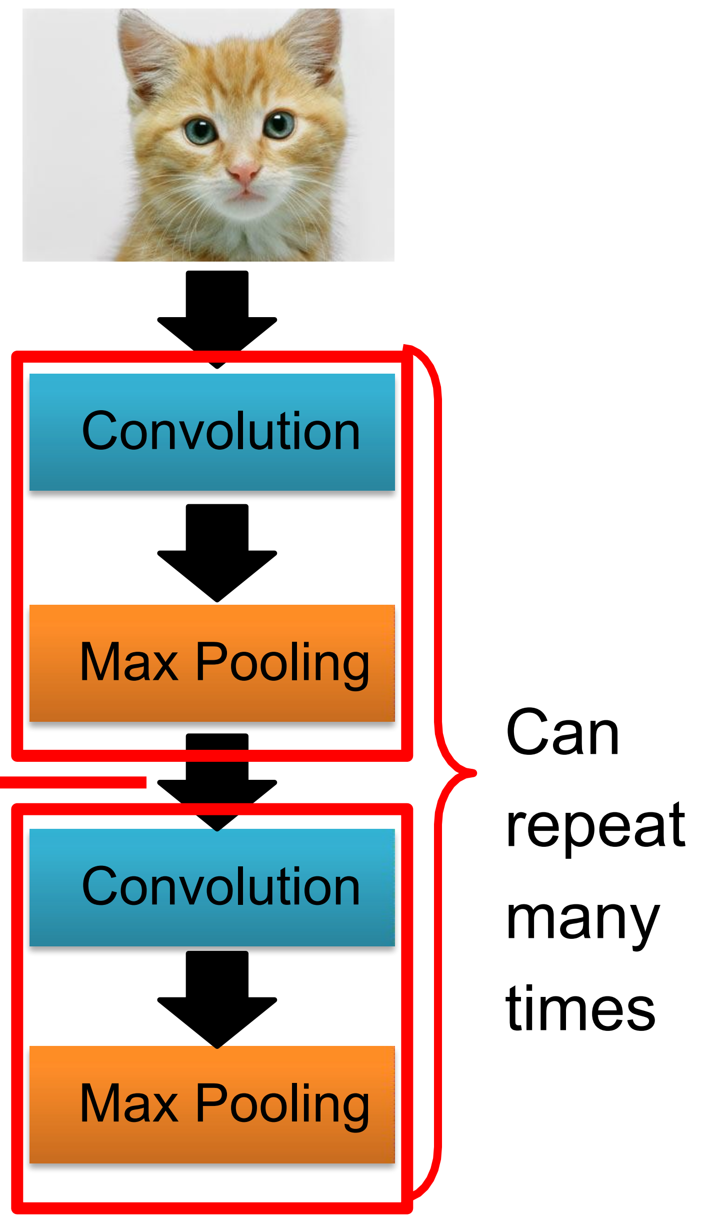


The whole CNN



Smaller than the original image

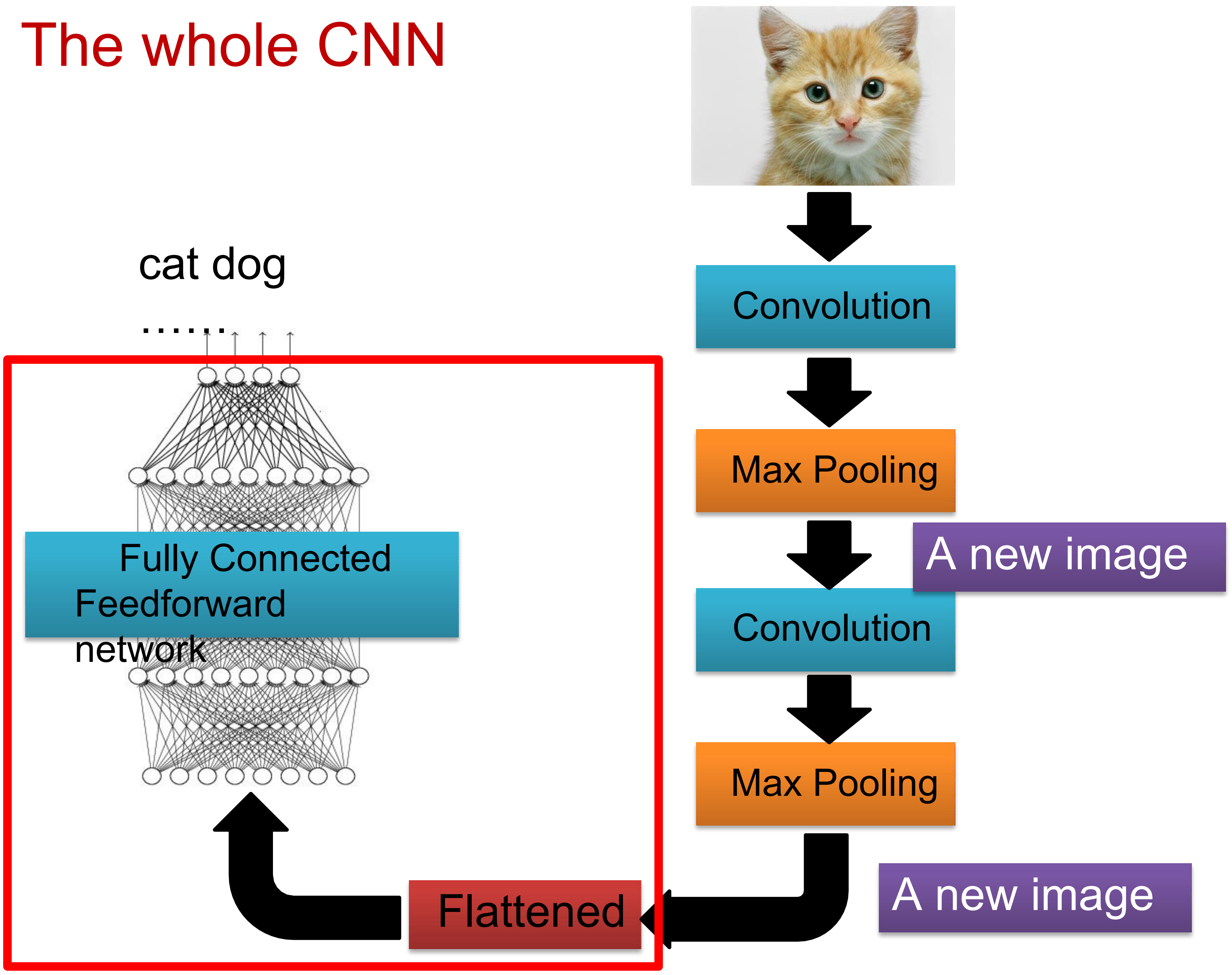
The number of channels is the number of filters



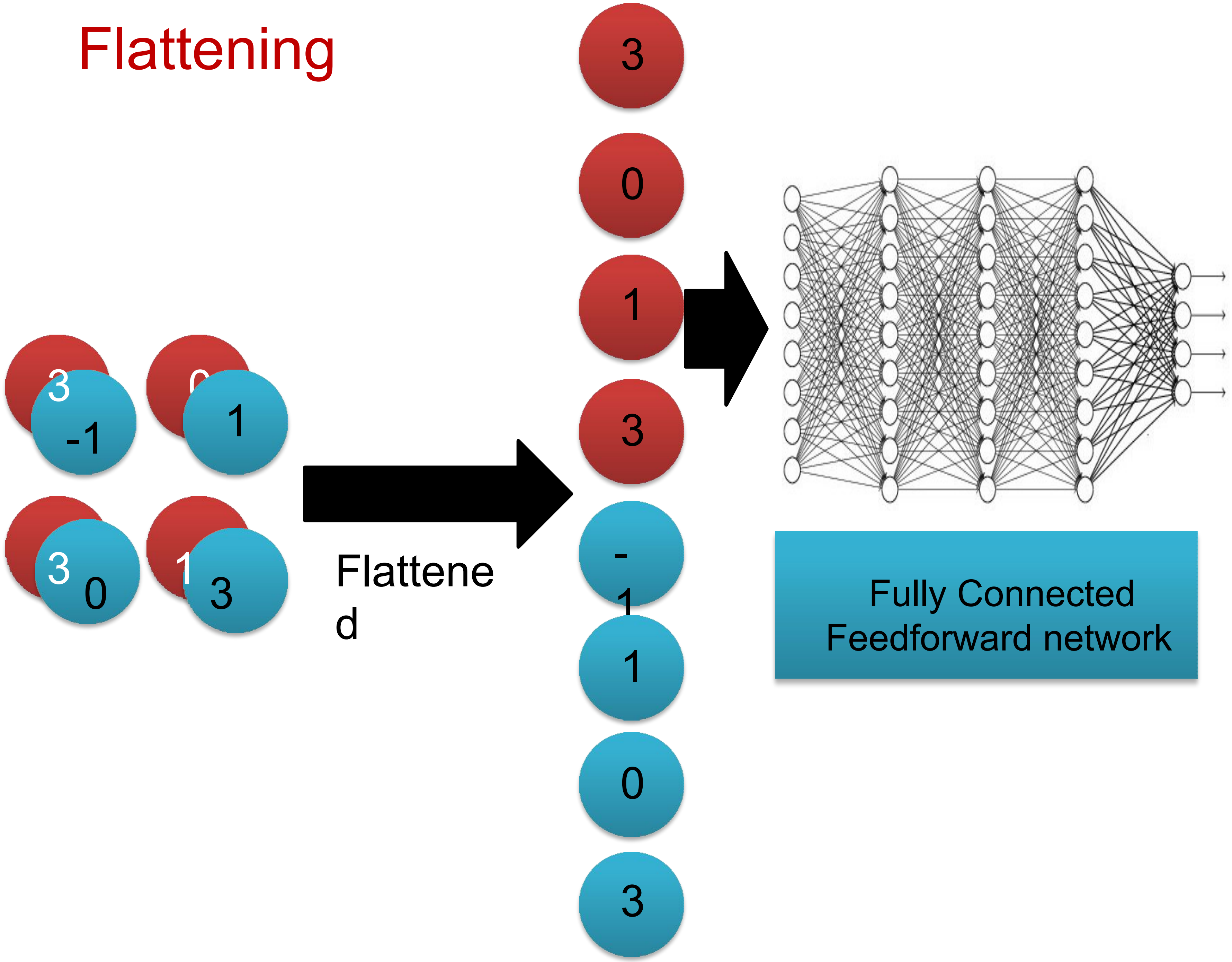
Fully Connected Layer of CNN

- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer.
- The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.
- The output from the convolutional and pooling layers represent high-level features of the input image.
- The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

The whole CNN



Flattening

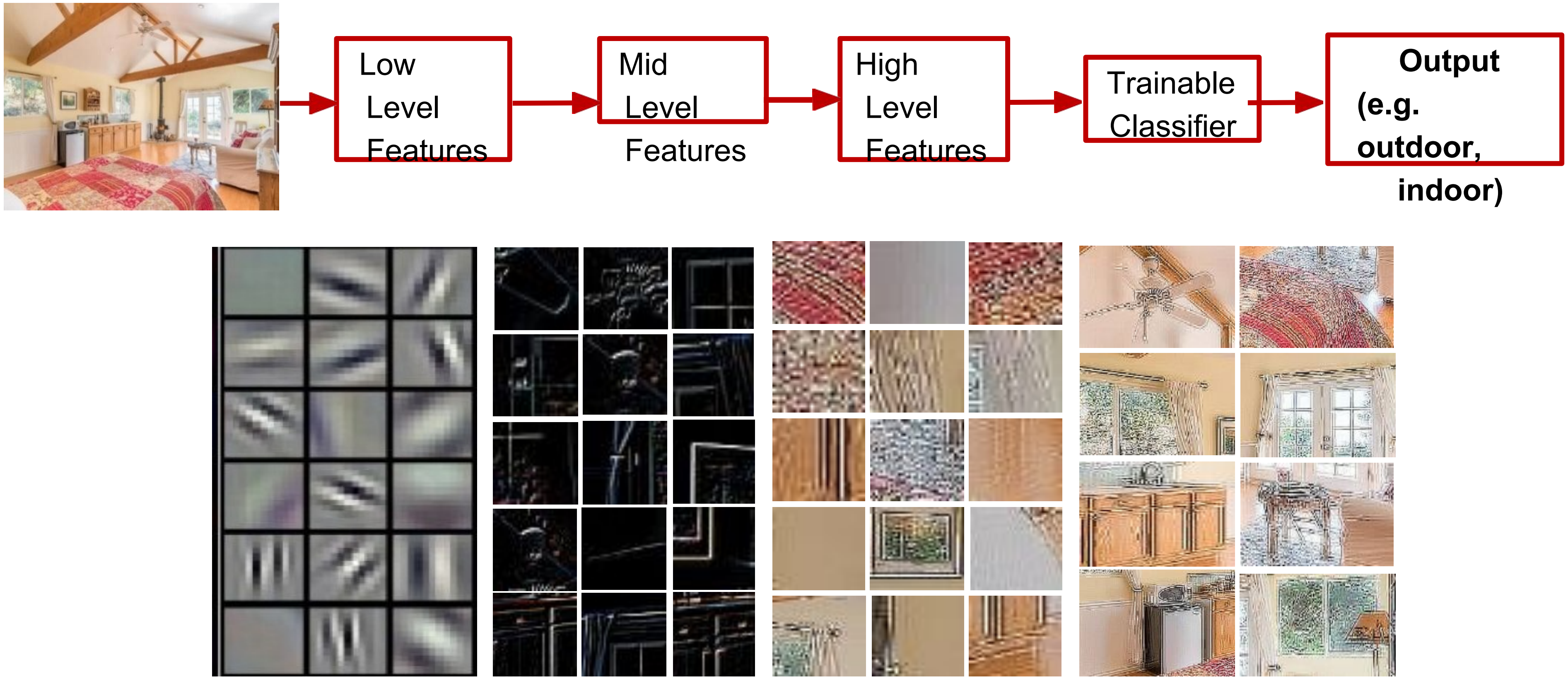

3
-1
0
1
3
0
1
3
Flattene
d
3
0
1
3
-
1
1
0
3
Fully Connected Feedforward network

A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

Deep Learning

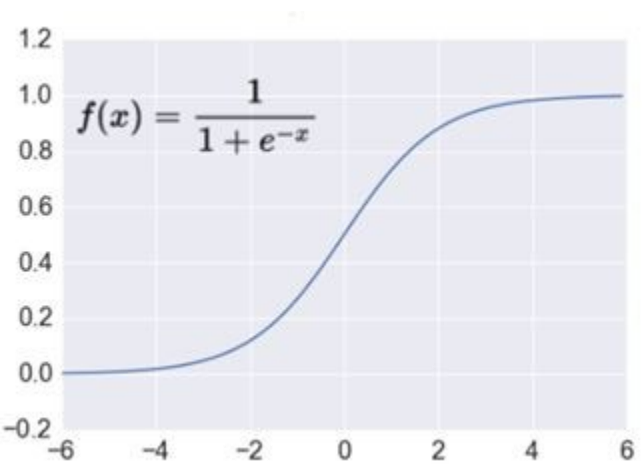
- Deep learning has an **inbuilt automatic multi stage feature learning process** that learns rich hierarchical representations (i.e. features).



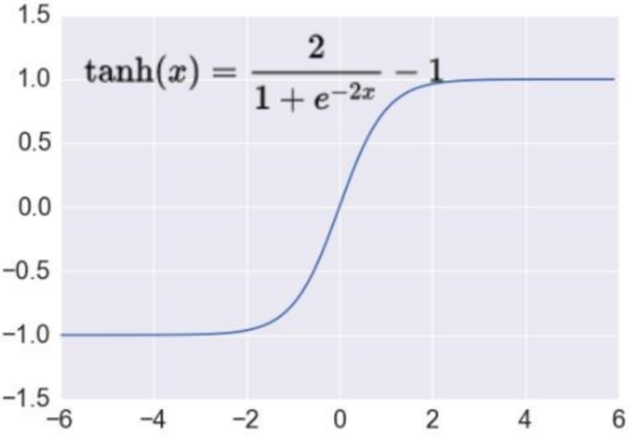
LAYERS USED TO BUILD CONVNETS

- A simple ConvNet is a sequence of layers.
- Every layer of a ConvNet transforms one volume of activations to another through a differentiable function.
- Three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer.**
- Stack these layers to form a full ConvNet **architecture**.

Activation Functions



- **Sigmoid** neurons **saturate** and **kill gradients**
- If initial weights are too large then most neurons would saturate

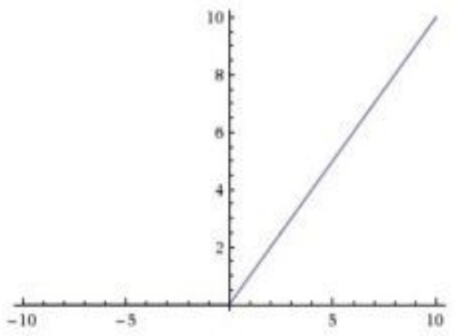


- Like Sigmoid, **tanh** neurons saturate.
- Unlike Sigmoid, **tanh** neurons zero centered and scaled sigmoid.

Non Linearity (ReLU)

- ReLU is a non-linear operation

Output = Max(zero, Input)

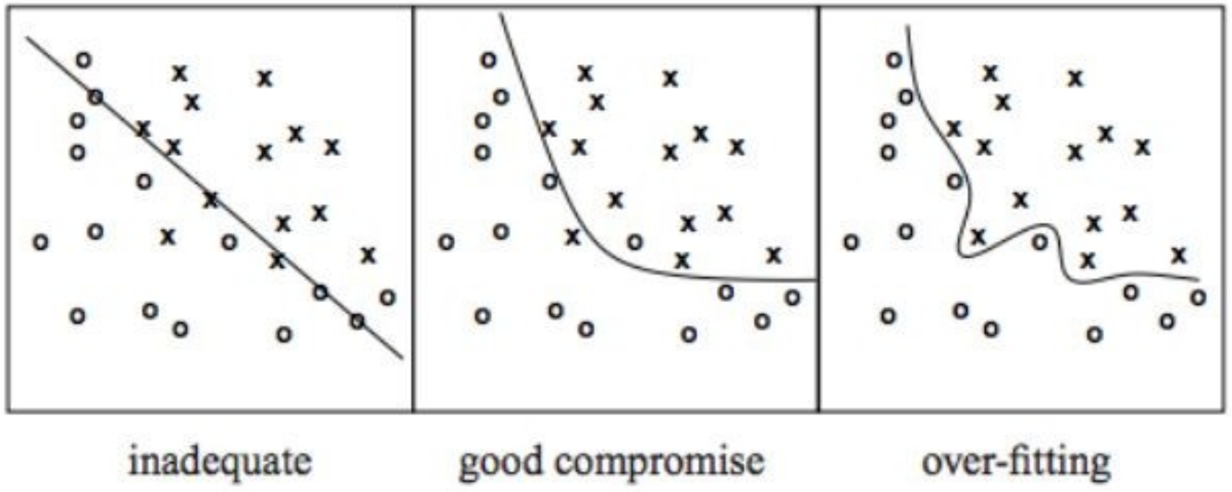


- ReLU is an element wise operation appliedper pixel and replaces all negative pixel values in the feature map by zero.

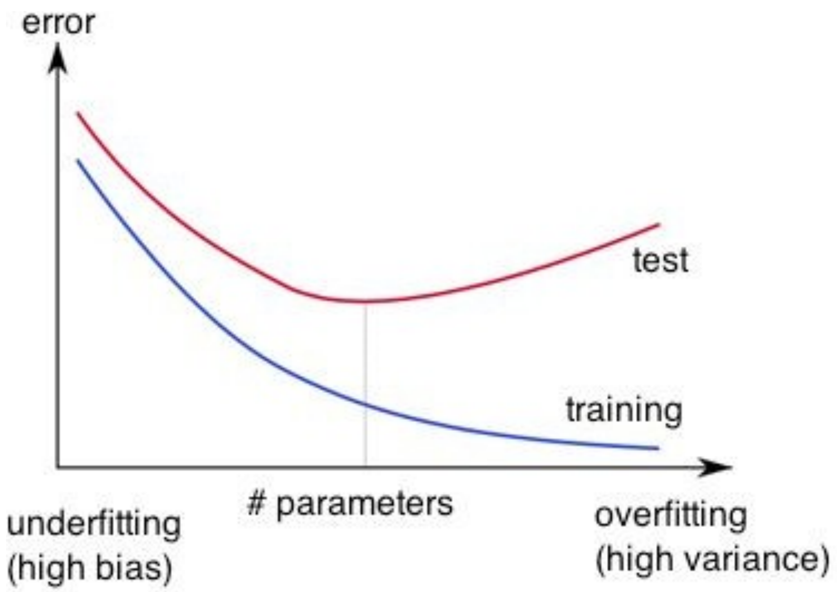
🙂 • Trains much faster due to linear, nonsaturating form.

□ Prevents the **gradient vanishing problem**

Overfitting



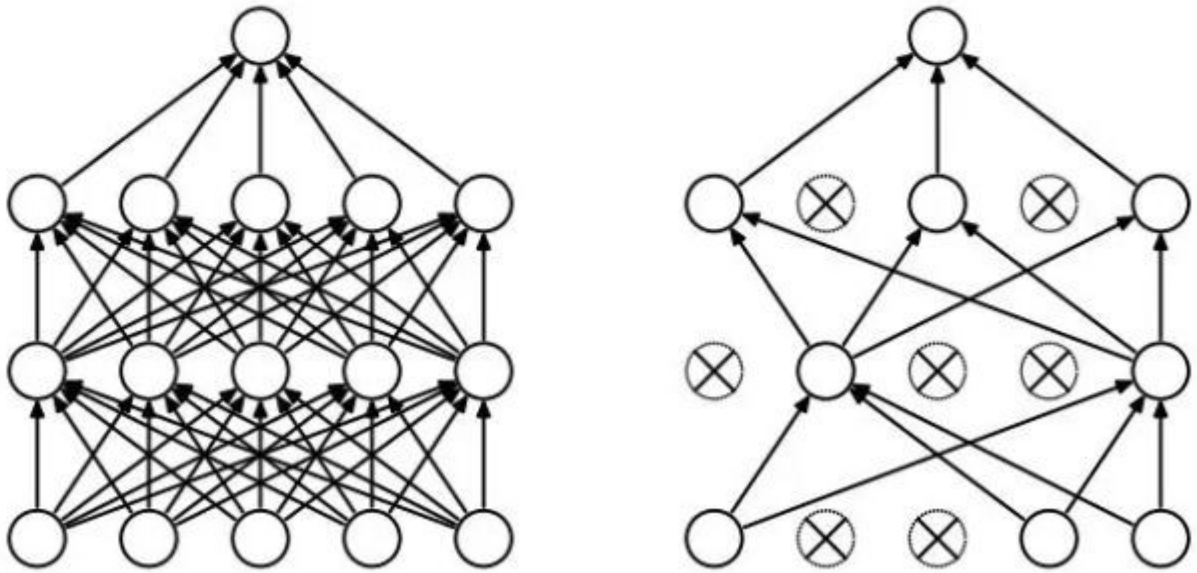
http://wiki.bethanycrane.com/overfitting-of-data



https://www.neuraldesigner.com/images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples (test data)

Regularization



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability p, independent of other units
- Hyper-parameter p to be chosen (tuned)

L2 = weight decay

- Regularization term that penalizes big weights, added to the objective $$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$
- Weight decay value determines how dominant regularization is during gradient computation.
- Big weight decay coefficient big penalty for big weights

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs
- n is called patience

Loss functions and output

Classification

Regression

Training examples

R_n x {class_1, ..., class_n} (one-hot encoding)

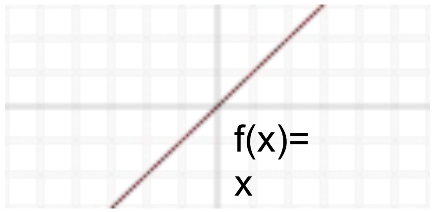
R^n x R^m

Output Layer

Soft-max
[map R^n to a probability

$$P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Linear (Identity) or Sigmoid



Cost (loss) function

Cross-entropy

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)}\log\hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$$

Mean Squared Error

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}-\hat{y}^{(i)}\right)^2$$

Mean Absolute Error

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)}-\hat{y}^{(i)}\right|$$

List of loss functions

How can we Train Deep Networks?

- Deep networks not performing better than shallow networks using stochastic gradient descent by backpropagation.

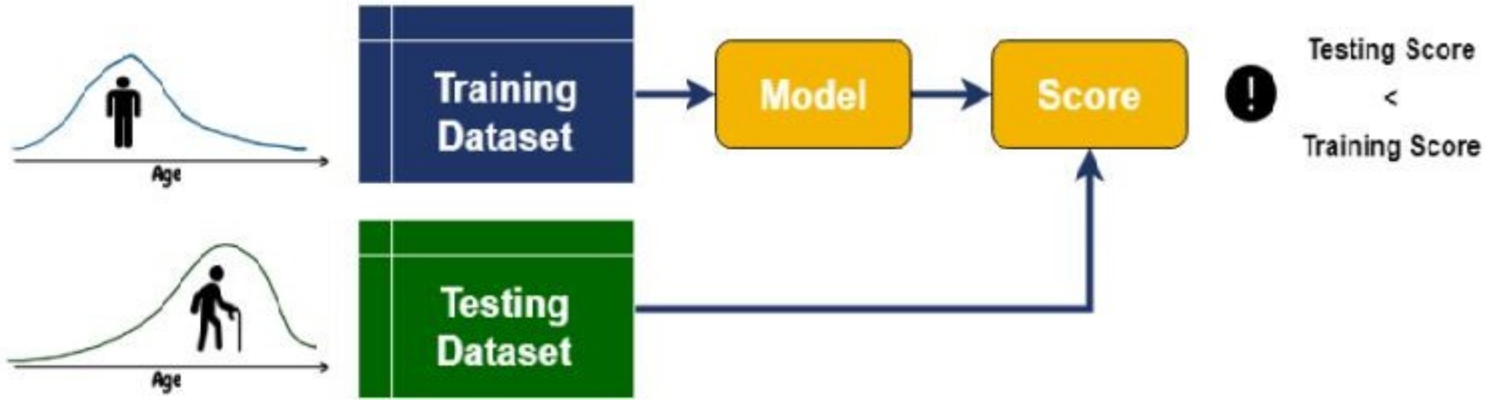
- Different layers in deep network are learning at vastly different speeds.

- When later layers in the network are learning well, early layers often get stuck during training, learning almost nothing at all.

- There's an intrinsic instability associated to learning by gradient descent in deep, many-layer neural networks resulting stuck at during training either at the early or the later layers.

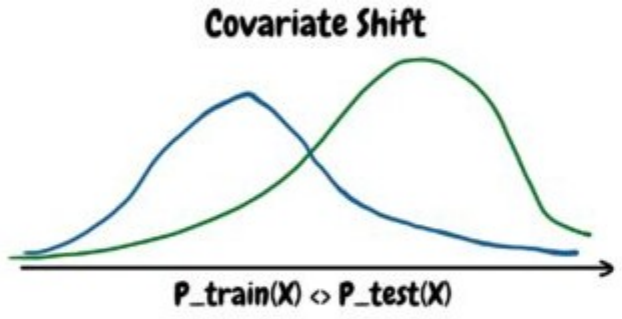
Dataset Shift

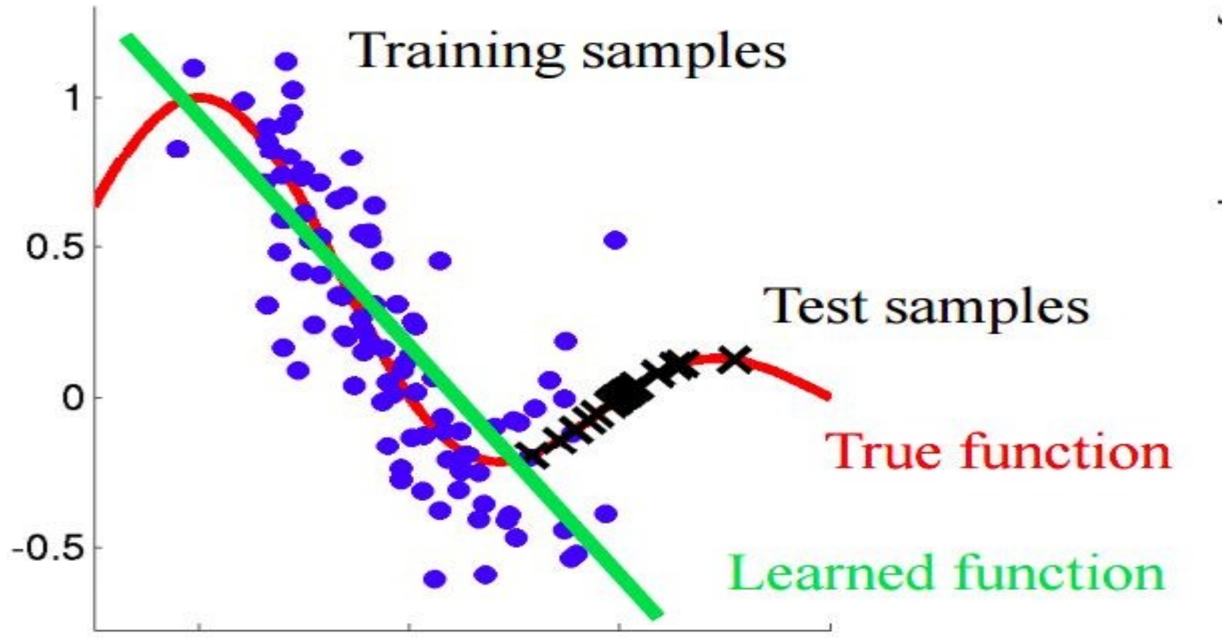
- In real world problems, often train and test datasets have not been generated by the same distribution.
- This phenomenon has an adversarial effect on the quality of a machine learning model, called **dataset shift** or **drift.**



Covariate Shift

- Covariate shift is the change in the distribution of the covariates specifically, that is, the independent variables.





Covariate shift

- Mathematically, covariate shift occurs if $p_{train}(X) \neq p_{test}(X)$ where X is a feature.

Say, an algorithm learned some X to Y mapping, now the distribution of X changes, so we might need to retrain the learning algorithm by trying to align the distribution of X with the distribution of Y.

- However, the notion of covariate shift can be extended beyond the learning system as a whole, but to apply to its parts, such as a sub-network or a layer.

Stochastic gradient descent(SGD)

- Stochastic gradient descent (SGD) has proved an effective way of training neural networks.
- SGD optimizes the parameters θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N}\sum_{i=1}^{N} \ell(x_i, \Theta)$$

•With SGD the training proceeds in steps and each step we consider a *mini batch* of size *m*, *m* << N.

•The mini batch is used to approximate the gradient of the loss function w.r.t. the parameters by computing

$$\frac{1}{m}\frac{\partial \ell(x_i, \Theta)}{\partial \Theta}.$$

Stochastic Gradient

- Stochastic gradient is simple and effective, but it requires careful tuning of the model hyper-parameters.
- Specifically the learning rate and the initial values for the model parameters.
- The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers .
- Small changes to the network parameters amplify

as the network becomes deeper.

- The change in the distributions of layers' inputs creates problem because the layers need to continuously adapt to the new distribution.

- A network computing $\ell = F2(F1(u,\Theta_1), \Theta_2)$ where F1 and F2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ.

- Learning Θ_2 can be viewed as if the inputs $x = F1(u, \Theta_1)$ are fed into the sub-network $\ell = F2(x, \Theta_2)$

$$\Theta_2 \leftarrow \Theta_2 - \frac{\propto}{m} \sum_{i=1}^{m} \frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2}$$

- A gradient descent step

for batch size m and learning rate α is exactly equivalent to that for a stand-alone network F2 with input x.

- Therefore for training more efficiently – must having the same distribution between the training and test data – apply to training the sub-network as well.

Internal Covariate Shift

- Fixed distribution of inputs to a sub-network would have positive consequences for the layers *outside the subnetwork,* as well.
- The change in the distributions of internal nodes of a deep network, in the course of training, called *Internal Covariate Shift.*
- Eliminating it offers a promise of faster training.
- A new mechanism, call Batch Normalization reduces internal covariate shift, which accelerates the training of deep neural nets.
- The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training.

Batch Normalization

- We normalize the input layer when feature have values wide spread and it speed up learning.
- We do the same thing for the values in the hidden layers, and get 10 times or more improvement in the training speed.
- Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift).

Batch Normalization

- The distribution of each layer’s inputs changes during training, as the parameters of the previous layers change.

- In a neural network, batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs. Zero means, unit variances.

- The network becomes more robust to different initialization schemes and learning rates.

Batch Normalization

- Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch.

- Standardizing the activations of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically.

- This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Implementation

- During training the mean and standard deviation of each input variable to a layer is calculated per mini-batch and used to perform the standardization.
- Allow the layer to learn two new parameters, namely a new mean and standard deviation, Beta and Gamma respectively.
- It allows the automatic scaling and shifting of the standardized layer inputs.
- These parameters are learned along with the original model parameters, as part of the training process and restore the representation power of the network.

Implementation

- Batch normalization may be used on the inputs to the layer before or after the activation function in the previous layer.
- It may be more appropriate **after** the activation function if for s-shaped functions like the hyperbolic tangent and logistic function.
- It may be appropriate **before** the activation function for activations that may result in non-Gaussian distributions like the rectified linear activation function.

Issues

- If the mean and standard deviations for each input feature are calculated over the mini-batch instead then the batch size must be sufficiently representative of the range of each variable.
- In a batch-normalized model, we have been able to achieve a training speedup from higher learning rates, with no ill side effects.
- Batch normalization can make training deep networks less sensitive to the choice of weight initialization method.
- Section – 8.7.1 Batch Normalization, Deep Learning, 2016.

Batch normalization - Summary

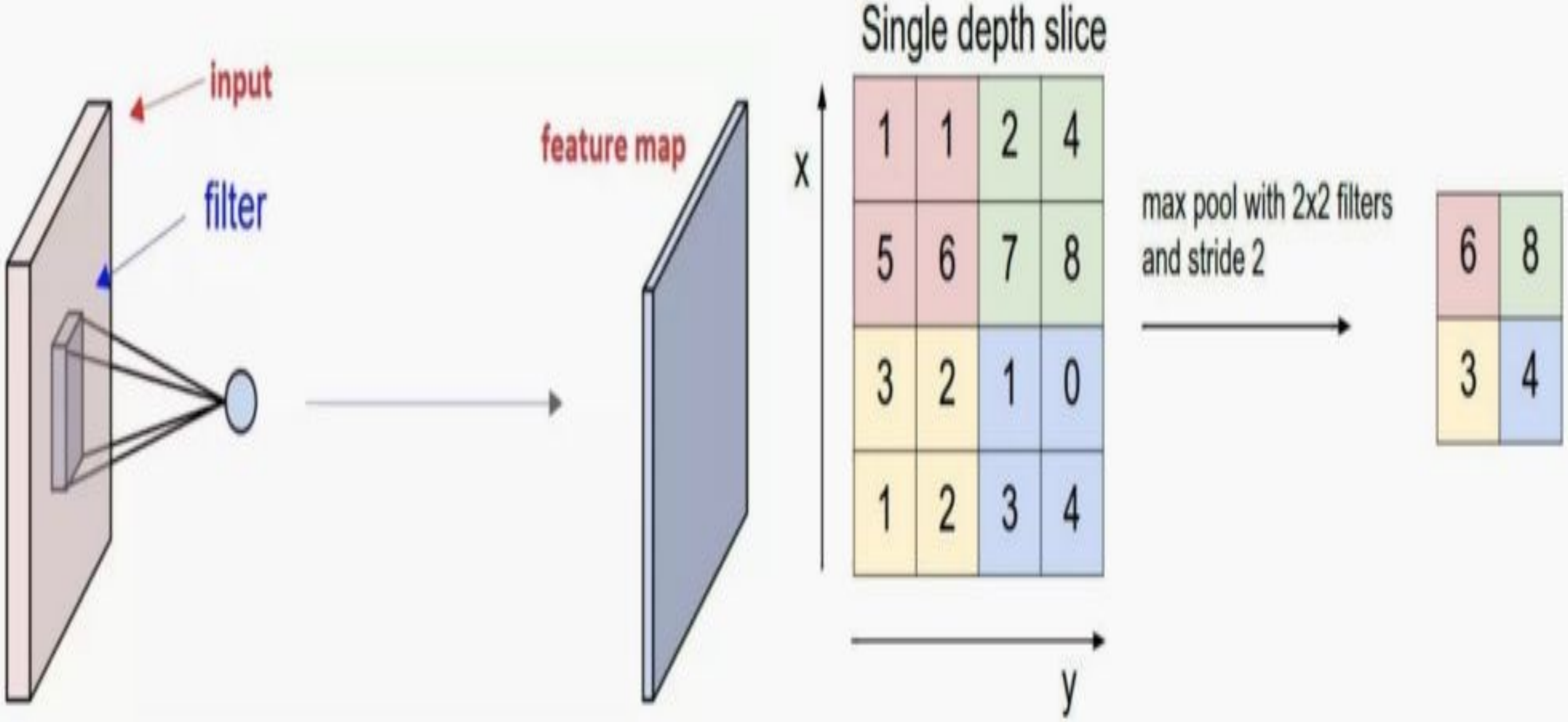
- Batch normalization allows each layer of a network to learn by itself, independently of other layers.
- Use higher learning rates because batch normalization makes sure that there’s no activation that’s gone really high or really low.
- It reduces overfitting because it has a slight regularization effects.
- Similar to dropout, it adds some noise to each hidden layer’s activations.
- Therefore, if we use batch normalization, we will use less dropout or no drop out which is a good thing because we are not going to lose a lot of information.

CNN REVIEW

- The motivation of using fully connected networks for image analysis with following benefits:

- Fewer parameters (weights and biases)
- Invariant to object translation
- Capable of generalizing and learning features.

- Convolutional layers are formed by **filters, feature maps, and activation functions.**


input
filter
feature map
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters and stride 2
6 8
3 4

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}:	number of input features
n_{out}:	number of output features
k:	convolution kernel size
p:	convolution padding size
s:	convolution stride size

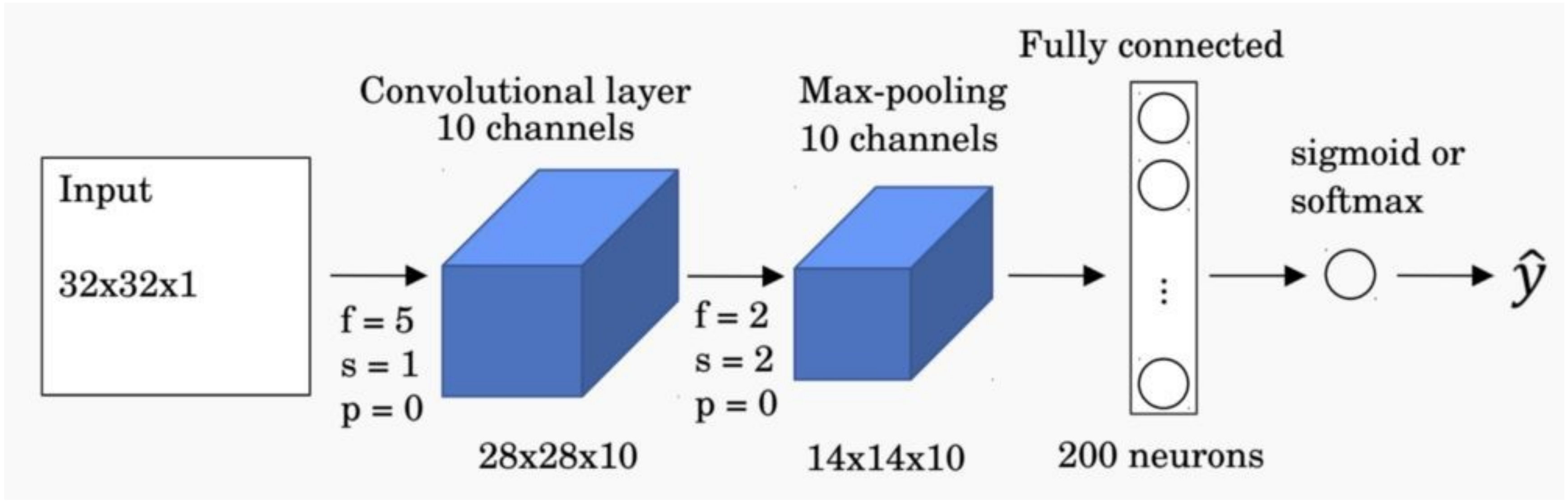
We can determine the number of output layers of a given convolutional block using number of layers in the input is n_i, the number of filters in that stage, f, the size of the stride, s, and the pixel dimension of the image, p(assuming it is square)

Layers and Features

- Pooling layers are used to reduce overfitting.
- Fully connected layers are used to mix spacial and channel features together.
- Each of the filter layers corresponds to the image aftera feature map has been drawn across the image.
- which is how features are extracted.

Weights

- It is important to know the number of input and output layers as this determines the number of weights and biases that make up the parameters of the neural network.
- The more parameters in the network, the more parameters need to be trained which results in longer training time.
- Training time is very important for deep learning as it a limiting factor unless you have access to powerful computing resources such as a computing cluster.



(i) 250 weights on the convolutional filter and 10 bias terms.

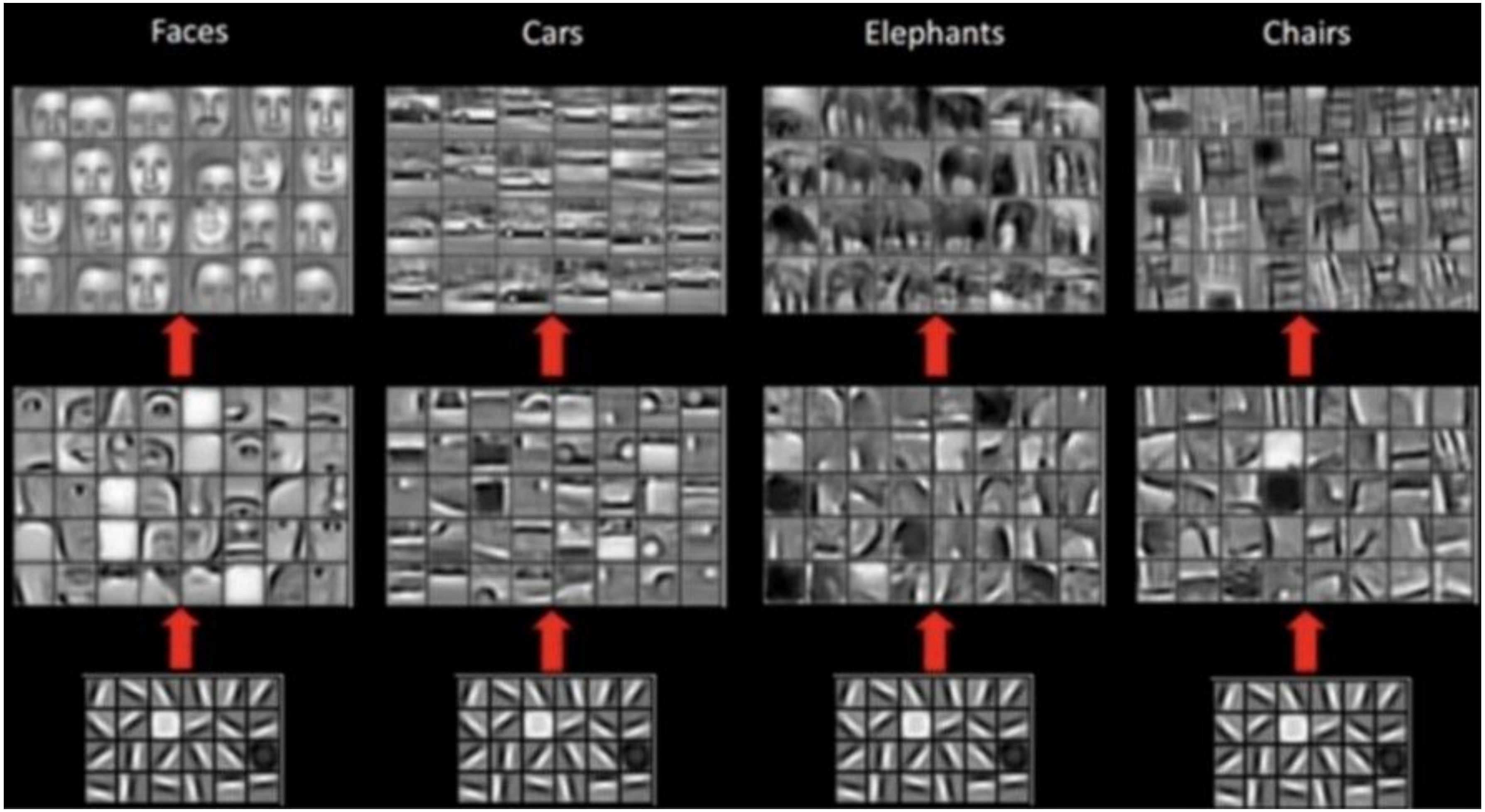
(ii) $13 \times 13 \times 10 = 1{,}690$ output elements after the max-pooling layer.

(iii) 200 node fully connected layer, which results in a total of $1{,}690 \times 200 = 338{,}000$ weights and 200 bias terms in the fully connected layer.

(iv) 338,460 parameters to be trained in the network. Majority of the trained parameters occur at the fully connected output layer.

Layer wise Complexity

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters such as edges and corners.
- The middle layers learn filters that detect parts of objects — for faces, they might learn to respond to eyes and noses.
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.

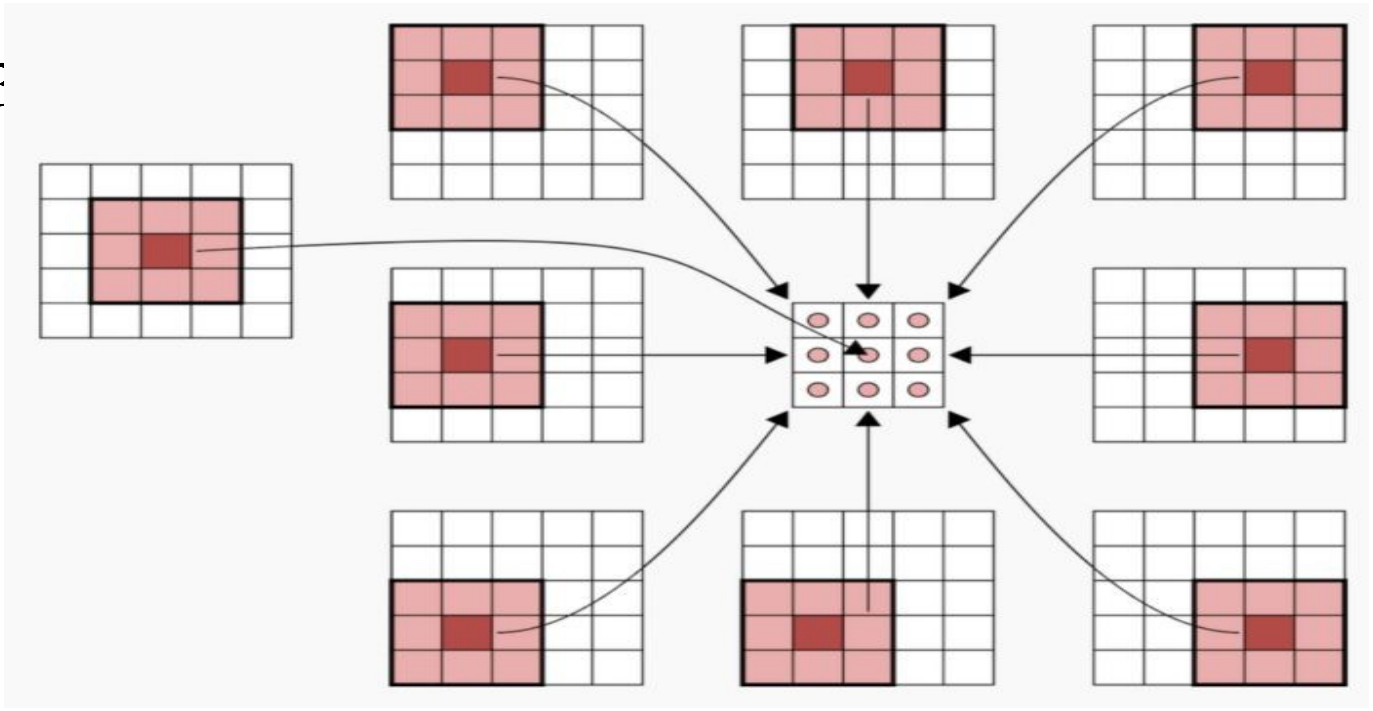


Feature maps showing increasing resolution of features through different convolutional layers of a neural network.

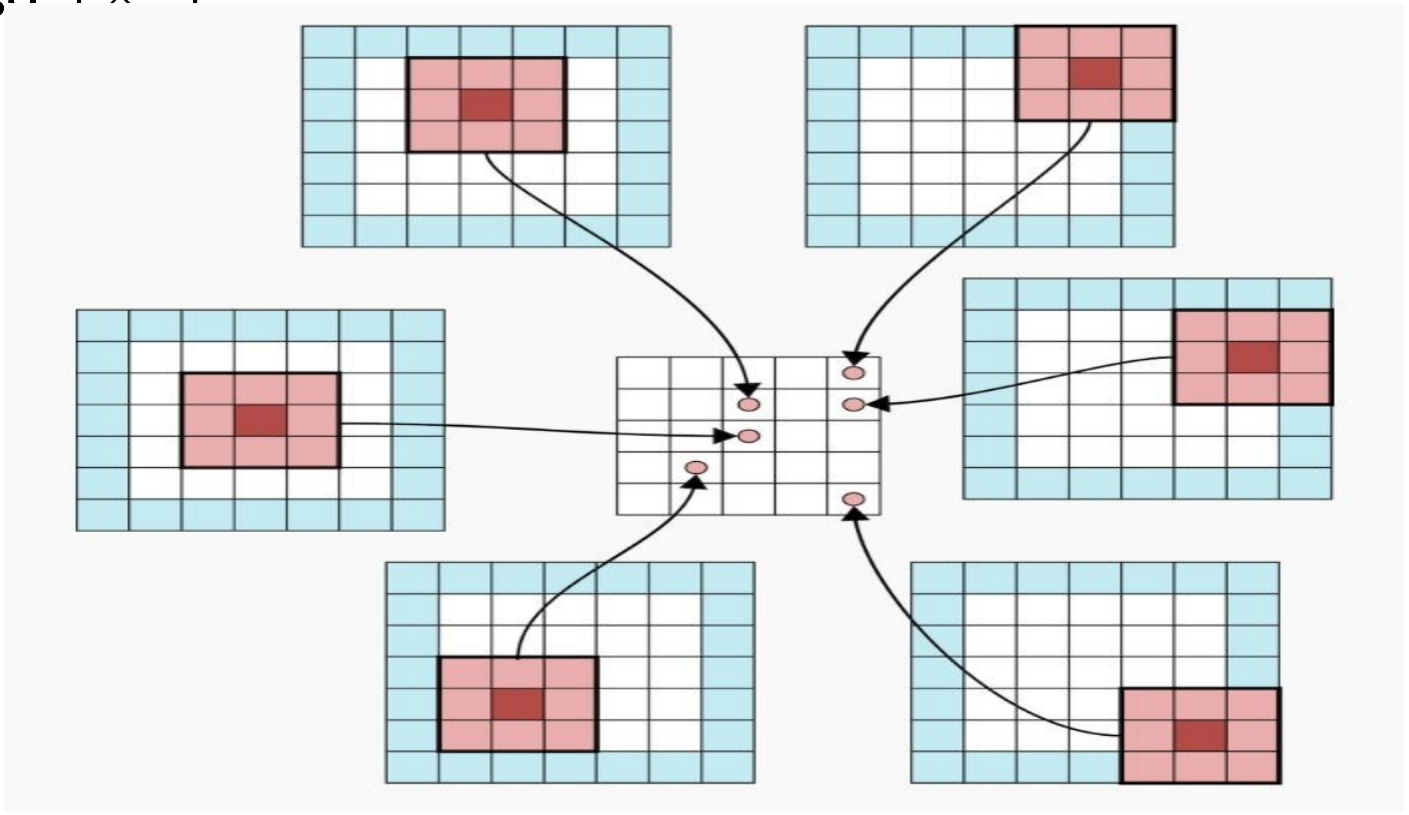
Transposed Convolution

- Upsampling of a convolution step, it is called transposed convolution or fractional striding.

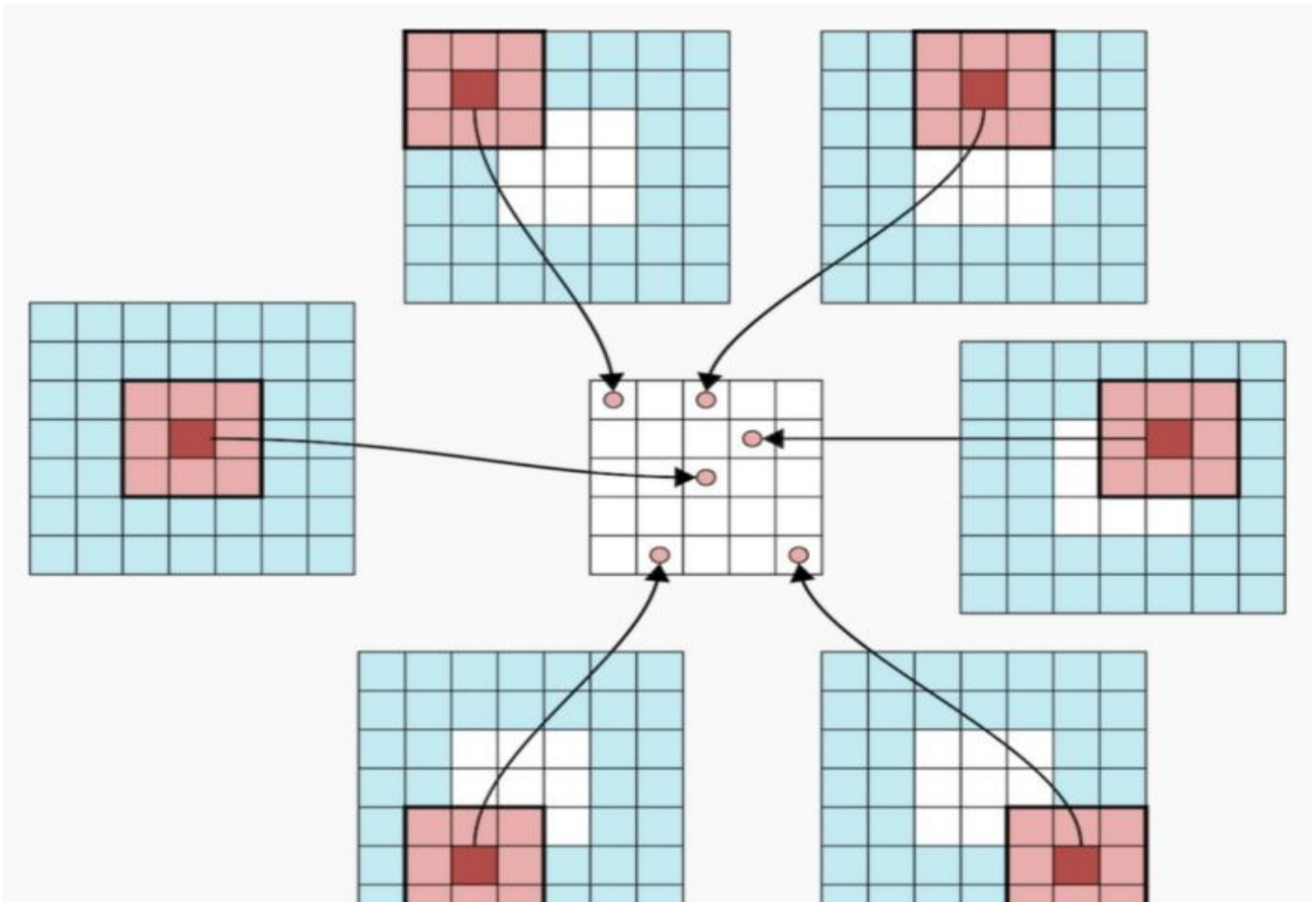
- A typical convolutional layer with no padding, acting on an image of size 5 × 5. After the convolution, we end up with a 3 × 3



- A convolutional layer with a padding of 1. The original image is 5 × 5, and the output image after the convolution is also 5 × 5



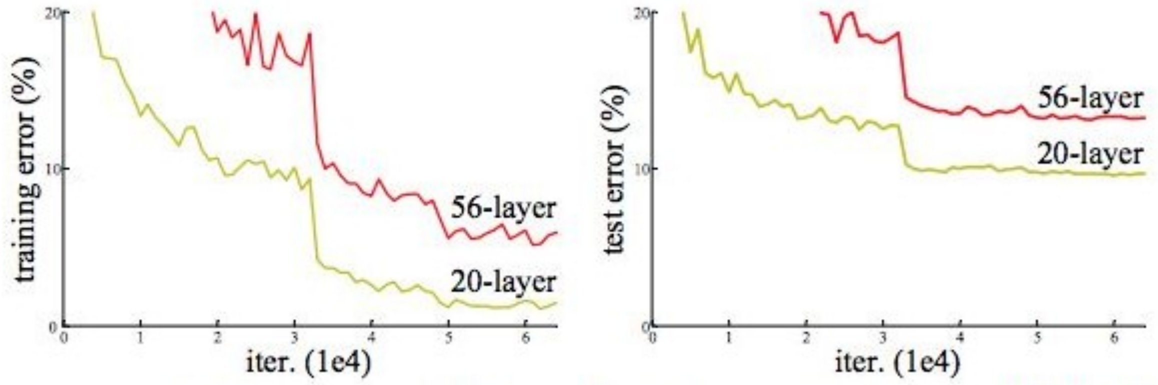
- A padding of 2, original image 3× 3, and the output image after convolution 5X5
- In the development of a variational autoencoder, these are implemented using an upsampling layer.



Deep Neural Networks

- Classical CNNs do not perform well as the depth of the network grows past a certain threshold.

- There is a maximum threshold for depth with the traditional CNN model.

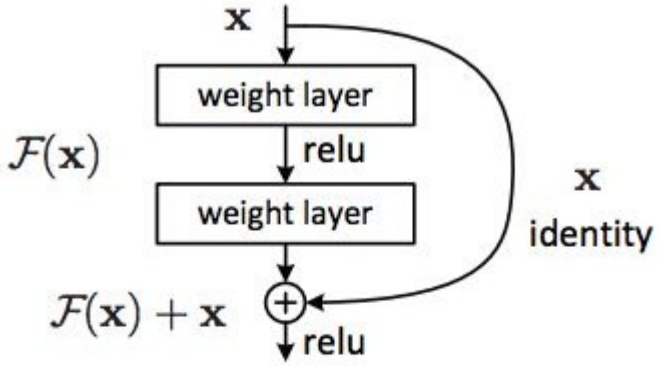


Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error.

The Residual Block NN

- The failure of the 56-layer CNN could for the optimization function, parameter initialization, or vanishing/exploding gradient problem.

- The problem of training very deep networks has been alleviated with the introduction of a new neural network layer
— **The Residual Block.**



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Residual learning: a building block.

Skip Connection and Identity mapping

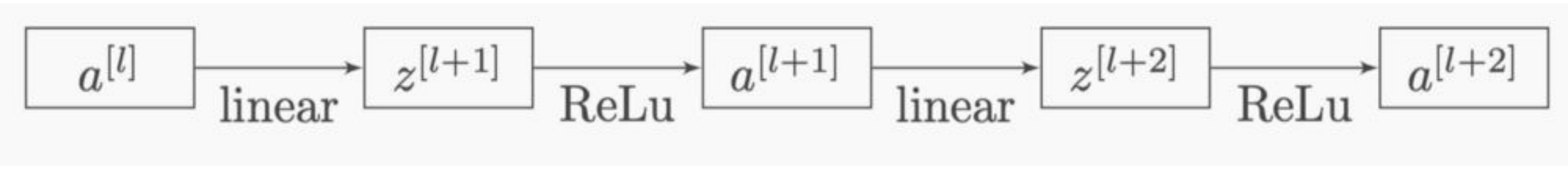
- Identity mapping does not have any parameters and is there to add the output from the previous layer to the layer ahead.
- However, x and F(x) will not have the same dimension, because a convolution operation typically shrinks the spatial resolution of an image.
- The identity mapping is multiplied by a linear projection W for the input x and F(x) to be combined as input to the next layer.

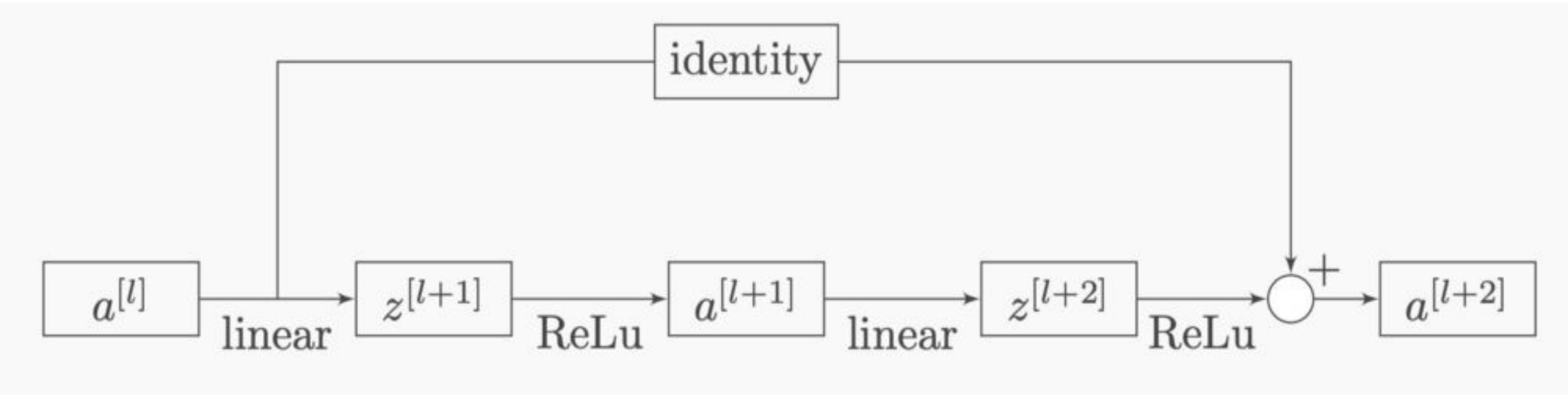
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

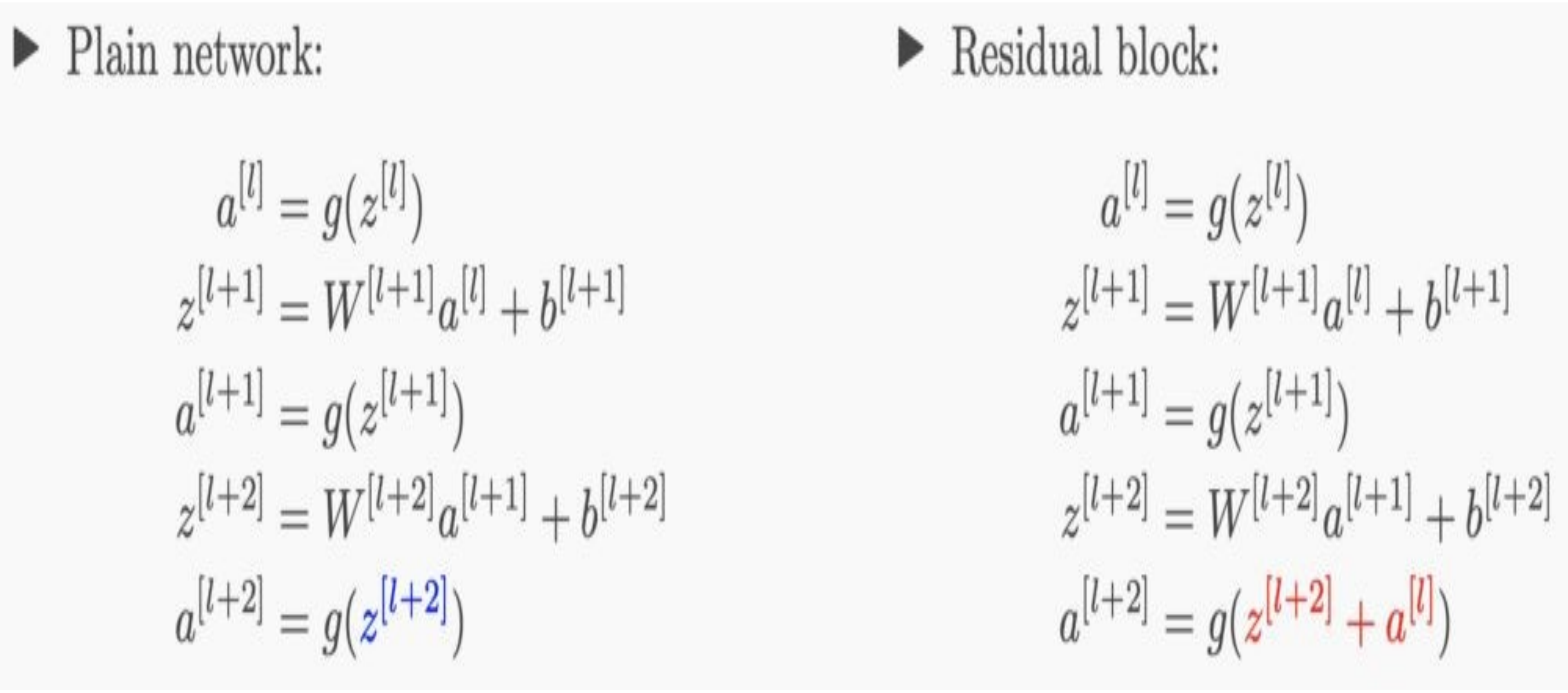
- Ws term can be implemented with 1×1 convolutions,
introducing additional parameters to the model.

Residual Networks

- The main idea behind this network is the residual block.
- The network can contain 100 layers or more.







- Plain network:

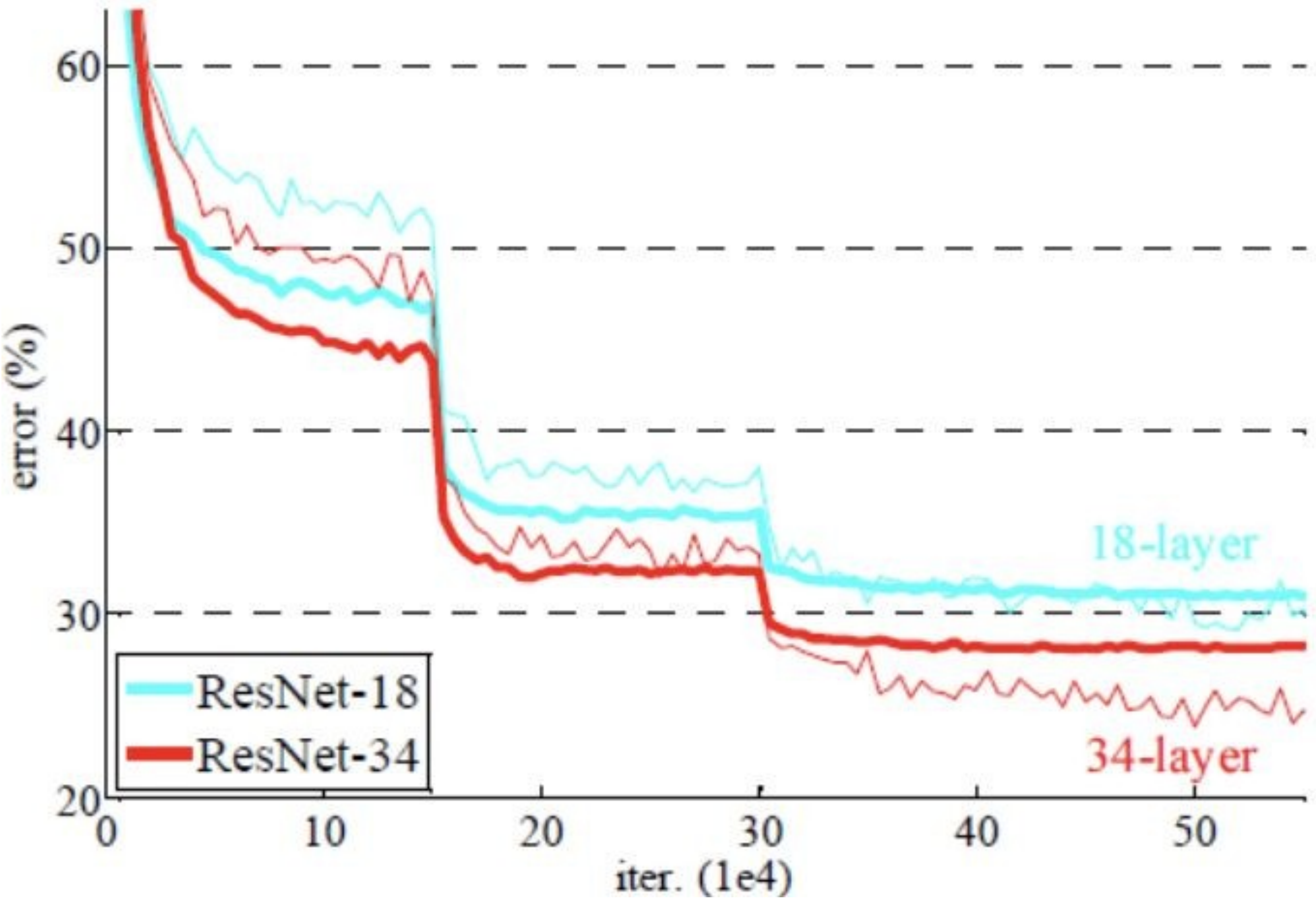
$$\begin{aligned} a^{[l]} &= g(z^{[l]}) \\ z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\ a^{[l+1]} &= g(z^{[l+1]}) \\ z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\ a^{[l+2]} &= g(z^{[l+2]}) \end{aligned}$$

- Residual block:

$$\begin{aligned} a^{[l]} &= g(z^{[l]}) \\ z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\ a^{[l+1]} &= g(z^{[l+1]}) \\ z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\ a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) \end{aligned}$$

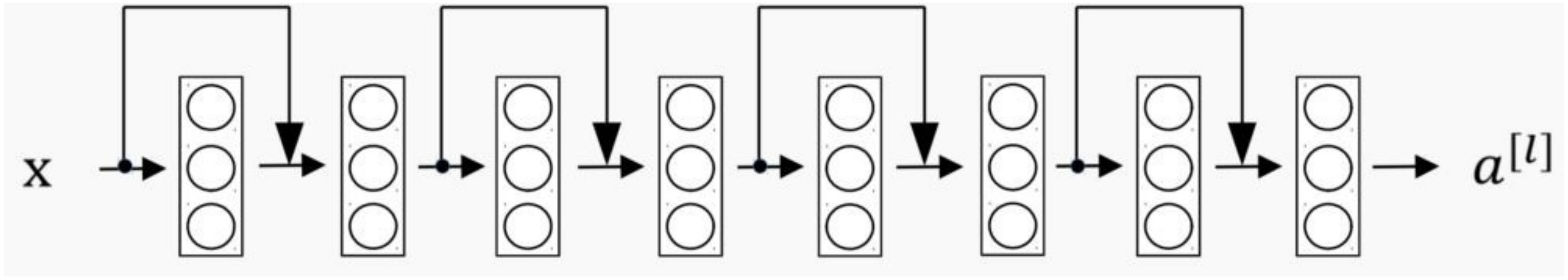
- With this extra connection, gradients can travel backward more easily.

- It becomes a flexible block that can expand the capacity of the network, or simply transform into an identity function that would not affect training

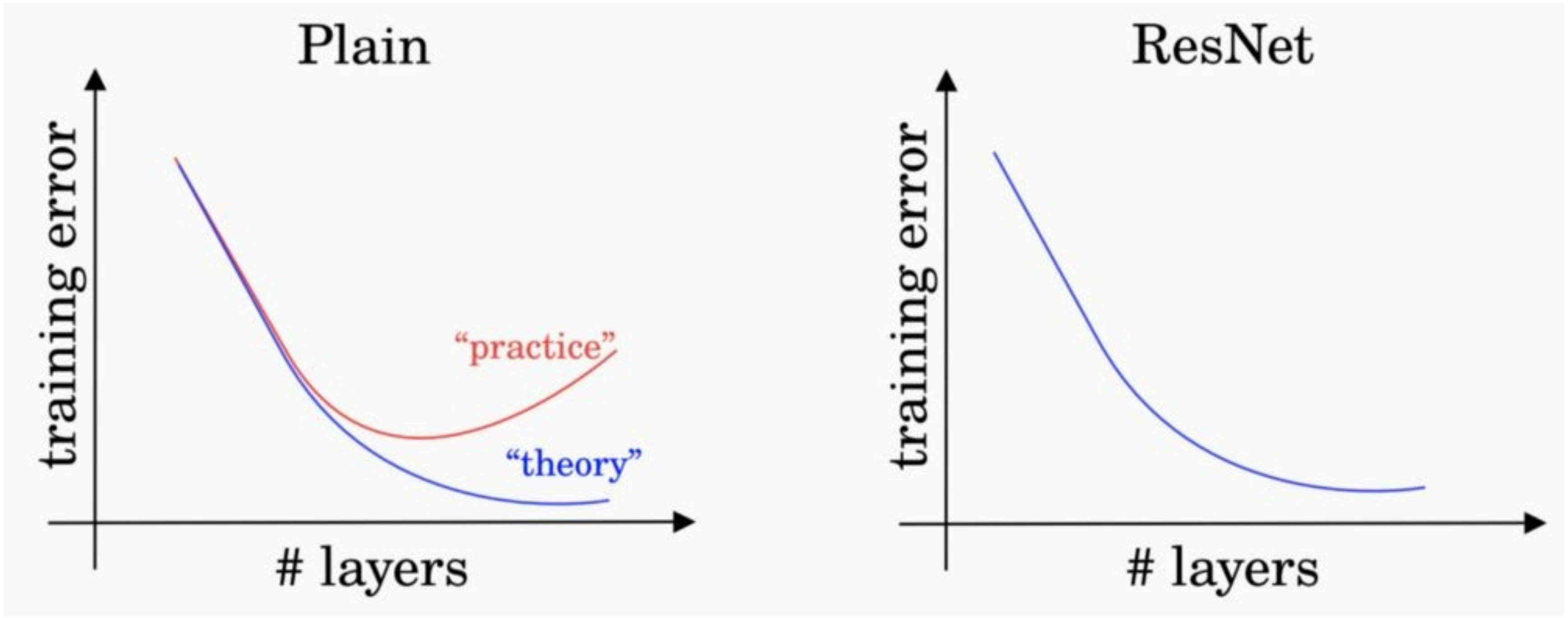


error (%)
60
50
40
30
20
0
10
20
30
40
50
iter. (1e4)
18-layer
34-layer
ResNet-18
ResNet-34

A residual network stacks residual blocks sequentially



- The idea is to allow the network to become deeper without increasing the training complexity.


Plain
training error
"practice"
"theory"
# layers
ResNet
training error
# layers

Reference

- Kaiming He, et al. in their 2015 paper titled "Deep Residual Learning for Image Recognition" used batch normalization after the convolutional layers in their very deep model referred to as ResNet and achieve then state-of-the-art results on the ImageNet dataset, a standard photo classification task.