
# AI COURSE

**Thanh Ha DO**

AI Faculty
Posts and Telecommumications Institute of Technology

August 8, 2025

# OUTLINE

# Ensemble Methods

## Ensemble Algorithms

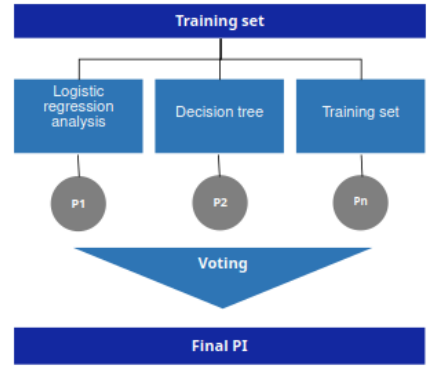
### Definition

A strong predictive model based on the weaker learner

- **Voting**
  - An ensemble of different kinds of learners, for example, Tree, KNN, SVM, etc
  - One identical training set
- **Bagging type**
  - An ensemble of different kinds of learners: Random Forest
  - Different training set
- **Boosting type**
  - A series of weak learners that adaptatively learn and predict, for example, AdaBoost, GBM, XGBoost, etc.
  - The series is grown by adding new learners multiplied by the boosting weights

# Ensemble Methods
## Voting

- Can be applied to classification and regression
- Two voting methods for the classifier
  - **Hard**: The predicted class label is decided by the majority rule voting
  - **Soft**: The predicted class label is decided by the **arg max** of the sum of the predicted probabilities



# Ensemble Methods

## Voting

- **Hard Voting**: Suppose the predictive values for classification are 1,0,0,1,1
  - 1 has three votes and 0 has 2 votes.
  - $\rightarrow$ 1 becomes the final predictive value
- **Soft Voting**: Soft voting method calculates the average value of each probability and then determines the one with the highest probability.
  - Suppose the probabilities of getting class 0 and 1 are (0.4, 0.9, 0.9, 0.4, 0.4), and (0.6, 0.1, 0.1, 0.6, 0.6), respectively.
  - The final probability of getting class 0 is (0.4+0.9+0.9+0.4+0.4) / 5 = 0.6; The final probability of getting class 1 is (0.6+0.1+0.1+0.6+0.6) / 5 = 0.4.
  - $\rightarrow$ 0 becomes the final predictive value.

In general, using the soft voting method is considered more reasonable than the hard voting method in competitions because the soft voting method provides a much better actual performance result.

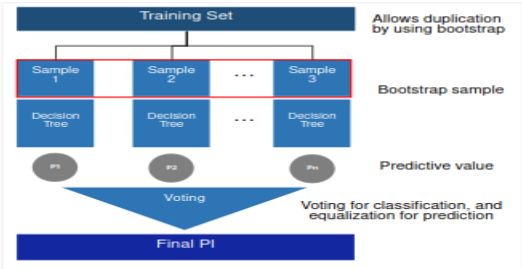
# Ensemble Methods

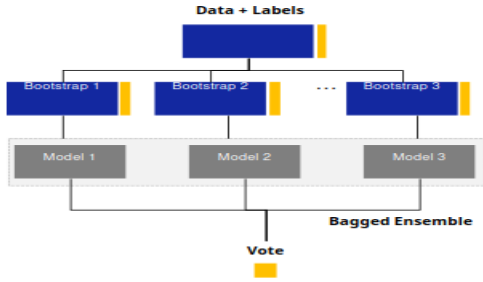
## Bagging & Random Forest

- Randomly chosen trees form a forest
- Prediction is decided by the majority vote
  1. Make trees with randomly selected variables and observations.
  2. Keep only those with the lowest Gini impurity (or entropy).
  3. Repeat step 1) a given number of times.
  4. Using the trees gathered during the training step, we can make predictions by majority vote.

# Ensemble Methods

## Bagging & Random Forest




- It creates **multiple decision trees** and performs sampling of different datasets while allowing **overlapped datasets**.
- If the dataset consists of **[1, 2, 3, 4, 5]: Group 1 = [1, 2, 3]; Group 2 = [1, 3, 4]; Group 3 = [2, 3, 5]** the classification problem, voting is done
- In regression problems, the average of each obtained value is calculated.

# Ensemble Methods

## Boosting Ensemble: AdaBoost

- A sequence of weak learners such as tree
- A **weight** ensemble of weak learners
  - One set of weights that increase the importance of the better-performing learners
  - One set of weights that increase the importance of the wrongly classified observations

# Ensemble Methods

## AdaBoost Classification Algorithm

- Lets suppose $n$ observations for the training step: $\mathbf{x}_i$ and $y_i$ that $y_i \in \{-1, 1\}$
- Set of weak learners $G_m(\mathbf{x})$ with $m = 1, 2, ..., M$
- The ensemble classifier is made up of a linear combination of these weak learners:

$$G_{ensemble}(\mathbf{x}) = sign(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x}))$$

where $\alpha_m$ are the "boosting weights" that need to be calculated.

# Ensemble Methods
## Boosting

1. For the first step ($m = 1$), equal weight is assigned to the observations:
2. For the boost sequence $m = 1, ..., M$
   a) Train the learner $G_m(\mathbf{x})$ using observations weighted by $w_i^{(m)}$
   b) Calculate the error ratio

$$\epsilon_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G_m(\mathbf{x}_i)}{\sum_{i=1}^{n} w_i^{(m)}}$$

$\Longrightarrow I(y_i \neq G_m(\mathbf{x}_i)$ give 1 for an incorrect prediction, else 0
$\Longrightarrow 0 \leq \epsilon_m \leq 1$

# Ensemble Methods
## Boosting

c) For the boost sequence (con't) $m = 1, 2, ..., M$
- Calculate the boosting weight: $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
  - $\Longrightarrow$ As $\epsilon_m \to 0$, $\alpha_m$ is large positive number. The learner is given more importance
  - $\Longrightarrow$ As $\epsilon_m \approx 0.5$, $\alpha_m \approx 0$
  - $\Longrightarrow$ As $\epsilon_m \to 1$, $\alpha_m$ is a large negative number

d) For the next step, the weights of the wrongly predicted observation are rescaled by a factor $e^{\alpha_m}$. This can be compactly expressed as

$$w_i^{(m+1)} = w_i^{(m)} \times e^{\alpha_m \cdot I(y_i \neq G_m(\mathbf{x}_i))} \text{ where } i = 1, ..., n$$

$\longrightarrow$ In the next sequence step, the wrongly predicted observations receive heavier weight.

# Ensemble Methods
## Boosting

4. The ensemble classifier is made up of a linear combination of weak learners:

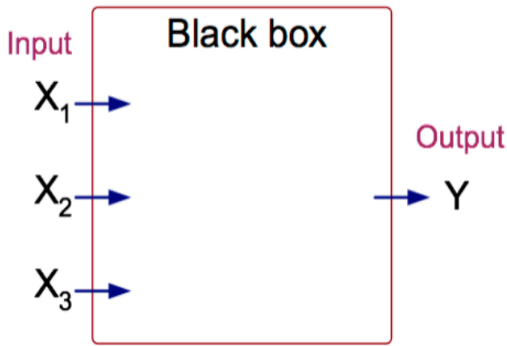
$$G_{ensemble}(\mathbf{x}) = sign(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x}))$$

5. For a new testing condition $\mathbf{x}'$, we can predict $y'$ by $G_{ensemble}(\mathbf{x}')$

For more: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`

# Artificial Neural Network (ANN)

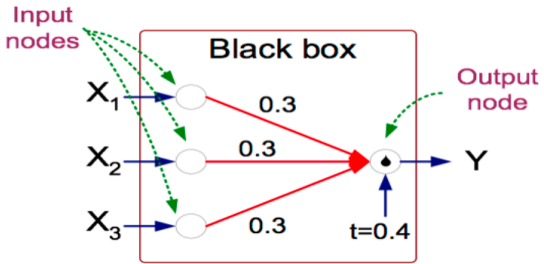
| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 1 | 0 | 0 | -1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | -1 |
| 0 | 1 | 0 | -1 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | -1 |




Output $Y$ is 1 if at least two of the three inputs are equal to 1

# Artificial Neural Network (ANN)

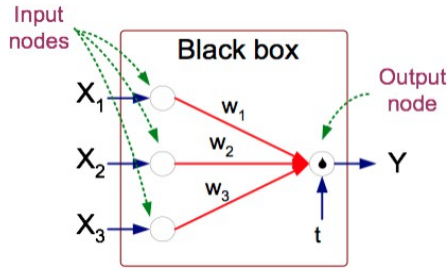
| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 1 | 0 | 0 | -1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | -1 |
| 0 | 1 | 0 | -1 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | -1 |




$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4) \text{ where } sign(x) = \begin{cases} 1 \text{ if } x \geq 0, \\ -1 \text{ if } x < 0 \end{cases}$$

# Artificial Neural Network (ANN)

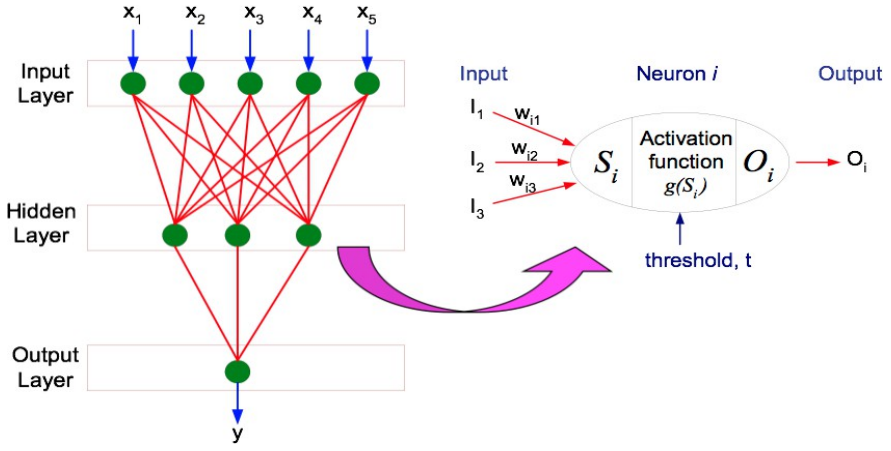
- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold $t$




**Perceptron Model**

$$Y = sign(\sum_{i=0}^{d} w_i X_i - t)$$

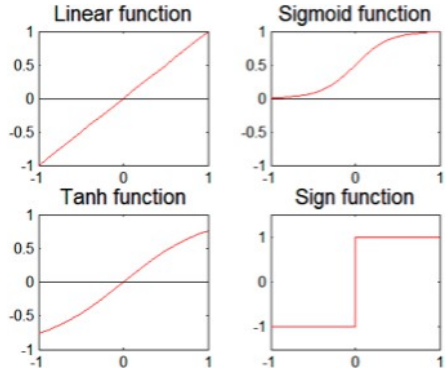
# General Structure of ANN




Training ANN means learning the weights of the neurons

# Artificial Neural Network (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi- layered network
  - Feed-forward versus recurrent network
- Various types of activation functions (f):



# Perceptron

- Single layer network: Contains only input and output nodes
- Activation function: $f = sign(w \cdot x)$
- Applying model is straightforward:

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4) \text{ where } sign(x) = \begin{cases} 1 \text{ if } x \geq 0, \\ -1 \text{ if } x < 0 \end{cases}$$

$\rightarrow X_1 = 1, X_2 = 0, X_3 = 1 \Longrightarrow sign(0.2) = 1$

# Perceptron Learning Rule

- Initialize the weights $(w_0, w_1, ..., w_d)$
- Repeat: for each training example $(x_i, y_i)$
  - Compute $f(w, x_i)$
  - Update the weights based on error, in which $\lambda$ is learning rate:

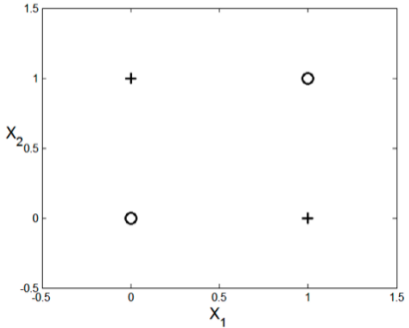
$$w^{(k+1)} = w^k + \lambda[y_i - f(w^k, x_i)]x_i$$

- Until stopping condition is met

# Perceptron Learning Rule

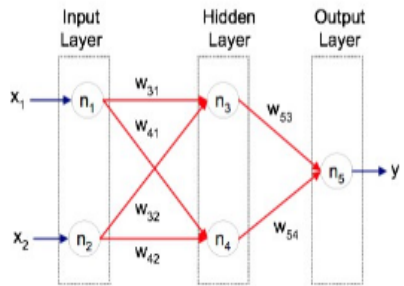
- Since $f(w, x)$ is a linear combination of input variables, decision boundary is linear
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly
- Example of Nonlinearly Separable Data

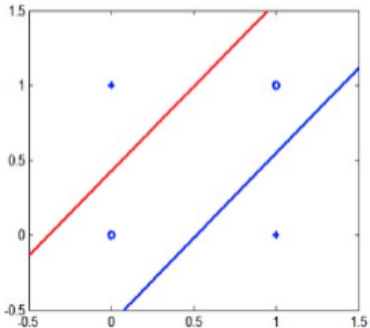
| $x_1$ | $x_2$ | y |
|---|---|---|
| 0 | 0 | -1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | -1 |



# Multilayer Neural Network

- Hidden layers: intermediary layers between input & output layers
- More general activation functions (sigmoid, linear, etc)
- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



# Multilayer Neural Network

Can we apply perceptron learning rule to each node, including hidden nodes?

- Perceptron learning rule computes error term $e = y - f(w, x)$ and updates weights accordingly
  - Problem: how to determine the true value of $y$ for hidden nodes?
- Approximate error in hidden nodes by error in the output nodes. However, there are problems:
  - Not clear how adjustment in the hidden nodes affect overall error
  - No guarantee of convergence to optimal solution

# Gradient Descent for Multilayer NN

- Weight update: $w_j^{(k+1)} = w_j^k - \lambda \frac{\partial E}{\partial w_j}$
- Error function:

$$E = \frac{1}{2} \sum_{i=1}^{N} (t_i - f(\sum_j w_j x_{ij}))$$

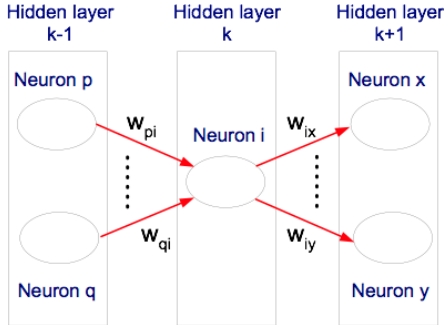
- Activation function $f$ must be differentiable
- For sigmoid function:

$$w_j^{(k+1)} = w_j^{(k)} - \lambda \sum_t (t_i - o_i) o_i (1 - o_i) x_{ij}$$

- Stochastic gradient descent (update the weight immediately)

# Gradient Descent for Multilayer NN

- For output neurons, weight update formula is the same as before (gradient descent for perceptron)




- For hidden neurons:

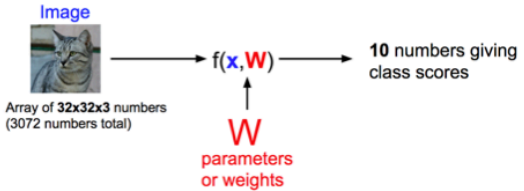
$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i(1 - o_i) \sum_{j \in \Phi_i} \sigma_j w_{ij} x_{pi}$$

- Output neurons: $\sigma_j = o_j(1 - o_j)(t_j - o_j)$
- Hidden neurons: $\sigma_j = o_j(1 - o_j) \sum_{k \notin \Phi_i} \sigma_k w_{jk}$

# Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large
- Gradient descent may converge to local minimum
- Model building can be very time consuming, but testing can be very fast
- Can handle redundant attributes because weights are automatically learnt
- Sensitive to noise in training data
- Difficult to handle missing attributes

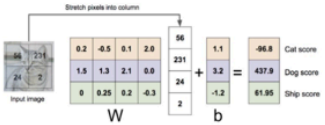
# Image classification with Linear Classifier




$$f(x,W) = Wx + b$$

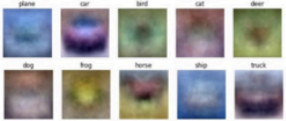
## Algebraic Viewpoint

$$f(x,W) = Wx$$



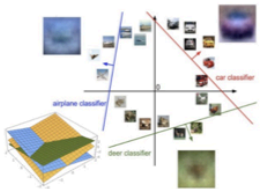

## Visual Viewpoint

One template per class



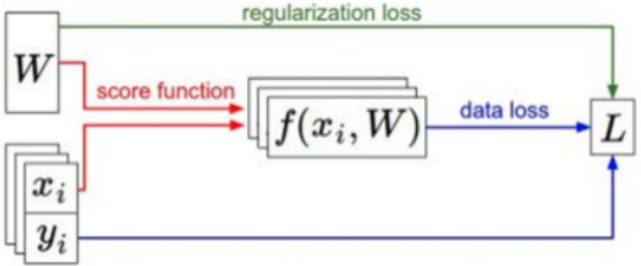

## Geometric Viewpoint

Hyperplanes cutting up space



# Revise: Loss function

- We have some dataset of $(\mathbf{x}, y)$
- We have a score function: $f(\mathbf{x}; W) = W\mathbf{x}$
- We have a loss function:
  - Softmax $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$
  - SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
  - Full loss: $L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$



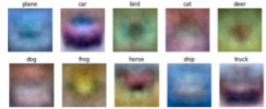
# Image Classification: A core task in Computer Vision

(assume given a set of labels)
{dog, cat, truck, plane, ...}

cat
dog
bird
deer
truck

f(x) = Wx

Class scores



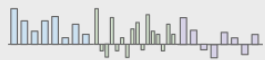

f(x) = Wx

Feature Representation

Class scores

# Image features vx. ConvNets

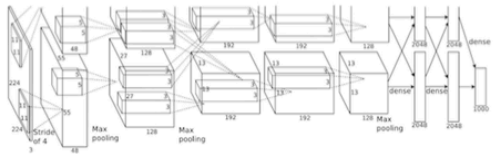
Feature Extraction



f

10 numbers giving scores for classes

training




Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

# Convolution Neural Networks

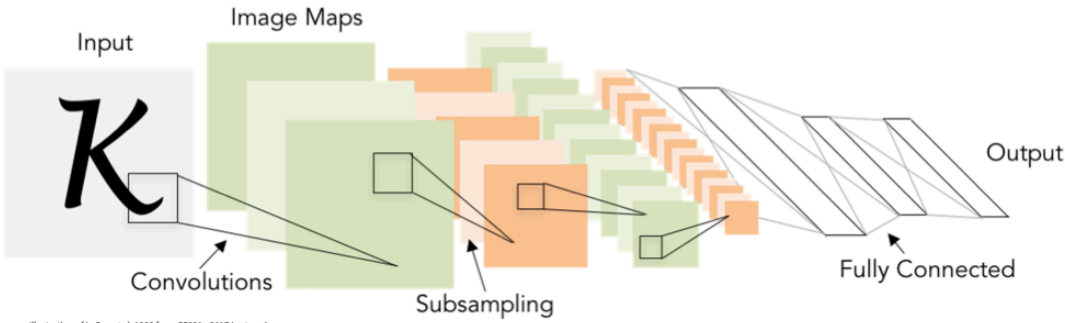



Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

# Deep Convolution Neural Network

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky, Sutskever, Hinton, 2012]

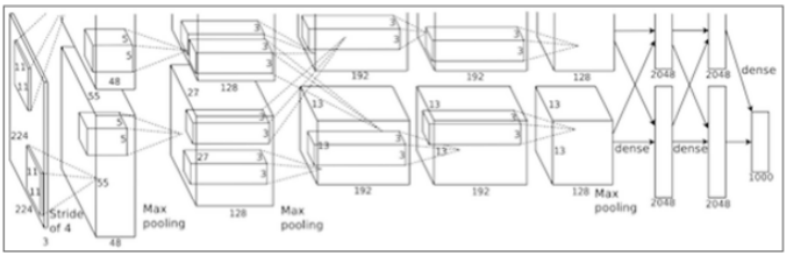
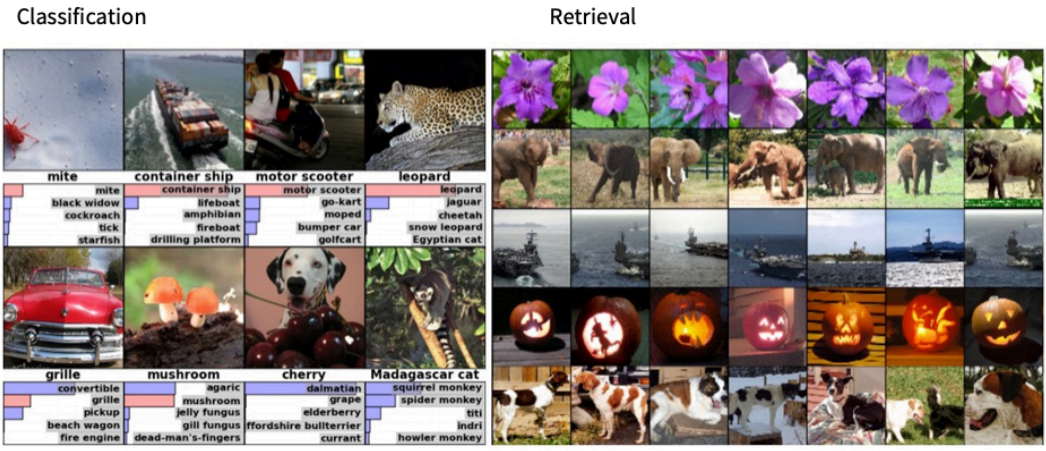




Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

# Fast-forward: ConvNets are everywhere



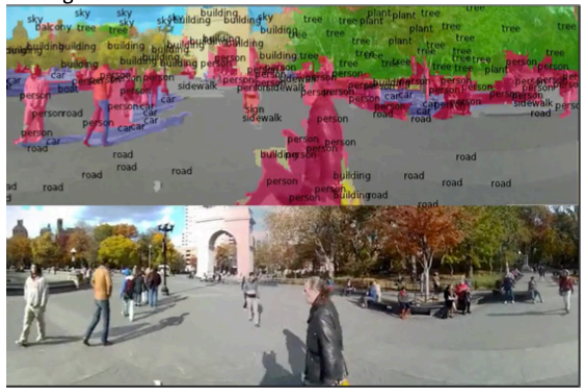
# Fast-forward: ConvNets are everywhere

Detection


Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

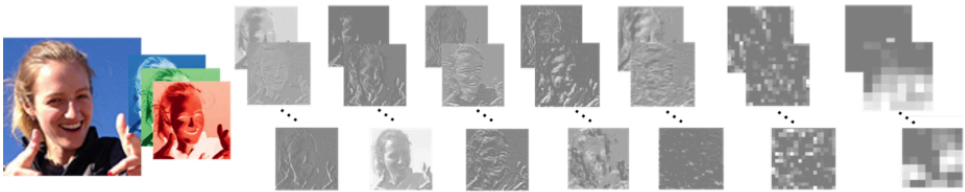
Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

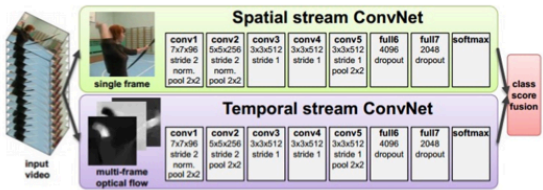
[Farabet et al., 2012]

# Fast-forward: ConvNets are everywhere




[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.




[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

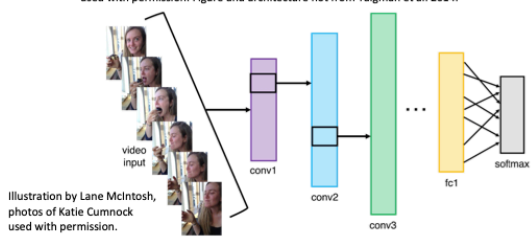
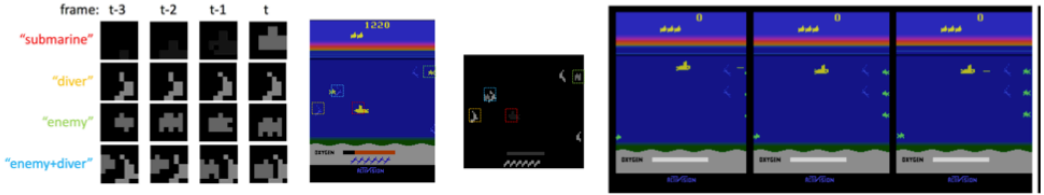



Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

# Fast-forward: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

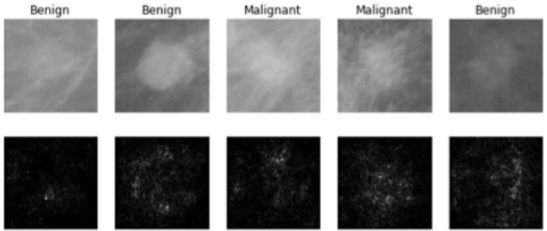
[Toshev, Szegedy 2014]




[Guo et al. 2014]

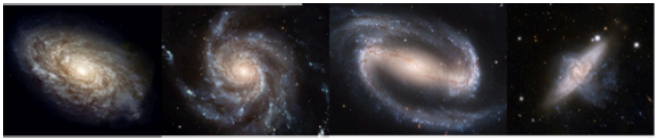
Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

# Fast-forward: ConvNets are everywhere




[Levy et al. 2016]





[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.


[Sermanet et al. 2011]
[Ciresan et al.]

# Fast-forward: ConvNets are everywhere

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

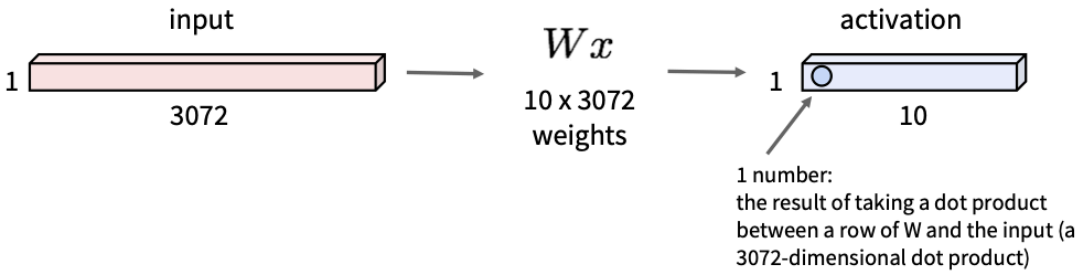
Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

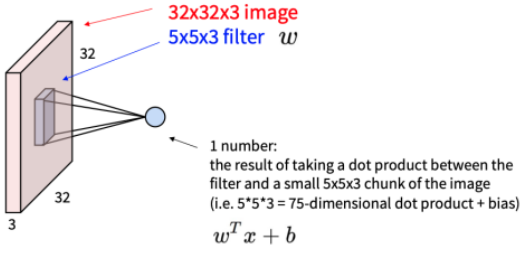
Mnih and Hinton, 2010

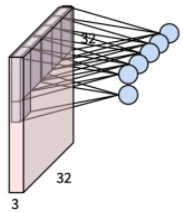
# Convolutional Neural Network

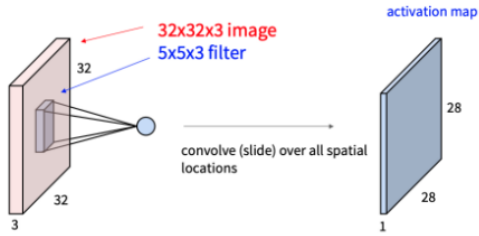
32x32x3 image -> stretch to 3072 x 1



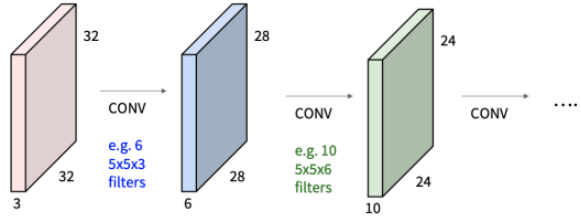
# Convolution Layer

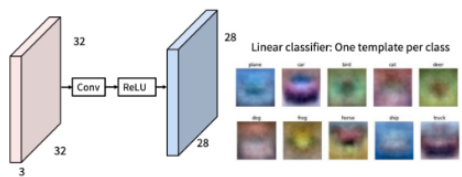


# Convolution Layer



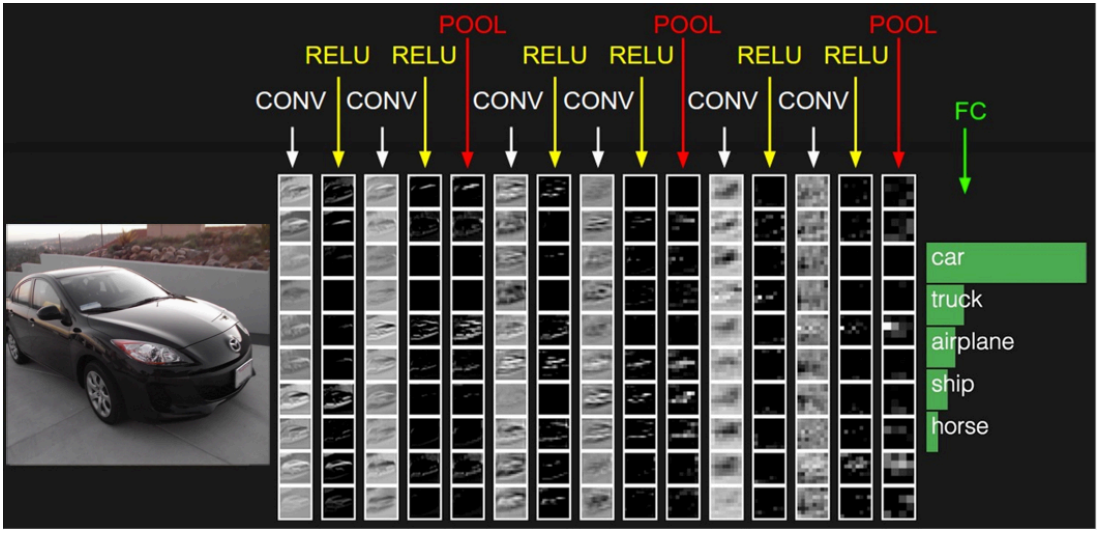



# Convolution Layer: summary

- Let's assume input is $W_1 \times H_1 \times C$
- Conv layer needs 4 hyperparameters:
  - Number of filters $K$
  - The filter size $F$
  - The stride $S$
  - The zero padding $P$
- This will produce an output of $W_2 \times H_2 \times K$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
- Number of parameters: $F^2CK$ and $K$ biases

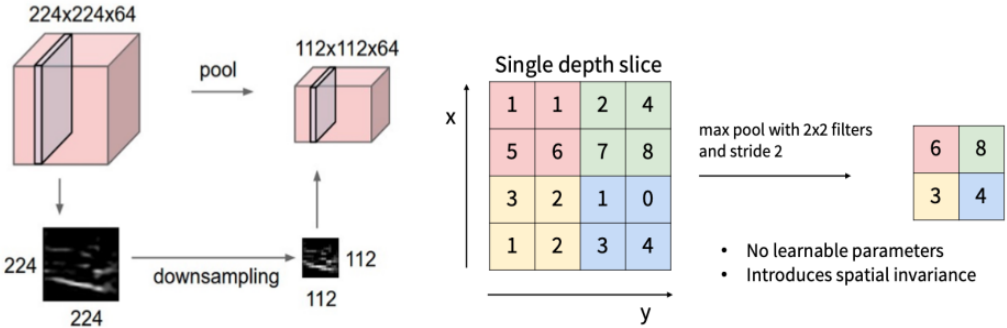
Comment setting:
K = (power of 2, eg. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ?? (whatever fits)
- F = 1, S = 1, P = 0


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

# Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently

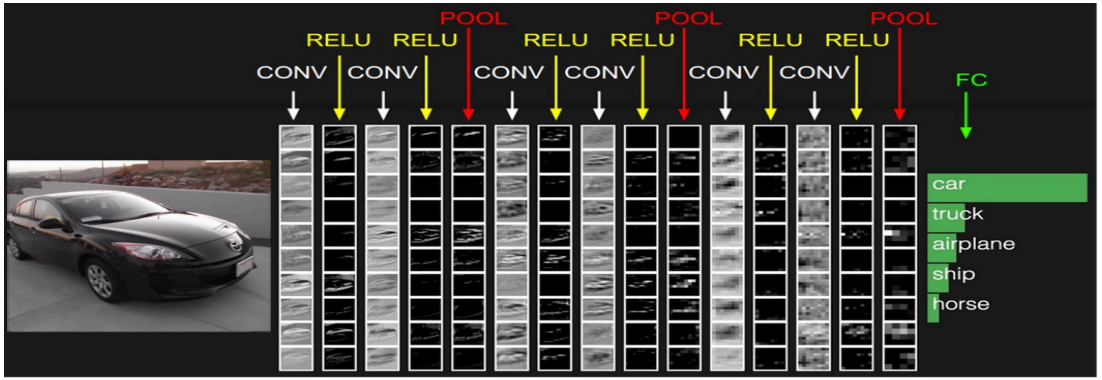


# Pooling layer: summary

- Let's assume input is $W_1 \times H_1 \times C$
- Conv layer needs 2 hyperparameters
  - The spatial extent $F$
  - The stride $S$
- This will produce an output of $W_2 \times H_2 \times C$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
- Number of parameters: **0**

# Fully Connected Layer (FC Layer)

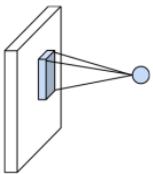
Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



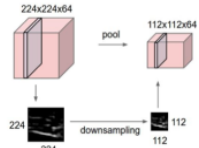

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

# Putting it All Together

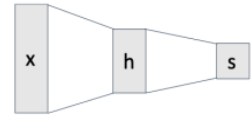
Convolution Layers



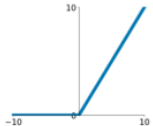
Pooling Layers



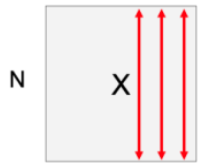

Fully-Connected Layers




Activation Function




**Input:** $x : N \times D$




$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

# Summary

- A ConvNets stack CONV, POOL, FC layers
- The trend towards smaller filters and deeper architectures - Trend towards getting rid of POOL/FC layers (just CONV) - Historically architectures looked like [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX where $N$ is usually up to $\approx 5$, $M$ is large, $0 \leq K \leq 2$ .

However, recent advances such as ResNet/GoogLeNet have challenged this paradigm.

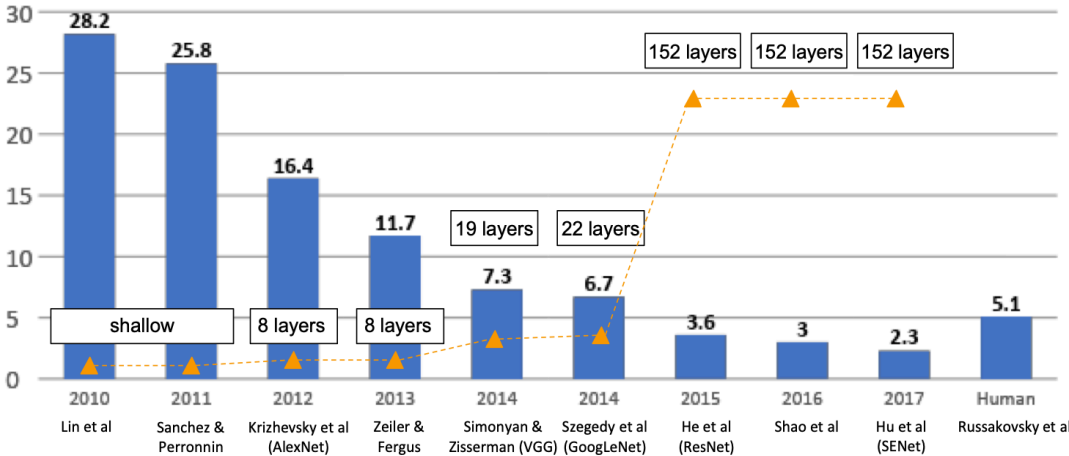
# CNN Architectures

- AlexNet
- VGG
- ResNet

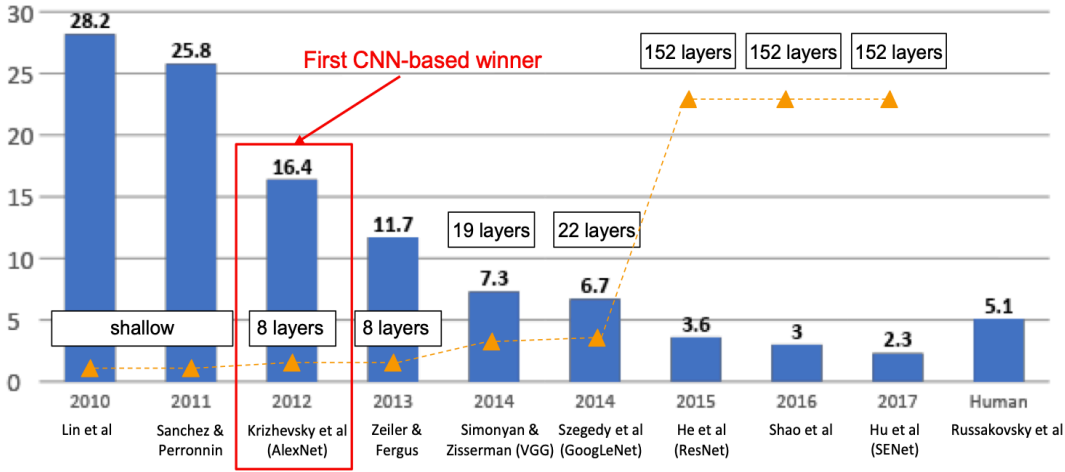
Also....

- GoogLeNet
- ZFNet
- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet

# ImageNet Large Scale Visual Recognition Challenge winners



# ImageNet Large Scale Visual Recognition Challenge winners



# AlexNet

**Architecture:**

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

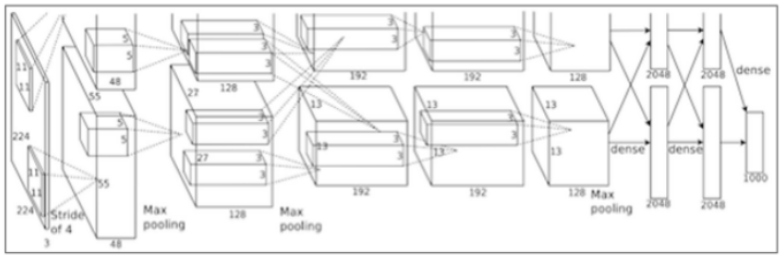



Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

# AlexNet

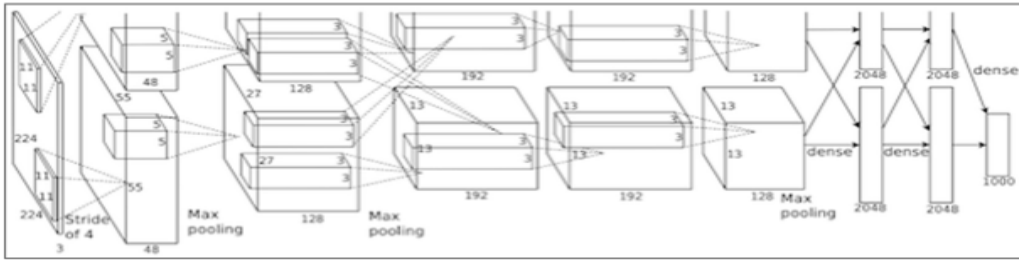



Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

## "AlexNet"

- Input: 227 x 227 x 3 images
- First Layer: (CONV1): 96 11 x 11 filters applied at stride 4
- Q: What is the output volumn size?
- Q: What is the total number of parameters in this layer?

# AlexNet

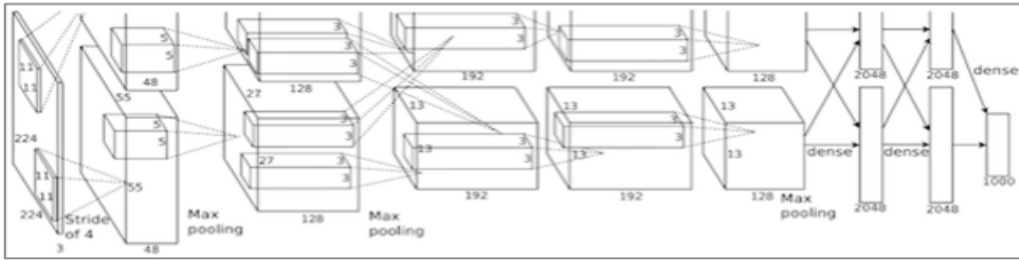



Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

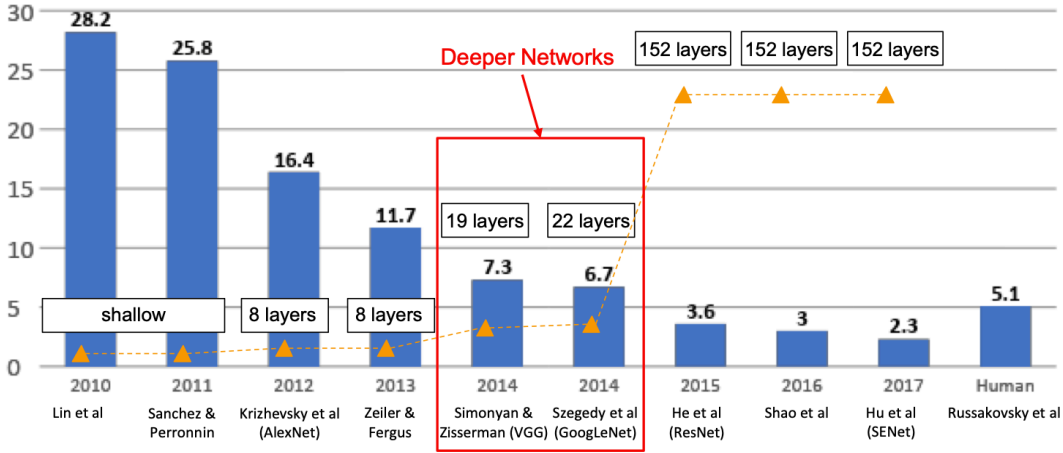
- Input: 227 x 227 x 3 images
- After CONV1: 55 x 55 x 96
- Second Layer (POOL1): 3 x 3 filters applied at stride 2
- Q: What is the output volumn size?
- Q: What is the total number of parameters in this layer?

# AlexNet

Full (simplified) AlexNet architecture:


[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

# AlexNet


30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
Deeper Networks
152 layers
152 layers
152 layers
19 layers
22 layers
shallow
8 layers
8 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

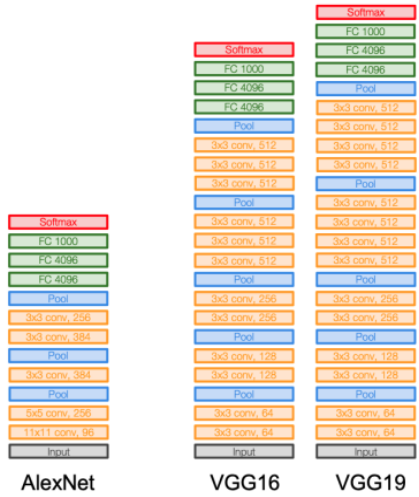
# VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:

Most memory is in early CONV

Most params are in late FC

(not counting biases)

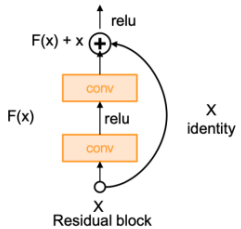
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

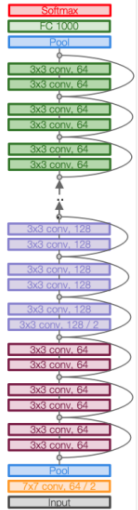
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

# ResNet

Very deep networks using residual connections

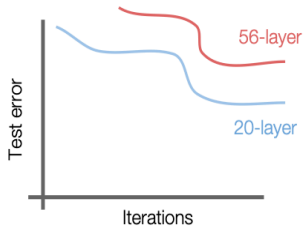
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

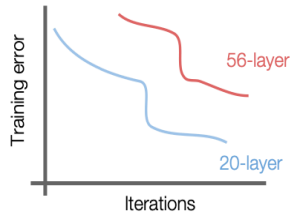


# ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?




56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

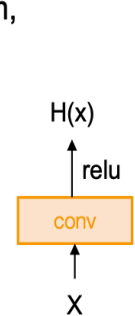
# ResNet

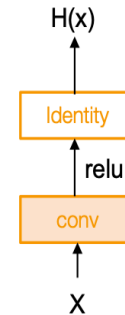
Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

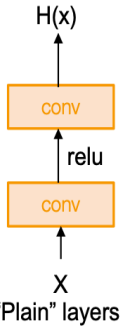
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

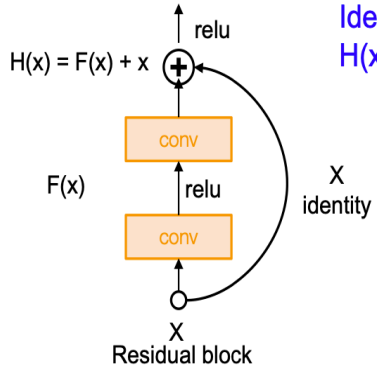


# ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



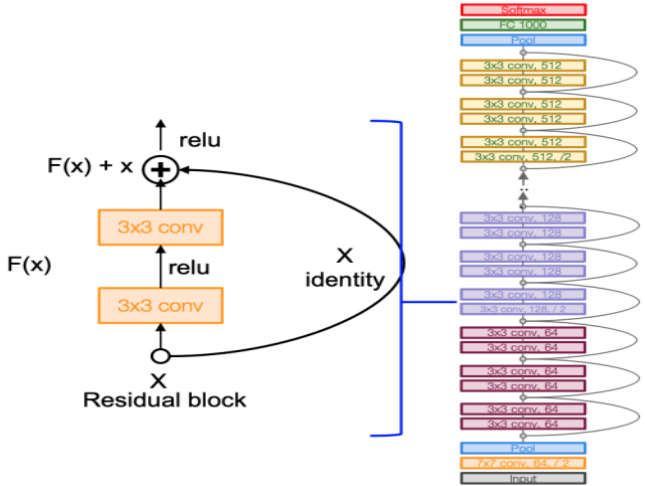




Identity mapping:
H(x) = x if F(x) = 0

# ResNet

Full ResNet architecture

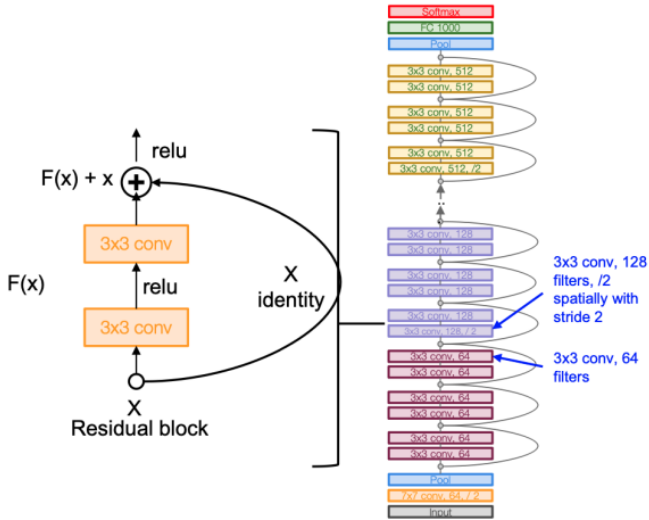
- Stack residual blocks
- Every residual block has two $3 \times 3$ conv layers



# ResNet

## Full ResNet architecture

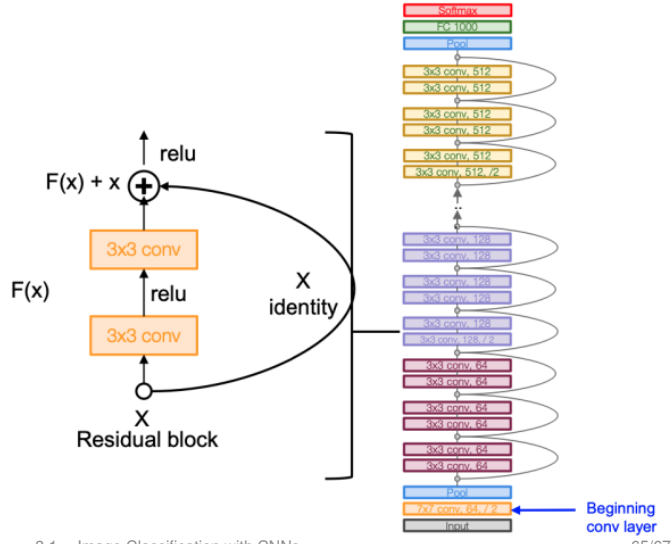
- Stack residual blocks
- Every residual block has two $3 \times 3$ conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volumn by half



# ResNet

## Full ResNet architecture

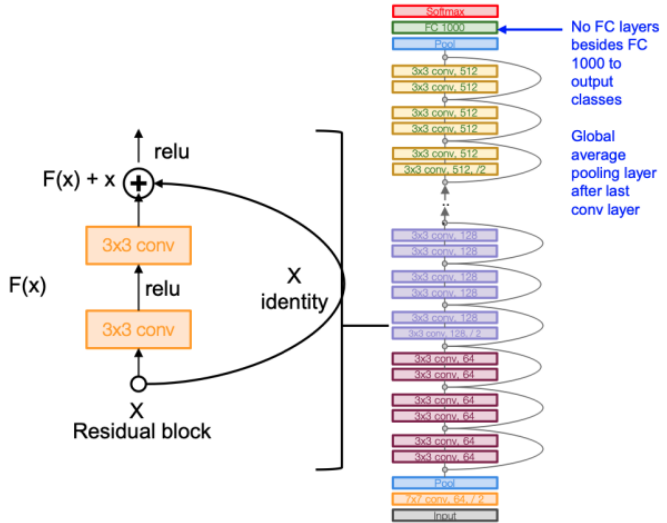
- Stack residual blocks
- Every residual block has two $3 \times 3$ conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)



# ResNet

## Full ResNet architecture

- Stack residual blocks
- Every residual block has two $3 \times 3$ conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)



# Exerecise

Do three exercises!