# Convolutional Neural Networks

Compared to standard feedforward neural networks with similarly-sized layers,

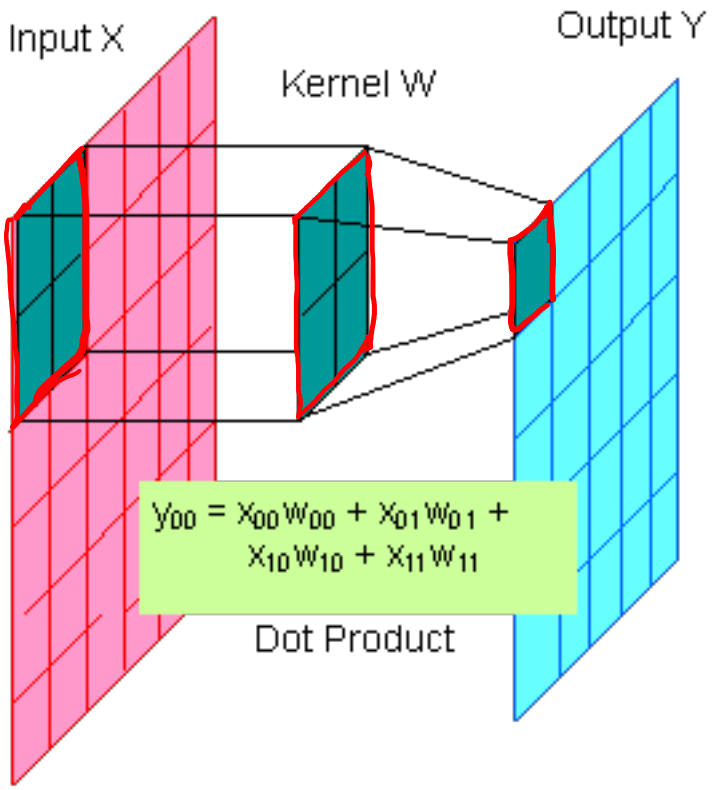
- CNNs have much fewer connections and parameters
- and so they are easier to train,
- while their performance is likely to be only slightly worse, particularly for images as inputs.

shared parameters

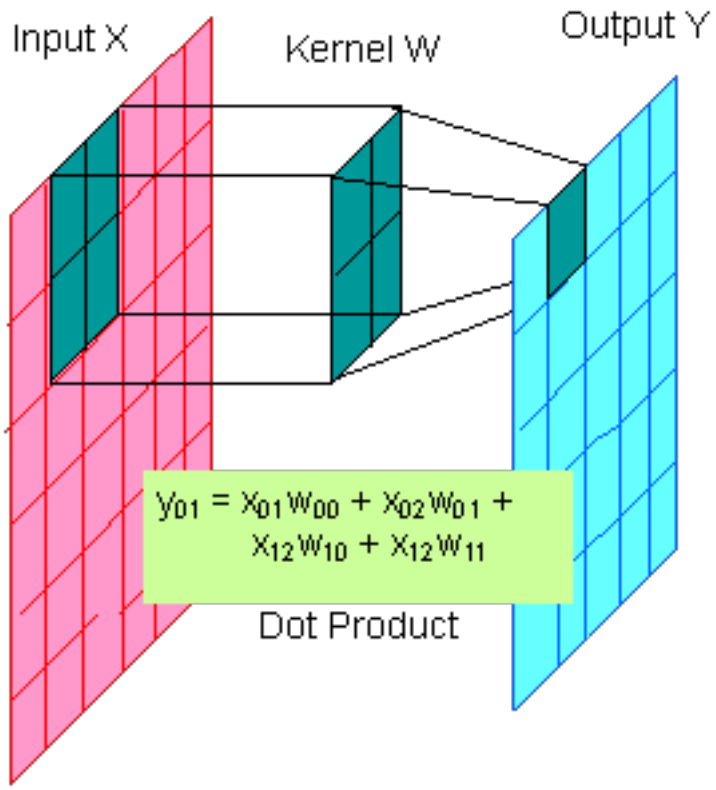
**LeNet 5**

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE, 86(11):2278-2324, November **1998***

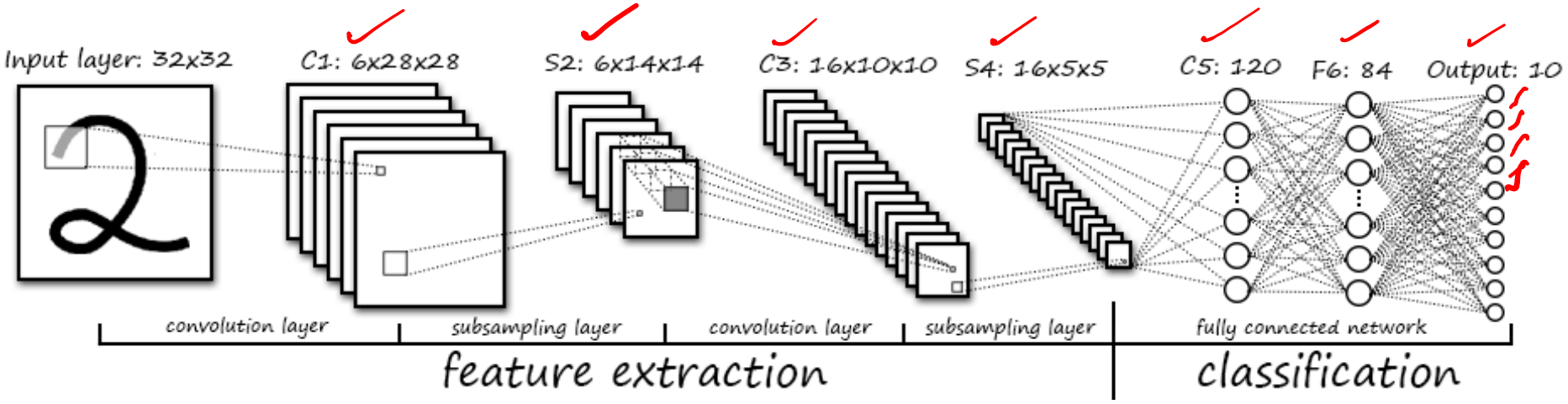
# 2-Dimensional Convolution



# 2-Dimensional Convolution



# LeNet 5, LeCun et al 1998

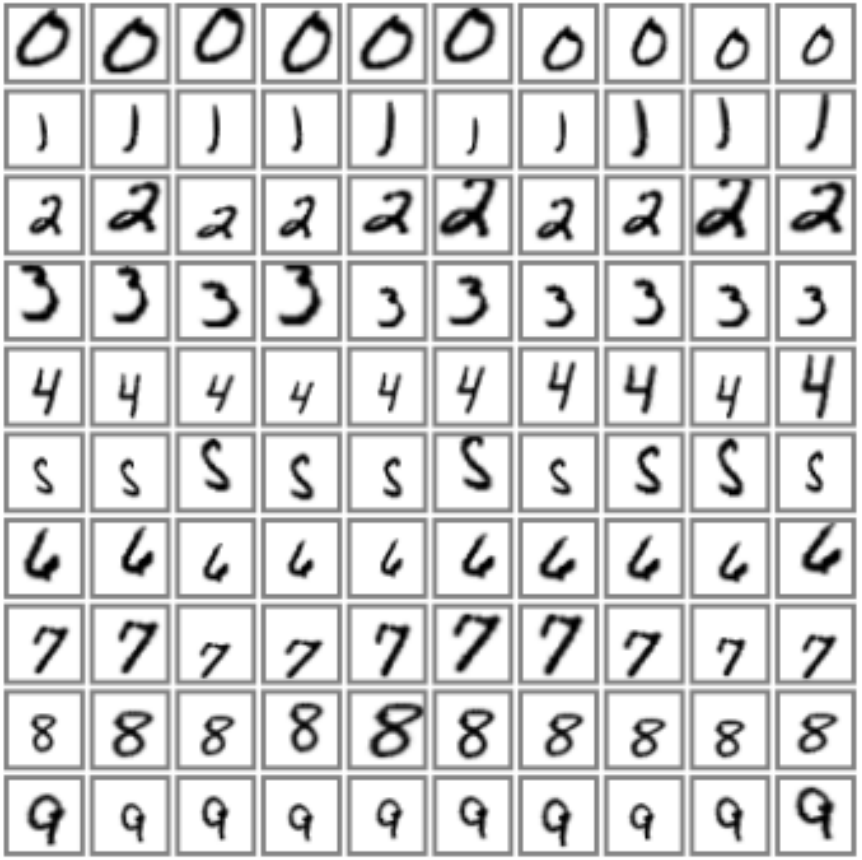



- **Input:** 32x32 pixel image. Largest character is 20x20
  (All important info should be in the center of the receptive fields of the highest level feature detectors)
- **Cx:** Convolutional layer (C1, C3, C5) tanh nonlinear units
- **Sx:** Subsample layer (S2, S4) average pooling
- **Fx:** Fully connected layer (F6) logistic/sigmoid units
- Black and White pixel values are normalized:
  E.g. White = -0.1, Black =1.175 (Mean of pixels = 0, Std of pixels =1)

# MINIST Dataset

60,000 original dataset

Test error: 0.95% ✓



540,000 artificial distortions

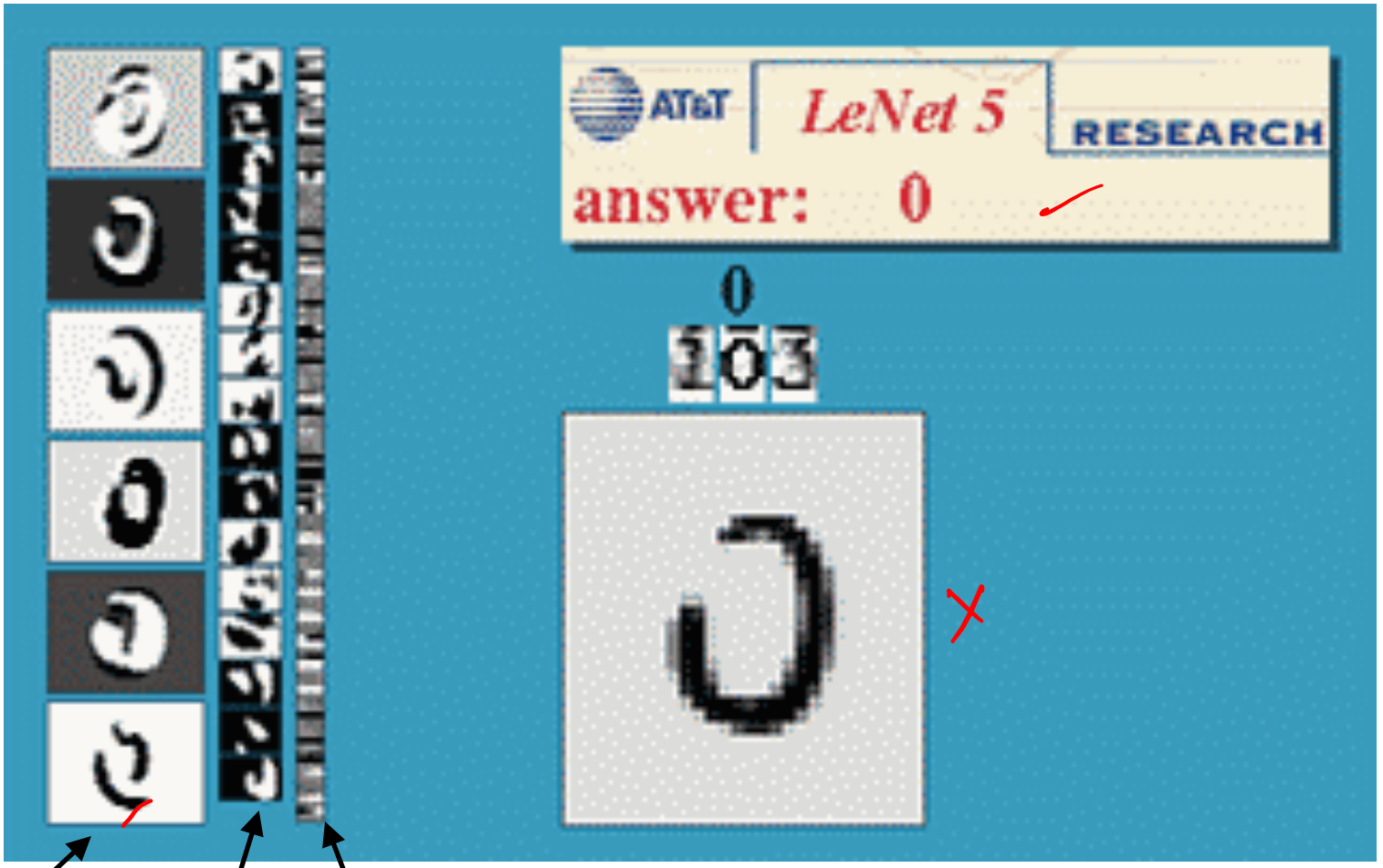
+ 60,000 original

Test error: 0.8% ✓

# Misclassified examples

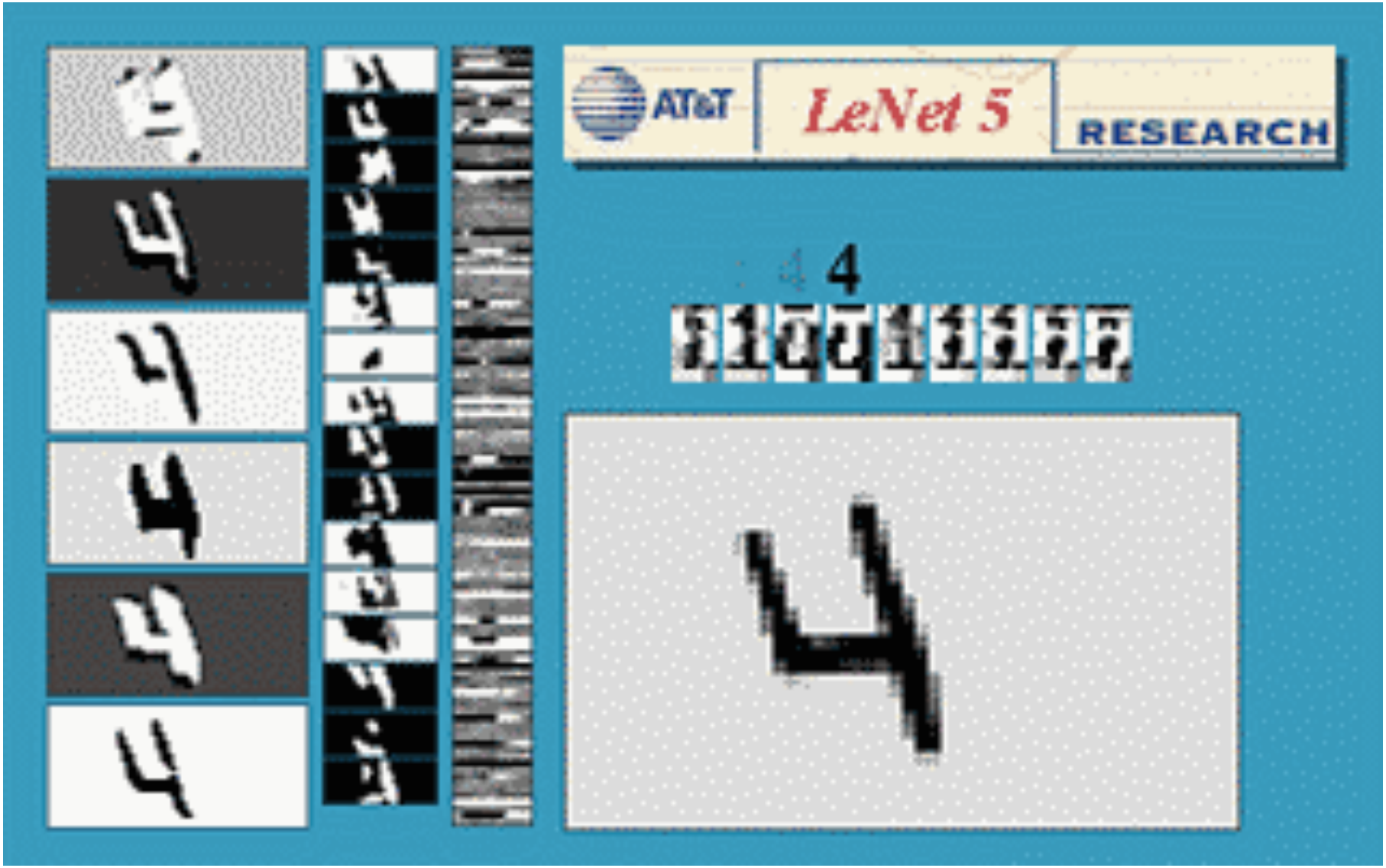
True label -> Predicted label

# LeNet 5 in Action

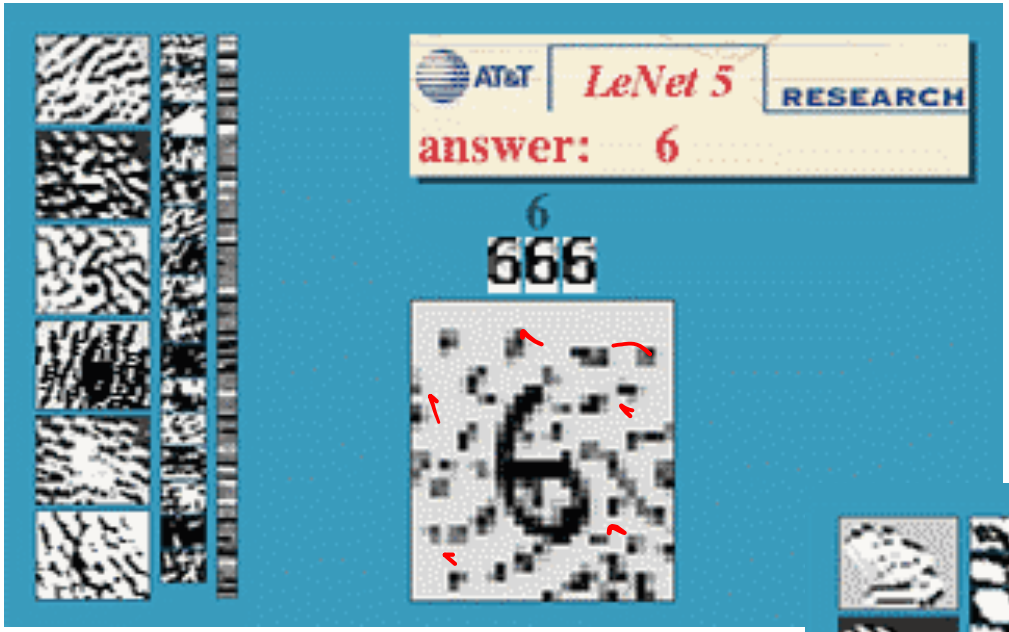


# LeNet 5, Shift invariance

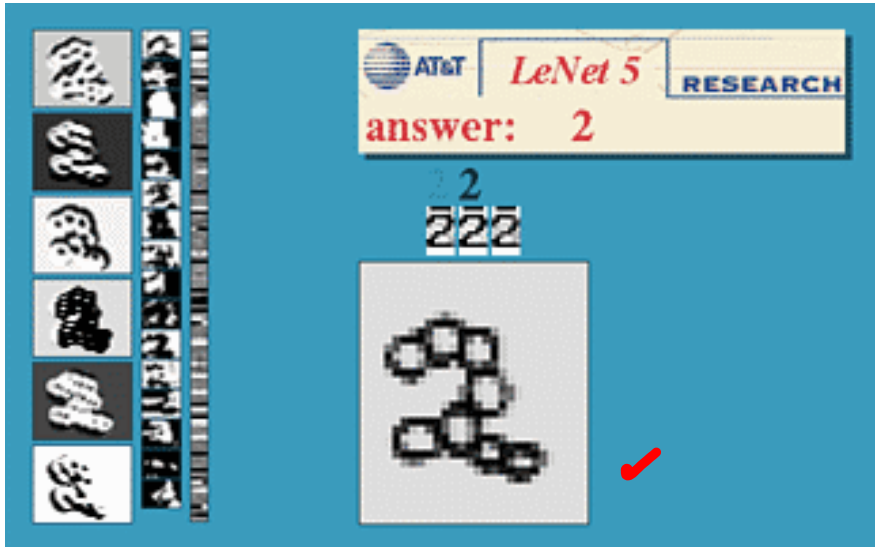
# LeNet 5, Rotation invariance

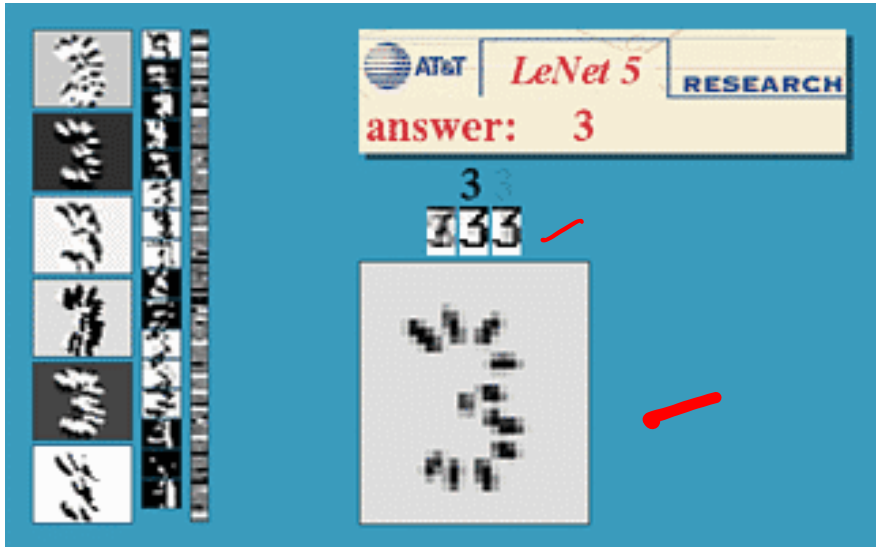


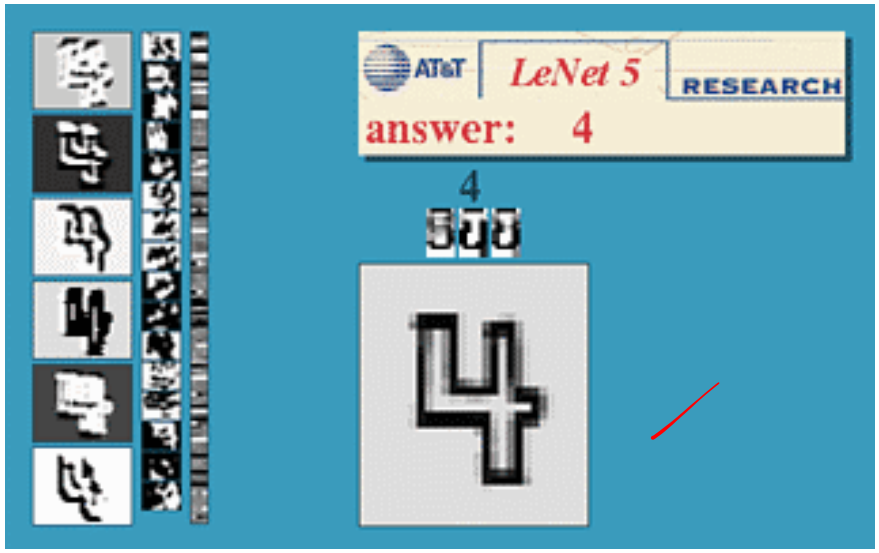
# LeNet 5, Noise resistance

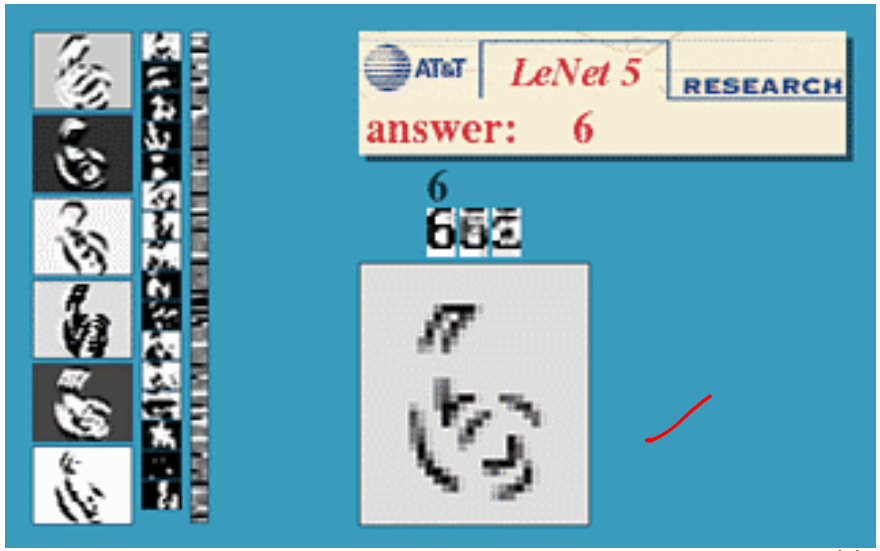



# LeNet 5, Unusual Patterns



# ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton,

Advances in Neural Information Processing Systems 2012

**Alex Net**

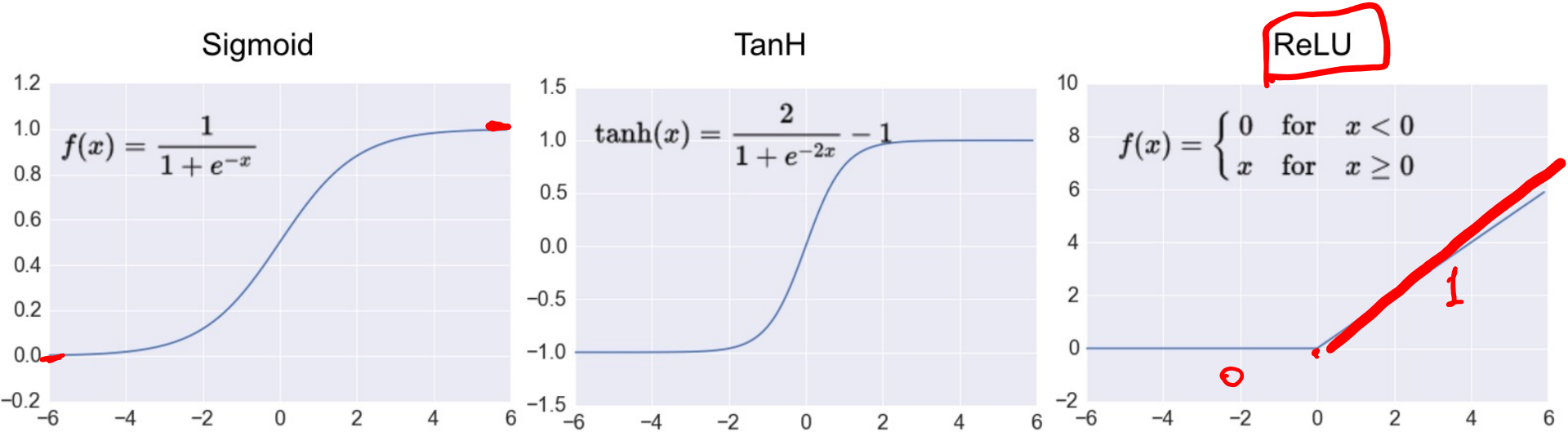
# The Architecture

Typical nonlinearities: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$f(x) = (1 + e^{-x})^{-1}$ (logistic function)

Here, **Rectified Linear Units (ReLU)** are used: $f(x) = \max(0, x)$

**Non-saturating/Gradients don't vanish** – faster training



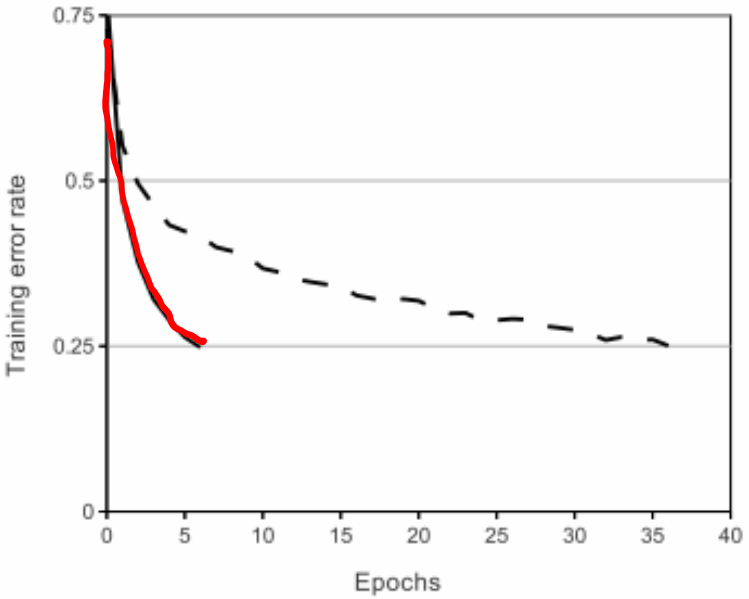
# The Architecture

Typical nonlinearities: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$f(x) = (1 + e^{-x})^{-1}$ (logistic function)

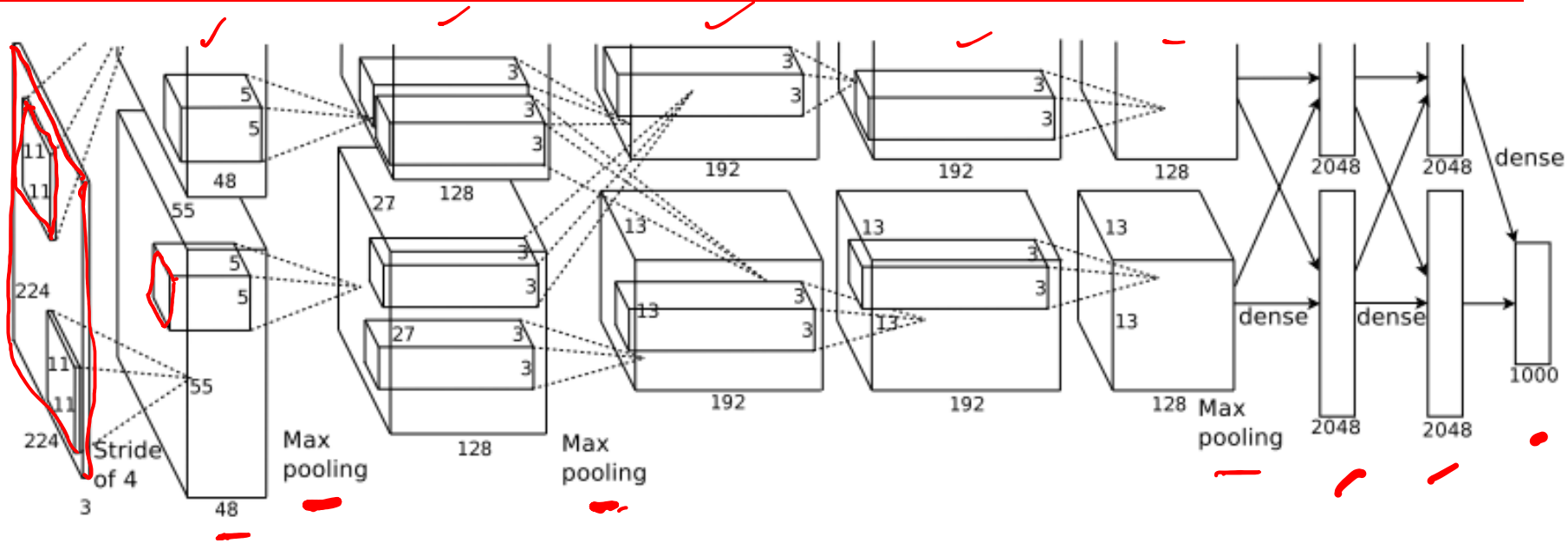
Here, **Rectified Linear Units (ReLU)** are used: $f(x) = \max(0, x)$

**Non-saturating/Gradients don't vanish** – faster training



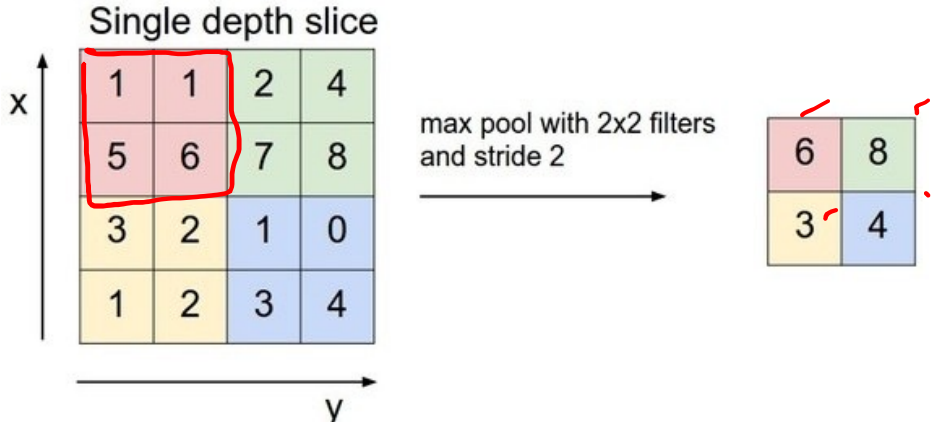

A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line)

# The Architecture

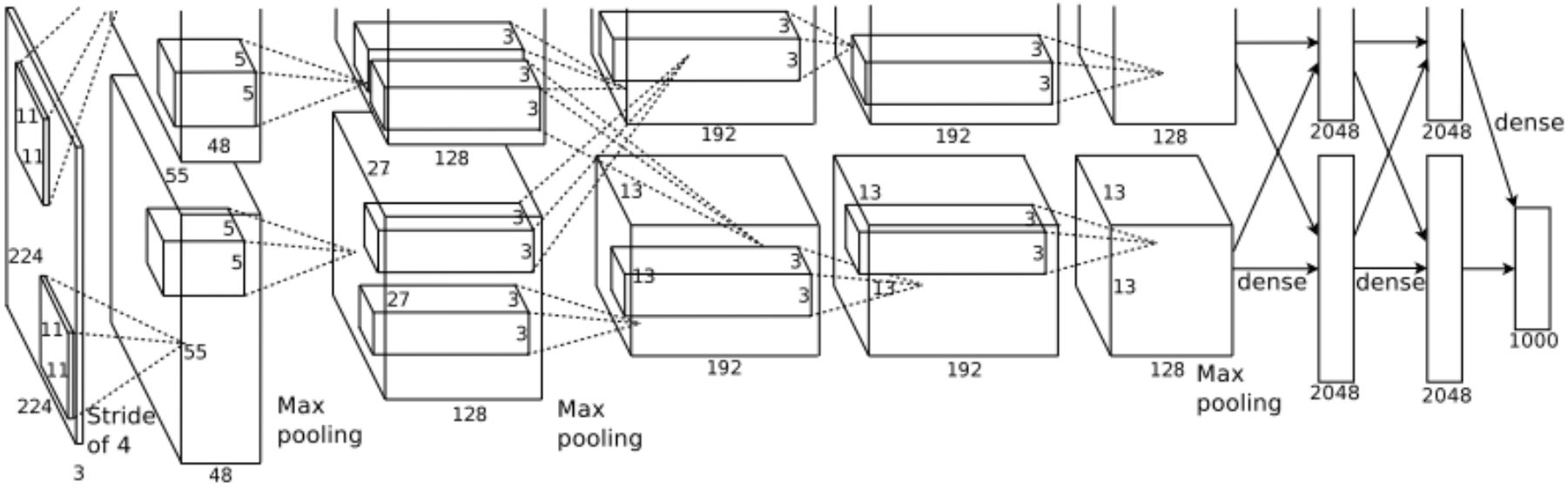



**5 convolution layers (ReLU)**

**3 overlapping max pooling** – nonlinear downsampling (max value of regions)



# The Architecture




**5 convolution layers (ReLU)**

**3 overlapping max pooling** – nonlinear downsampling (max value of regions)

**2 fully connected layers**

**output softmax**

# Training

- Trained with stochastic gradient descent
- on two NVIDIA GTX 580 3GB GPUs
- for about a week

- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- 5 convolutional layer with Rectified Linear Units (ReLUs), 3 overlapping max pooling, 2 fully connected layer
- Final feature layer: 4096-dimensional

- Prevent overfitting – data augmentation, dropout trick
- Randomly extracted 224x224 patches for more data

# Preventing overfitting

1) **Data augmentation**: The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations:

- image translation
- horizontal reflections
- changing RGB intensities

2) **Dropout**: set the output of each hidden neuron to zero w.p. 0.5.

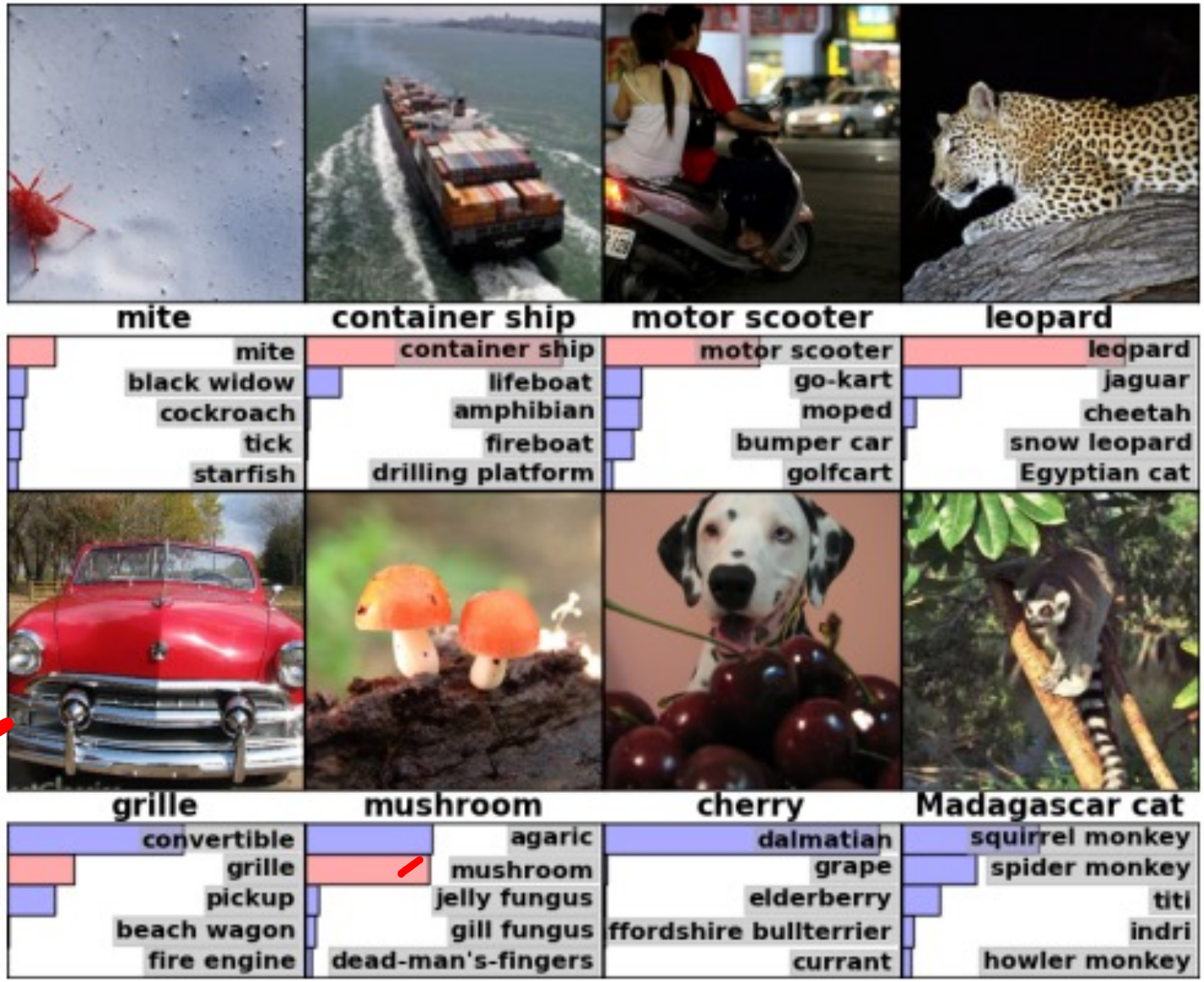
- So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.
- forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

# ImageNet

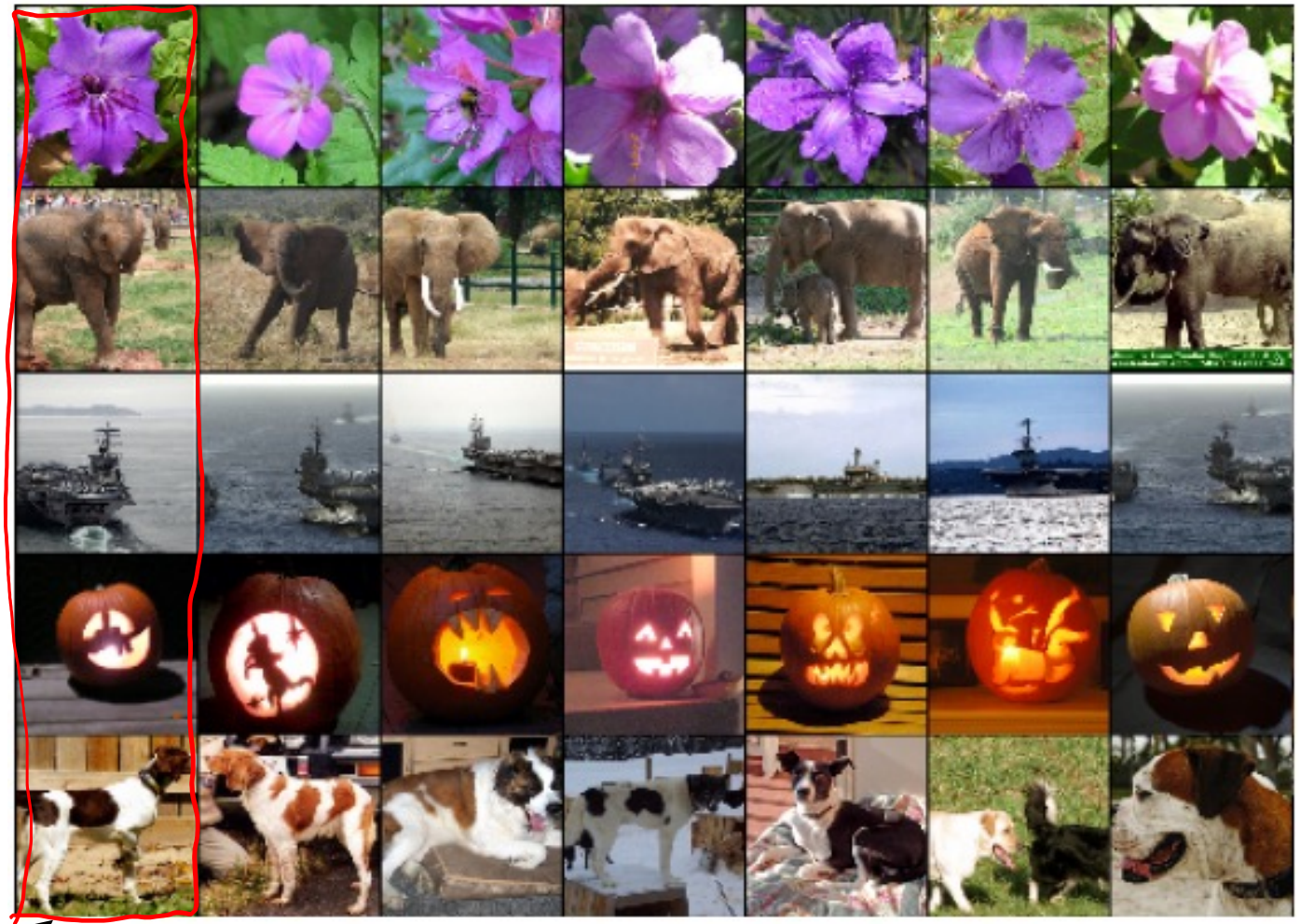
- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images

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

# 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

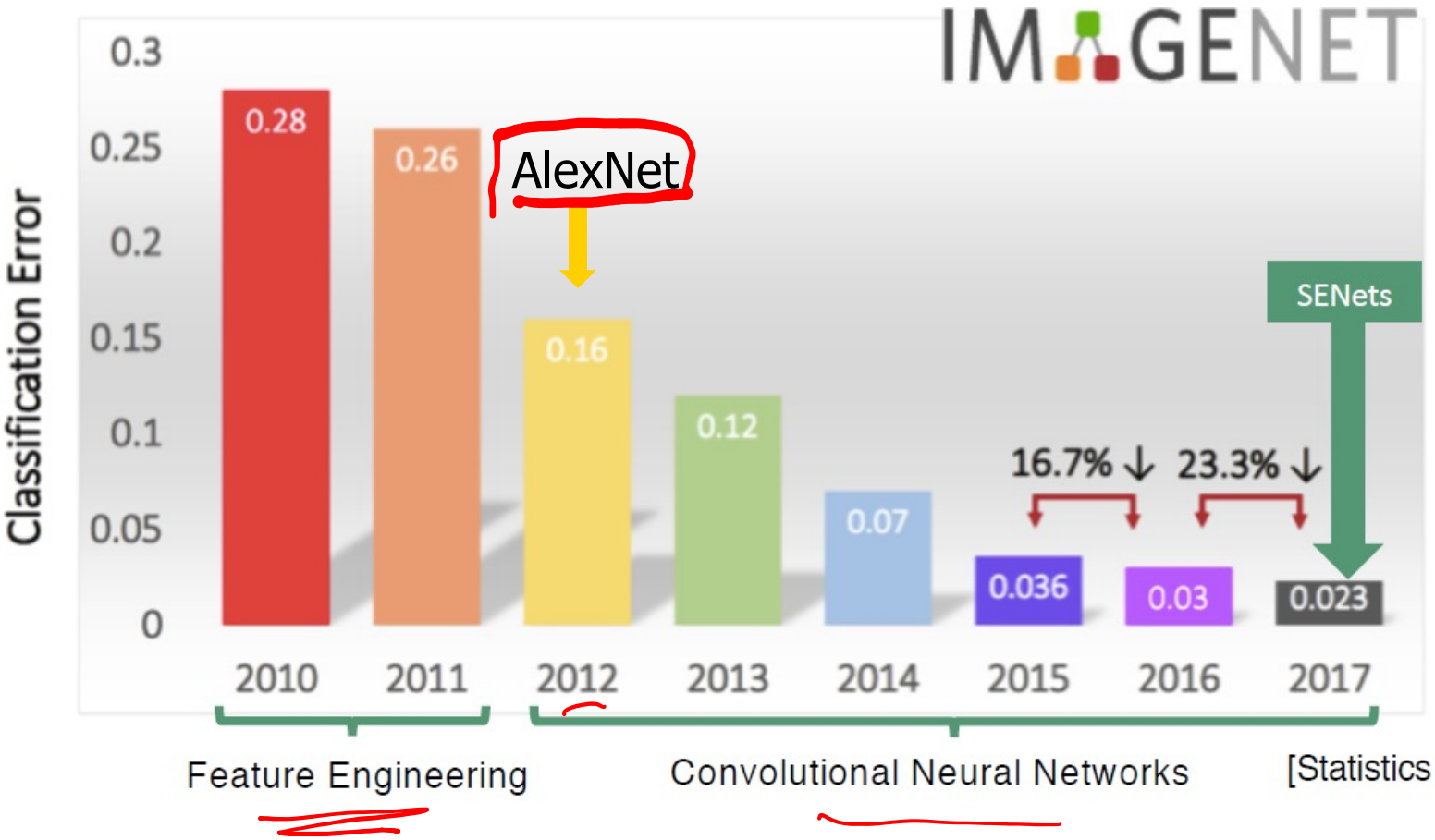
# Results: Image similarity



Test column

six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

# Results


IMAGENET
Classification Error
0.3
0.25
0.2
0.15
0.1
0.05
0
0.28
0.26
AlexNet
0.16
0.12
0.07
0.036
0.03
0.023
SENets
16.7% ↓
23.3% ↓
2010
2011
2012
2013
2014
2015
2016
2017
Feature Engineering
Convolutional Neural Networks
[Statistics provided by ILSVRC]

# Other optimization tips and tricks

- **Momentum:** use exponentially weighted sum of previous gradients

$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

can get pass plateaus more quickly, by "gaining momentum"

- **Initialization:** cannot initialize to same value, all units in a hidden layer will behave same; randomly initialize unif[-b,b]

- **Adaptive learning rates:** one learning rate per parameter

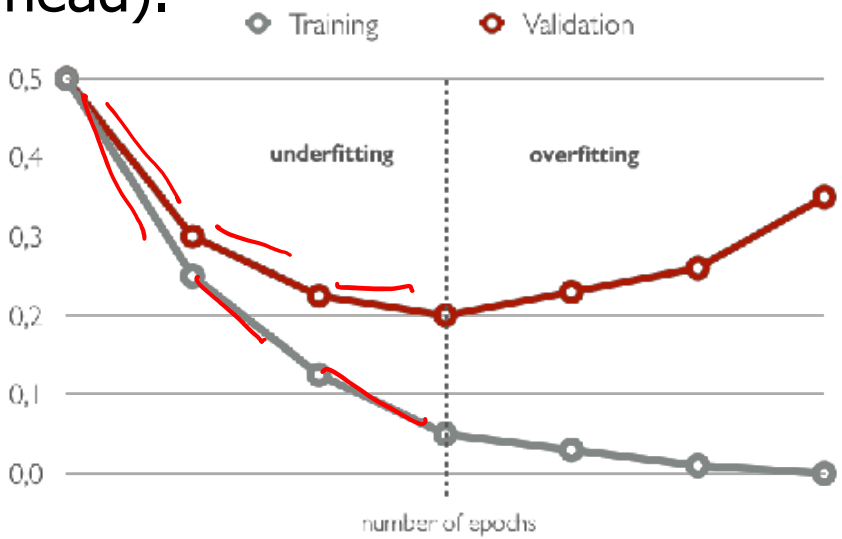
e.g. RMSProp uses exponentially weighted average of squared gradients

$$\gamma^{(t)} = \beta \gamma^{(t-1)} + (1-\beta) \left( \nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) \right)^2 \qquad \overline{\nabla}_{\theta}^{(t)} = \frac{\nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})}{\sqrt{\gamma^{(t)} + \epsilon}}$$

Adam combines RMSProp with momentum

# Tips and tricks for preventing overfitting

- **Dropout**
- **Data augmentation**
- **Early stopping:** stop training when validation set error increases (with some look ahead).




- **Neural Architecture search:** tune number of layers and neurons per layer using grid search or clever optimization

# Tips and Tricks for training deep NNs

- First hypothesis (underfitting): better optimize
  - ➢ Increase the capacity of the neural network
  - ➢ Check initialization
  - ➢ Check gradients (saturating units and vanishing gradients)
  - ➢ Tune learning rate

- Second hypothesis (overfitting): use better regularization
  - ➢ Dropout
  - ➢ Data augmentation
  - ➢ Early stopping
  - ➢ Architecture search

- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

# Artificial Neural Networks: Summary

- Used to mimic distributed computation in brain
- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate - overfitting
- Hidden layers learn intermediate representations – how many to use?

- Prediction – Forward propagation
- Gradient descent (Back-propagation), local minima problems

- Coming back in new form as deep networks

# Decision Trees

Aarti Singh

Machine Learning 10-315
Feb 20 , 2023

# Parametric methods

- Assume some model (Gaussian, Bernoulli, Multinomial, logistic, network of logistic units, Linear, Quadratic) with fixed number of parameters
  - Gaussian Bayes, Naïve Bayes, Logistic Regression, Support vector machines, Neural Networks

- Estimate parameters ($\mu, \sigma^2, \theta, w, \beta$) using MLE/MAP and plug in

- **Pro** – need few data points to learn parameters
- **Con** – Strong modeling assumptions, not satisfied in practice

# Non-Parametric methods

- Typically don't make any modeling assumptions
- As we have more data, we should be able to learn more complex models
- Let number of parameters scale with number of training data

- Some nonparametric methods

  **Classification:** Decision trees, k-NN (k-Nearest Neighbor) classifier

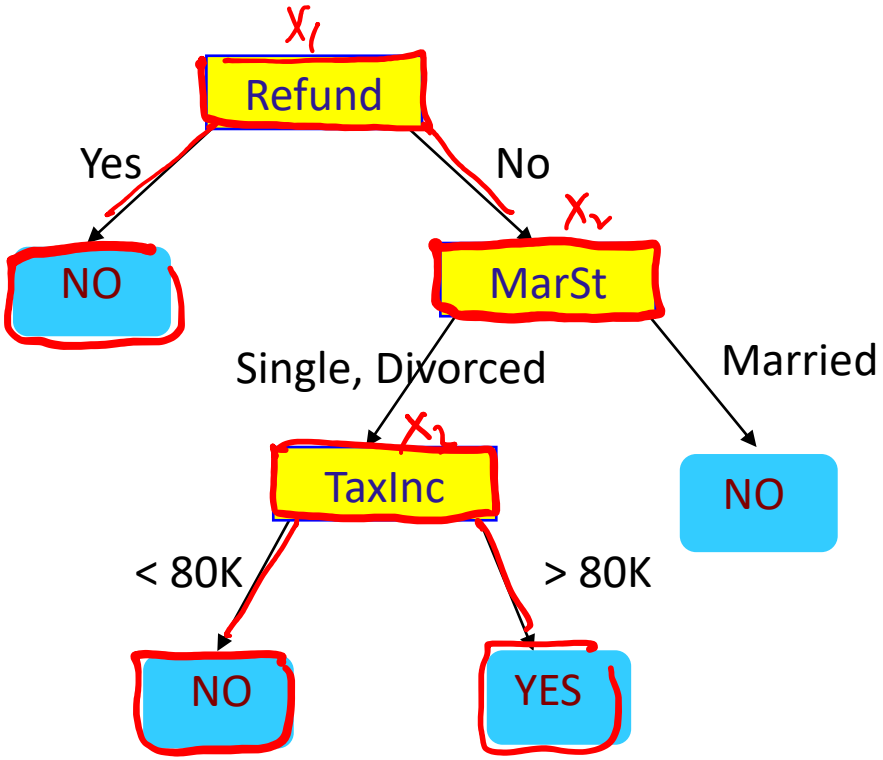
  **Density estimation:** k-NN, Histogram, Kernel density estimate

  **Regression:** Kernel regression

# Decision Trees

- A nonparametric method
  - Complexity increases with more data
  - No fixed set of parameters
- Start with discrete features, then discuss continuous
- What does a decision tree represent?

# Decision Tree for Tax Fraud Detection




$X =$

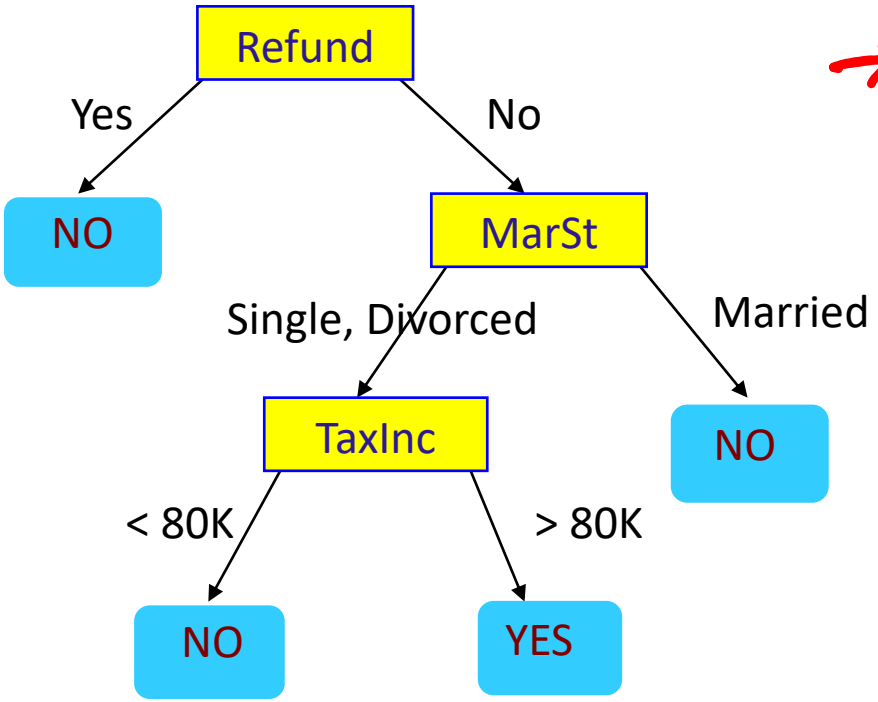
| $X_1$ | $X_2$ | $X_3$ | $Y$ |
|---|---|---|---|
| Refund | Marital Status | Taxable Income | Cheat |
| | | | |

- Each internal node: test one feature $X_i$
- Each branch from a node: selects some value for $X_i$
- Each leaf node: prediction for Y

# Prediction

- Given a decision tree, how do we assign label to a test point

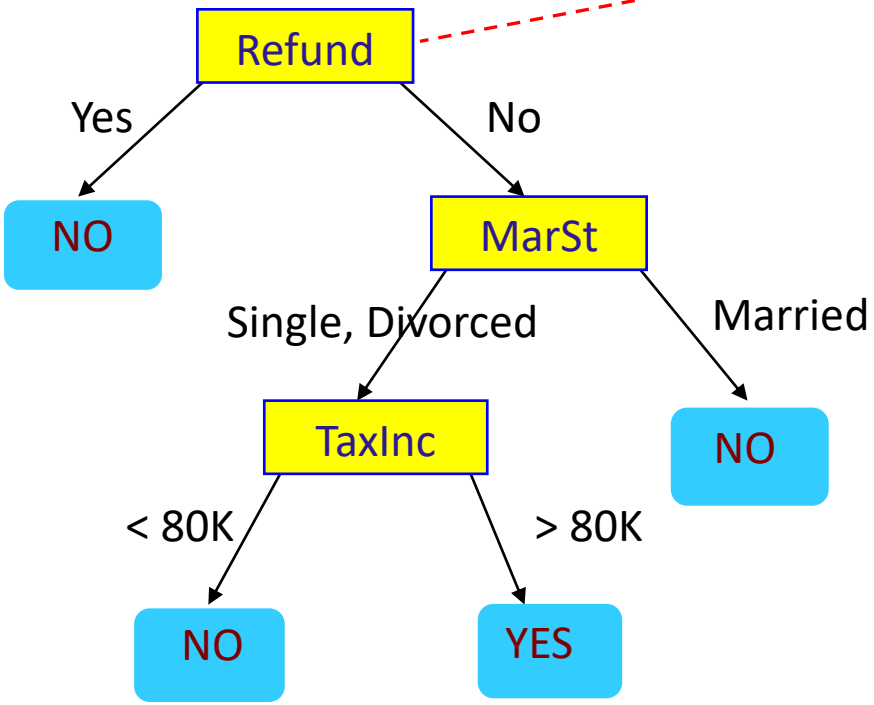
# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

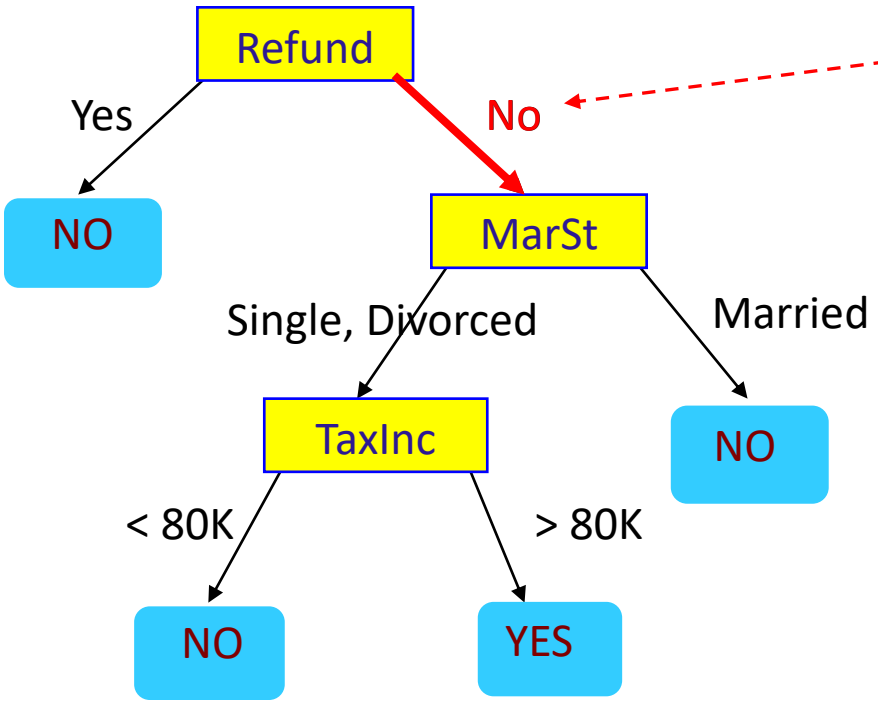
# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

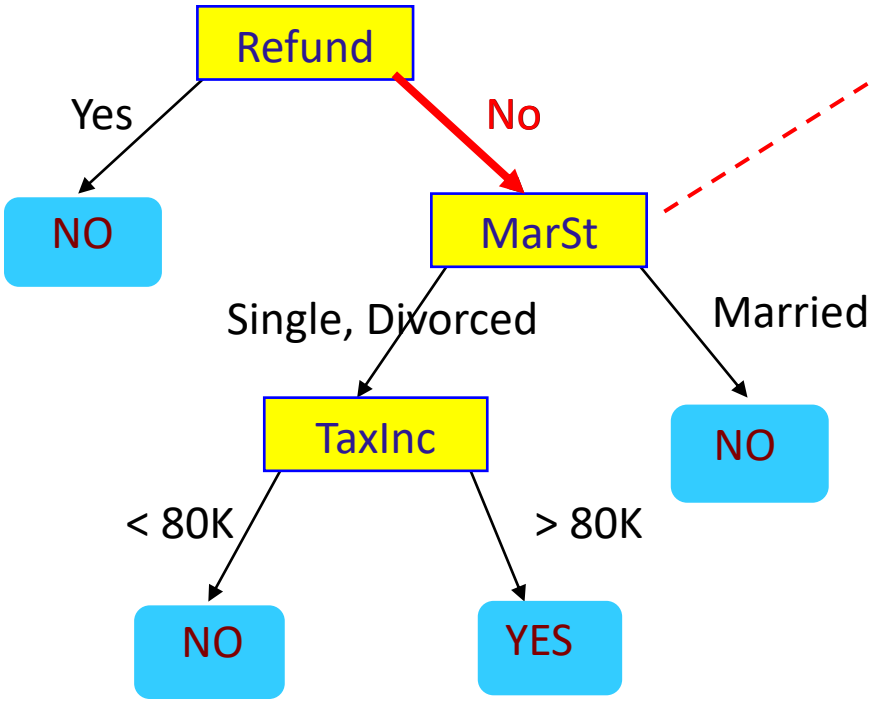
# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

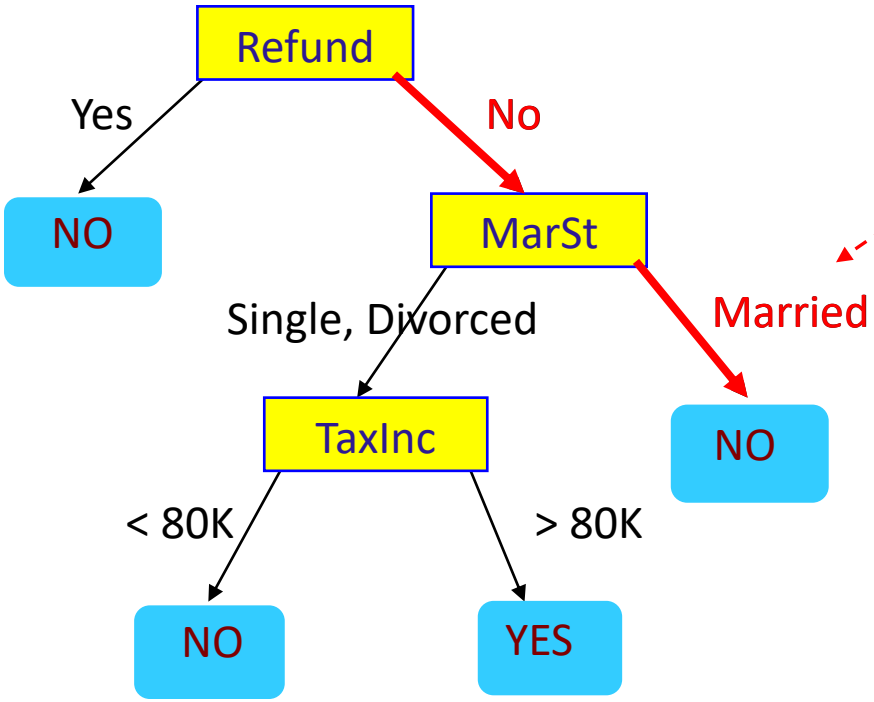
# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

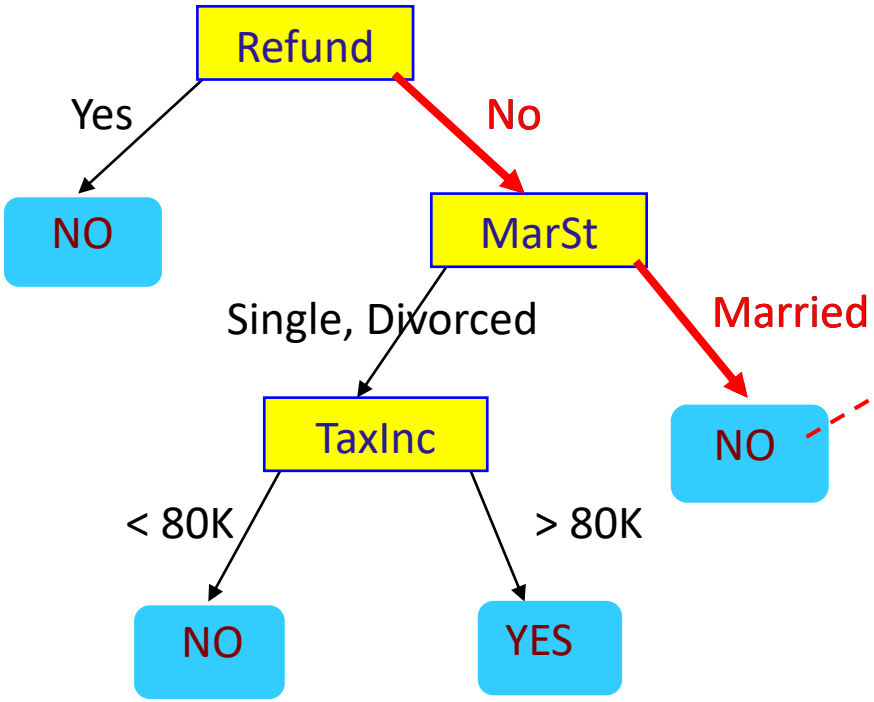
# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

# Decision Tree for Tax Fraud Detection




Query Data

| $X_1$ Refund | $X_2$ Marital Status | $X_3$ Taxable Income | $Y$ Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

Assign Cheat to “No”

# So far…

- What does a decision tree represent
- Given a decision tree, how do we assign label to a test point

Discriminative or Generative?

# Now …

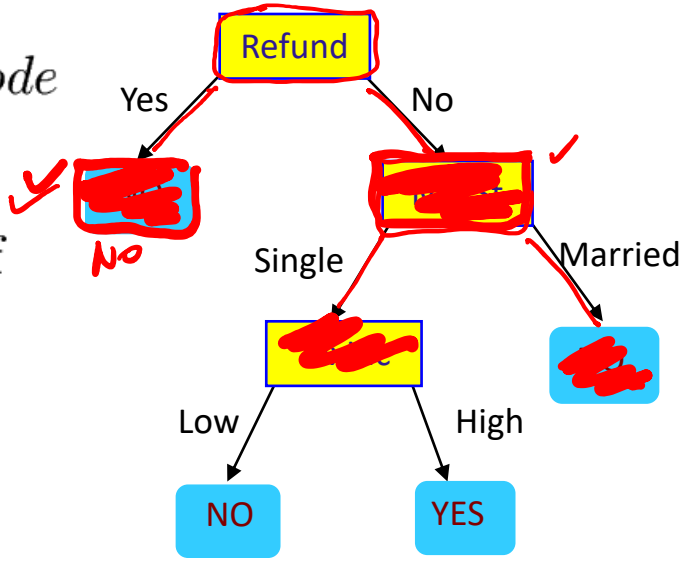
- How do we learn a decision tree from training data

# How to learn a decision tree

- Top-down induction [ID3]

Main loop:

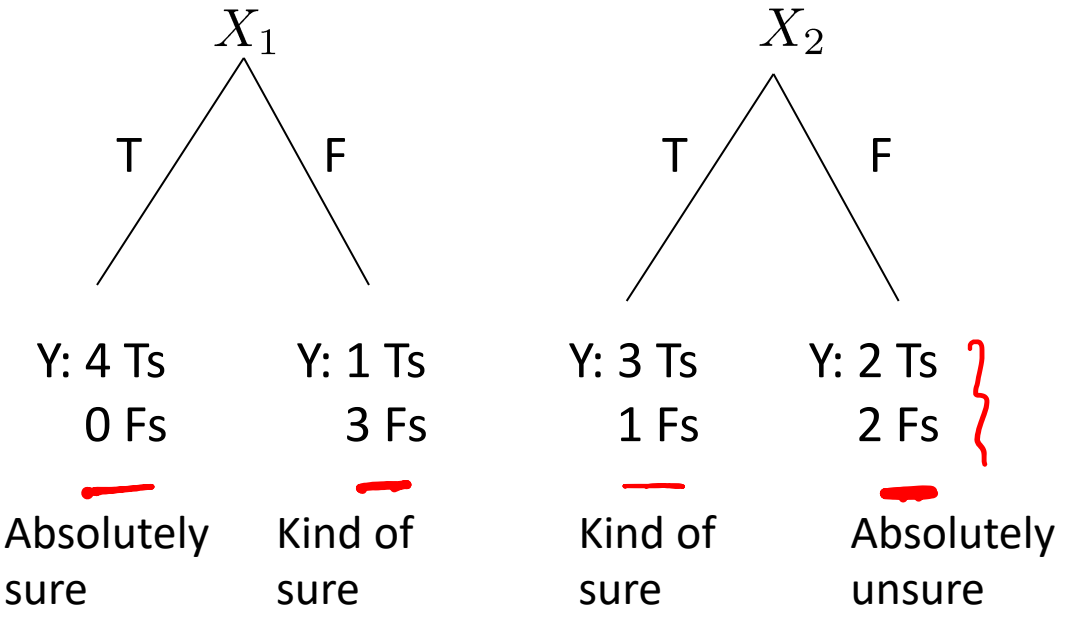
1. $X \leftarrow$ the "best" decision feature for next *node*
2. Assign $X$ as decision feature for *node*
3. For each value of $X$, create new descendant of *node* (Discrete features)
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes
   (steps 1-5) after removing current feature




6. When all features exhausted, assign majority label to the leaf node

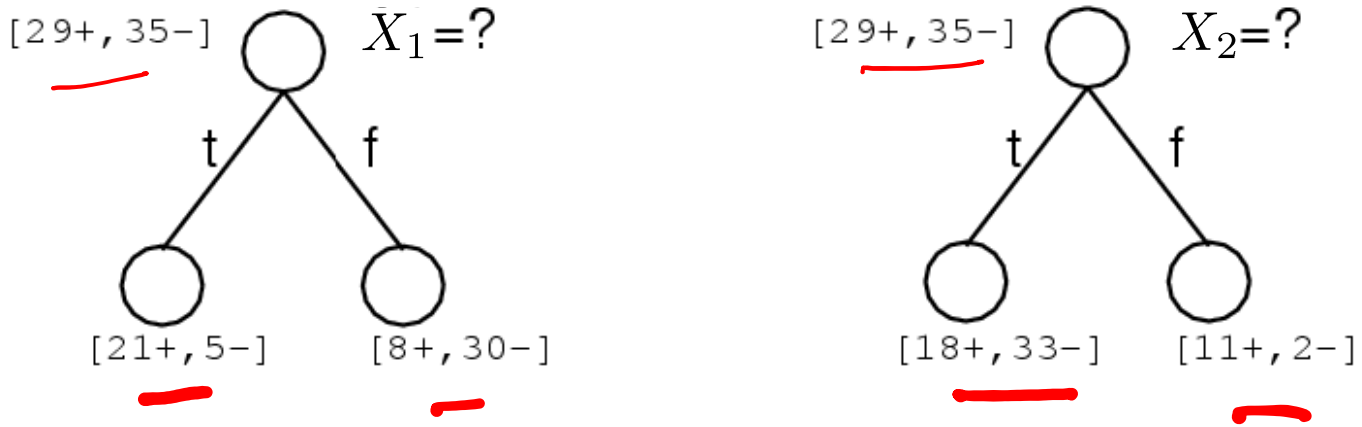
# Which feature is best?

| $X_1$ | $X_2$ | Y |
|---|---|---|
| T | T | T |
| T | F | T |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |
| F | T | F |
| F | F | F |




Good split if we are more certain about classification after split –
Uniform distribution of labels is bad

# Which feature is best?




Pick the attribute/feature which yields maximum information gain:

$$\arg\max_i I(Y, X_i) = \arg\max_i [H(Y) - H(Y|X_i)]$$

H(Y) – entropy of Y  H(Y|$X_i$) – conditional entropy of Y

# Andrew Moore's Entropy in a Nutshell

Low Entropy

..the values (locations of soup) sampled entirely from within the soup bowl

High Entropy

..the values (locations of soup) unpredictable... almost uniformly sampled throughout our dining room

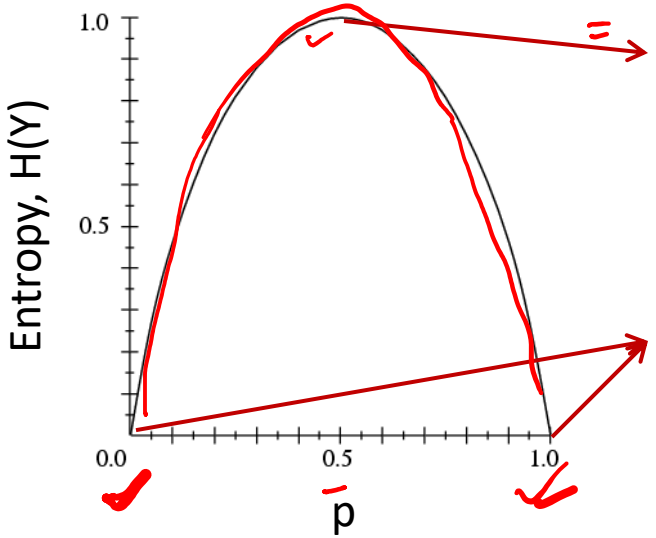
# Entropy

- Entropy of a random variable Y

$$H(Y) = -\sum_{y} P(Y = y) \log_2 P(Y = y)$$

***More uncertainty, more entropy!***

Y ~ Bernoulli(p)




- **Entropy**: $H(Y) = H(P)$ is the expected number of bits needed to encode a randomly drawn value of $Y \sim P$ under most efficient code optimized for distribution $P$

# Information Gain

- Advantage of attribute = decrease in uncertainty
  - Entropy of Y before split

$$H(Y) = -\sum_y P(Y = y) \log_2 P(Y = y)$$

  - Entropy of Y after splitting based on $X_i$
    - Weight by probability of following each branch

$$\begin{aligned} H(Y \mid X_i) &= \sum_x P(X_i = x) H(Y \mid X_i = x) \\ &= -\sum_x P(X_i = x) \sum_y P(Y = y \mid X_i = x) \log_2 P(Y = y \mid X_i = x) \end{aligned}$$

- Information gain is difference

$$I(Y, X_i) = H(Y) - H(Y \mid X_i)$$

**Max Information gain = min conditional entropy**

# Which feature is best to split?

Pick the attribute/feature which yields maximum information gain:

$$\arg\max_i I(Y, X_i) = \arg\max_i [H(Y) - H(Y|X_i)]$$

$$= \arg\min_i H(Y|X_i)$$

Entropy of Y $\quad H(Y) = -\sum_y P(Y=y)\log_2 P(Y=y)$

Conditional entropy of Y $\quad H(Y \mid X_i) = \sum_x P(X_i = x) H(Y \mid X_i = x)$

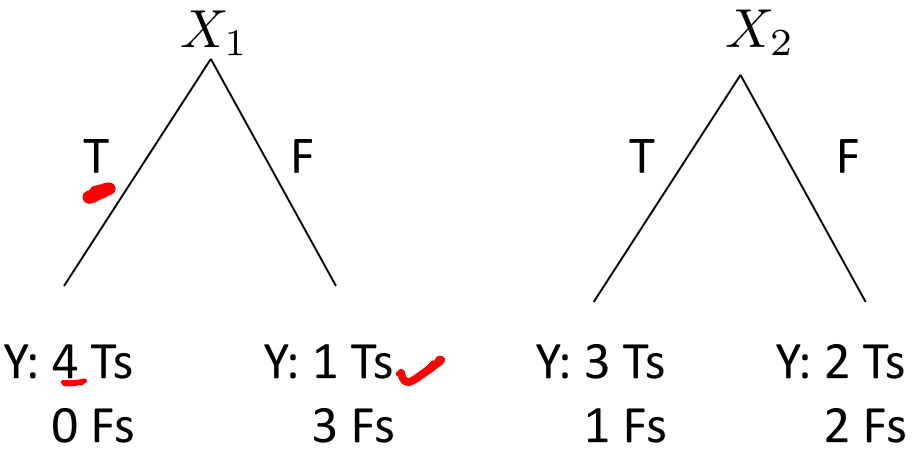
Feature which yields maximum reduction in entropy (uncertainty) provides maximum information about Y

# Information Gain

$$H(Y \mid X_i) = -\sum_x P(X_i = x) \sum_y P(Y = y \mid X_i = x) \log_2 P(Y = y \mid X_i = x)$$

$H(Y|X_i = x)$

| $X_1$ | $X_2$ | Y |
|---|---|---|
| T | T | T |
| T | F | T |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |
| F | T | F |
| F | F | F |




$H(Y|X_1 = T)$

$1 \cdot \log_2 1 + 0 \cdot \log_2 0 = 0$

$H(Y|X_2 = T)$

$H(Y|X_1 = F)$

$\frac{1}{4} \log_2 \frac{1}{4} + \frac{3}{4} \log_2 \frac{3}{4}$

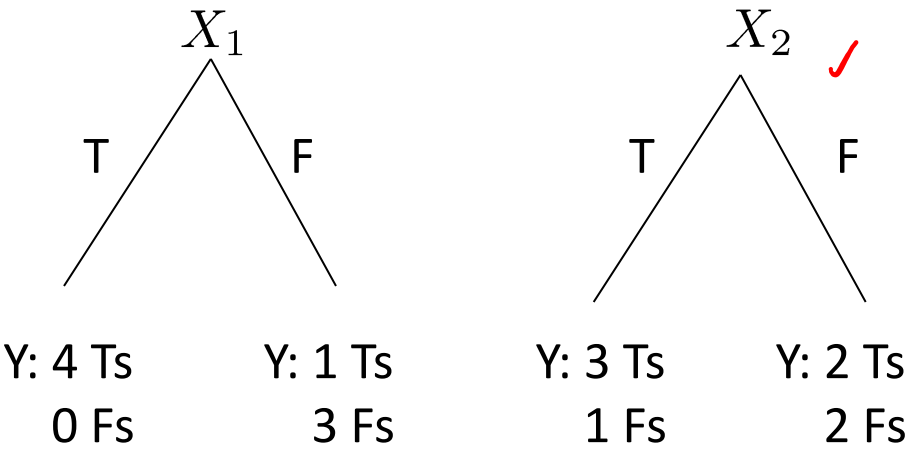
$H(Y|X_2 = F)$

# Information Gain

$$H(Y \mid X_i) = -\sum_x P(X_i = x) \sum_y P(Y = y \mid X_i = x) \log_2 P(Y = y \mid X_i = x)$$

$H(Y|X_i=x)$

| $X_1$ | $X_2$ | Y |
|---|---|---|
| T | T | T |
| T | F | T |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |
| F | T | F |
| F | F | F |




$$\widehat{H}(Y|X_1) = -\frac{1}{2}[1\log_2 1 + 0\log_2 0] - \frac{1}{2}[\frac{1}{4}\log_2\frac{1}{4} + \frac{3}{4}\log_2\frac{3}{4}]$$

$$\widehat{H}(Y|X_2) = -\frac{1}{2}[\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}] - \frac{1}{2}[\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}]$$

$> 0$

$$\widehat{H}(Y|X_1) < \widehat{H}(Y|X_2)$$

# How to learn a decision tree

- Top-down induction [ID3, C4.5, C5, …]

Main loop: C4.5

1. $X \leftarrow$ the "best" decision feature for next *node*
2. Assign $X$ as decision feature for *node*
3. For "best" split of $X$, create new descendants of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes
6. Prune back tree to reduce overfitting
7. Assign majority label to the leaf node