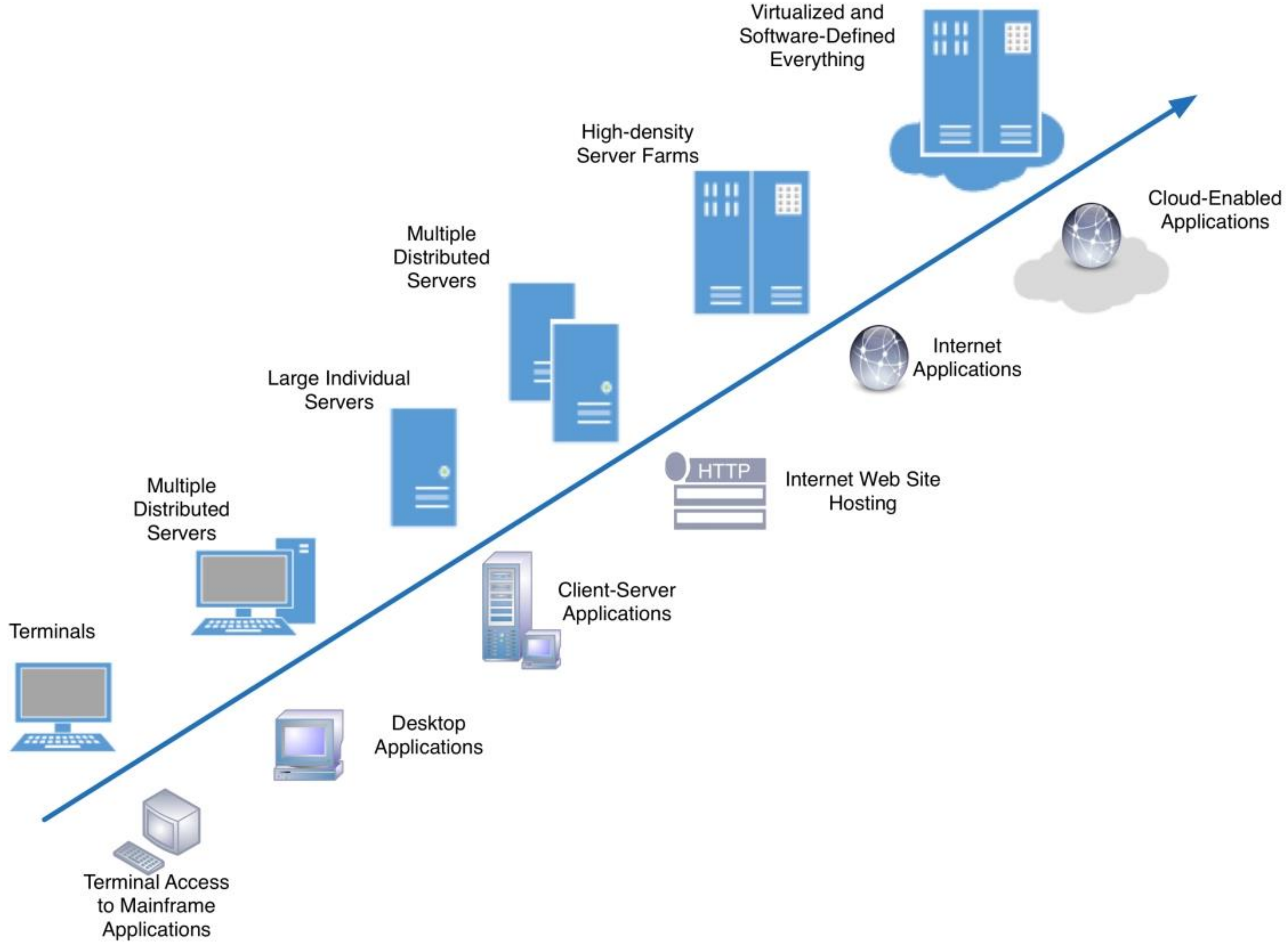
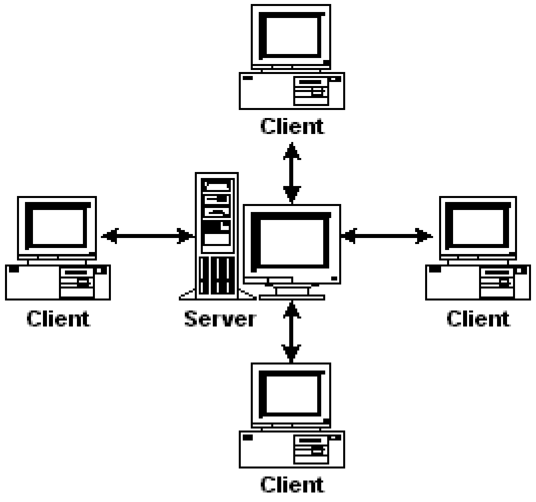
AI in Resource Constrained Devices

Evolution in Computers

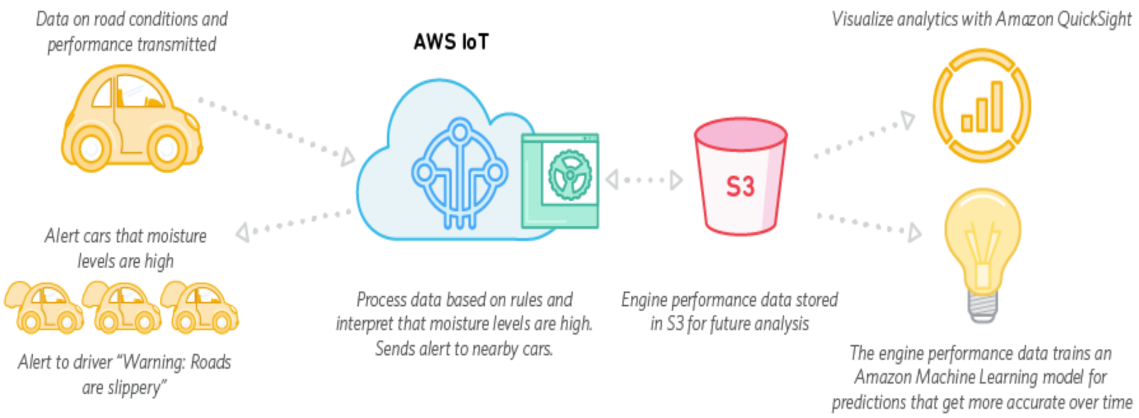


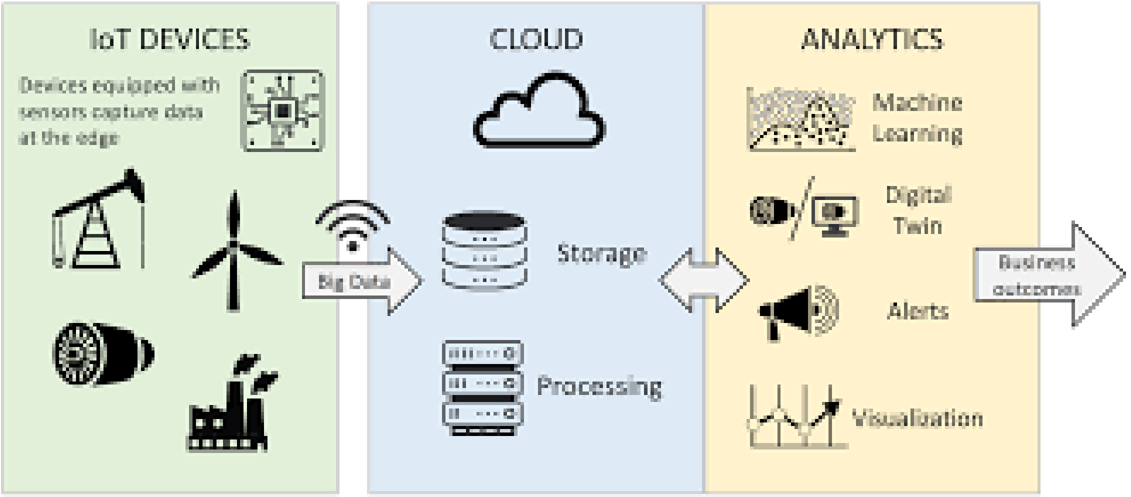
Computing in Era 2K and Today's

Server – Terminal
Era 2K



Cloud/Big Server – PC/ mobile/IoT
Today





Machine Learning In Resource Constrained Devices

Deep Learning in Resource constraint Devices (Edge Devices)

Phones

Drones

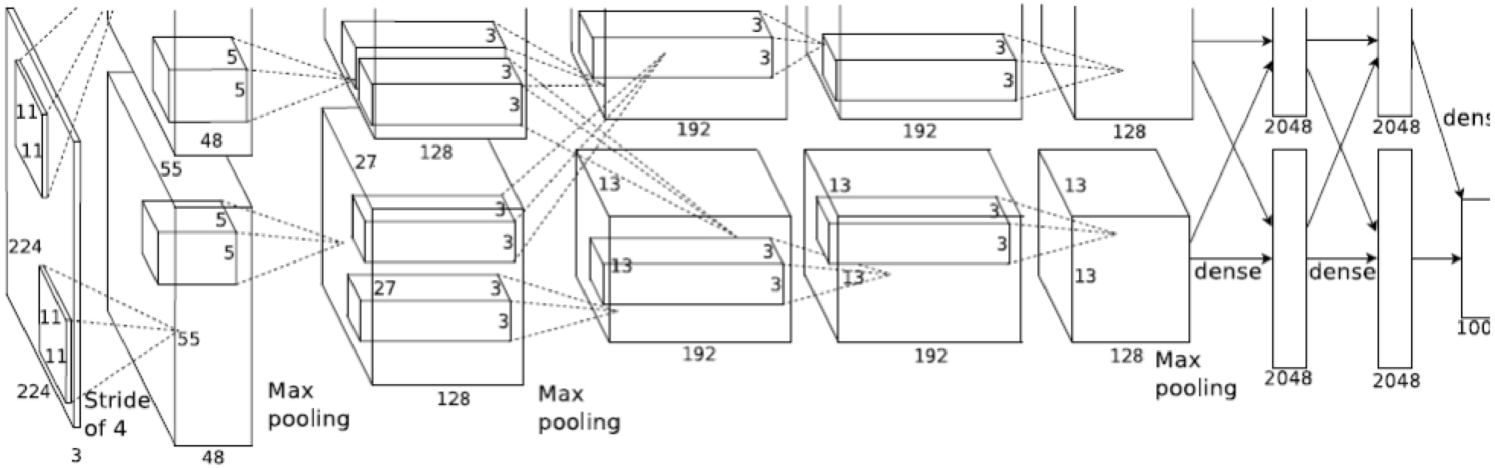
Robots

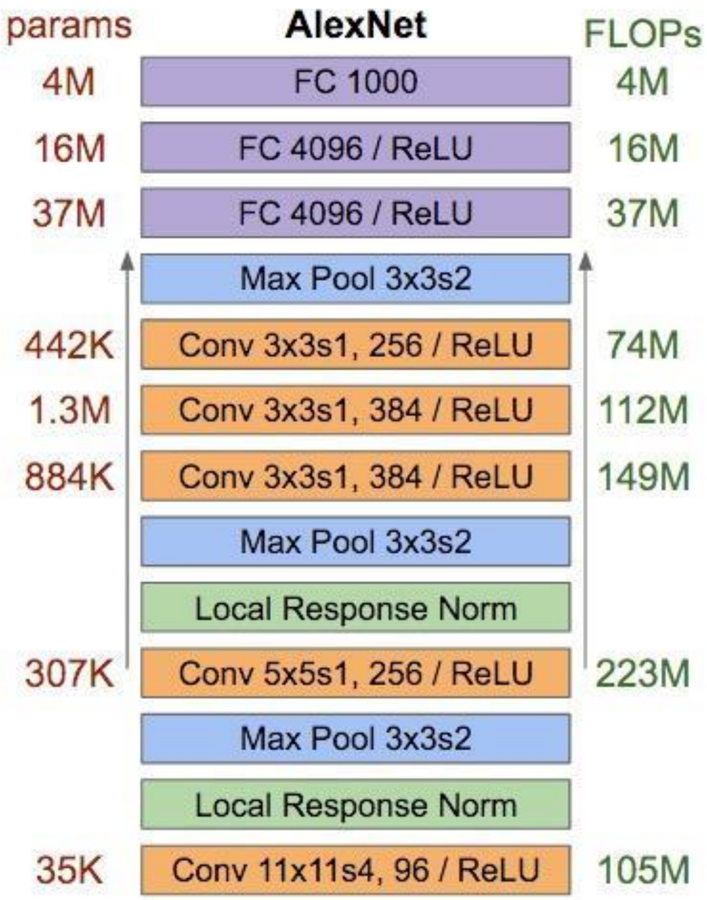
Glasses

Self Driving Cars

Battery Constrained!

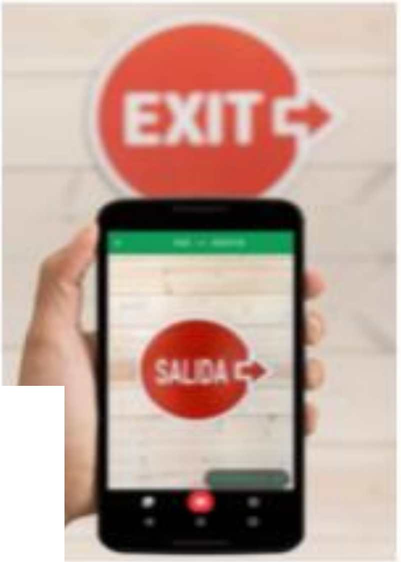
Big Huge Neural Network!





AlexNet - 60 Million Parameters = 240 MB

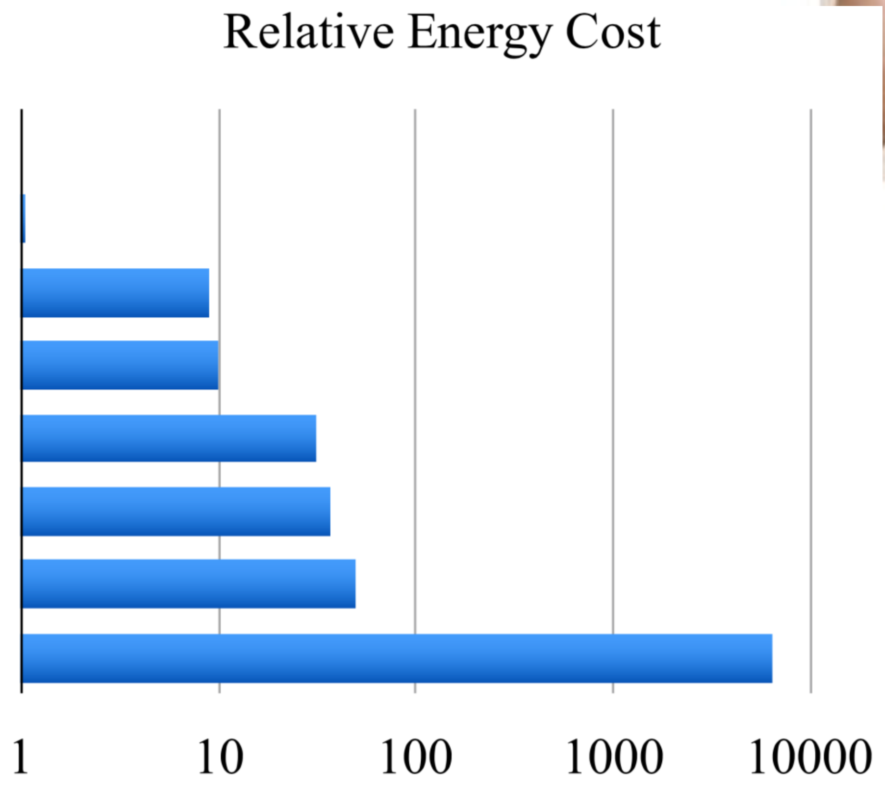
The Mobile Phone, IoT



1 GB RAM

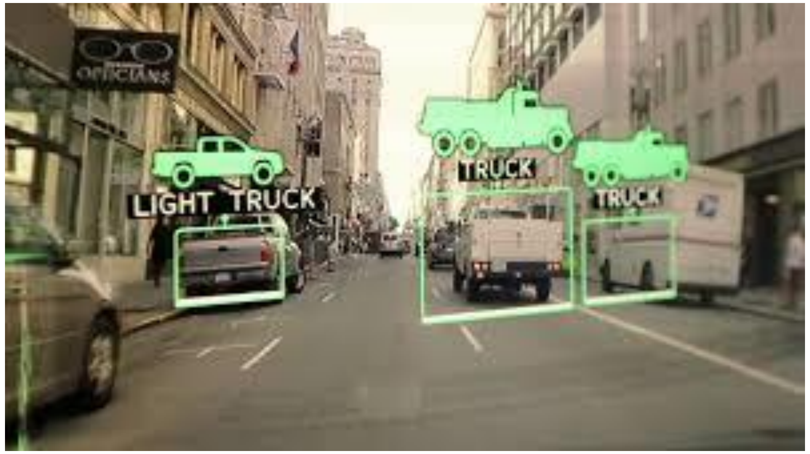
1/2 Billion FLOPs

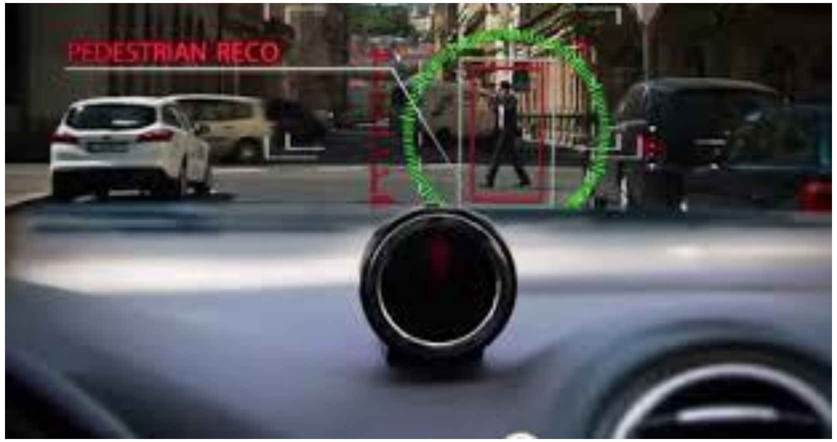
Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	640	6400



But wait! What about battery life?

Self Driving Cars!





Can we do 30 fps?

ORCAM! : Blind AID

Can we do 30 fps?

Running Model in the Cloud

1. Network Delay
2. Power Consumption
3. User Privacy

Issues on Edge Devices

1. RAM Memory Usage
2. Running Time
3. Power Usage
4. Download / Storage size

Can these “Edge Devices” take decision

Yes, they can do this if..

- Make these devices powerful

- Make decision making process less computational expensive and small

How ?

- Compress heavy models to small models

- Advantage
  - Needed less space to store
  - Energy efficient
  - Low latency
  - Lesser computational expensive
- Disadvantage
  - Accuracy may reduce

What are Neural Networks made of?

- Fully Connected Layer : Matrices

- Convolutional Layer : Kernals (Tensors)

Reducing Memory Usage

1. Compressing Matrices
   a. Sparse Matrix => Special Storage formats
   b. Quantization
2. Architecture

Neural Network Algorithms

1. Matrix Multiplications
2. Convolutions

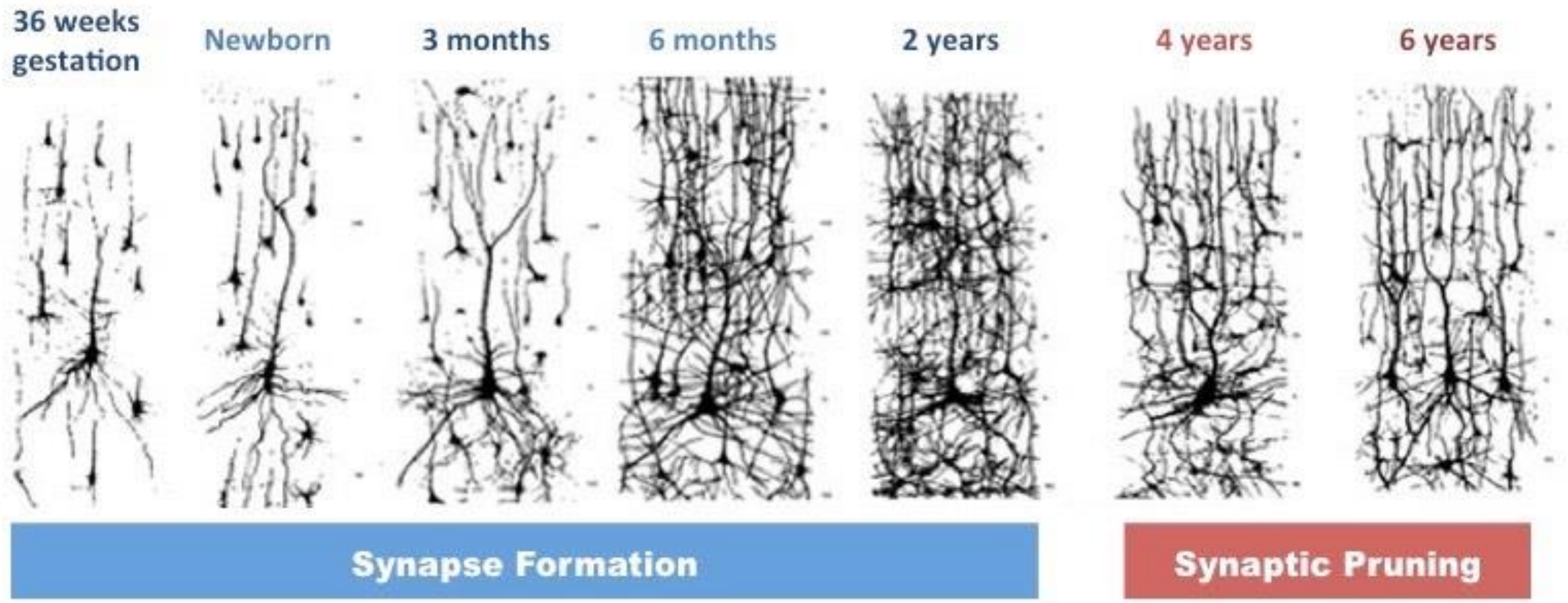
Various Techniques for Compression

- Pruning
- Quantization
- Weight Matrix Factorization
  - Singular value decomposition (SVD)
  - Sparse coding
- Convolution Architecture
  - Global average pooling layer
  - Convolution kernel sparse decomposition
  - depth-wise separable convolution

PRUNING

Compressing Matrices by making them Sparse

Pruning in human brain



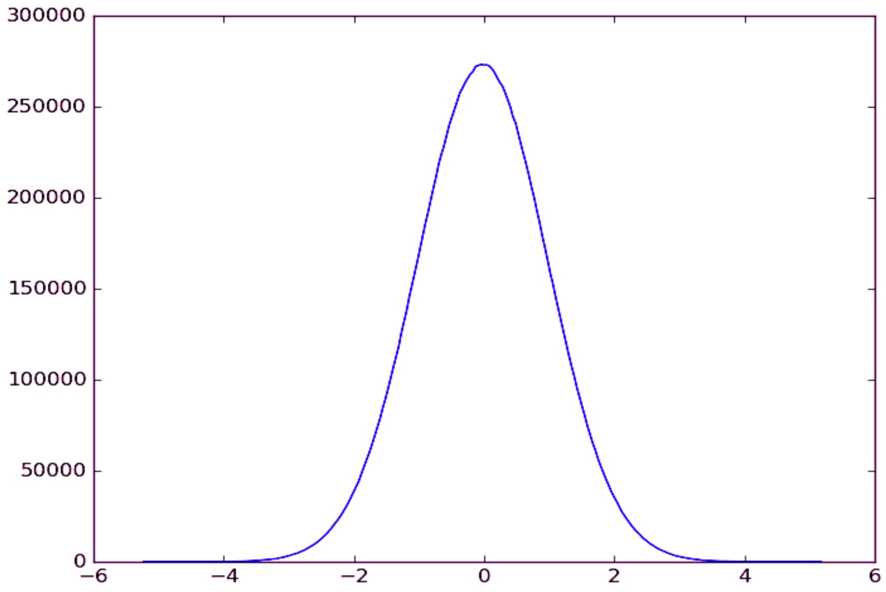
WHY PRUNING ?

Deep Neural Networks have redundant parameters.

Such parameters have a negligible value and can be ignored.

Removing them does not affect performance.

Figure: Distribution of weights after Training



Why do you need redundant parameters?
Redundant parameters are needed for training to converge to a good optima.

Optimal Brain Damage by Yann Le Cunn in 90's
https://papers.nips.cc/paper/250-optimal-brain-damage

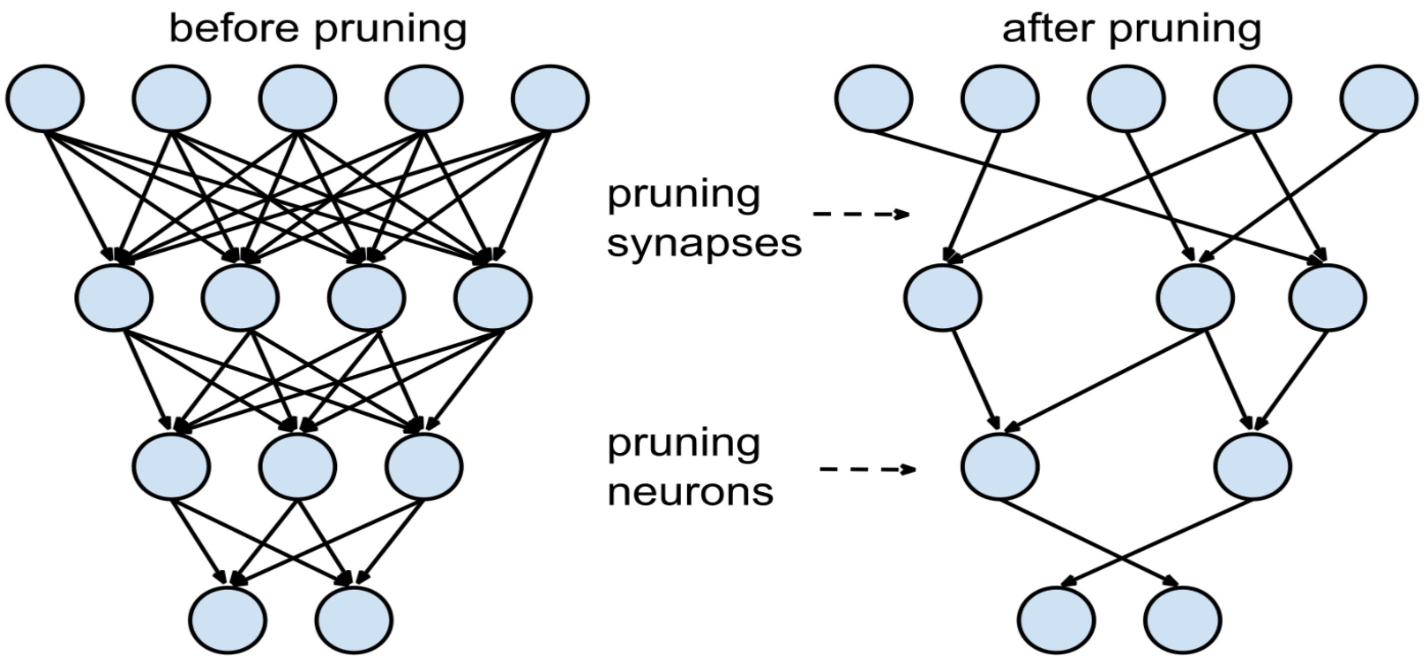
TYPES OF PRUNING

- Fine Pruning : Prune the weights
- Coarse Pruning : Prune neurons and layers

- Static Pruning : Pruning after training
- Dynamic Pruning : Pruning during training time

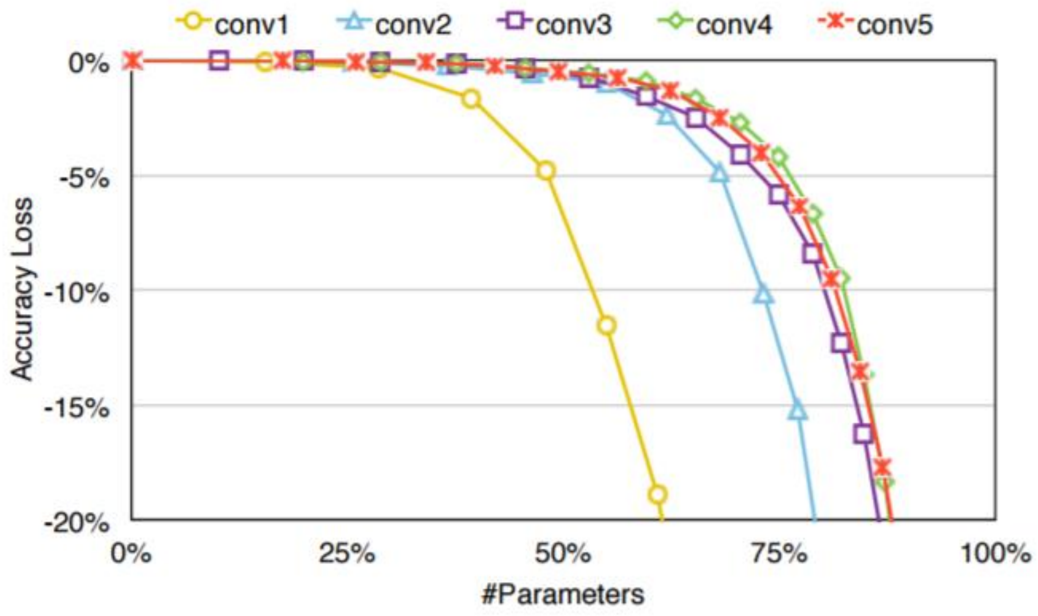
Weight Pruning

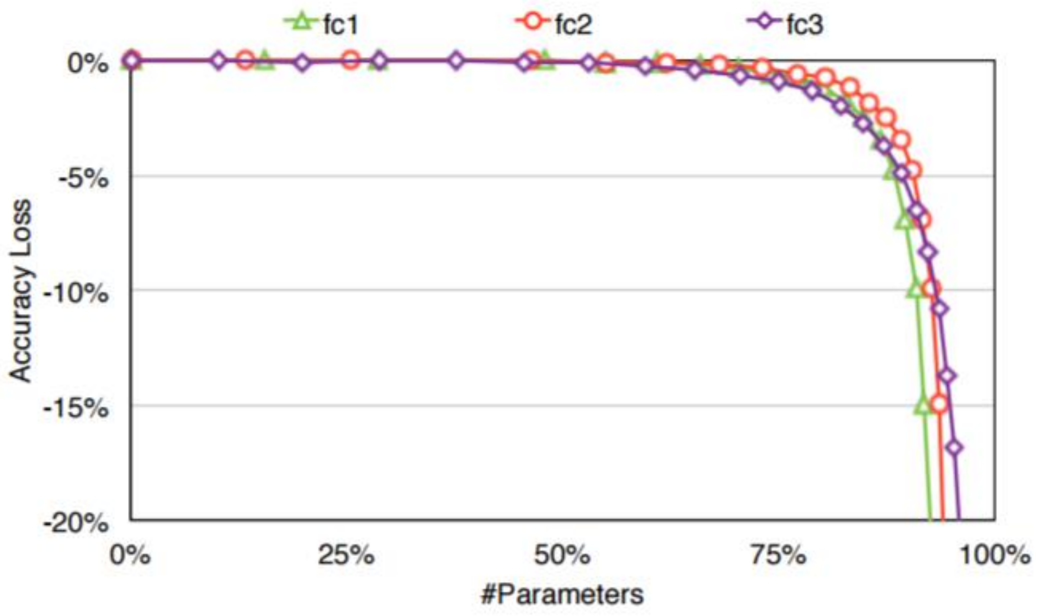
(After Training)



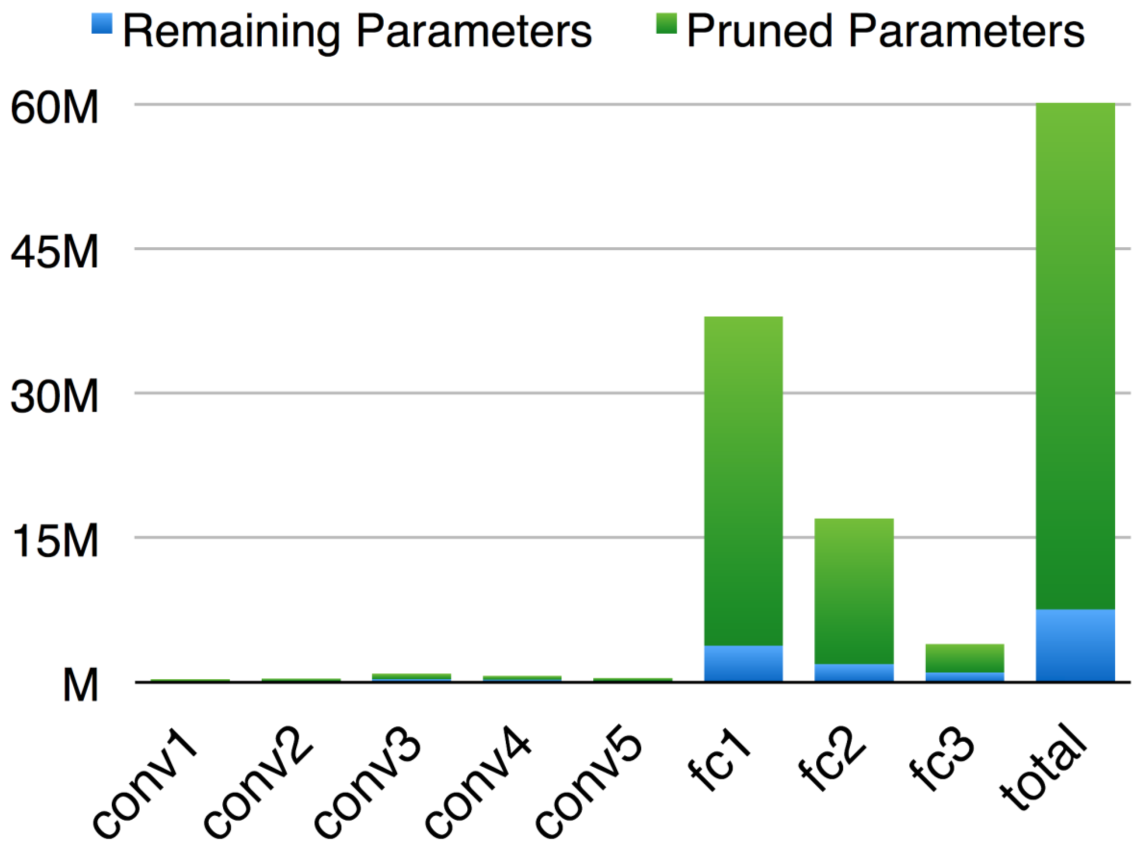
- The matrices can be made sparse. A naive method is to drop those weights which are 0 after training.
- Drop the weights below some threshold.
- Can be stored in optimized way if matrix becomes sparse

Sensitivity of layers to pruning





Magnitude-based method: Iterative Pruning + Retraining (AlexNet)



Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

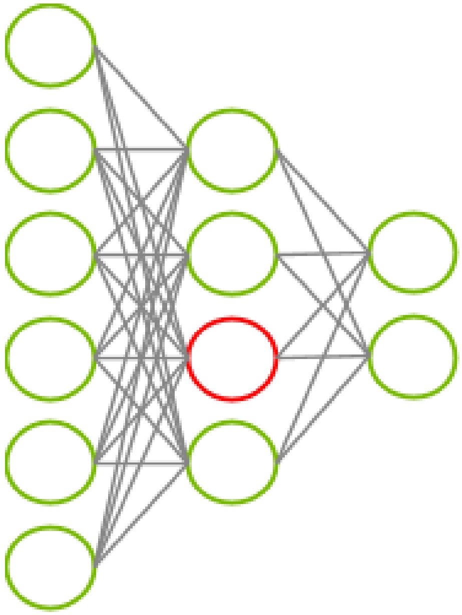
Comments on Weight Pruning

1. Matrices become sparse.
2. Storage in HDD is efficient.
3. Same memory in RAM is occupied by the weight matrices.
4. Matrix multiplication is not faster since each 0 valued weight takes as much computation as before.

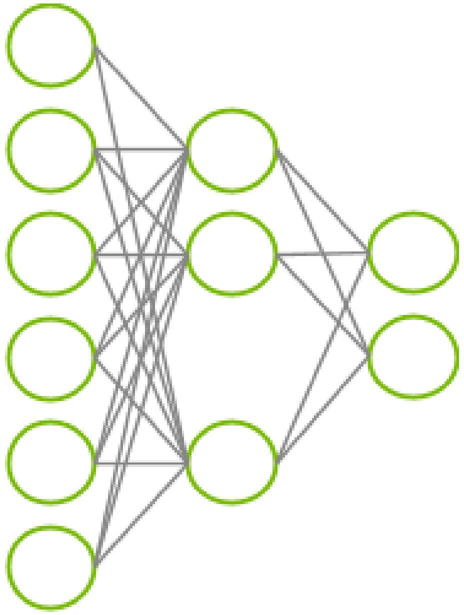
$$\begin{pmatrix} 1.0 & 0 & 5.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 9.0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.0 & 0 & 0 & 0 & 0 \\ 2.0 & 0 & 0 & 0 & 0 & 10.0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 & 0 & 0 & 0 & 12.0 \end{pmatrix}$$

Sparse Weight Matrices

Neuron Pruning



6 inputs, 6 neurons (including 2 outputs), 32 connections



6 inputs, 5 neurons (including 2 outputs), 24 connections

- ➔ Removing rows and columns in a weight matrix.
- ➔ Matrix multiplication will be faster improving test time.

Dropping Neurons by Regularization

$$\texttt{li\_regulariser} := \lambda_{\ell_i} \sum_{\ell=1}^{L} \sum_{j=1}^{n^\ell} \|\mathbf{W}^\ell_{:,j}\|_2 = \lambda_{\ell_i} \sum_{\ell=1}^{L} \sum_{j=1}^{n^\ell} \sqrt{\sum_{i=1}^{n^{\ell-1}} \left(W^\ell_{ij}\right)^2}$$

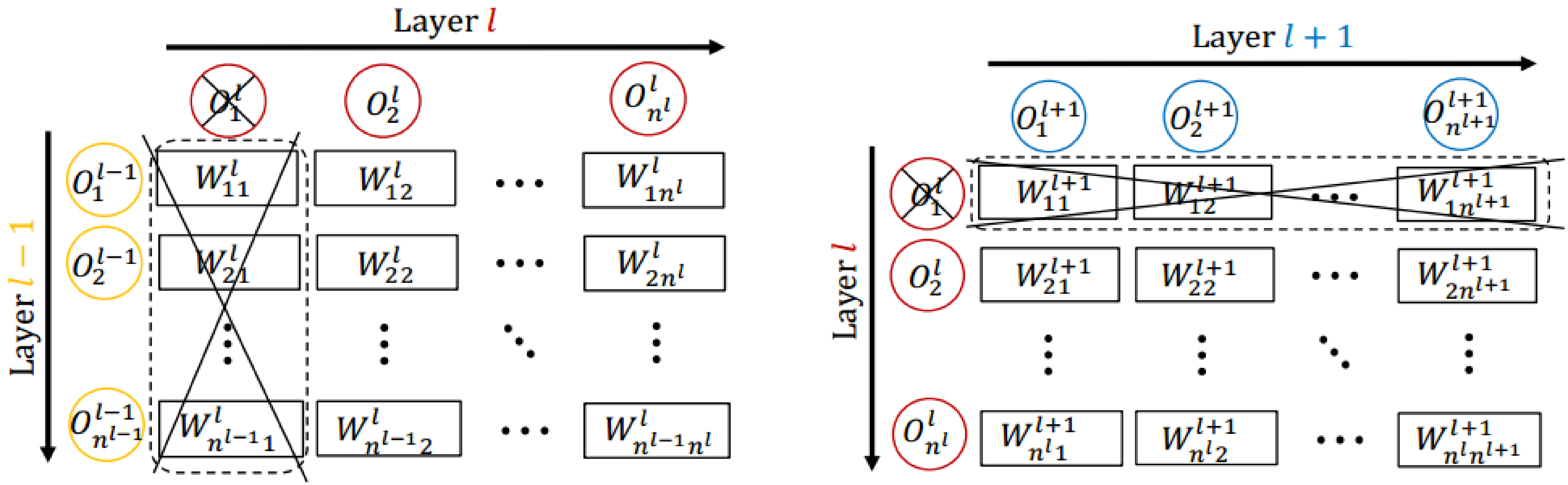
$$\texttt{lo\_regulariser} := \lambda_{\ell_o} \sum_{\ell=1}^{L} \sum_{i=1}^{n^{\ell-1}} \|\mathbf{W}^\ell_{i,:}\|_2 = \lambda_{\ell_o} \sum_{\ell=1}^{L} \sum_{i=1}^{n^{\ell-1}} \sqrt{\sum_{j=1}^{n^\ell} \left(W^\ell_{ij}\right)^2}$$

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Dropping principles

- All input connections to a neuron is forced to be 0 or as close to 0 as possible. (force li_regulariser to be small)
- All output connections of a neuron is forced to be 0 or as close to zero as possible. (force lo_regulariser to be small)
- Add regularisers to the loss function and train.
- Remove all connections less than threshold after training.
- Discard neuron with no connection.

Effect of neuron pruning on weight matrices



(c) Removal of incoming connections to neuron O_1^ℓ, i.e., the group of weights in the dashed box are all zeros

(d) Removal of outgoing connections from neuron O_1^ℓ, i.e., the group of weights in the dashed box are all zeros

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

QUANTIZATION

Binary Quantization

$$\hat{W}_{ij} = \begin{cases} 1 & \text{if } W_{ij} \geq 0, \\ -1 & \text{if } W_{ij} < 0. \end{cases}$$

Size Drop : 32X

Runtime : Much faster (7x) matrix multiplication for binary matrices.

Accuracy Drop : Classification error is about 20% on the top 5 accuracy on ILSVRC dataset.

COMPRESSING DEEP CONVOLUTIONAL NETWORKS USING VECTOR QUANTIZATION Yunchao Gong, Liu Liu , Ming Yang, Lubomir Bourdev

8-bit uniform quantization

- Divide the max and min weight values into 256 equal divisions uniformly.
- Round weights to the nearest point
- Store weights as 8 bit ints

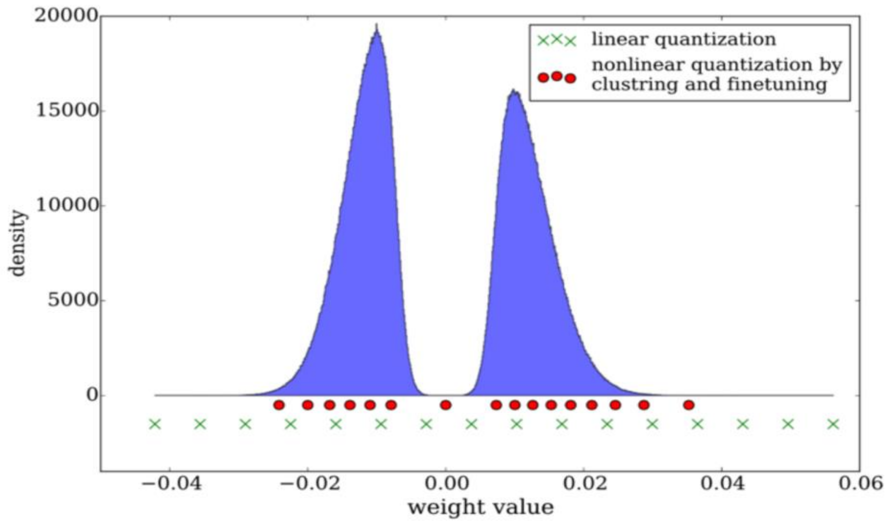
Size Drop : 4X

Runtime : Much faster matrix multiplication for 8 bit matrices.

Accuracy Drop : Error is acceptable for classification for non critical tasks

https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/

Non Uniform Quantization/ Weight Sharing



- Perform k-means clustering on weights.

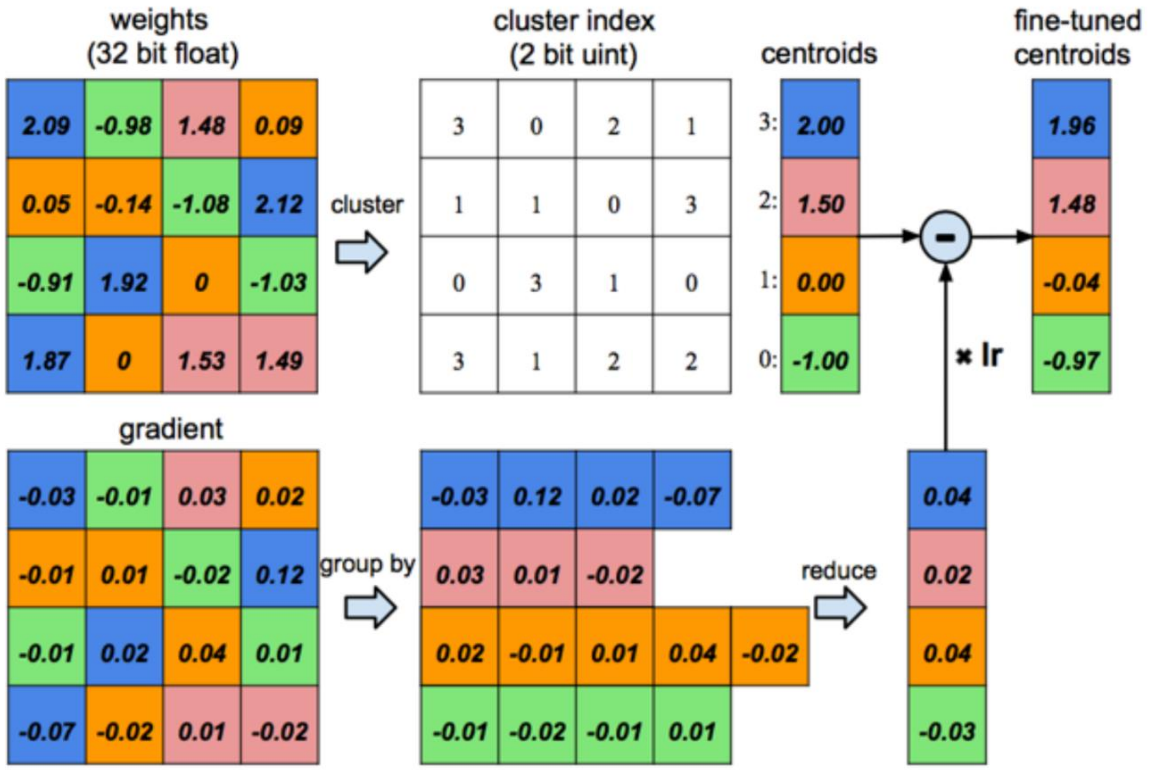
- Need to store mapping from integers to cluster centers. We only need log (k) bits to code the clusters which results in a compression factor rate of 32/ log (k). In this case the compression rate is 4.

DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

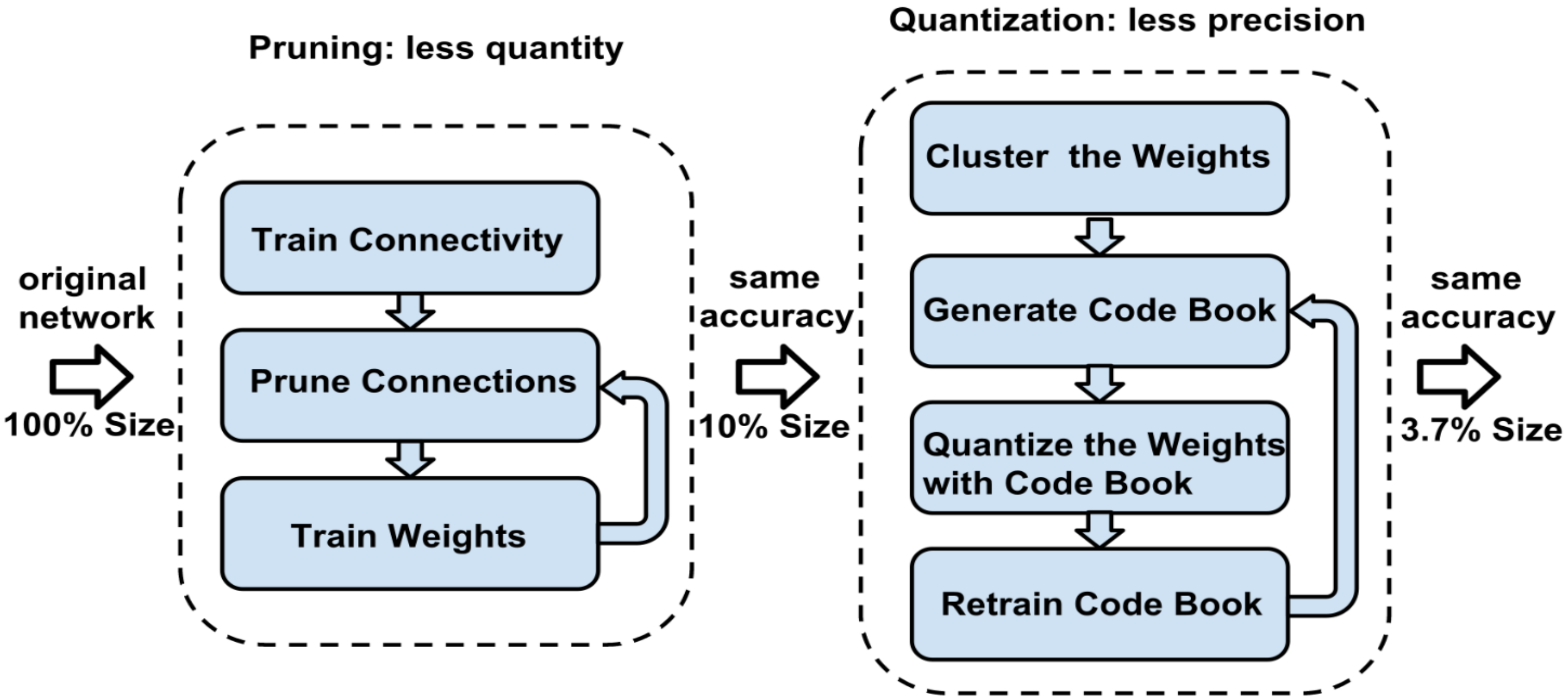
Weight Sharing while Training

- Iterate
  - Train
  - Cluster weights
  - Make them same

- Compute gradients with respect to centroids so that weight sharing is preserved during gradient update.



Deep Compression by Song Han



DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

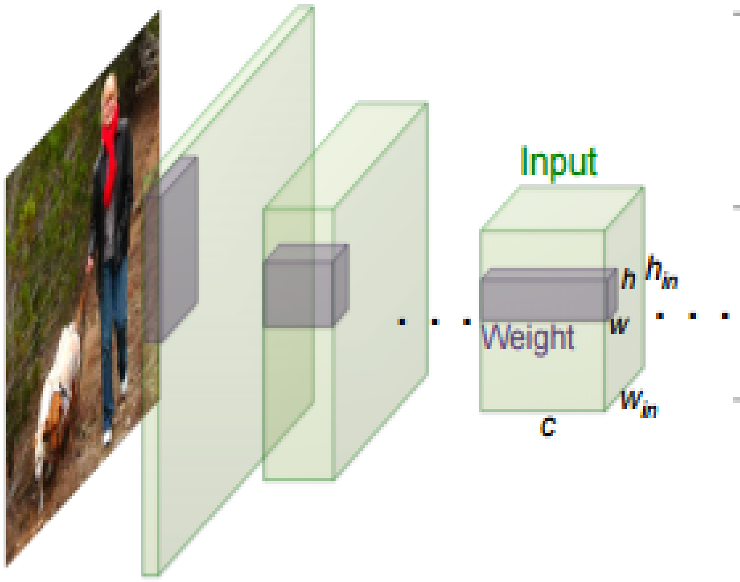
XNOR Net

- **Binary Weight Networks :**
  - Estimate real time weight filter using a binary filter.
  - Only the weights are binarized.
  - Convolutions are only estimated with additions and subtractions (no multiplications required due to binarization).

- **XNOR Networks:**
  - Binary estimation of both inputs and weights
  - Input to the convolutions are binary.
  - Binary inputs and weights ensure calculations using XNOR operations.

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Results



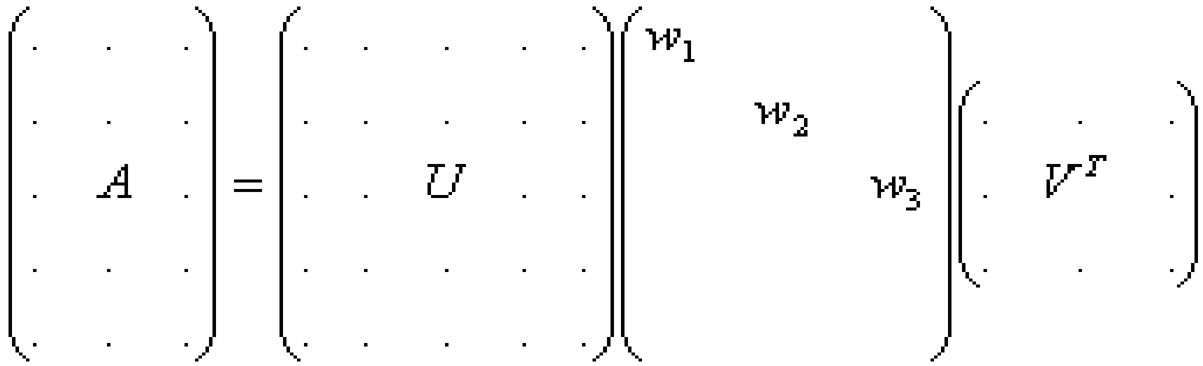
	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

MATRIX FACTORIZATION

Fully Connected Layers: Singular Value Decomposition

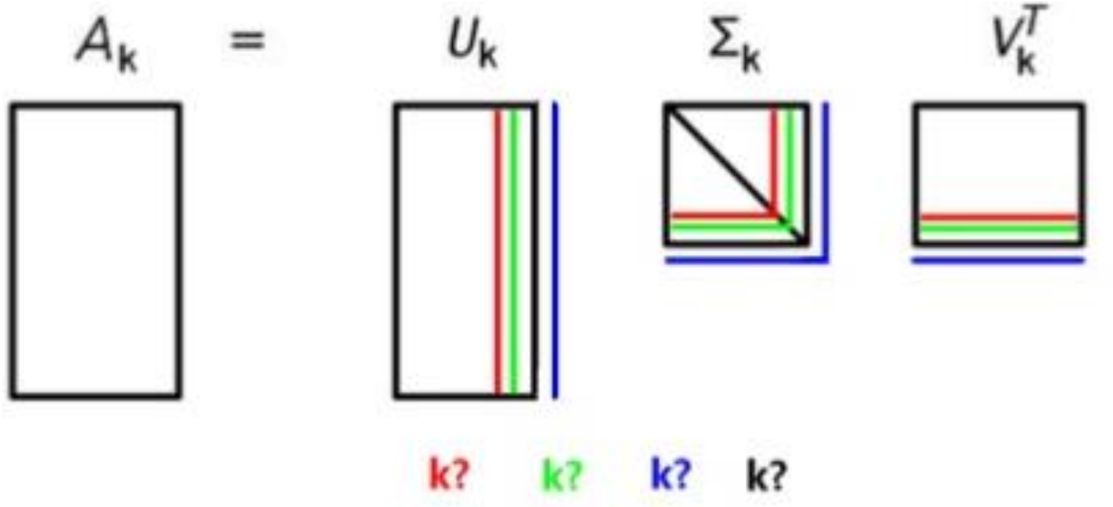
- Most weights are in the fully connected layers (according to Denton et al.)
- $W = USV^\top$
  - $W \in \mathbb{R}^{m\times k}, U \in \mathbb{R}^{m\times m}, S \in \mathbb{R}^{m\times n}, V^\top \in \mathbb{R}^{n\times n}$
- S is diagonal, decreasing magnitudes along the diagonal



Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." *arXiv preprint arXiv:1412.6115* (2014).

Singular Value Decomposition

- By only keeping the t singular values with largest magnitude:
- $\widetilde{W} = \widetilde{U}\tilde{S}\widetilde{V}^{\top}$
  - $\widetilde{W} \in \mathbb{R}^{m\times n}, \widetilde{U} \in \mathbb{R}^{m\times k}, \tilde{S} \in \mathbb{R}^{k\times k}, \widetilde{V}^{k} \in \mathbb{R}^{k\times n}$
- $Rank(\widetilde{W}) = k$



Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." *arXiv preprint arXiv:1412.6115* (2014).

SVD: Compression

- $W = USV^\top, W \in \mathbb{R}^{m\times n}, U \in \mathbb{R}^{m\times m}, S \in \mathbb{R}^{m\times n}, V^\top \in \mathbb{R}^{n\times n}$
- $\widetilde{W} = \widetilde{U}\tilde{S}\tilde{V}^\top, \widetilde{W} \in R^{m\times n}, \widetilde{U} \in R^{m\times k}, \tilde{S} \in R^{k\times k}, \tilde{V}^\top \in R^{k\times n}$

- Storage for W: $O(mn)$
- Storage for $\widetilde{W}$: $O(mk + k + kn)$
- Compression Rate: $O\left(\frac{mn}{k(m+n+1)}\right)$

Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." *arXiv preprint arXiv:1412.6115* (2014).

COMPRESSED ARCHITECTURES

Design small architectures

Compress scheme on **pre-trained model**

Vs

Design **small CNN architecture** from scratch
(also preserve accuracy?)

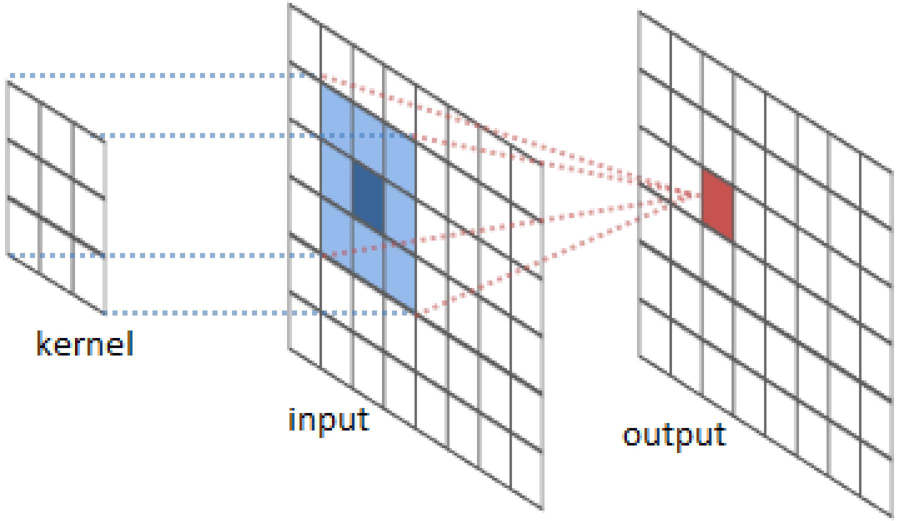
GoogLe Net

- First architecture with improved utilization of the computing resources inside the network while increasing size, both depth and width
- 22 layers deep when counting only layers with parameters
- Significantly more accurate than AlexNet
- 12 times lesser parameters than AlexNet.
- Computational cost “less than 2X compared to AlexNet”

Szegedy, Christian, et al. "Going deeper with convolutions." *CVPR*, 2015.

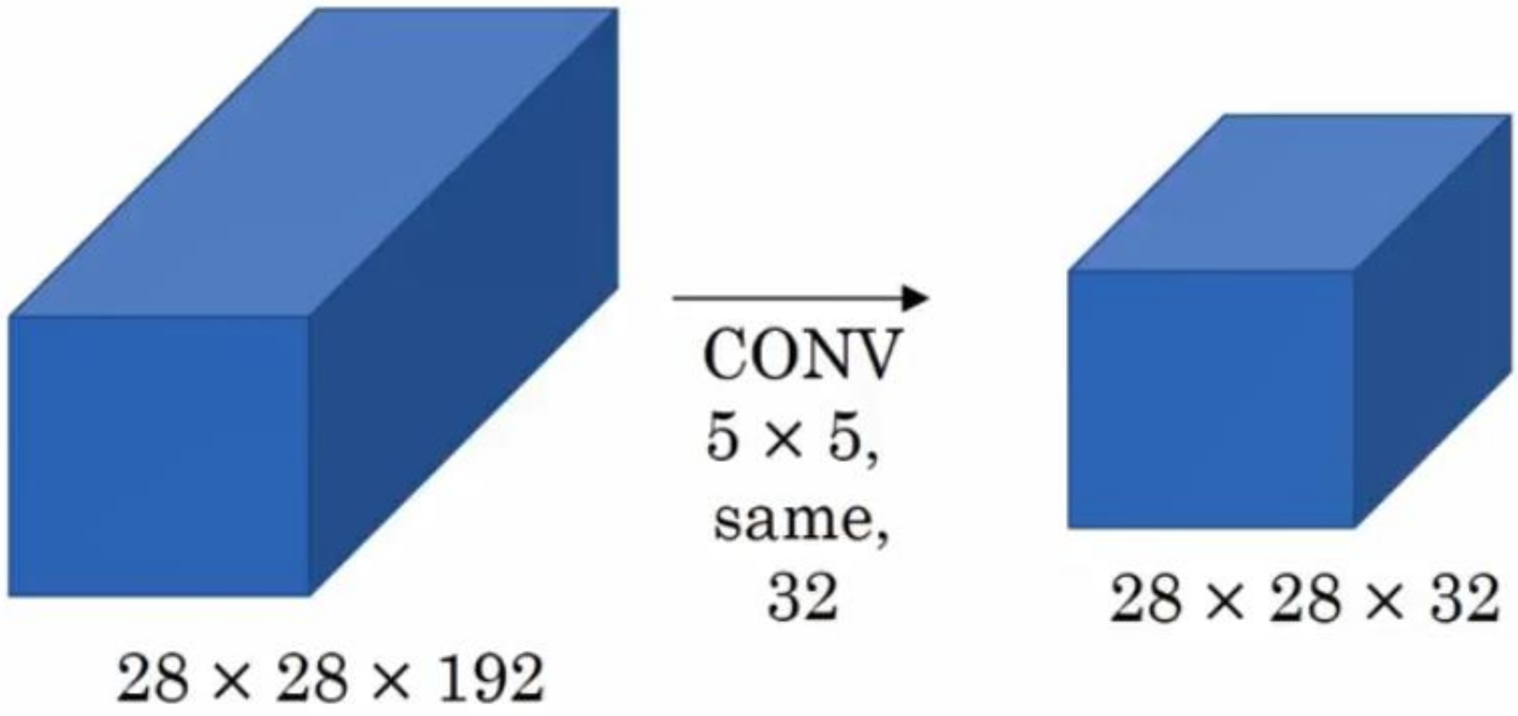
Convolutions:
Matrix Multiplication

Most time is spent in the convolutional layers



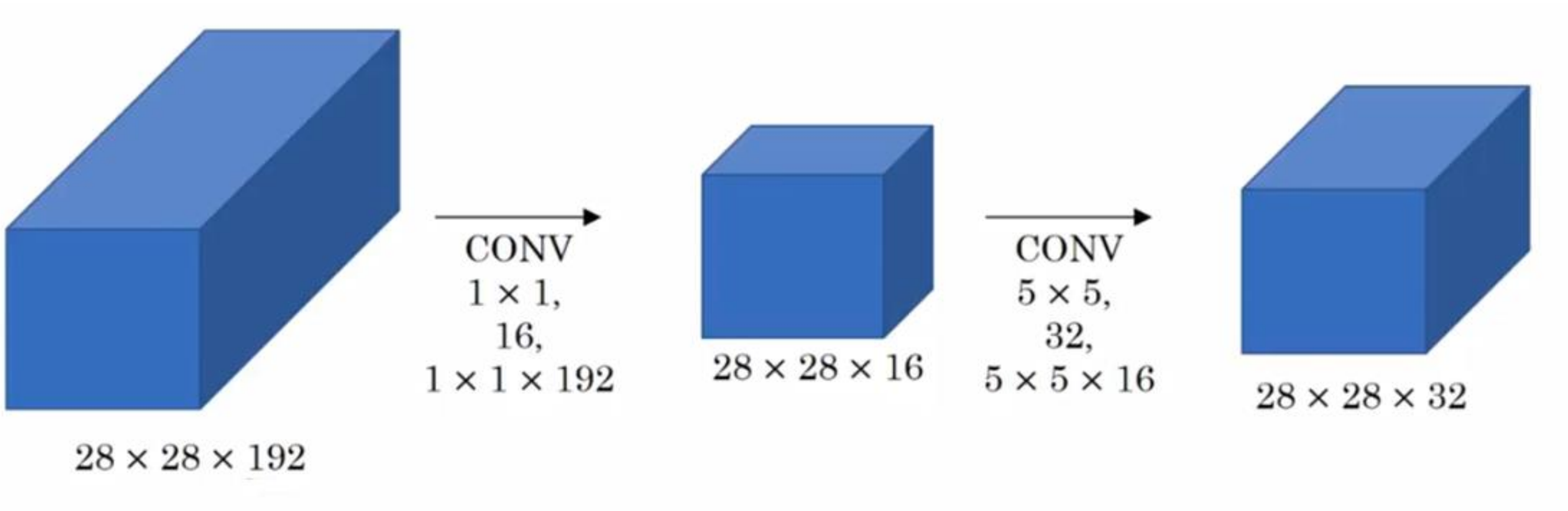
$$F(x, y) = I * W$$

Fire Layer



28X28X32X5X5X192=120 Million Calculations
32X5X5X192 = 153600 parameters

Fire Layer



28X28X16X192 + 28X28X32X5X5X16 = 12.4 Million Calculations

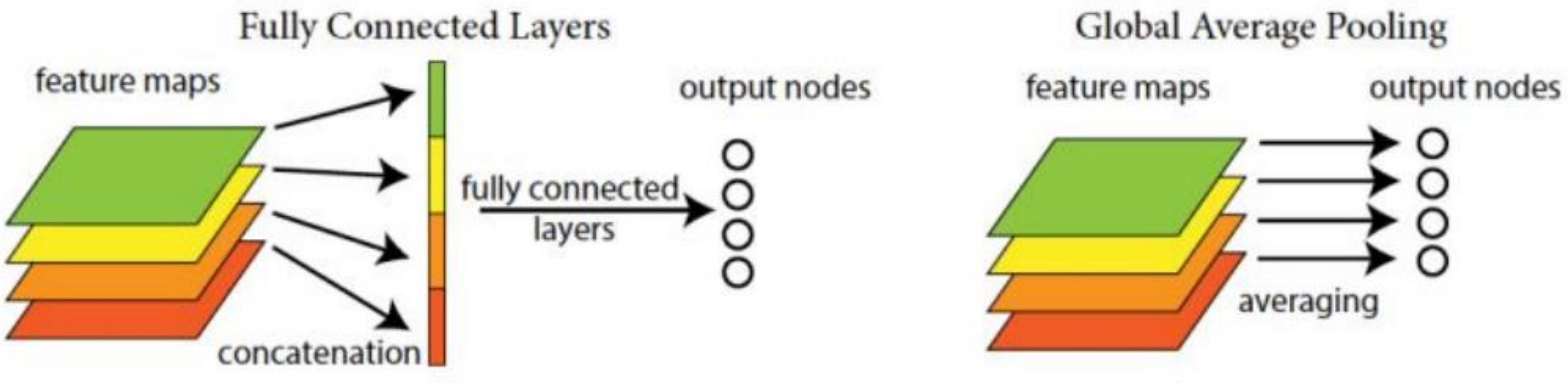
16X192 + 32X5X5X16 = 15873 parameters 10x less parameters

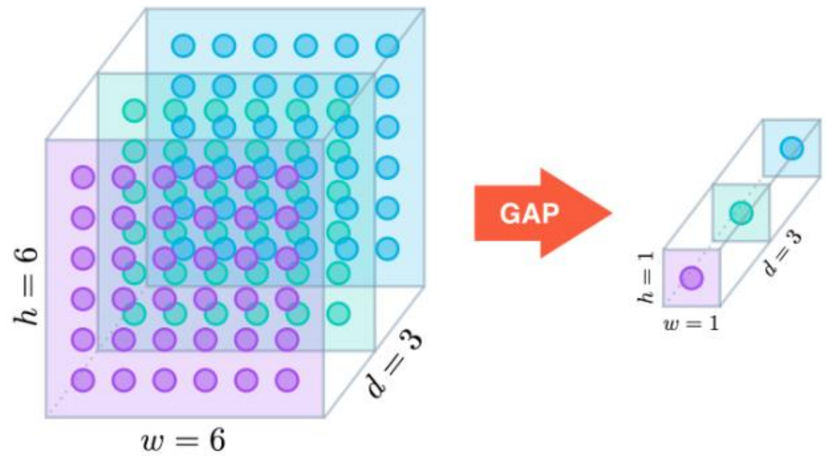
GoogLe Net: Global Average Pooling

Problems with fully connected (FC) layers:

- More than 90% parameters of Alexnet and VGG are in the Fully Connected layers.
- One single particular layer in VGG contains 100 million parameters alone.
- Prone to overfitting.
- Heavily dependent on regularization methods like dropout.

GoogLe Net: Global Average Pooling





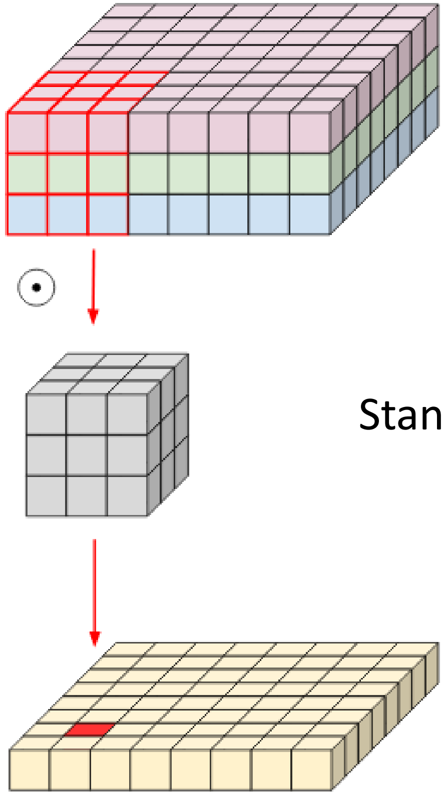
Source: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

GoogLe Net: Global Average Pooling

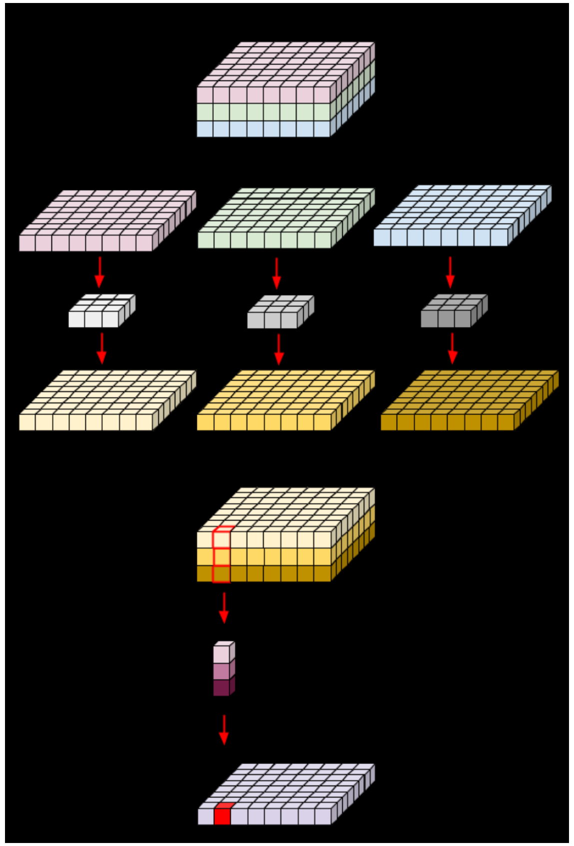
Global Average Pooling as replacement to FC layers:

- An alternative is to use spatial average of feature maps.
- Huge reduction the number of parameters as compared to the Fully Connected layer.
- Stronger local modelling using the micro network.
- It is itself a structural regularizer and hence doesn’t need dropout.

Modified Convolution Operation

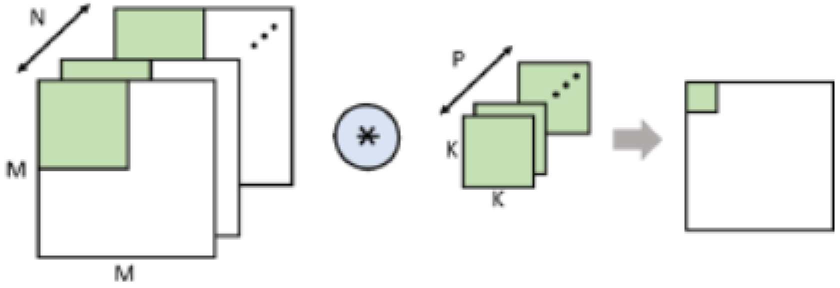


Standard Convolution



Depth-wise Convolution

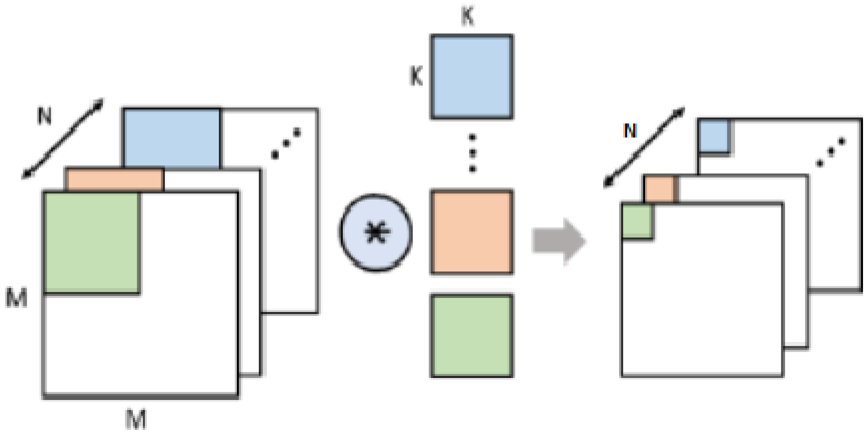
Point-wise Convolution



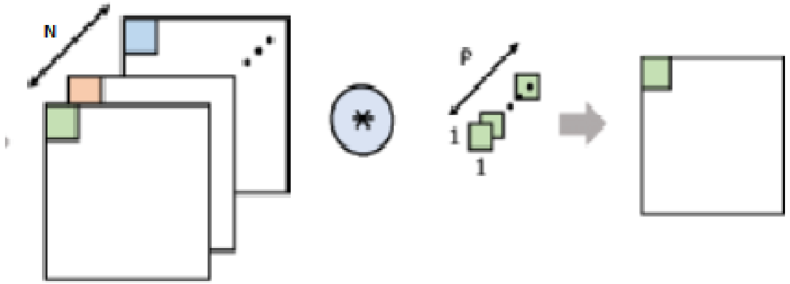
(a) standard convolution

Input feature map - M × M ×N
kernel size - K × K × N × P
number of weights
WSC=K×K×N×P

Number of operations
OSC=M×M×K×K×N×P



(b) depthwise convolution



(c) pointwise convolution

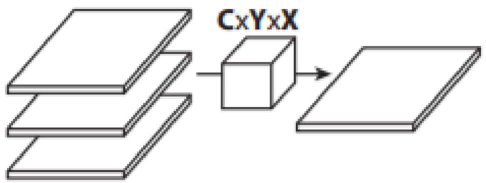
In case of Depth wise + Point wise number of weights is-

WDSC=K×K×N + N×P

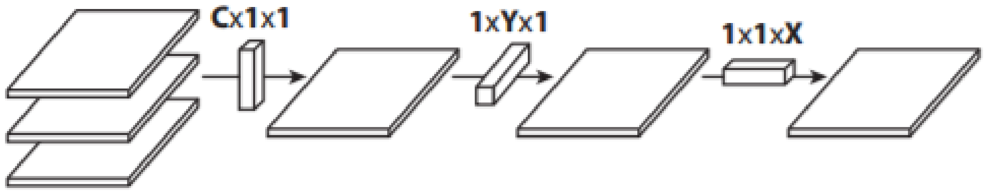
number of operations
ODSC=M×M×K×K×N + M×M×N×P

Flattened Convolutions: Fire Layer

- Replace $c \times y \times x$ convolutions with $c \times 1 \times 1$, $1 \times y \times 1$, and $1 \times 1 \times x$ convolutions



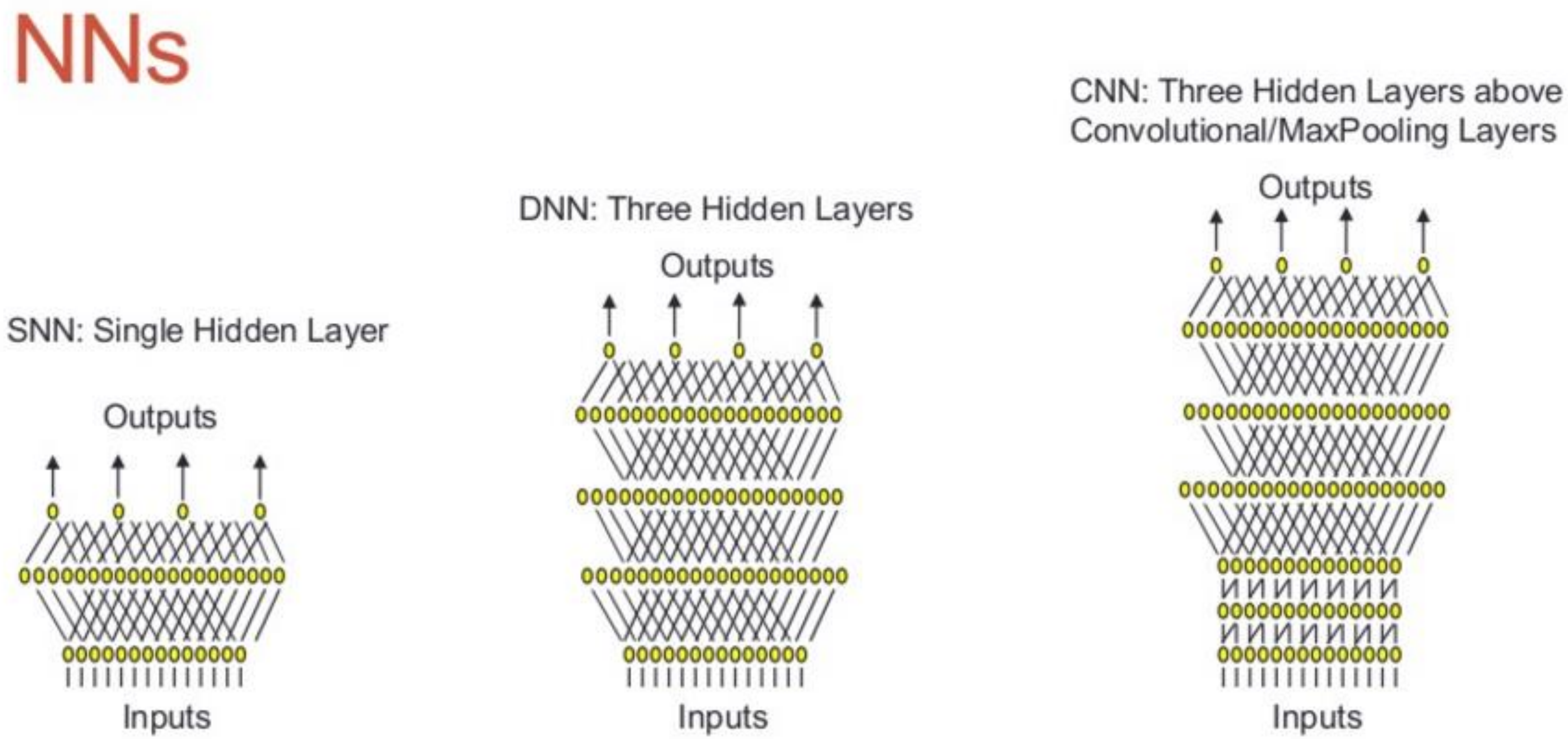
3D convolution



1D convolution over different direction

Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

Student – Teacher Networks

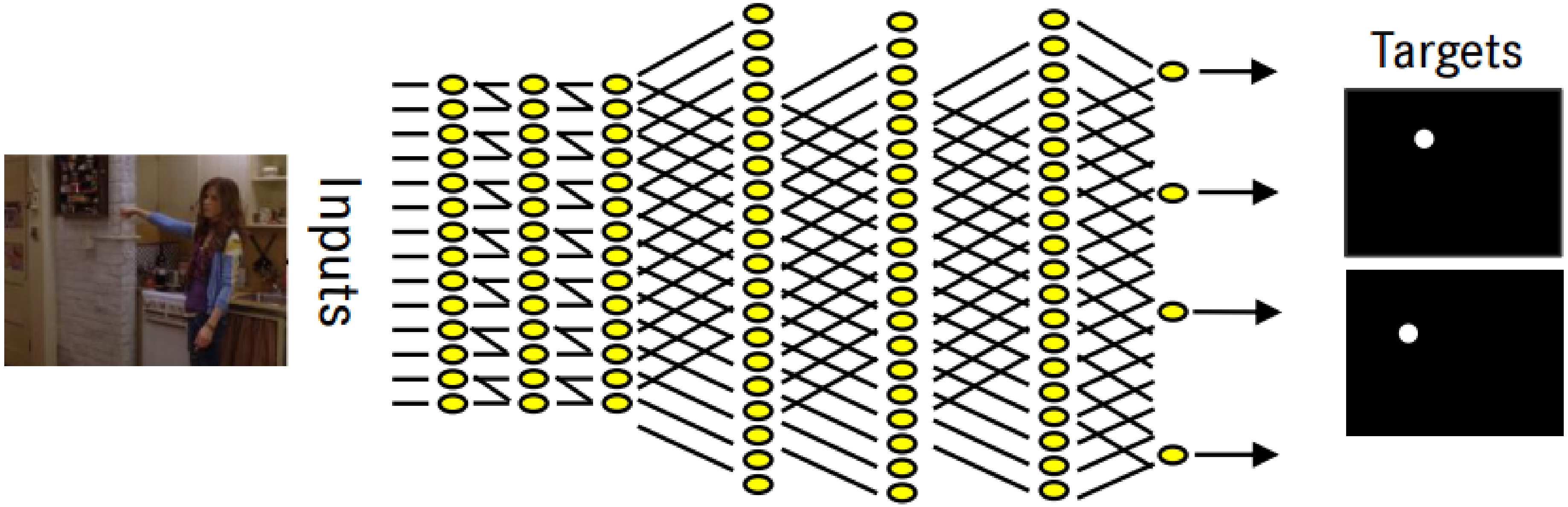


Ba, Jimmy, and Rich Caruana. "Do deep nets really need to be deep?." *NIPS,*

Student – Teacher Networks

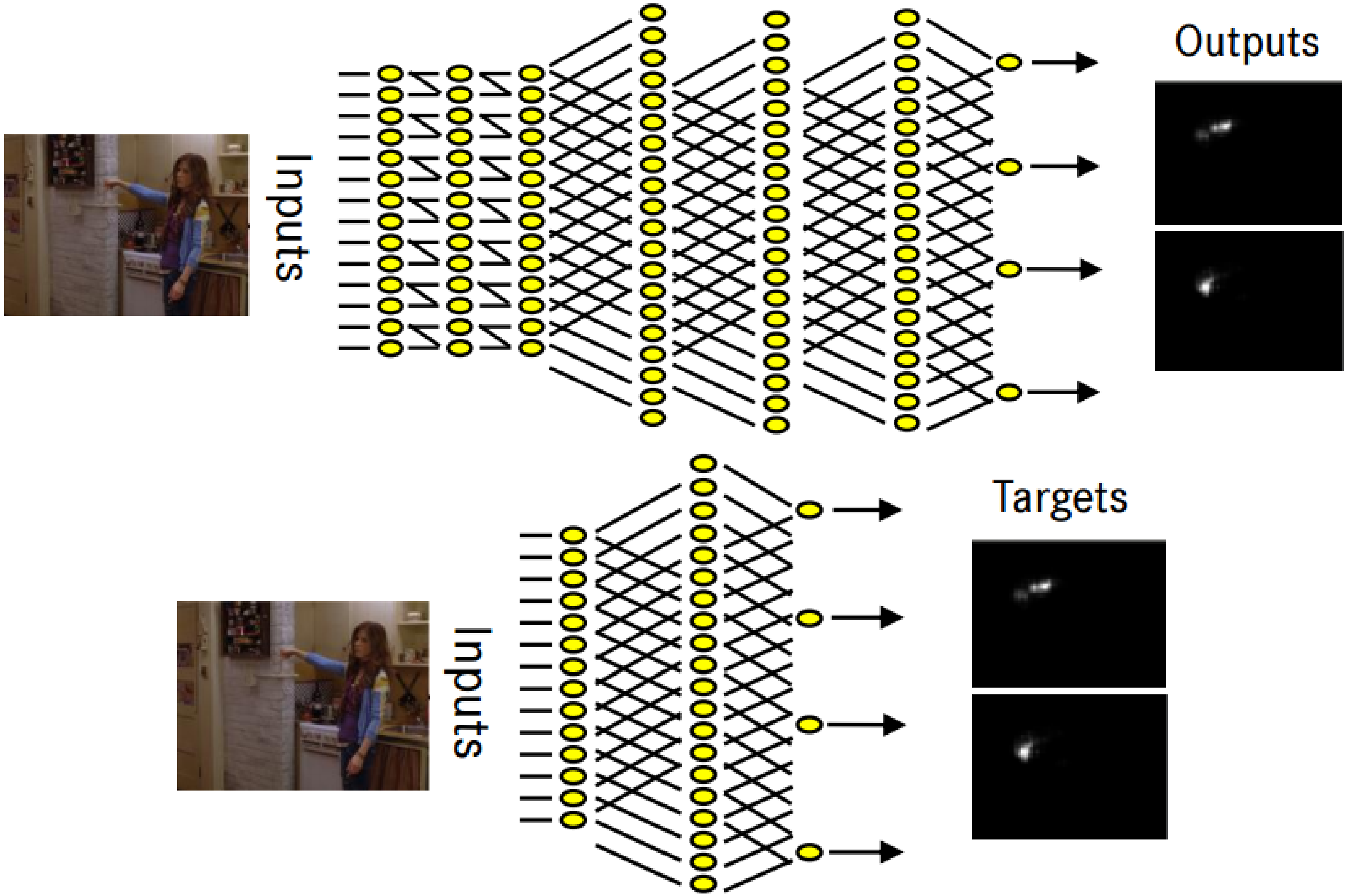
- Train a shallower network (student) with outputs of the larger network (teacher) as target.
- Deep Neural Networks (DNNs) excel over Shallow Neural Networks (SNNs). Why?
  - DNNs have more parameters.
  - DNNs can learn more complex functions?
  - Convolution gives a plus?
- Methods:
  - Do deep nets really need to be deep?
  - Knowledge Distillation
  - Fit Nets

Student-Teacher Networks: Training - 1



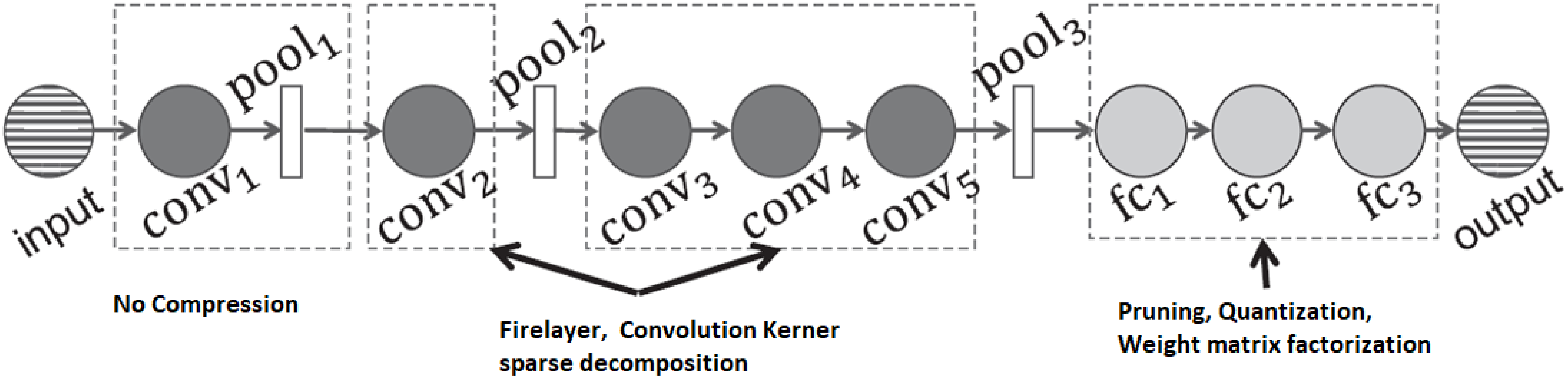
Train a DNN with original labelled data

Student-Teacher Networks: Training - 2



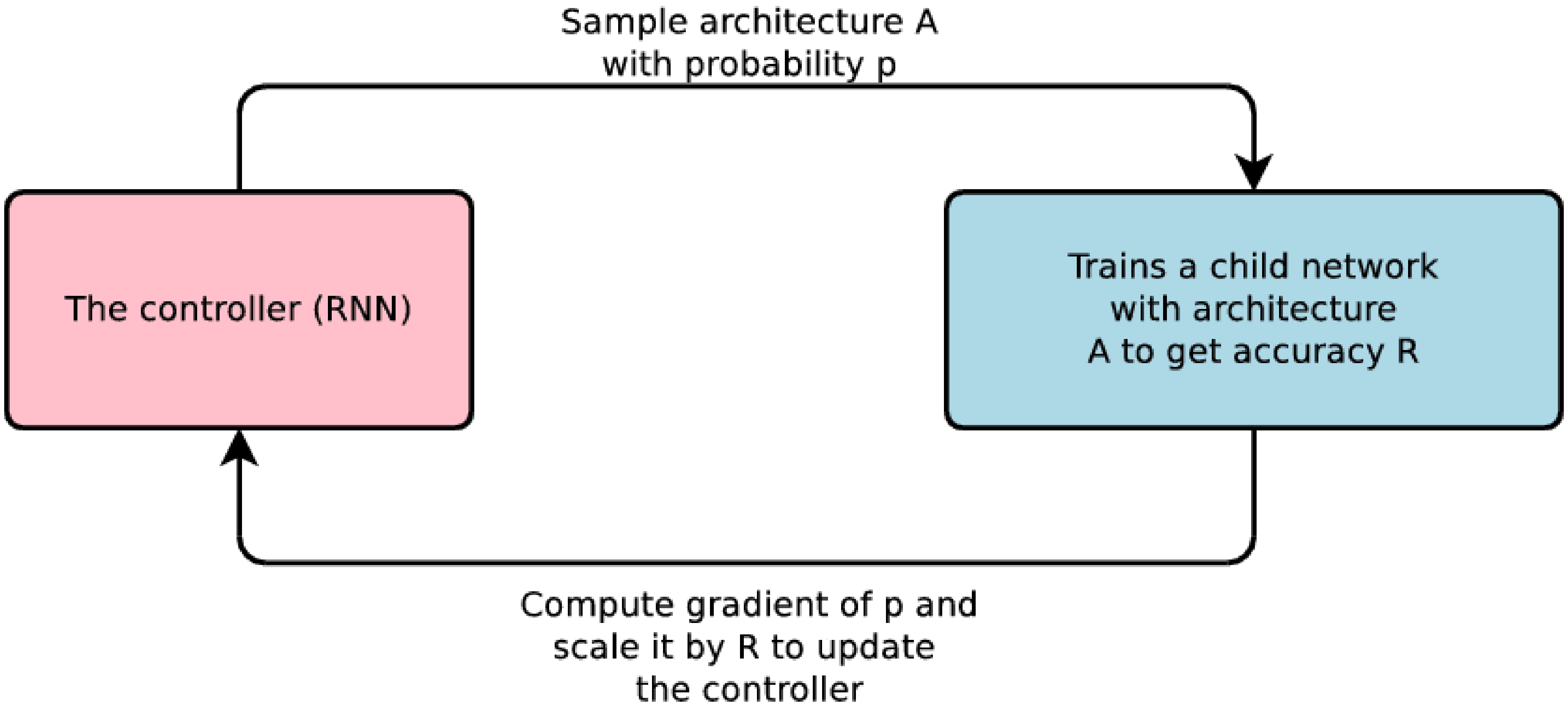
Train SNN with the output of DNN

Can we apply multiple techniques together

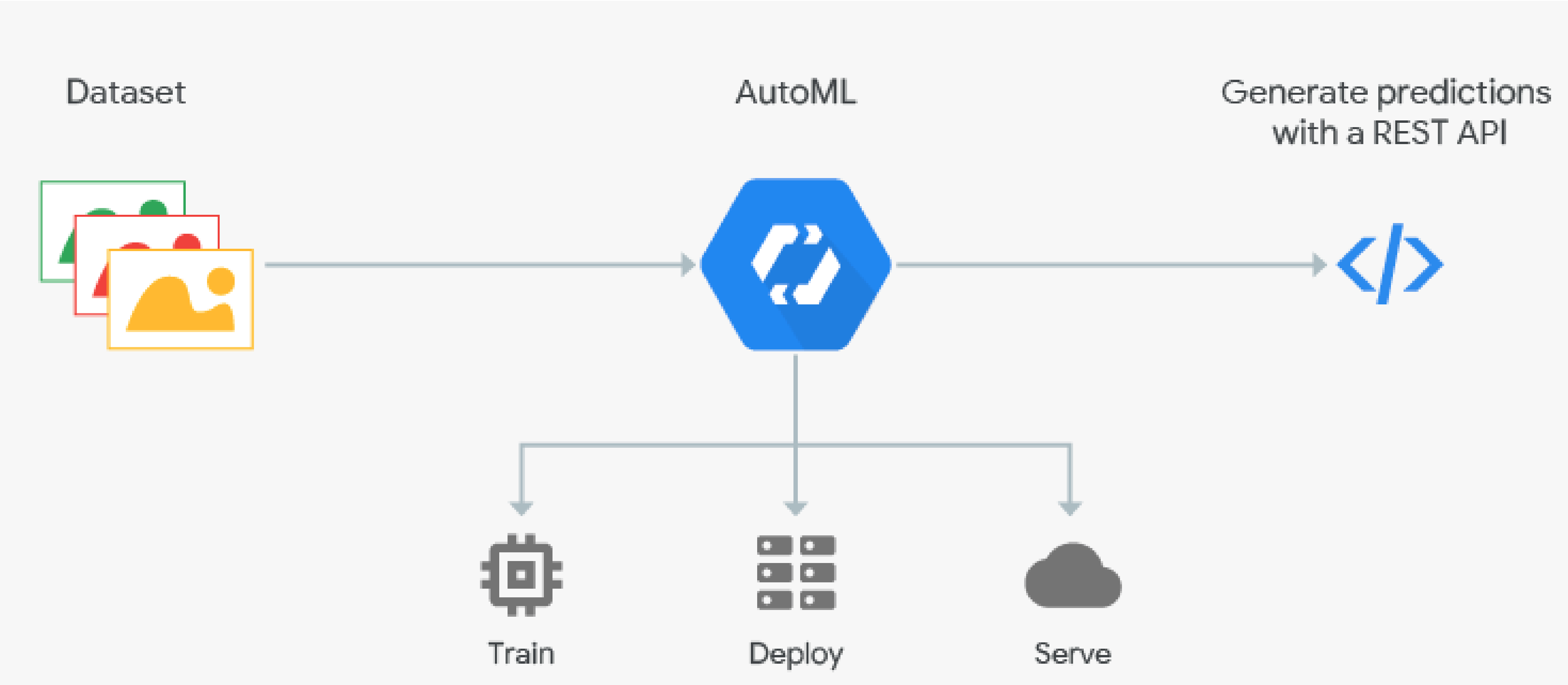


Neural Architecture Search

NAS is an algorithm that *searches* for the best*neural network architecture*

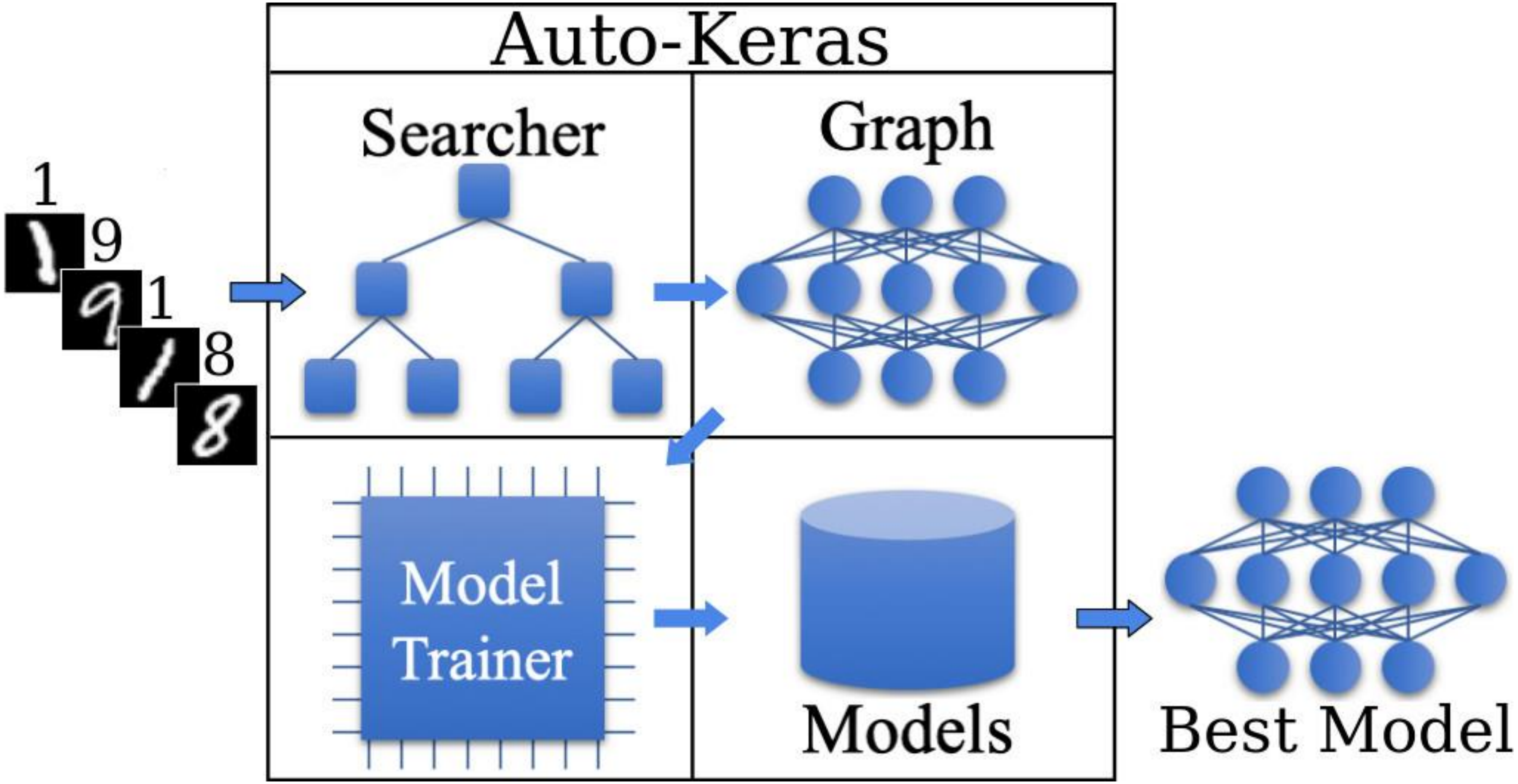


AutoML



Dataset
AutoML
Generate predictions
with a REST API
Train
Deploy
Serve

AutoKeras


Auto-Keras
Searcher
Graph
Model
Trainer
Models
Best Model
1
9
1
8

Model training with AutoKeras

```
Import autokeras as ak
model = ak.ImageClassifier(verbose=True)
model.fit(trainX, trainY, time_limit=seconds)
model.final_fit(trainX, trainY, testX, testY, retrain=True)
```


Thank you!

Lab

https://github.com/mishravipul/AdvanceDeepLearning