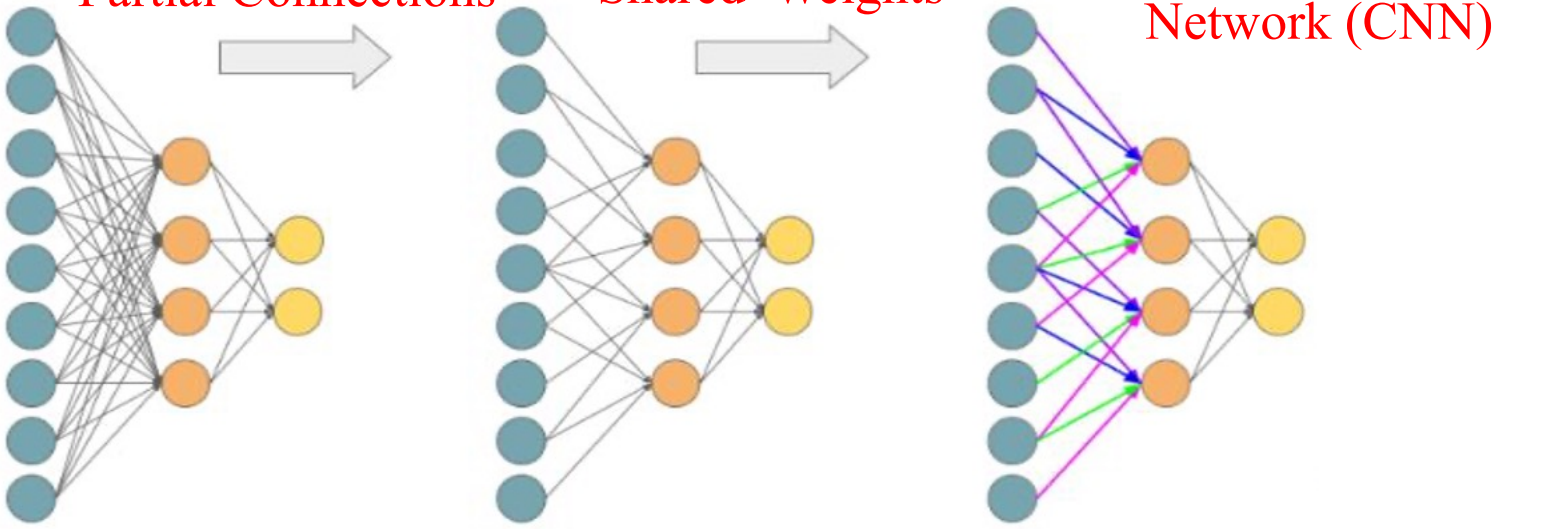
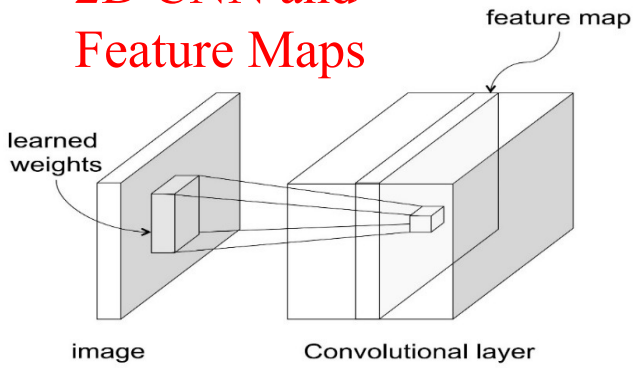
Convolution Neural Networks and Object Classification

Jenq-Neng Hwang, Professor
Department of Electrical & Computer Engineering
University of Washington, Seattle WA
hwang@uw.edu

Convolution Neural Network



2D CNN and Feature Maps

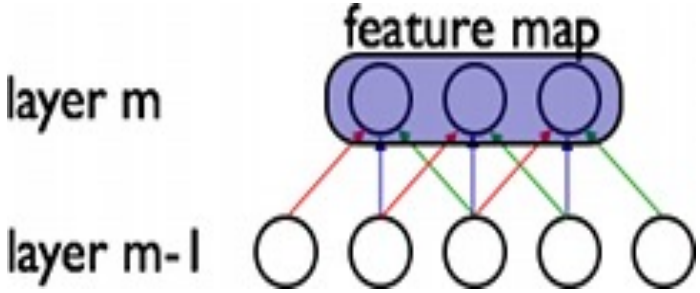


Backpropagation Learning for MLP can still be applied

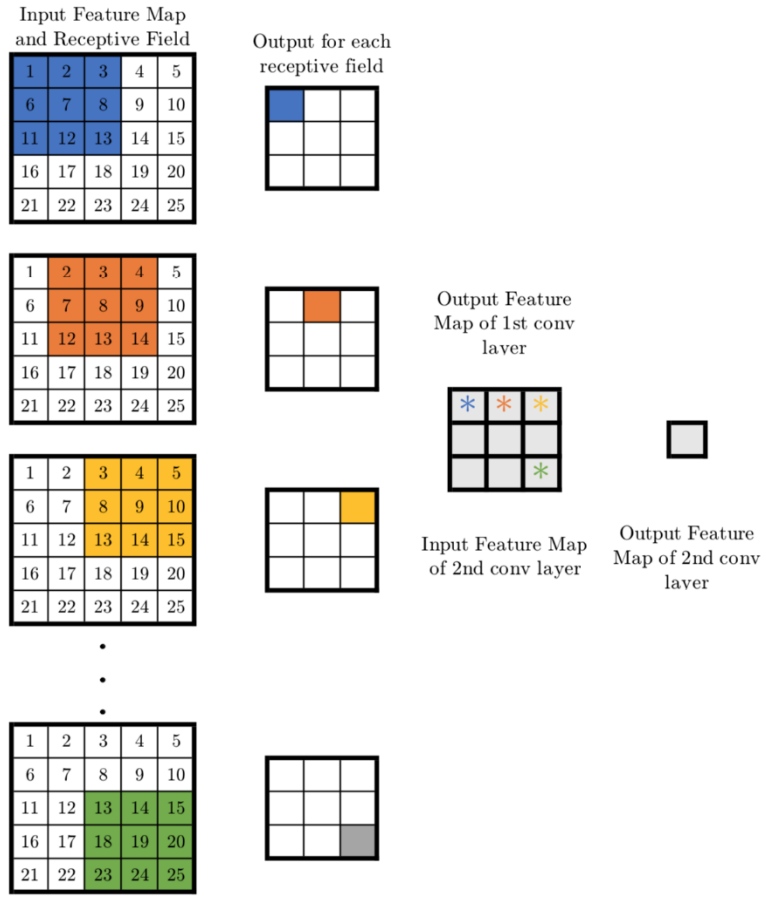
Partial Connections & Shared Convolutional Kernels

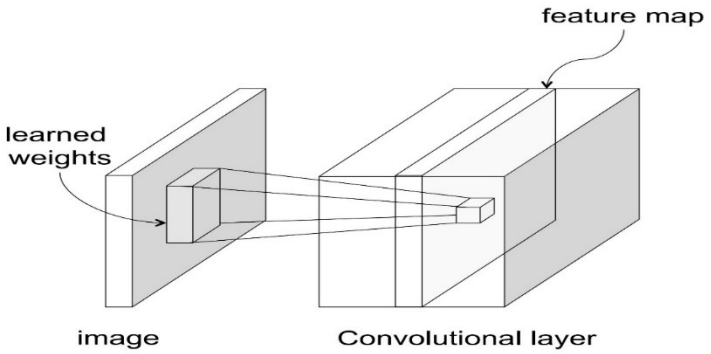
- Partial connections imply only local (locally connected) neurons are more correlated and dependent on one another.
- Replicating units in this way allows for features to be "detected" regardless of their position in the visual field.
- Additionally, weight sharing increases learning efficiency by greatly reducing the number of learned free parameters.
- The constraints on the model enable CNNs to achieve better generalization on vision problems.

Convolution & Pooling
Shared weights and simplified nonlinearity [LeCun 1989]



2D Convolution Example



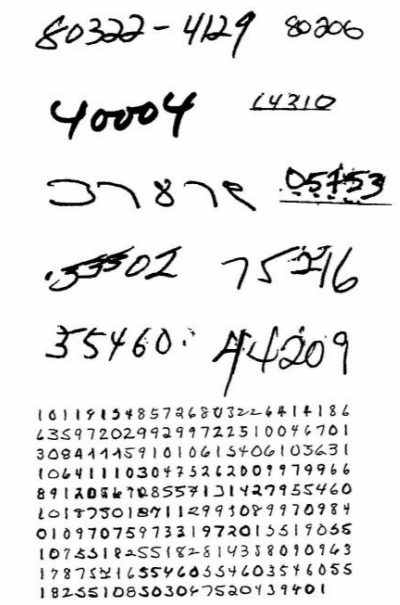


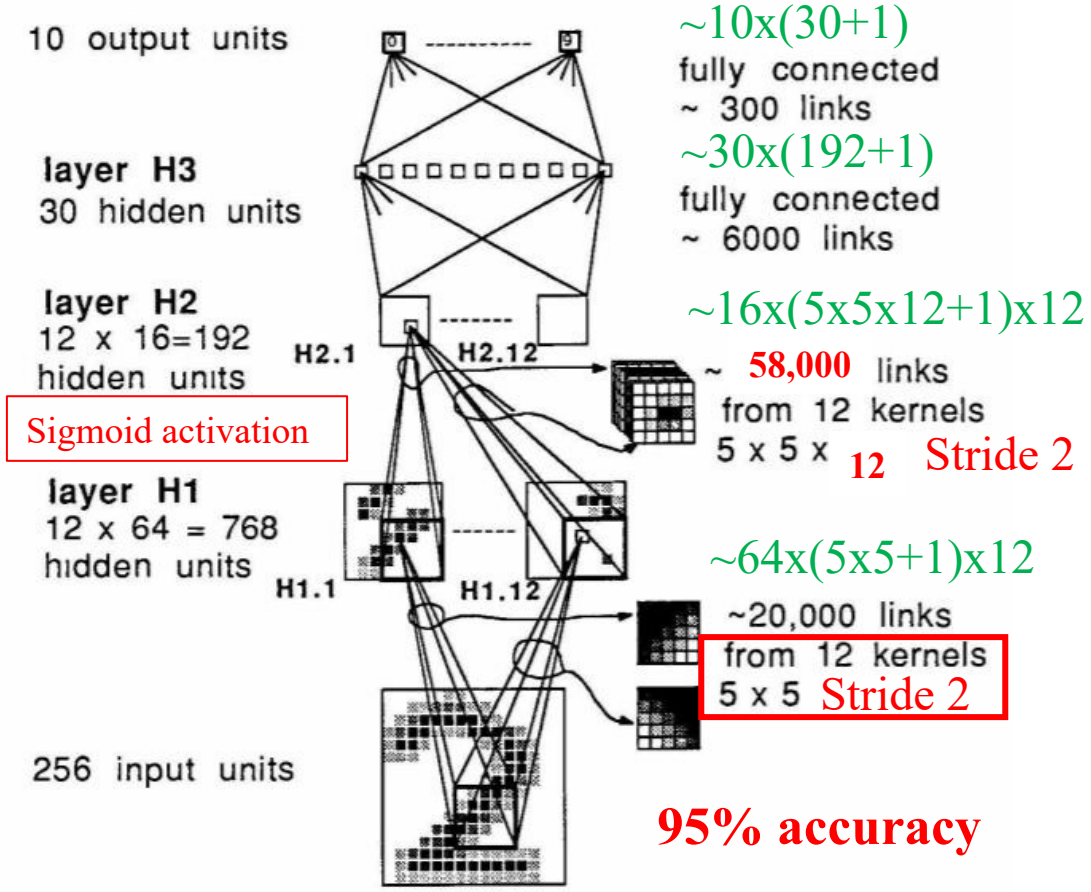
$F_m(k1,k2)=x(n1,n2)*w_m(n1,n2)$

- Stride
- Padding
- Pooling (Subsampling)

Let the convolution kernels $w_m(n1, n2)$ learnable in CNNs

LeNet for Handwritten Digits (1989)



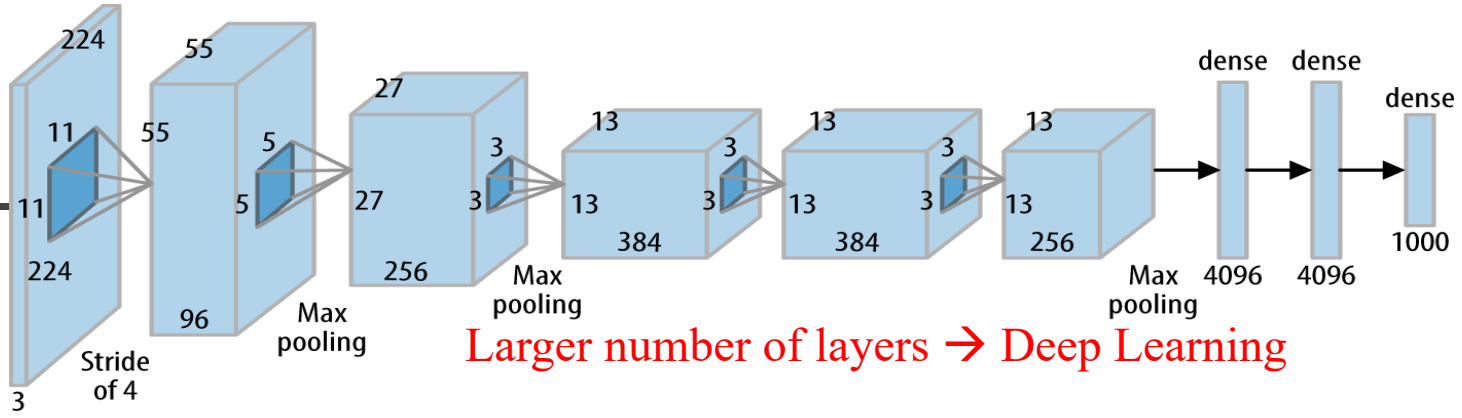


- For layer H1: 64x12=768 hidden units, 768x256=199608 connections (if fully connected), but only 1068 distinct trainable weights (25x12+768 biases)

Y. LeCun, et al, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, Winter 1989.

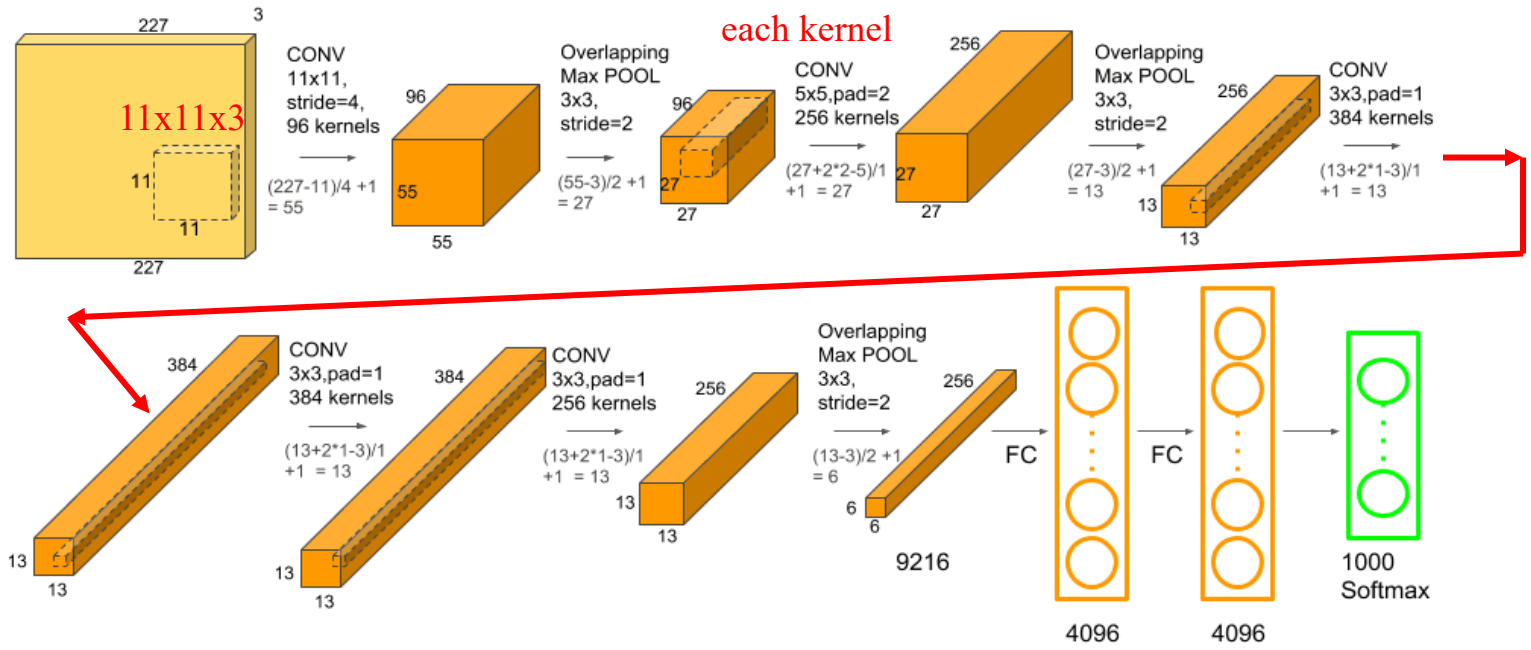
Ice Age of Neural Network based Learning 1992-2012

AlexNet for ImageNet



Larger number of layers → Deep Learning

5x5x96 for each kernel



Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, et al. "ImageNet Classification with Deep Convolutional Neural Networks," NIPS 2012.

AlexNet Parameters

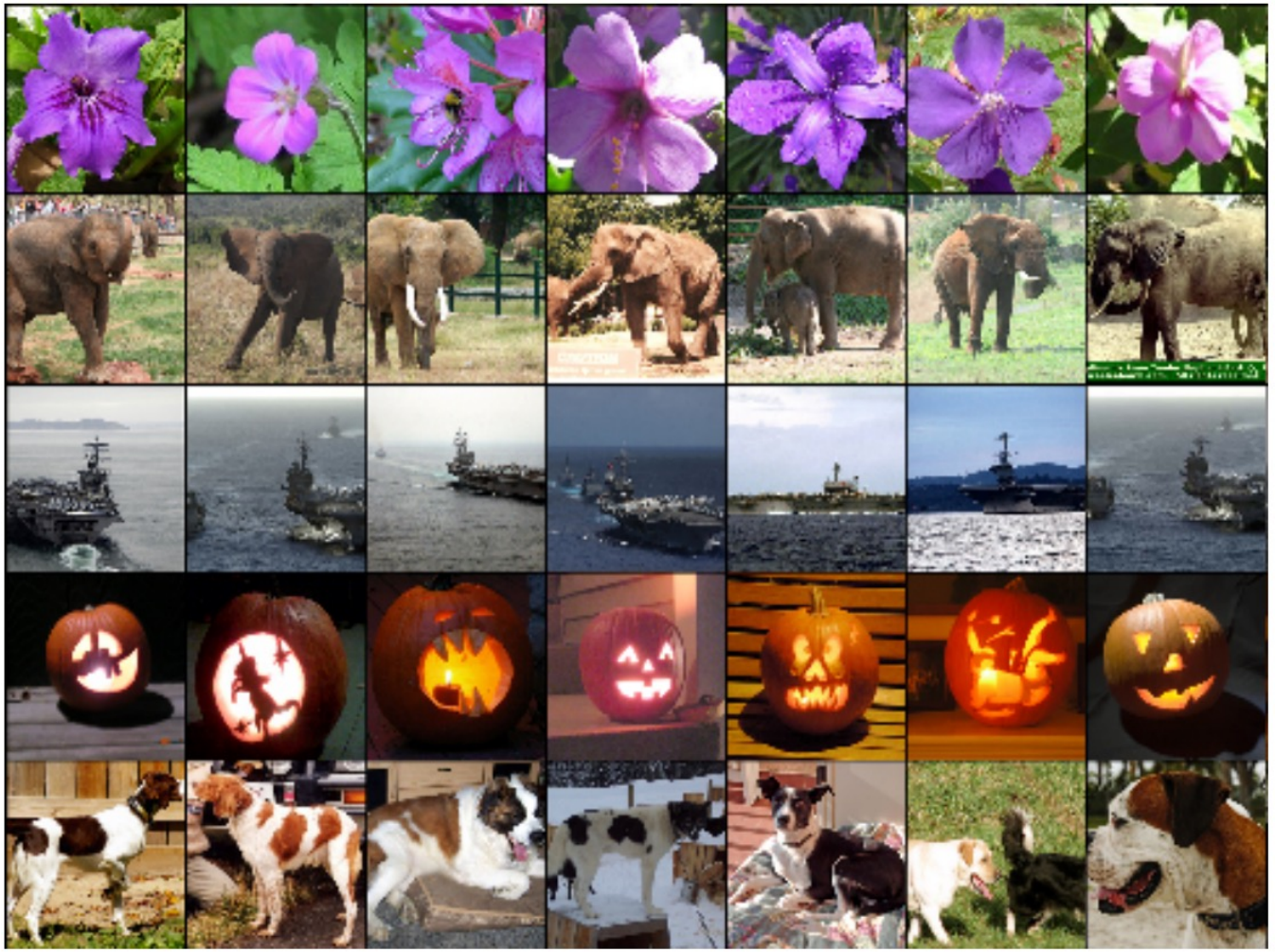
Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
-	-	-	-	-	-	-
-	-	-	-	-	-	-
-	-	-	-	-	-	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	11x11x3x96 34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	5x5x96x256 614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	3x3x256x384 884,736	384	885,120
Conv-4	13x13x384	3x3x384x384 1,327,104	384	1,327,488
Conv-5	13x13x256	3x3x384x256 884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	6x6x256x4096 37,748,736	4,096	37,752,832
FC-2	4096×1	4096x4096 16,777,216	4,096	16,781,312
FC-3	1000×1	4096x1000 4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

ImageNet and ILSVRC

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



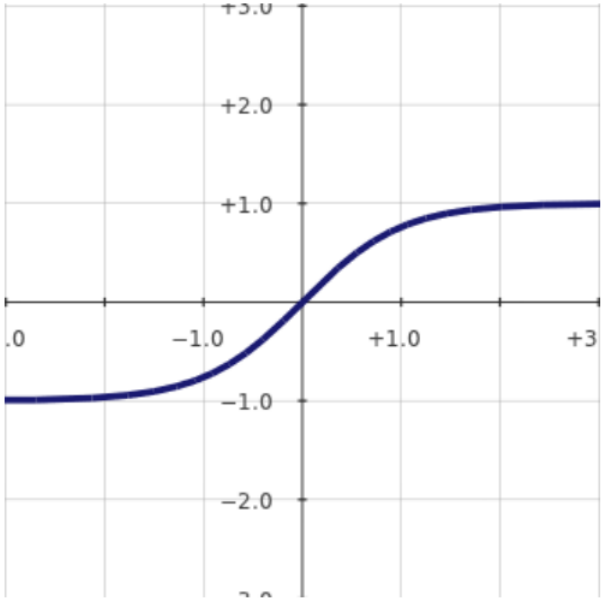
Deep Learning for AlexNet

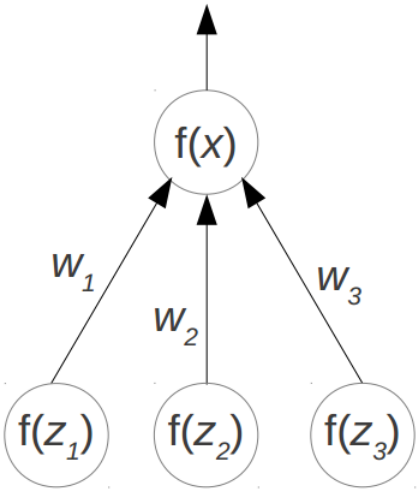
- Deep Convolutional Neural Network
  - 5 convolutional and 3 fully connected layers
  - 650,000 neurons, 60 million (62,378,344) parameters, 630 millions connections

- Some techniques for boosting up performance (top 5 accuracy)
  - SoftMax and Cross-Entropy Loss
  - ReLU nonlinearity for convolution layers (6x faster)
  - Overlapping Max pooling (0.5% error reduced)
  - Batch Normalization (1.2% error)
  - Data augmentation (spatial random crop, 4 corners+center & reflection, RGB perturbation) → x2048 (1% error reduced)
  - Dropout (less local minimum)

Rectified Linear Units (ReLU)

f(x) = tanh(x)





$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$

x is called the total input to the neuron, and f(*x*) is its output

f(x) = max(0, x)

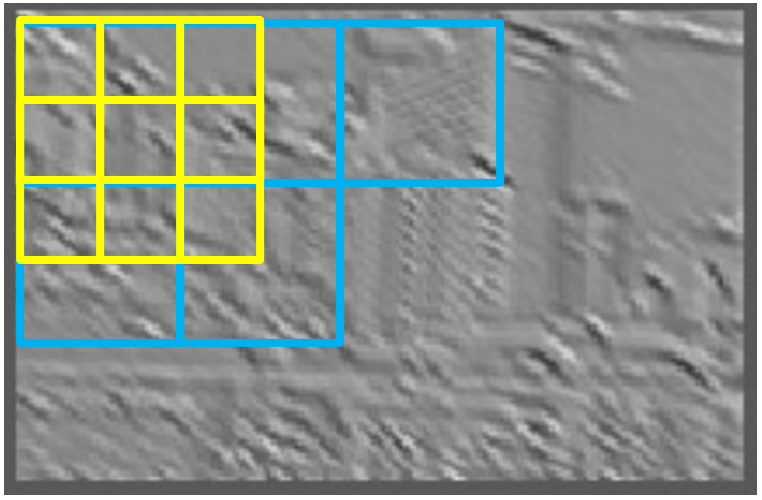
Very bad (slow to train)

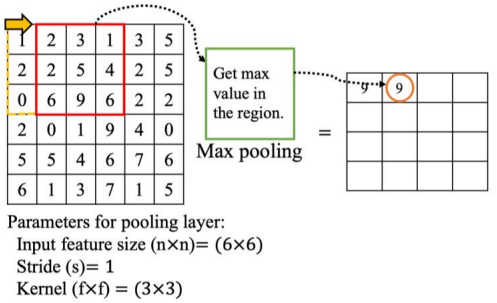
Very good (quick to train)

reference : http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf

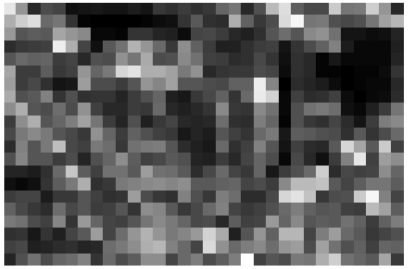
Maximum Pooling

- Spatial Pooling
  - Non-overlapping/overlapping
  - Sum or max (3x3 size)

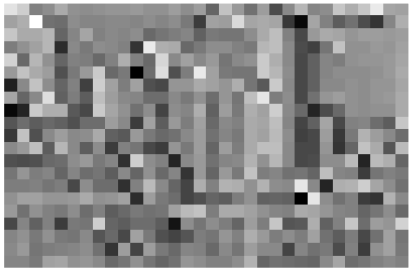




Max



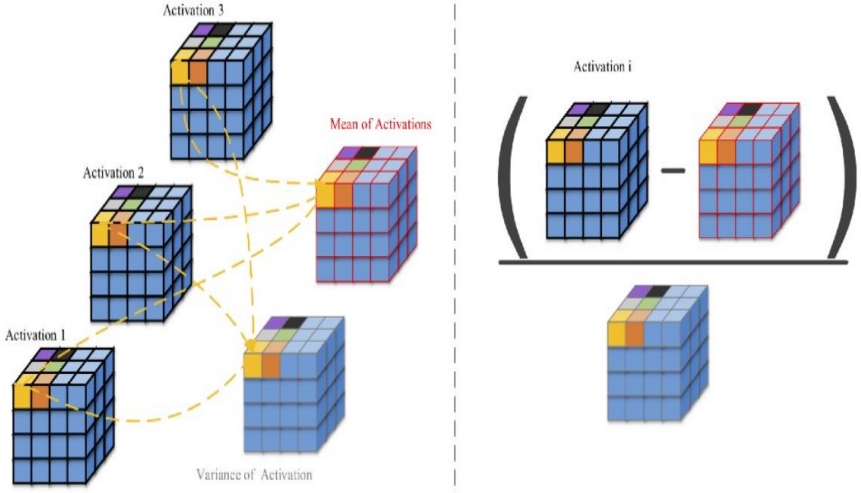
Sum



reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Batch Normalization (BN)

- To limit the unbounded activation from increasing the output layer values, normalization is used just before the activation function (e.g., ReLU) → accelerates training, better generalization.
- The normalization is carried out for each pixel across all the activations in a batch.



e.g., Batch of 3

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

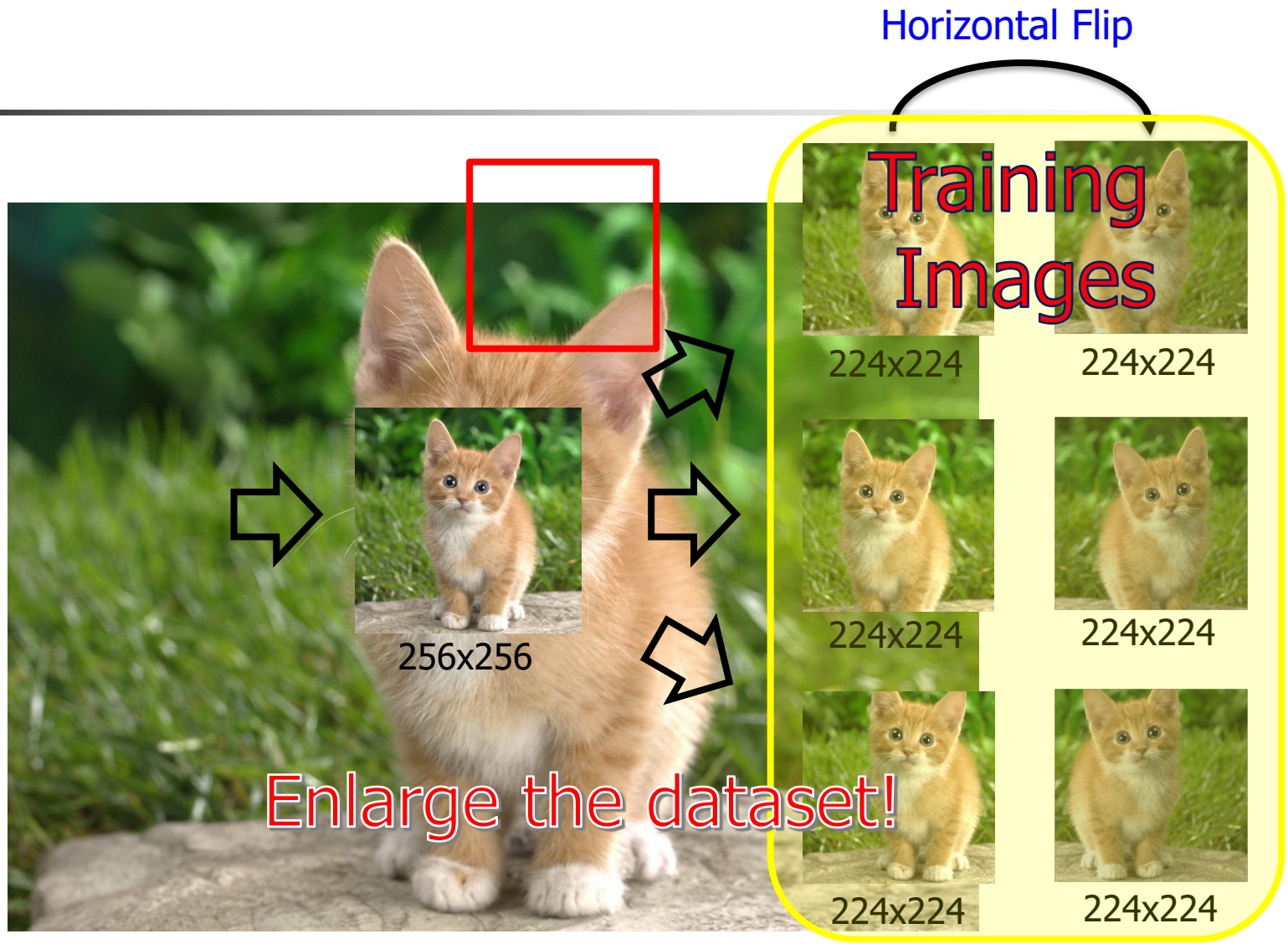
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

Data Augmentation



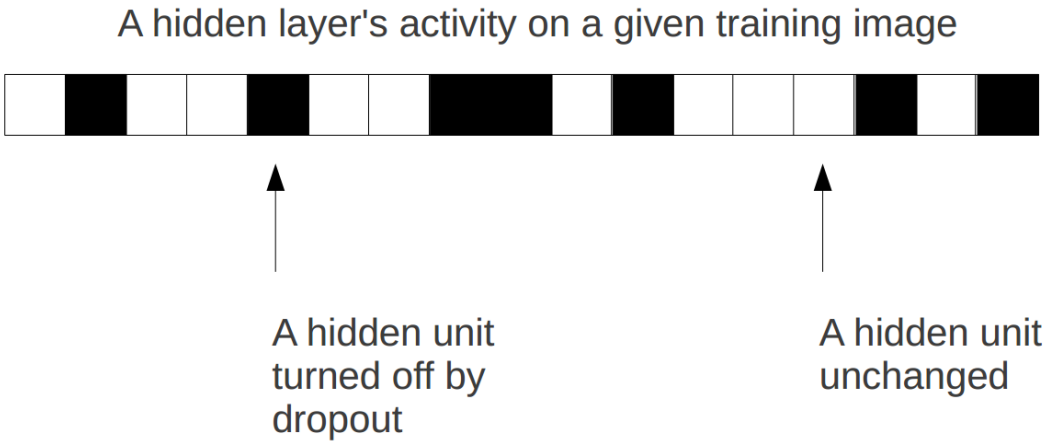
Color Data Augmentation

- Alter the intensities of the RGB channels in training images
- Perform PCA on the set of RGB pixel values
- To each training image, add multiples of the found 3 principal components to each RGB image pixel $I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T$ with the following quantity $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$

- $\mathbf{p}_i$, λ_i : i-th eigenvector and eigenvalue
- α_i : random variable drawn from a Gaussian with mean 0 and standard deviation 0.1
- This reduces top-1 error rate by over 1%

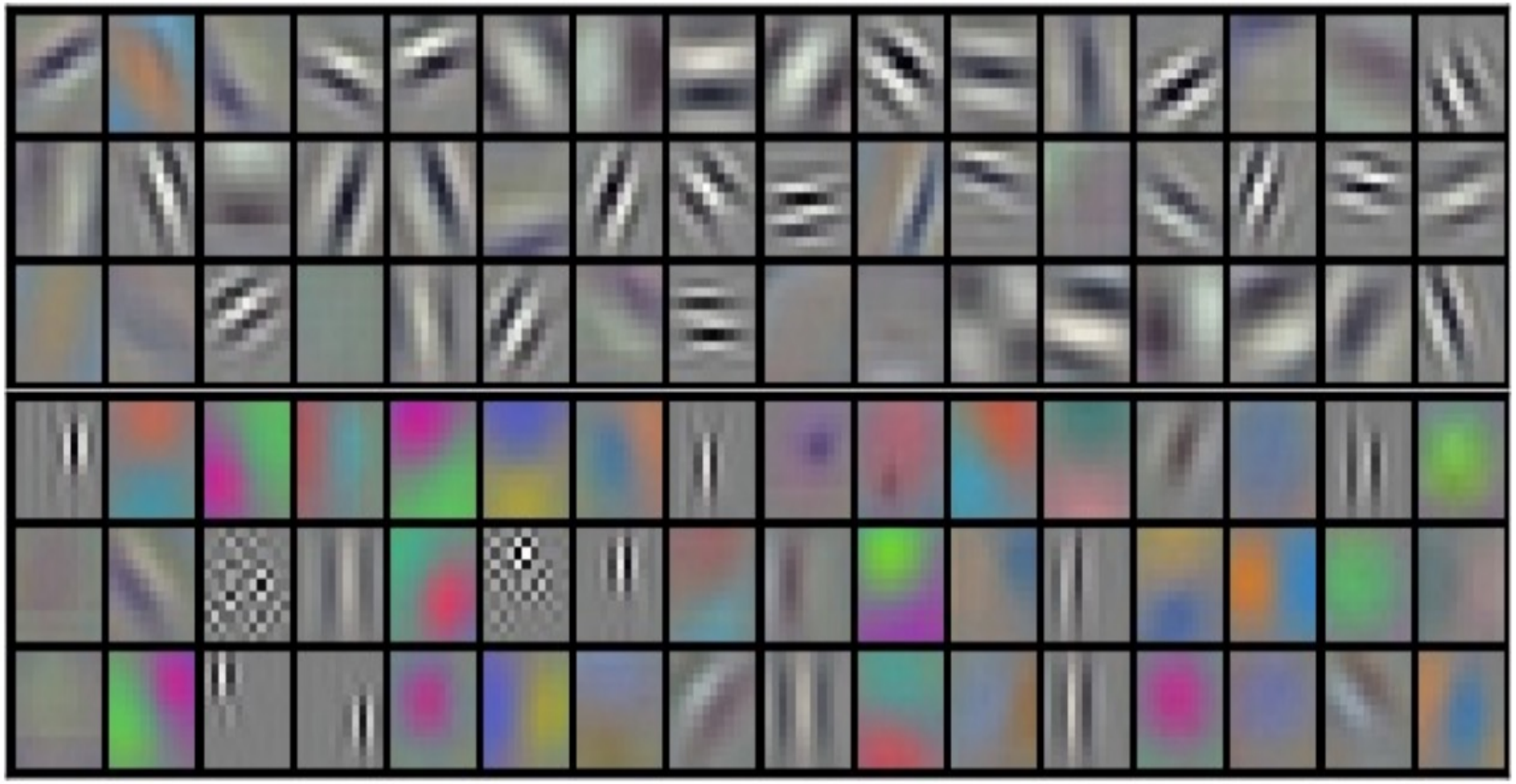
Dropout

- Independently set each hidden unit learning activity to zero with 0.5 probability for every training image
- Used in the two globally-connected hidden layers at the network's output



Exemplar Kernel Filters

- 96 learned low-level (1st layer) *11x11x3* kernels



Convolution Outputs

Input

Feature Map

reference :http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Details of CNN Learning

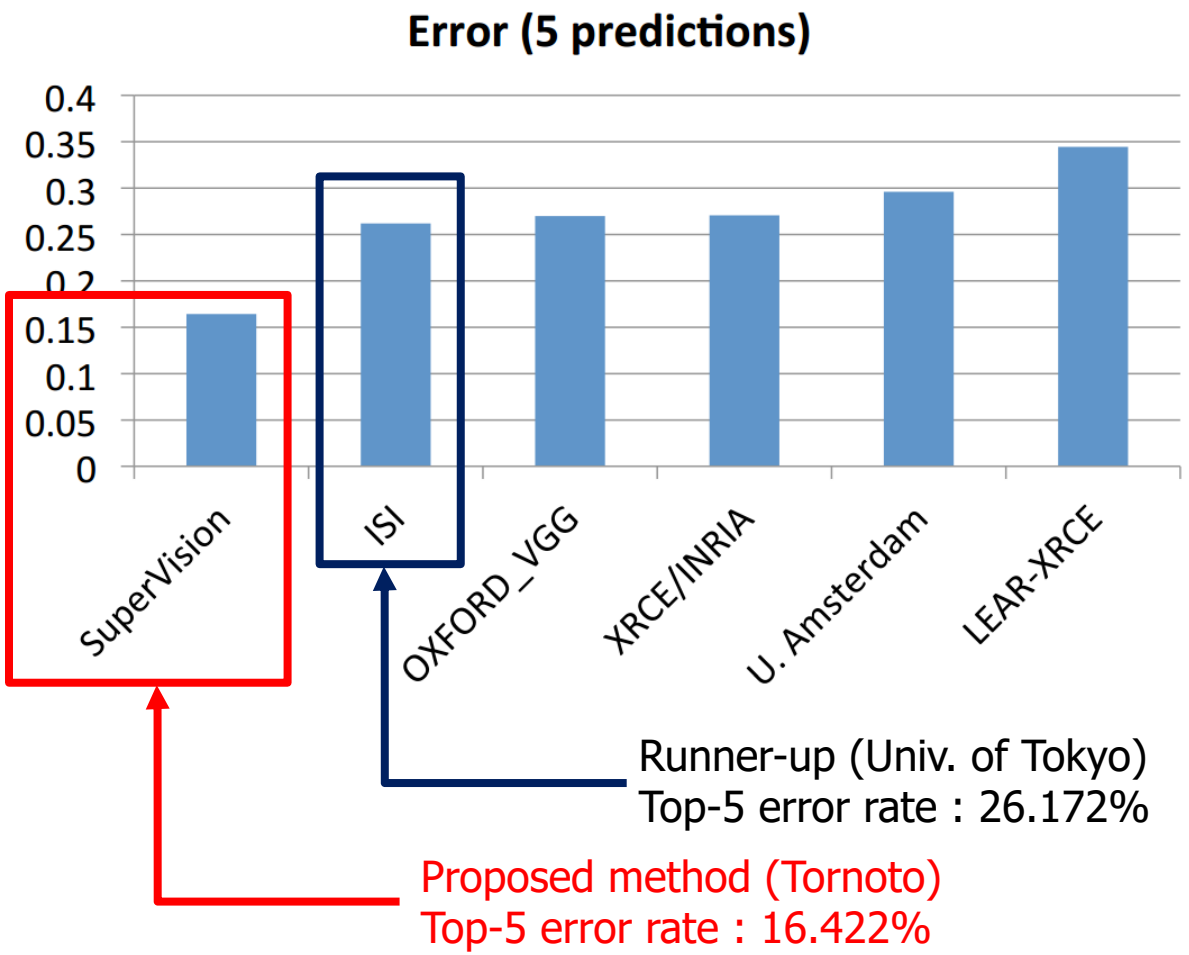
- Use stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weigh decay of 0.0005
- The update rule for weight w was

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

- Train for 90 cycles through the training set of 1.2 million images

ILSVRC 2012 Results



reference : http://image-net.org/challenges/LSVRC/2012/ilsvrc2012.pdf

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

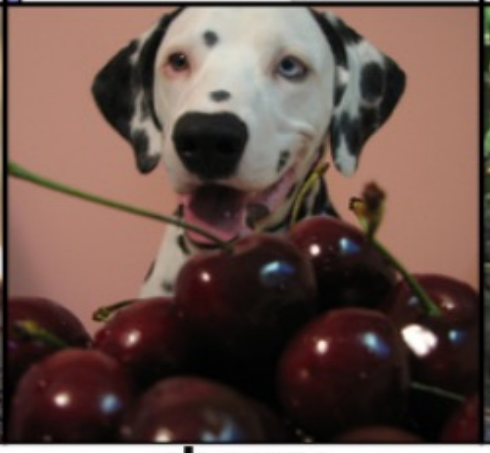
leopard
jaguar
cheetah
snow leopard
Egyptian cat

grille

convertible
grille
pickup
beach wagon
fire engine

mushroom

agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers



cherry

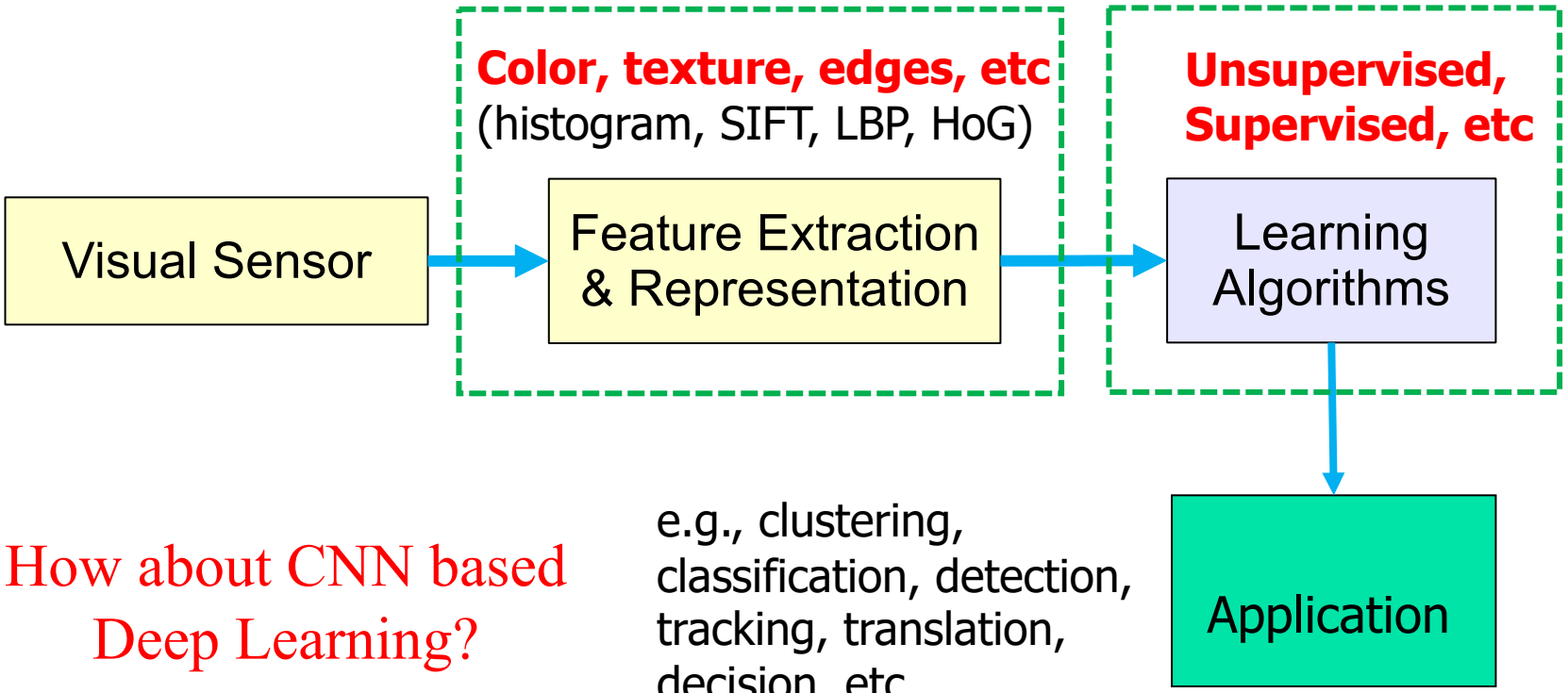
dalmatian
grape
elderberry
ffordshire bullterrier
currant

Madagascar cat

squirrel monkey
spider monkey
titi
indri
howler monkey

Recall Traditional ML

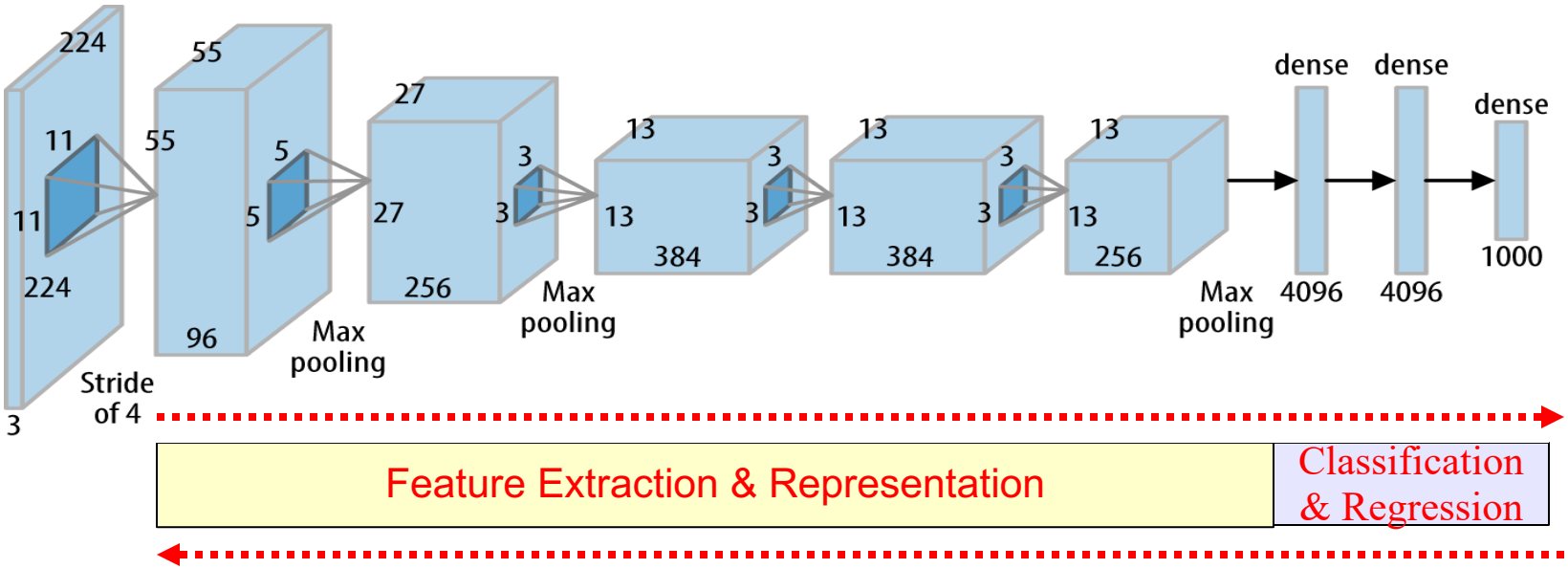
Recall the two-stage sequential machine learning



How about CNN based Deep Learning?

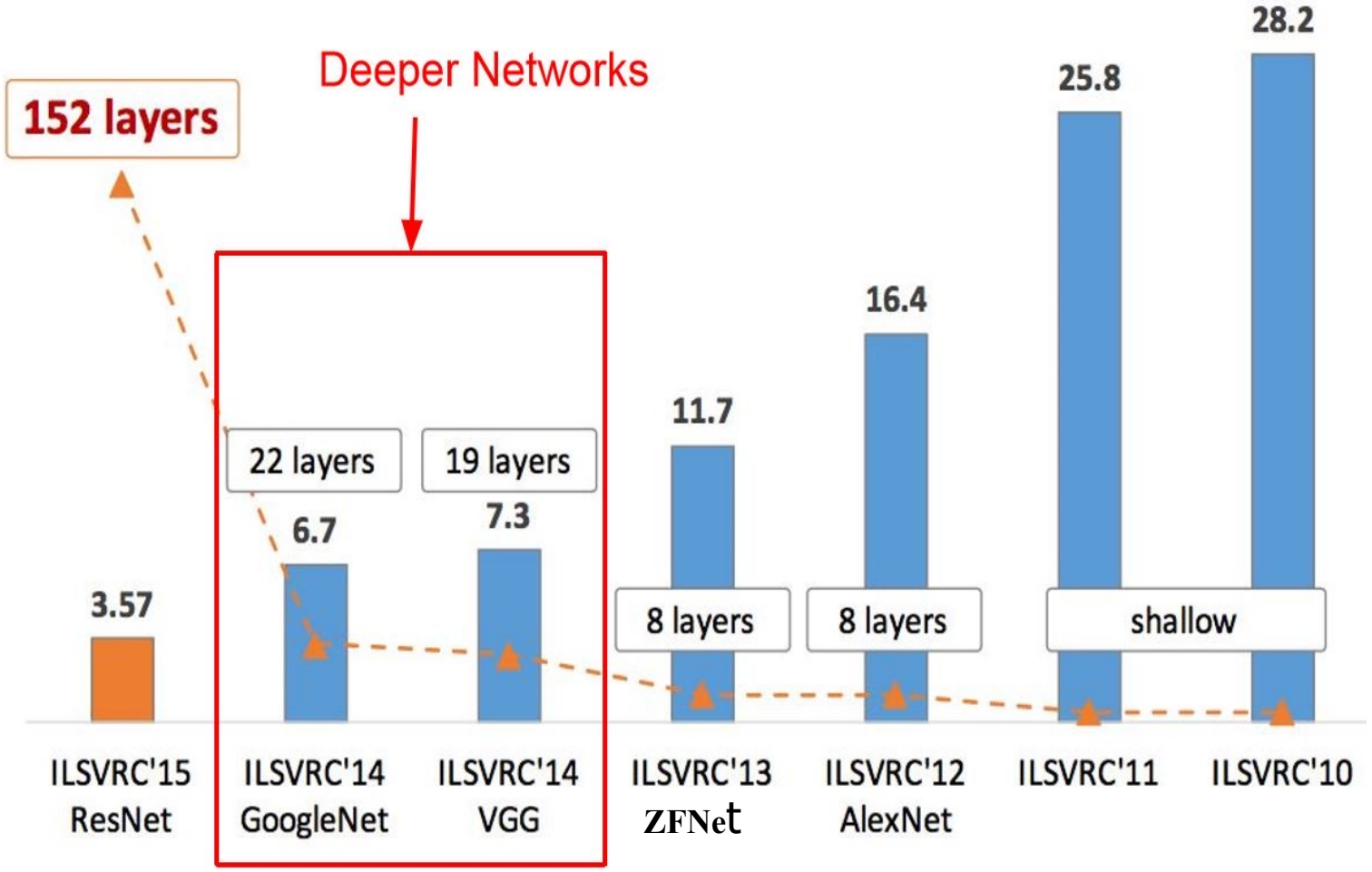
e.g., clustering, classification, detection, tracking, translation, decision, etc

Why CNNs Are Better?



A single-stage (end-to-end) joint machine learning (enough big training data, powerful computing resources)

Success of CNNs for ILSVRC

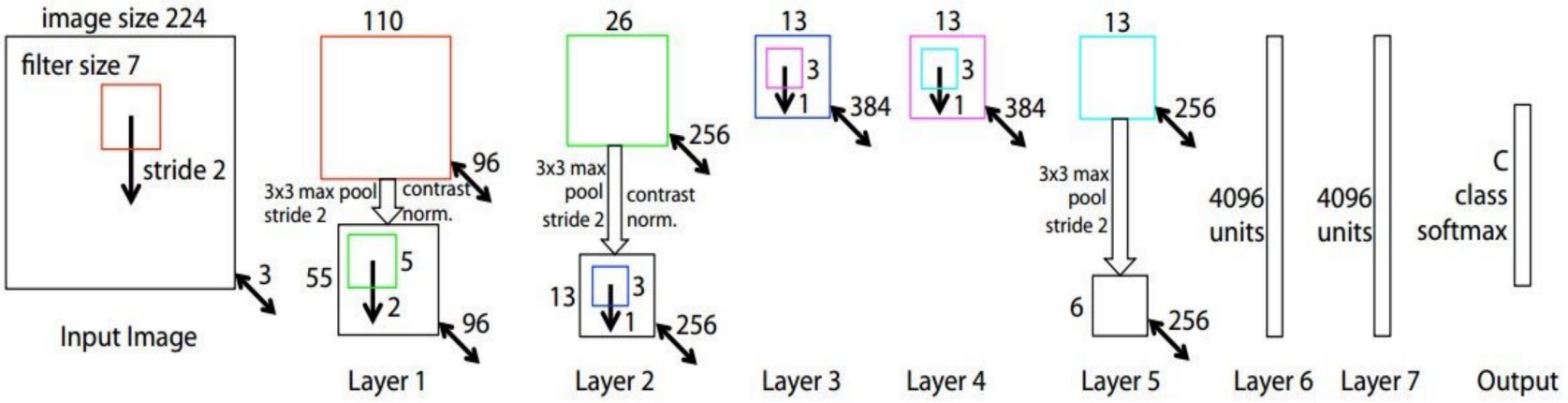


ZFNet, ILSVRC 2013 Winner

- Similar to 8-layer AlexNet but:
  - CONV1: change from (11x11 stride 4) to (7x7 stride 2)
  - CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ZFNet

[Zeiler and Fergus, 2013] NYU



Oxford VGG Net

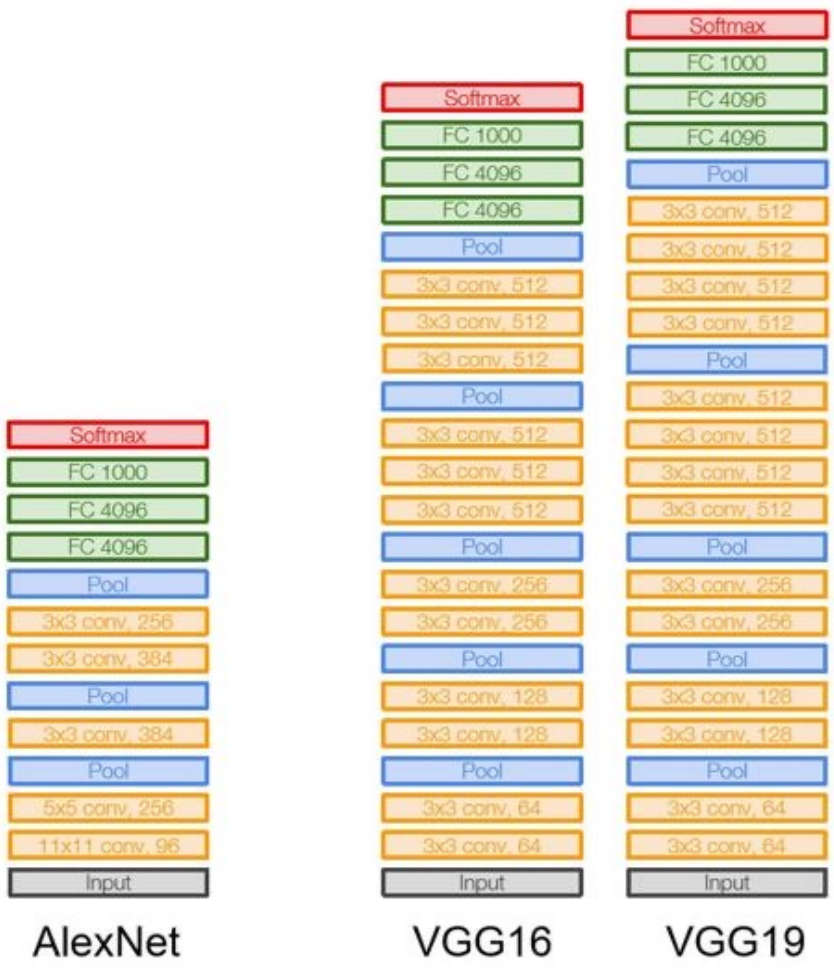
Small filters, Deeper networks

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

[Simonyan and Zisserman, 2014]

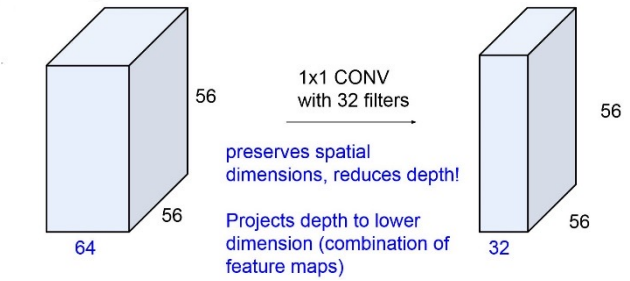


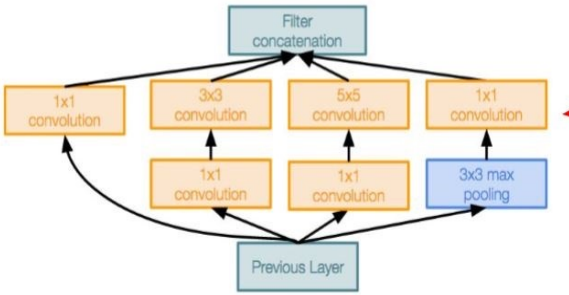
GoogLeNet

Deeper networks, with computational efficiency

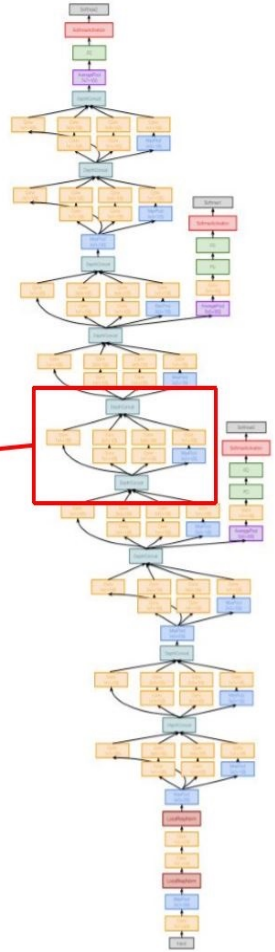
- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)

[Szegedy et al., 2014]





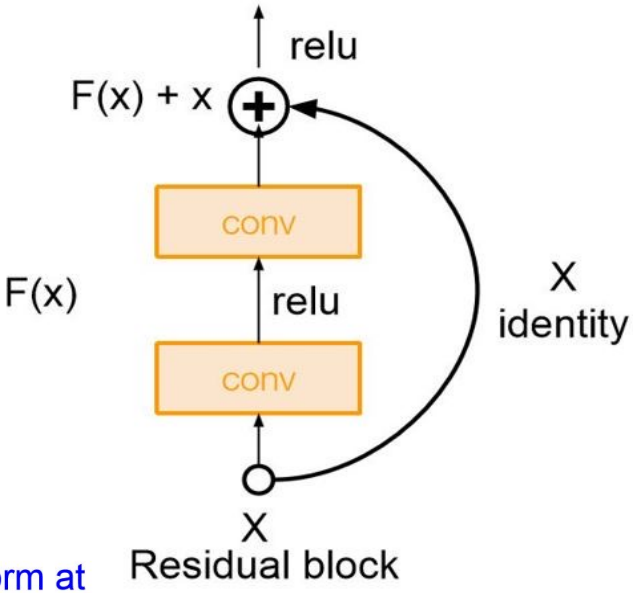
Inception module



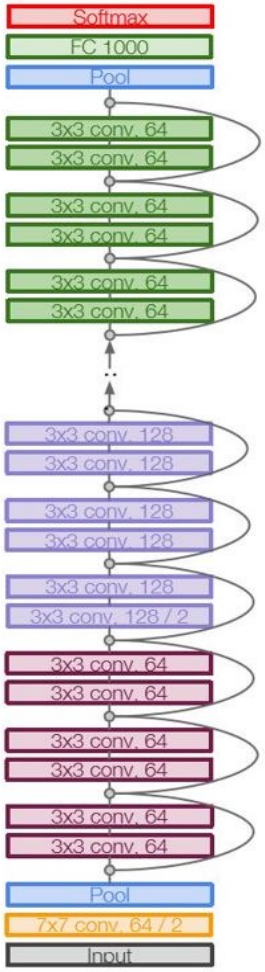
Microsoft ResNet

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!



[He et al., 2015]

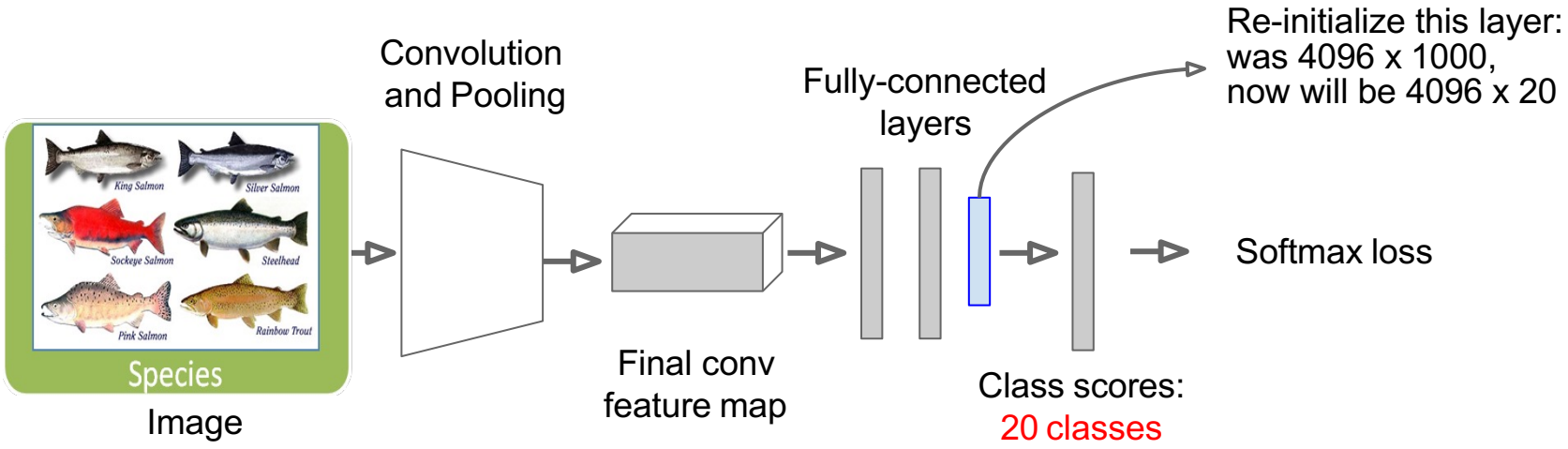


The deeper model should be able to perform at least as well as the shallower model.

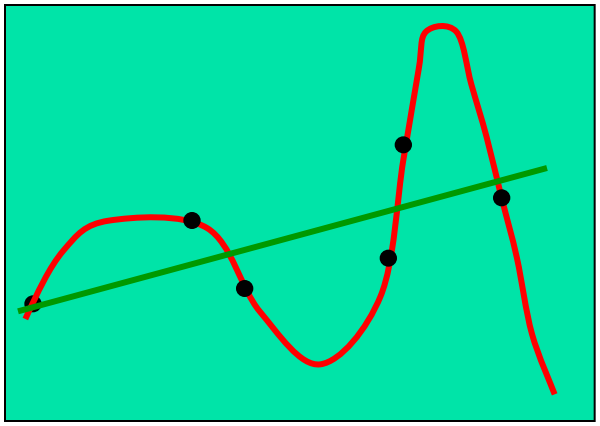
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

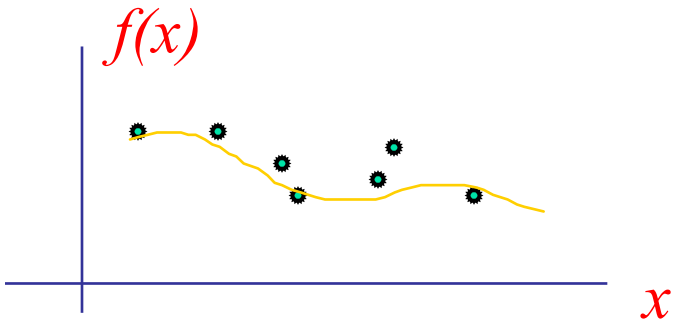
New Task Transfer Learning

- Transfer a pre-trained model (e.g., AlexNet, VGG) for a new task
- For example, instead of 1000-class ImageNet classification task, which has been a successful feature extraction architecture, we want transfer to a 20-class task
  - Throw away final fully-connected layer, reinitialize from scratch
  - Keep training model using the labelled data from the new task



Object Classification Overfitting





- Which model do you believe?
  - The complicated model fits the data better (can be noise)
- The objective of learning is to achieve good *generalization* to new cases, otherwise just use a look-up table.
- Generalization can be defined as a mathematical *interpolation* or *regression* over a set of training points:

Cross Validation

- Hold-out cross-validation (early stopping): split the dataset T into three mutually disjoint subsets – training, validation, and testing.
  - The model is trained on the training subset, while the validation subset is periodically used to evaluate the model performance during the training to avoid over-training. The training is stopped, when the performance on validation subset is good enough or when it stops improving.
  - When comparing $m > 1$ computational models $L_1, \cdots, L_m$ against each other, the testing subset is used to evaluate the models' performance.
- K-Fold cross-validation: Divide T into k parts of the same size. One part forms the validation (testing) set, the other parts form the training set. This process is repeated for each validation part of the data.

Relevant \ Selected	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive(FP)	True Negative (TN)

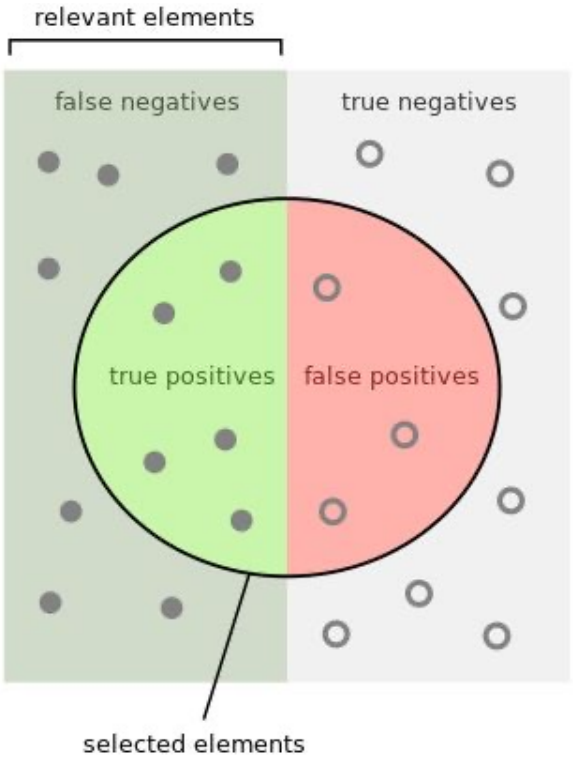
Assume only two classes

Classification Performance Metrics

- **Accuracy (ACC)**

ACC=(TP+TN)/(TP+FP+FN+TN)

outcome prediction



What is Wrong with Accuracy Alone?

- A 2-category classifier (imbalanced):

accuracy = (10 + 100)/(10 + 5 + 15 + 100) = **84.6%**

- A dumb "negative" classifier: (when TP < FP)

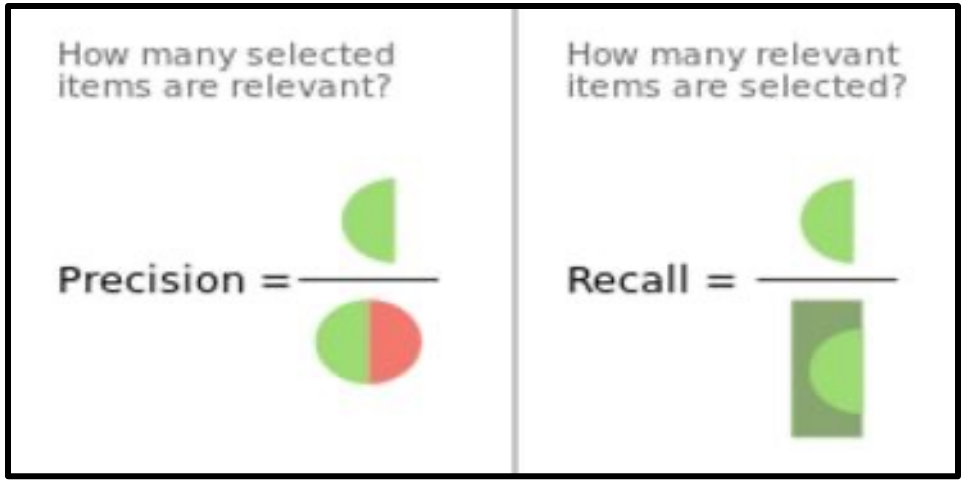
accuracy = (0 + 115)/(0 + 15 + 0 + 115) = **88.5%**.

	Classified positive	Classified negative
Positive class	10	5
Negative class	15	100

	Classified positive	Classified negative
Positive class	0	15
Negative class	0	115

Precision and Recall

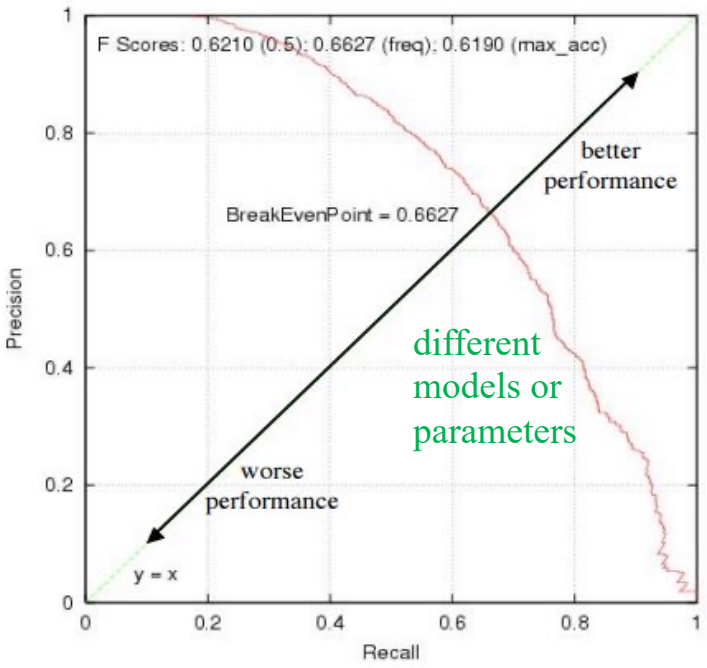
- Precision: positive predictive value, the correct fraction out of all the examples the classifier predicted as positive
  Pre = TP / (TP + FP) (no cancer → predict cancer)
- Recall: true positive rate, also called sensitivity, hit rate, the correct fraction out of all the positive examples
  Rec = TP / (TP + FN)
  (with cancer → predict no cancer)



Pre & Rec Tradeoffs

- A perfect recall (zero FN simply label all the examples as positive) → horrible precision
- Increase precision (low FP, only label the most certain examples as positive) → horrible recall
- Need to optimize a measure that combines precision and recall into a single value, such as the F1 Score.

Average Precision (AP) = area under the red curve



F1 Score (F1 Measure)

- F1 score (balanced F-score) can be interpreted as a weighted average (harmonic mean) of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

- This measure is approximately the average of the two when they are close, and is more generally the square of the geometric mean divided by the arithmetic mean.

Multi-Class Precision & Recall

predicted

actual	Cat	Fish	Chicken
Cat	7	1	3
Fish	8	2	2
Chicken	9	3	1

- Accuracy: (7+2+1)/(7+1+3+8+2+2+9+3+1)=10/36=0.28
- For Cat
  - TP=7, TN=2+3+2+1=8, FP=8+9=17, FN=1+3=4
  - Precision=7/(7+17)=0.29; Recall=7/(7+4)=0.64; F1-score=0.40

Class	Precision	Recall	F1-Score
Cat	0.29	0.64	0.40
Fish	0.33	0.17	0.22
Chicken	0.17	0.08	0.11

Multi-Class Average Precision

Class	Precision	Recall	F1-Score
Cat	0.29	0.64	0.40
Fish	0.33	0.17	0.22
Chicken	0.17	0.08	0.11

- Micro-Average Precision (in case of imbalance)

$$\frac{TP_{cat} + TP_{fish} + TP_{chicken}}{TP_{cat} + TP_{fish} + TP_{chicken} + FP_{cat} + FP_{fish} + FP_{chicken}}$$

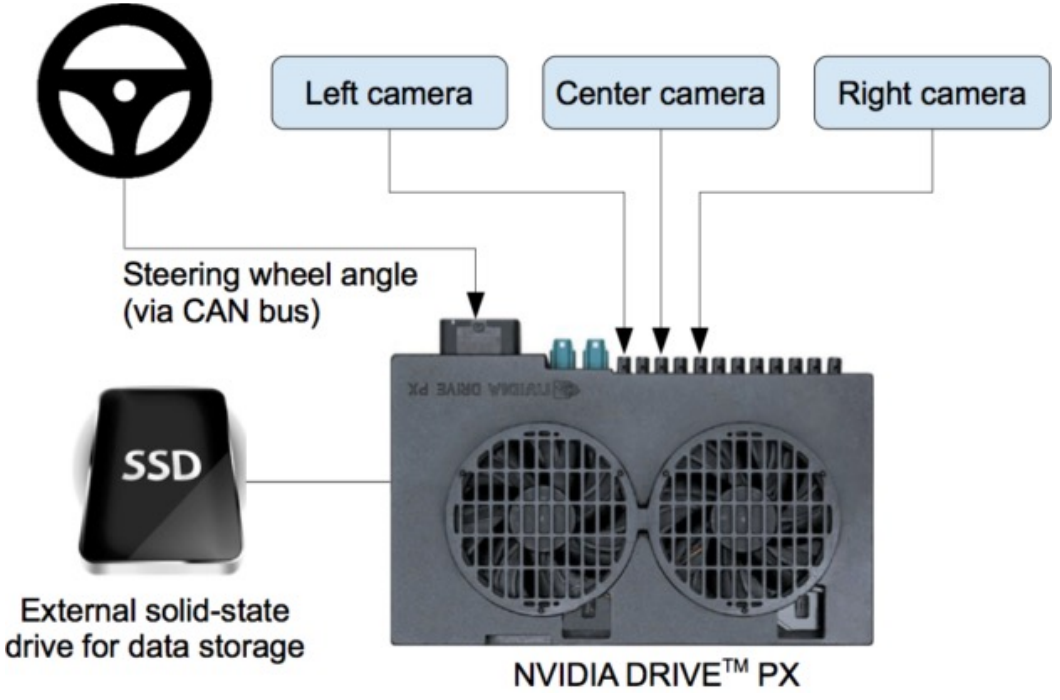
- Macro-Average Precision (insensitive to class imbalance)

$(0.29+0.33+0.17)/3=0.26$

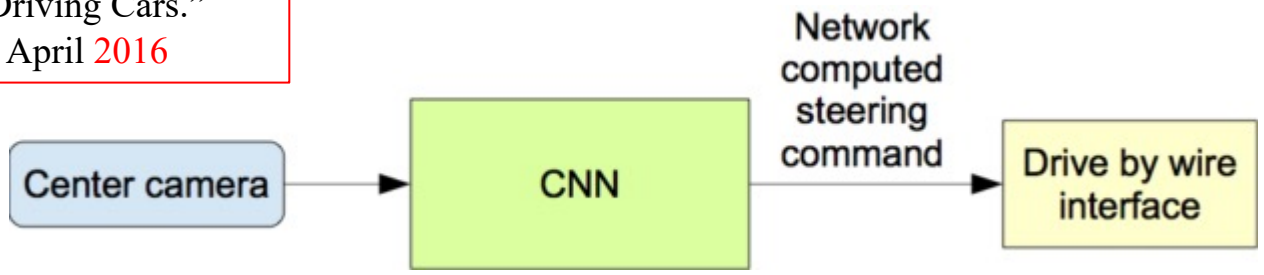
CNNs for Regressions too

Nvidia DAVE-2 (Early Days' Autonomous Vehicle)

CAN (controller area network) bus allows electronic control units and devices to communicate with each other.

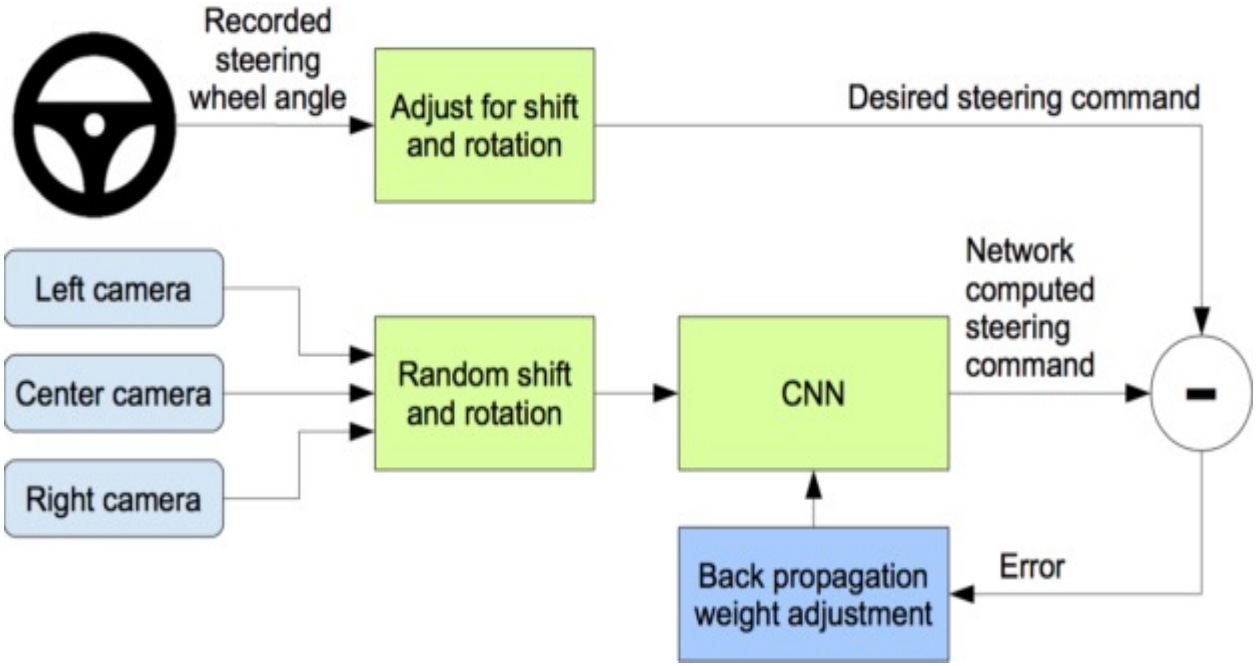


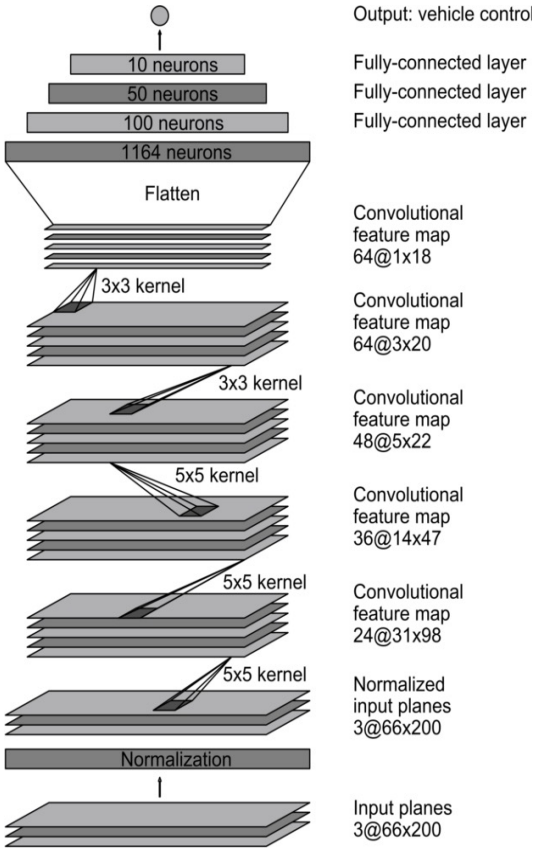
Mariusz Bojarski, et al. "End to End Learning for Self-Driving Cars." arXiv:1604.07316, April 2016



Training the CNN

- To learn how to recover from mistakes, the training data is augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road (72 hours).





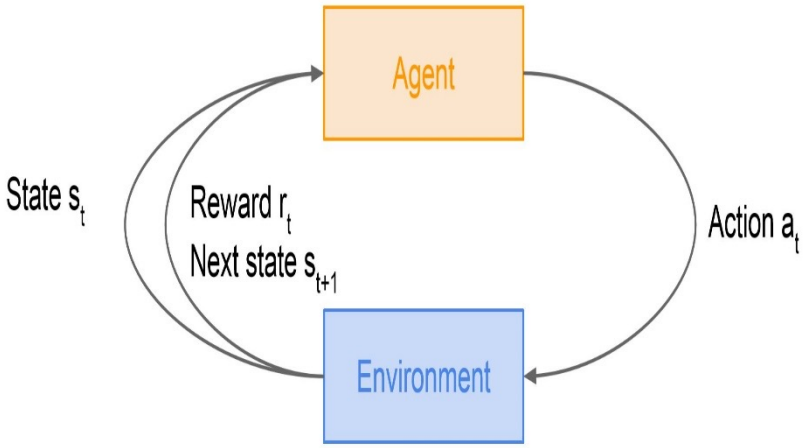

Dave-2 A Neural Network Drives A Car
0:24
1:27 / 14:13


Dave-2 A Neural Network Drives A Car
In-vehicle camera
2:40 / 14:13

Reinforcement Learning (RL)

- How can an agent learn behaviors when it doesn't have a teacher to tell it how to perform, except a delayed scalar feedback (a number called reward).
- The goal is to get the agent (with a policy) to act, by learning from interaction with environment, to maximize its future accumulated rewards (values)

- Potential applications
  - Backgammon/chess playing
  - Game playing (Atari)
  - job or shop scheduling
  - Car-pole control

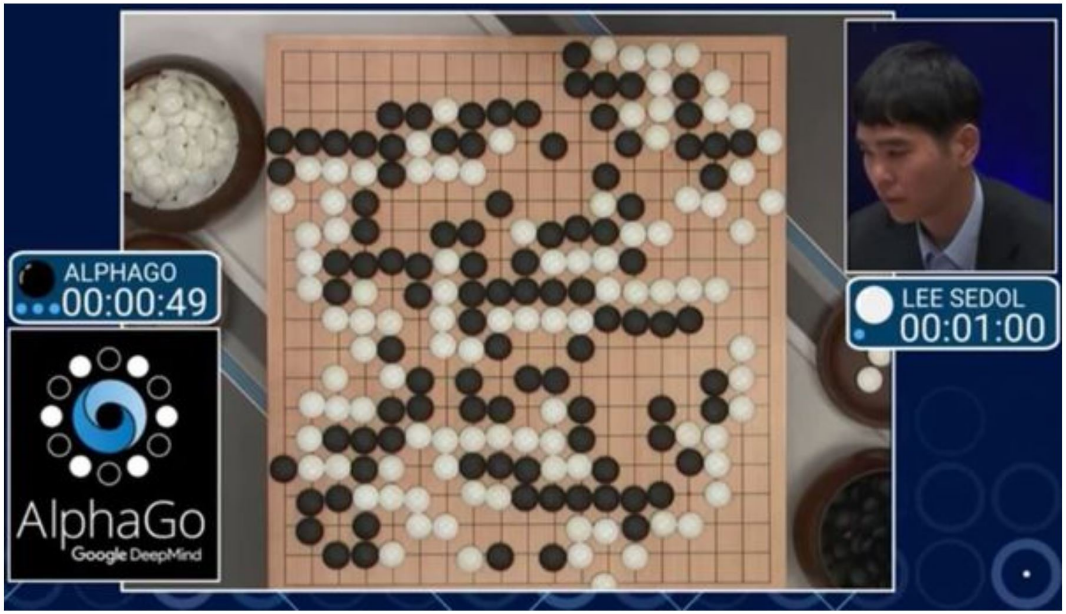


Computer Based Go Games

- Brute force search intractable:
  - Search space is huge
  - "Impossible" for computers to evaluate

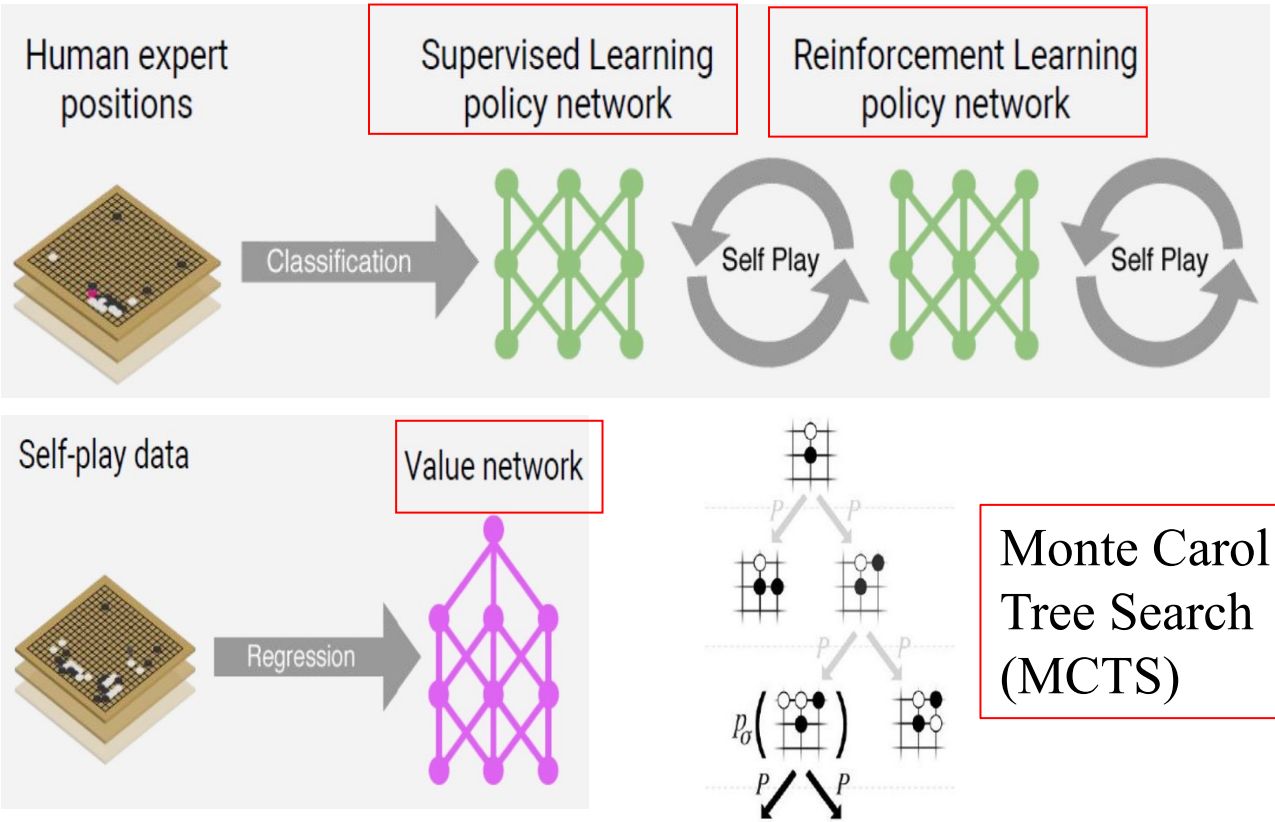
AlphaGo

- AlphaGo, developed by Google DeepMind in London, the first Computer Go program to beat a professional human Go player, 2-dan Hui Fan, in a 5-0 match, without handicaps on a full-sized 19×19 board in Oct. 2015
- In March 2016, it beat 9-dan Sedol Lee, in a 4-1 match.



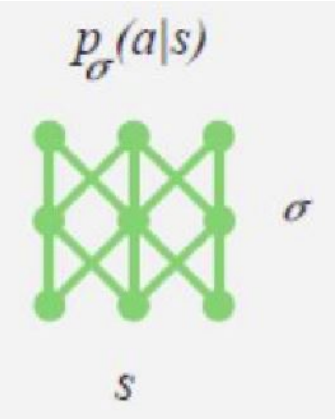
David Silver, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature* (529):484–489, 2016

Supervised Learning (SL) + Reinforcement Learning (RL)



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016).

Supervised Learning of Policy Network

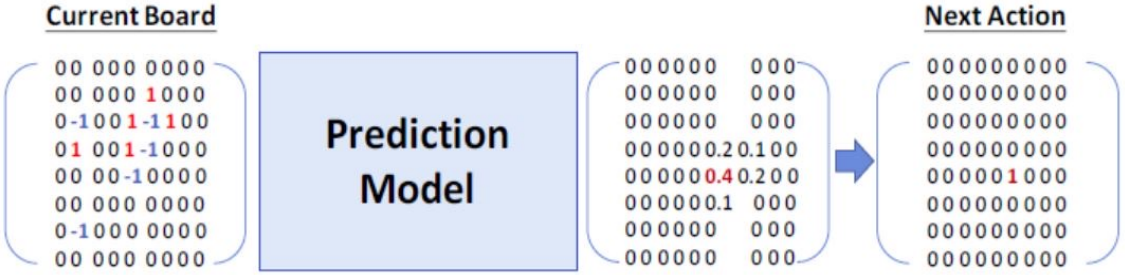


- Policy network: 13-layer convolutional neural network
- Training data: 160K games, 39M positions from human expert games (KGS, 5+ dan)
- Training algorithm: maximize likelihood (softmax output)

same as minimize cross-entropy

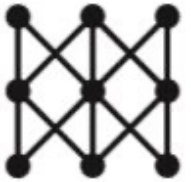
$$\Delta\sigma \propto \frac{\partial \log_\sigma p_\sigma(a \mid s_t)}{\partial \sigma}$$ stochastic gradient ascent

- 4-weeks training on 50 GPUs (TPUs) using Google Cloud

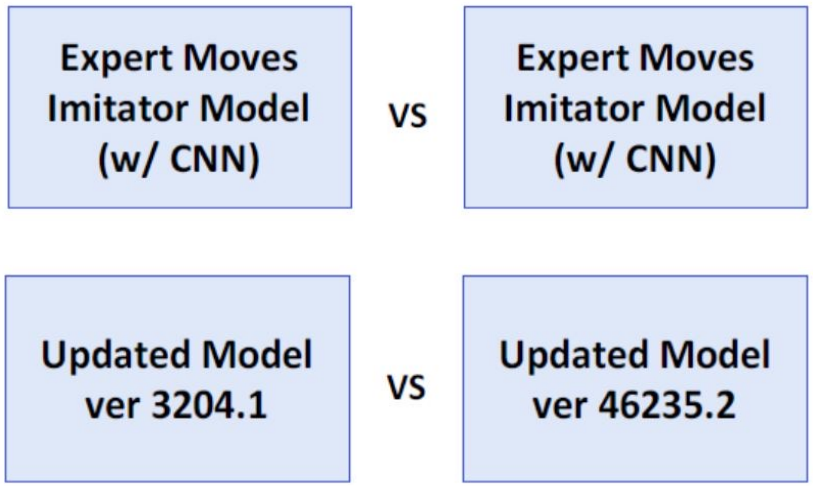


There are 19x19=361 possible actions (with different probabilities)

Reinforcement Learning of Policy Network



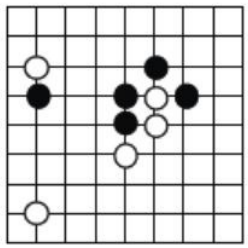
- Policy network: 13-layer convolutional neural network
- Training data: games of self-play between policy nets, different versions of updated policy nets
- 1 week training on 50 GPUs using Google Cloud



Return: board positions, win/lose info

30 million games

Reinforcement Learning of Policy Network

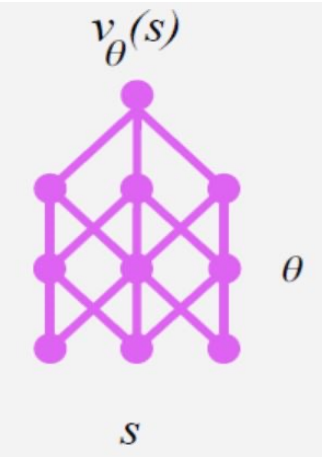


- Training algorithm: maximize wins z by policy gradient reinforcement learning, a reward function $r(s)$ that is zero ($p_t=0$) for all non-terminal time-steps $t < T$

$$\rho \Leftarrow \rho - \eta \frac{\partial E}{\partial \rho} = \rho + \eta \sum_{t=1}^{m} (z - p_t) \frac{\partial p_t}{\partial \rho} \Rightarrow \boxed{\Delta\sigma \propto \frac{\partial \log_\sigma p_\sigma(a_t \mid s_t)}{\partial \sigma} z_t}$$

- The final model wins 80% of the time when playing against the first SL model

Reinforcement Learning of Value Network



- Value network: 12-layer convolutional neural network
- Training data: 30-million games of self-play (p_ρ).
- Training algorithm: minimize MSE by stochastic gradient descent ($z_t = 1$ or -1), one random "s" per game

$$\Delta\theta \propto -\eta \frac{\partial E}{\partial \theta}, \text{where } \mathrm{E} = \frac{1}{2}(\mathrm{z_t} - \mathrm{v}_\theta(s_t))^2 \Rightarrow \boxed{\Delta\theta \propto \frac{\partial v_\theta(s_t)}{\partial \theta}(z_t - v_\theta(s_t))}$$

- 1 week training on 50 GPUs using Google Cloud
- Results: First strong position evaluation function - previously thought impossible

Monte Carlo Tree Search (MCTS)

- Simulate games starting from the current board position.
- Search for moves and play simulated games until the end.
  - If enough games are simulated, will get an idea which move likely gets the most wins.
  - A search tree recording sequences (how many wins for each move) of simulated moves is built
- To avoid the huge search space, need to prioritize searches based on information estimated by the policy and the value networks.
- Should explore moves that we know little in the simulations so far (exploration), on the other hand, if we win a simulated game, try the corresponding moves more often (exploitation).