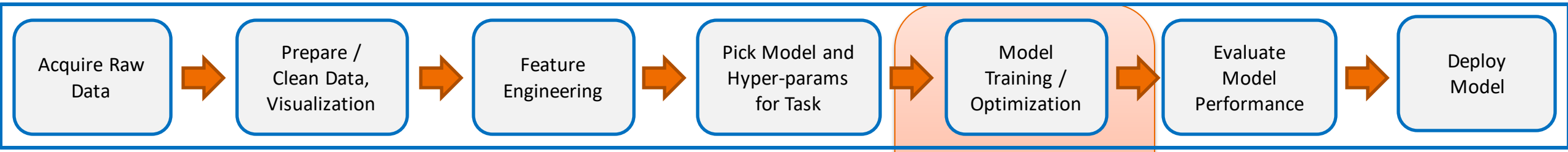
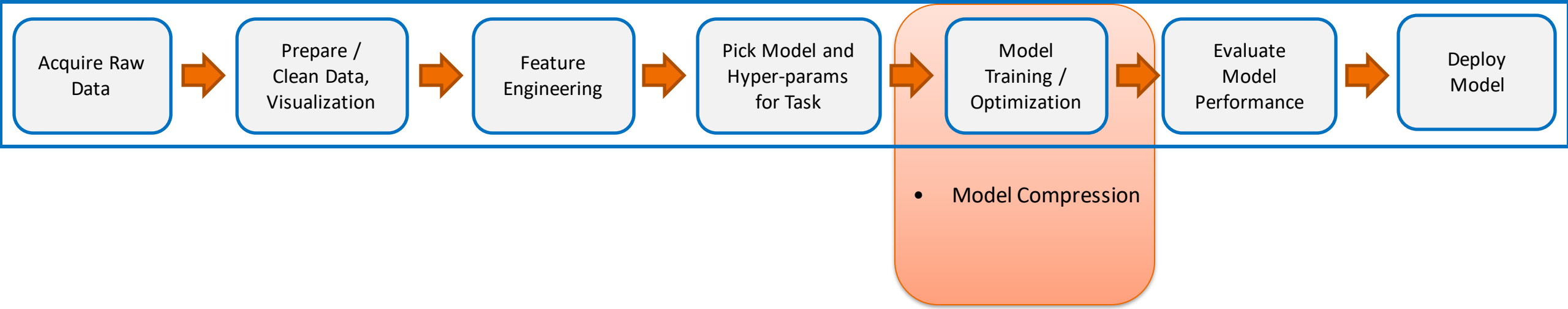
Focus for this lecture

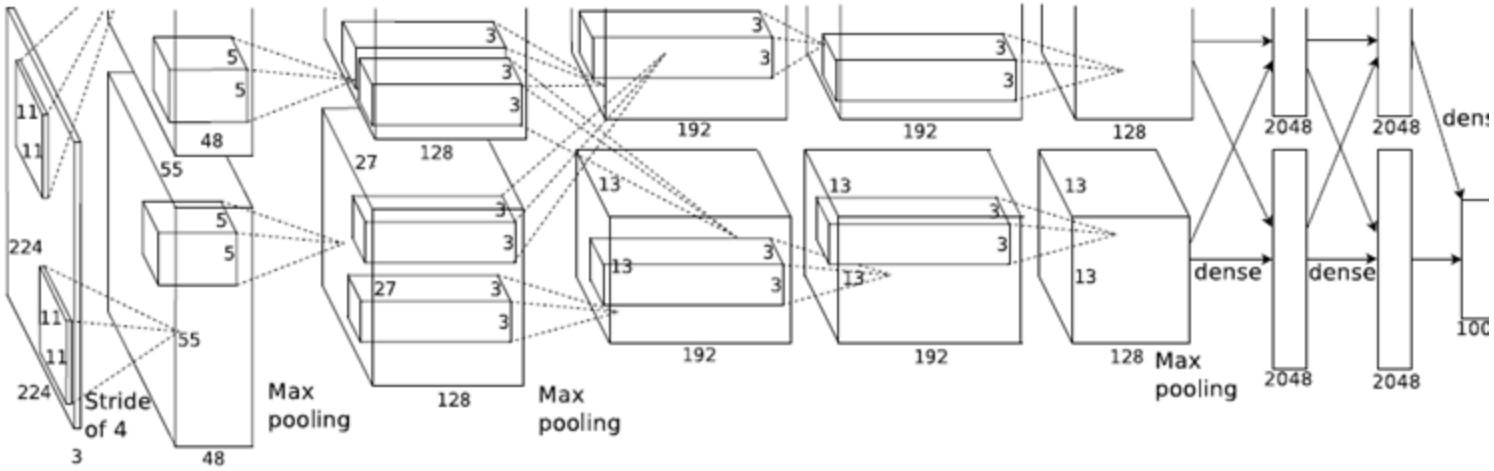


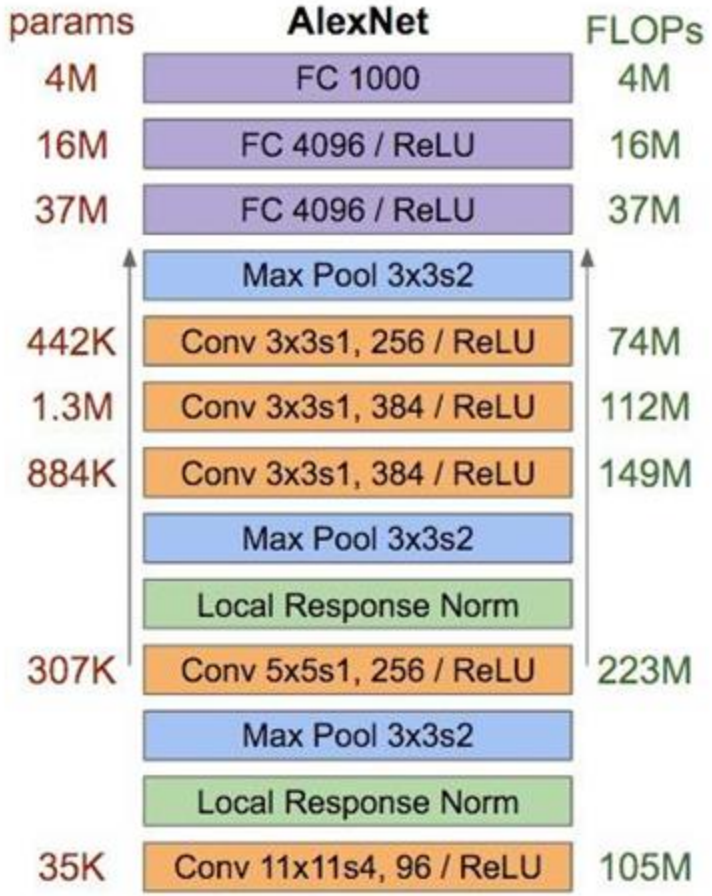
- Model Compression
- Pruning
- Quantization



Model Compression

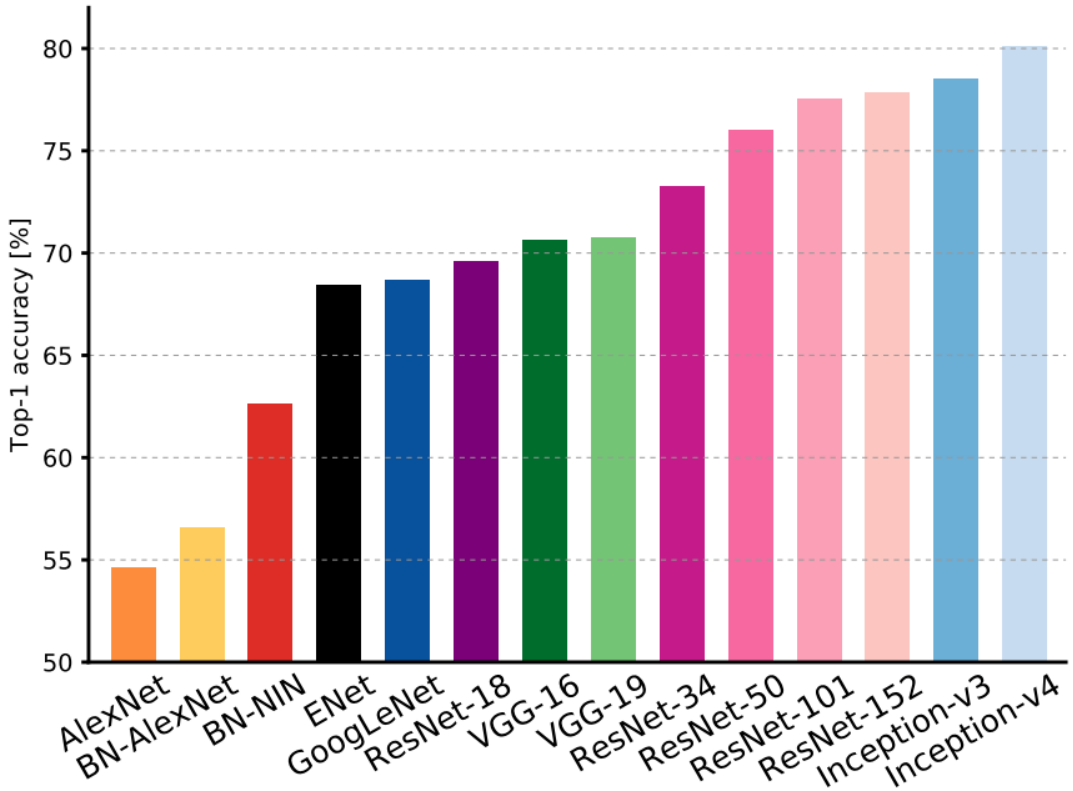
Big Huge Neural Network!

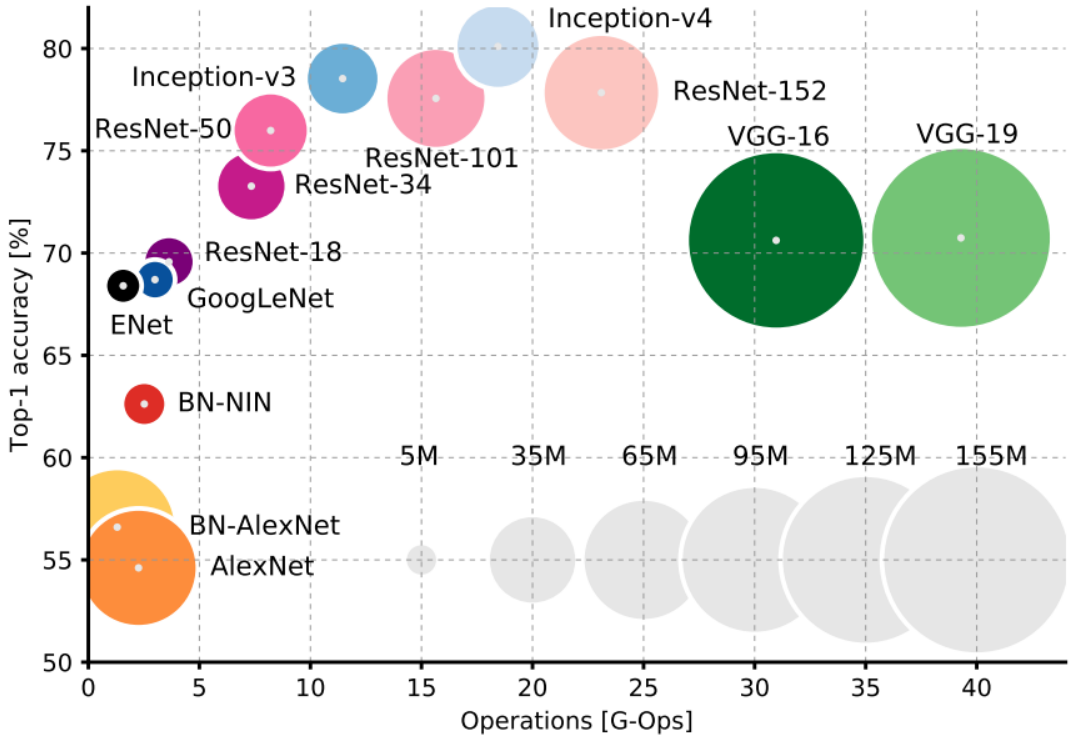




AlexNet - 60 Million Parameters = 240 MB

Performance Trade-offs





An analysis of deep neural network models for practical applications, 2017

& the Humble Mobile Phone

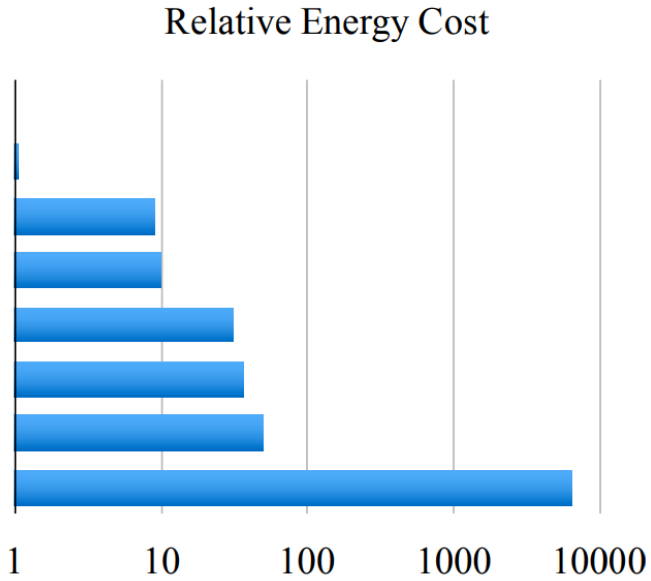
1 GB RAM

1/2 Billion FLOPs

NOT SO BAD!

& the Humble Mobile Phone

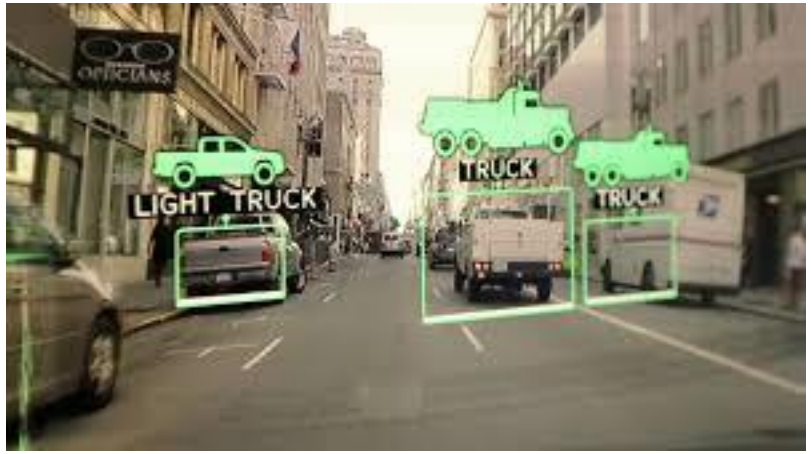
Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

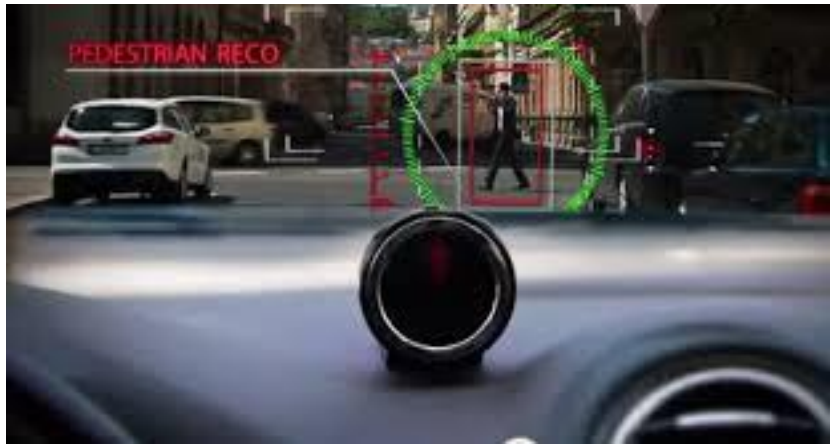


But wait! What about battery life?

Large networks do not fit in on-chip storage and hence require the more costly DRAM accesses. Running a 1 billion connection neural network, for example, at 20fps would require (20Hz)(1G)(640pJ) = 12.8W just for DRAM access - well beyond the power envelope of a typical mobile device.

Self Driving Cars!





Can we do 30 fps?

Aiding the blind

Can we do 30 fps?

Running Model in the Cloud

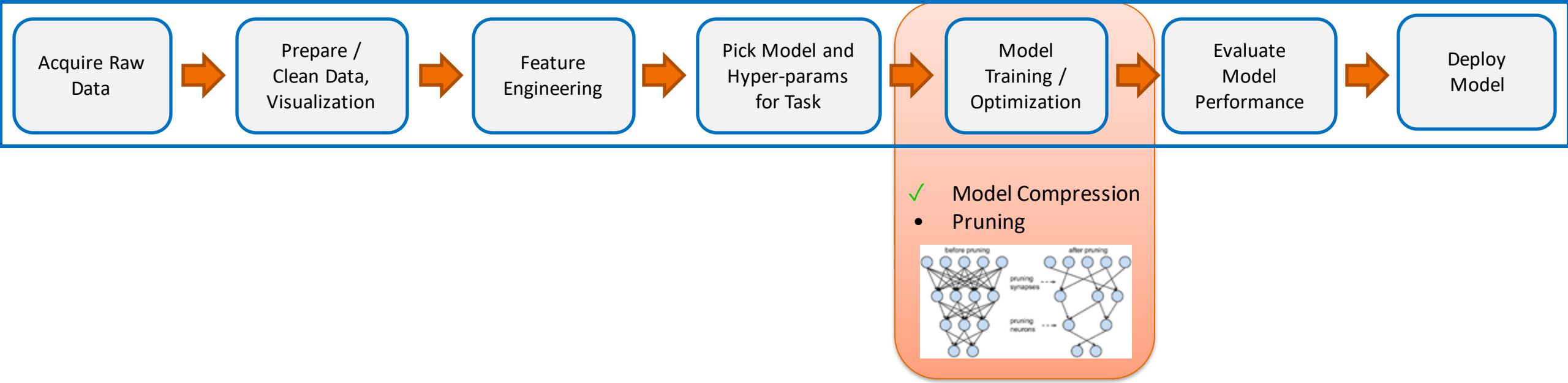
1. Network Delay
2. Power Consumption
3. User Privacy

Issues on Mobile Devices

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

What are Neural Networks made of?

- Fully Connected Layer : Matrices
- Convolutional Layer : Kernels (Tensors)



PRUNING

Compressing Matrices by making them Sparse

Why Pruning ?

- Deep Neural Networks have redundant parameters.
- Such parameters have a negligible value and can be ignored.
- Removing them does not affect performance.
- 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

Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

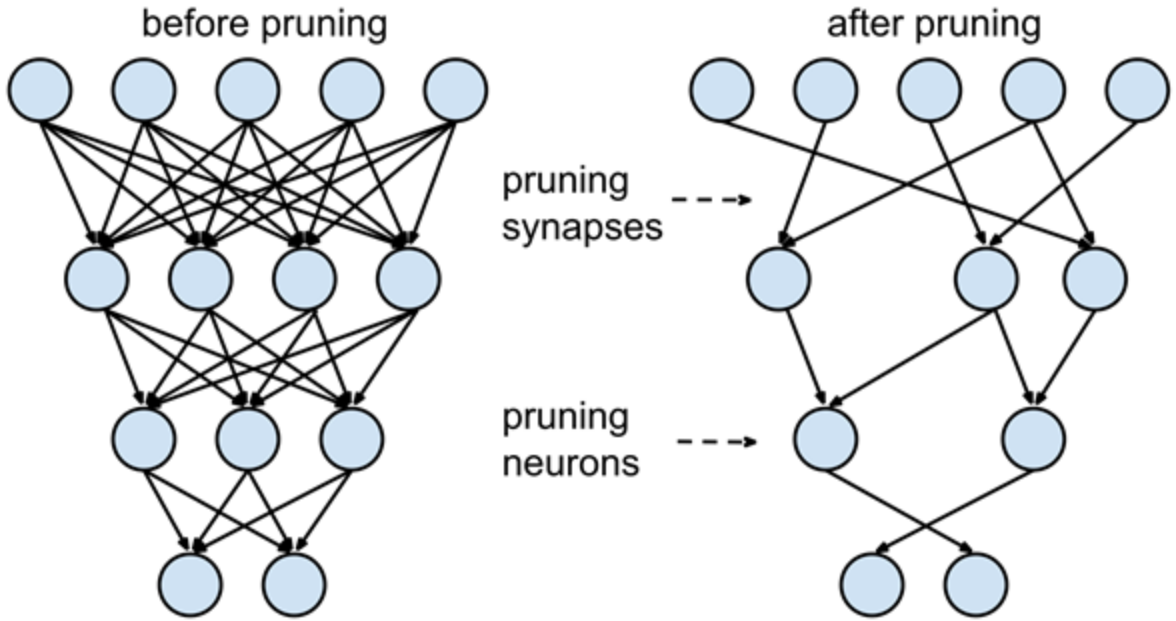
We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

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

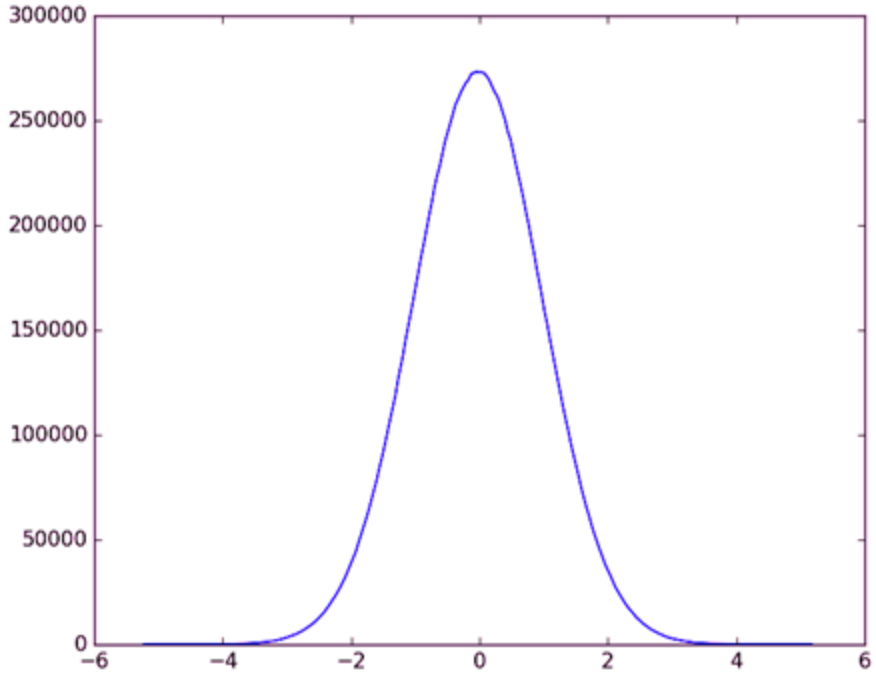
(After Training)

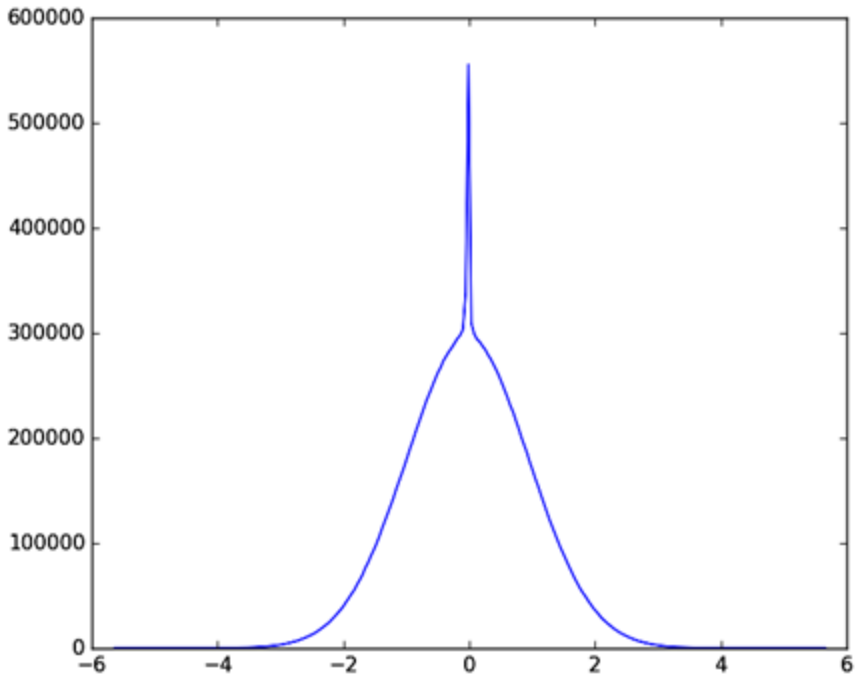


- The matrices can be made sparse. A naive method is to drop those weights which are 0 after training.
- Drop the weights below some threshold.
- Can be stored in optimized way if matrix becomes sparse.
- Sparse Matrix Multiplications are faster.

Ensuring Sparsity

- Addition of L1 regulariser to ensure sparsity.
- L1 better than L2.





Sparsify at Training Time

Iterative pruning and retraining

Train for few epochs → Prune weights which are 0 or below some threshold → Start training again

Add regularizer (weight decay) to loss function

$$\text{Loss} + \lambda \sum |w_i|^r$$

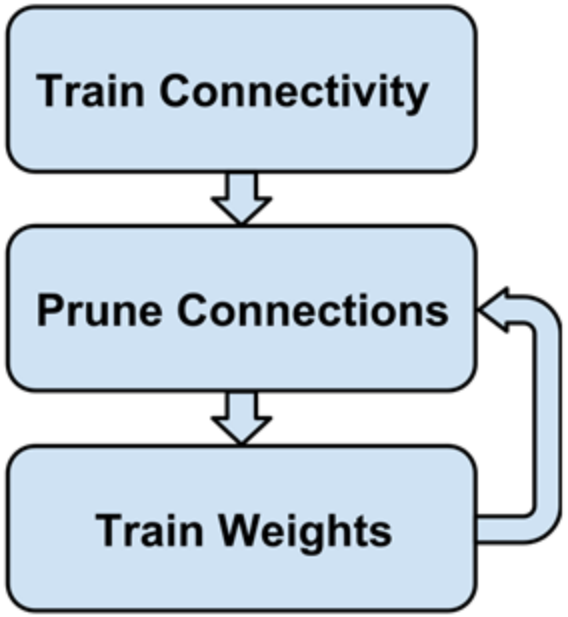
L1 regularizer : r = 1, L2 regularizer : r = 2

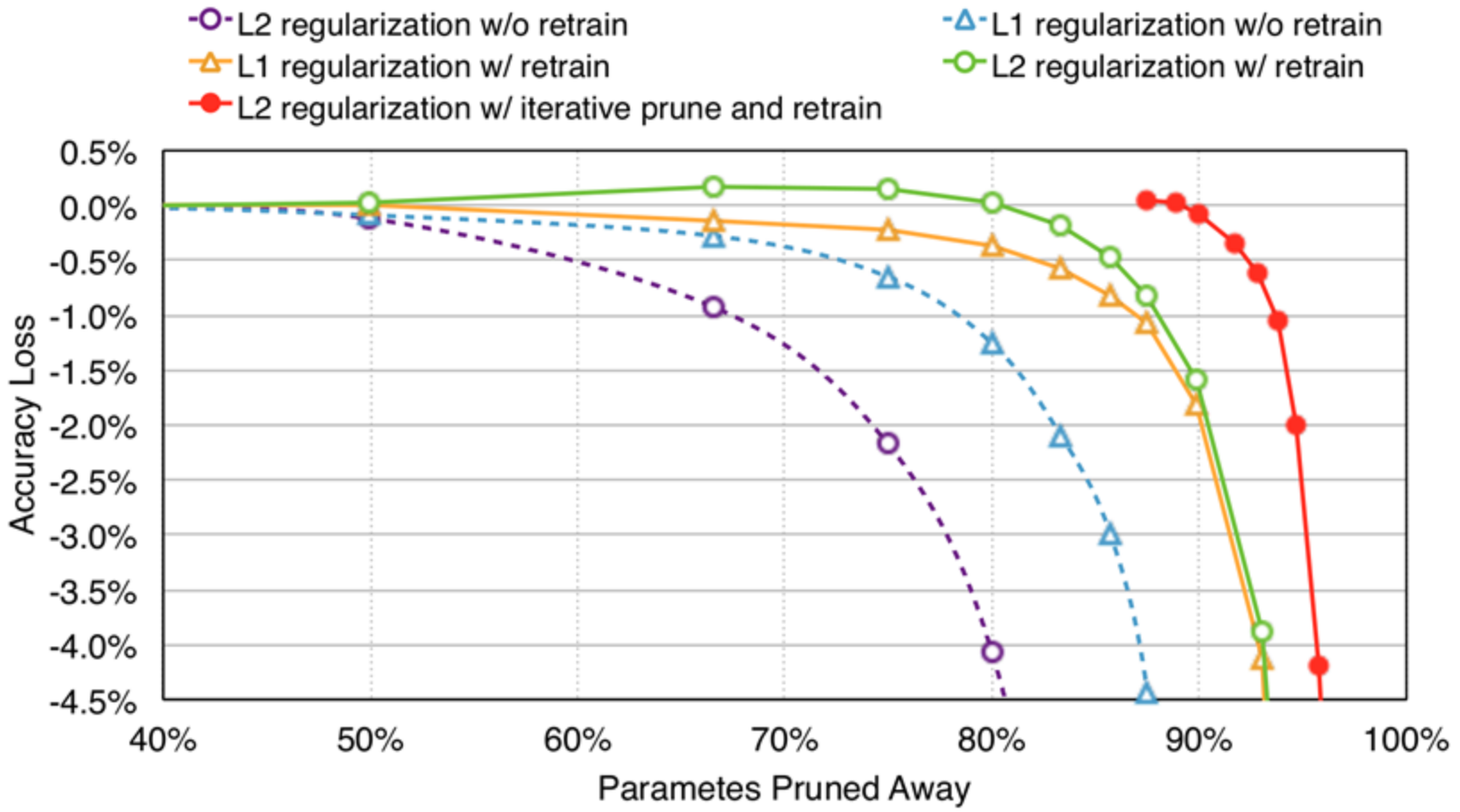
Learning both Weights and Connections for Efficient Neural Networks

https://arxiv.org/pdf/1506.02626

by S Han - 2015 - Cited by 233 - Related articles

Sparsify at Training Time





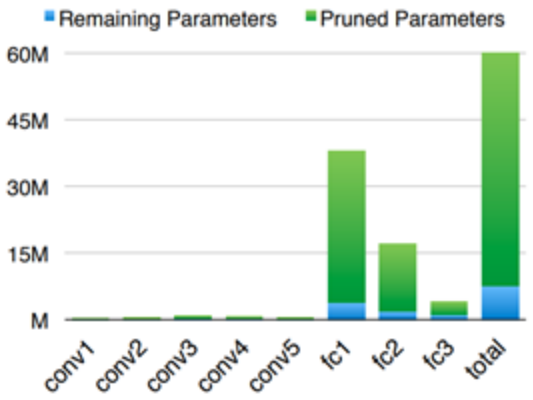
Learning both Weights and Connections for Efficient Neural Networks

https://arxiv.org/pdf/1506.02626

by S Han - 2015 - Cited by 233 - Related articles

- For AlexNet, pruning reduces the number of weights by 9× and computation by 3×.

Layer	Weights	FLOP	Weights%	FLOP%
conv1	35K	211M	84%	84%
conv2	307K	448M	38%	33%
conv3	885K	299M	35%	18%
conv4	663K	224M	37%	14%
conv5	442K	150M	37%	14%
fc1	38M	75M	9%	3%
fc2	17M	34M	9%	3%
fc3	4M	8M	25%	10%
Total	61M	1.5B	**11%**	**30%**



- For VGG-16, pruning reduces the number of weights by 12× and computation by 5×.

Layer	Weights	FLOP	Weights%	FLOP%
conv1_1	2K	0.2B	58%	58%
conv1_2	37K	3.7B	22%	12%
conv2_1	74K	1.8B	34%	30%
conv2_2	148K	3.7B	36%	29%
conv3_1	295K	1.8B	53%	43%
conv3_2	590K	3.7B	24%	16%
conv3_3	590K	3.7B	42%	29%
conv4_1	1M	1.8B	32%	21%
conv4_2	2M	3.7B	27%	14%
conv4_3	2M	3.7B	34%	15%
conv5_1	2M	925M	35%	12%
conv5_2	2M	925M	29%	9%
conv5_3	2M	925M	36%	11%
fc6	103M	206M	4%	1%
fc7	17M	34M	4%	2%
fc8	4M	8M	23%	9%
total	138M	30.9B	**7.5%**	**21%**

Sensitivity of layers to pruning

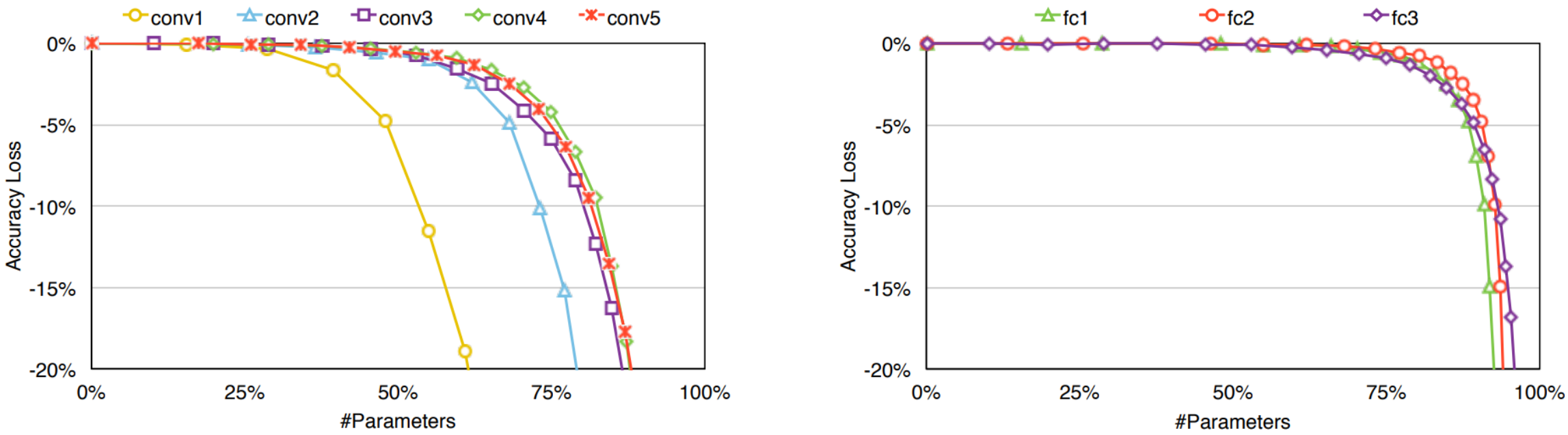
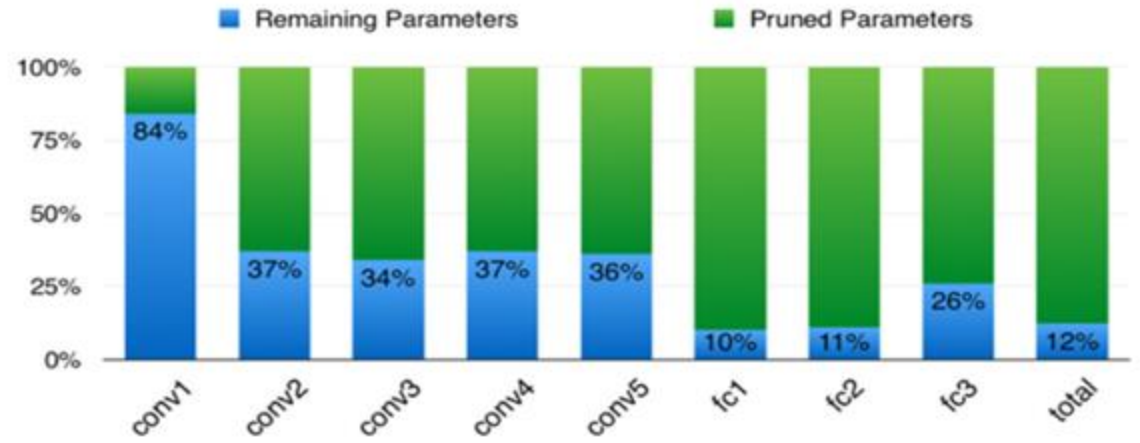
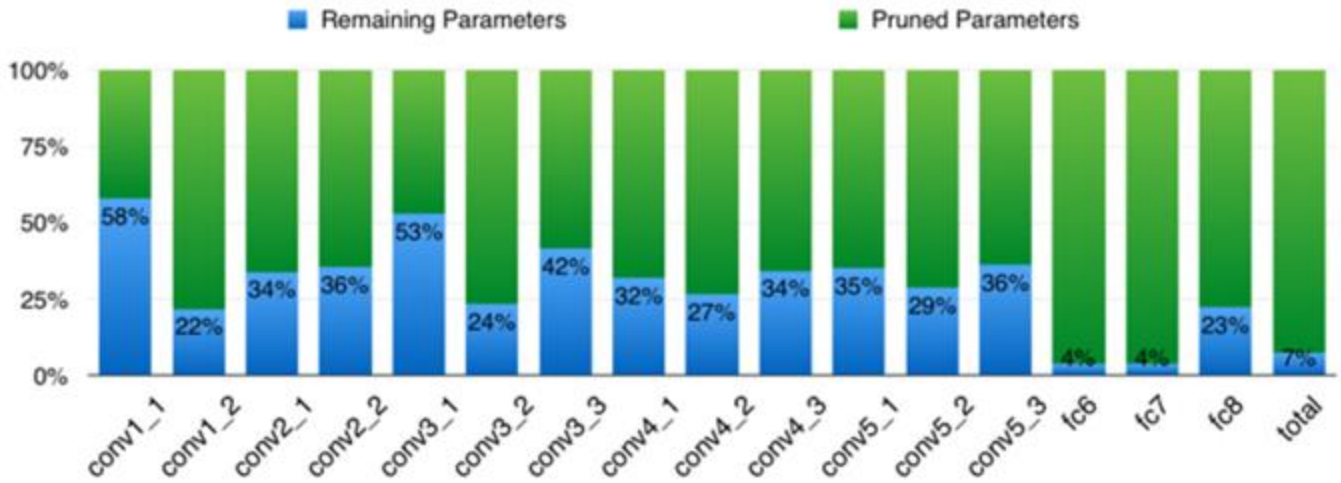


Figure 6: Pruning sensitivity for CONV layer (left) and FC layer (right) of AlexNet.

Remaining parameters in Different Layers



ALEXNET



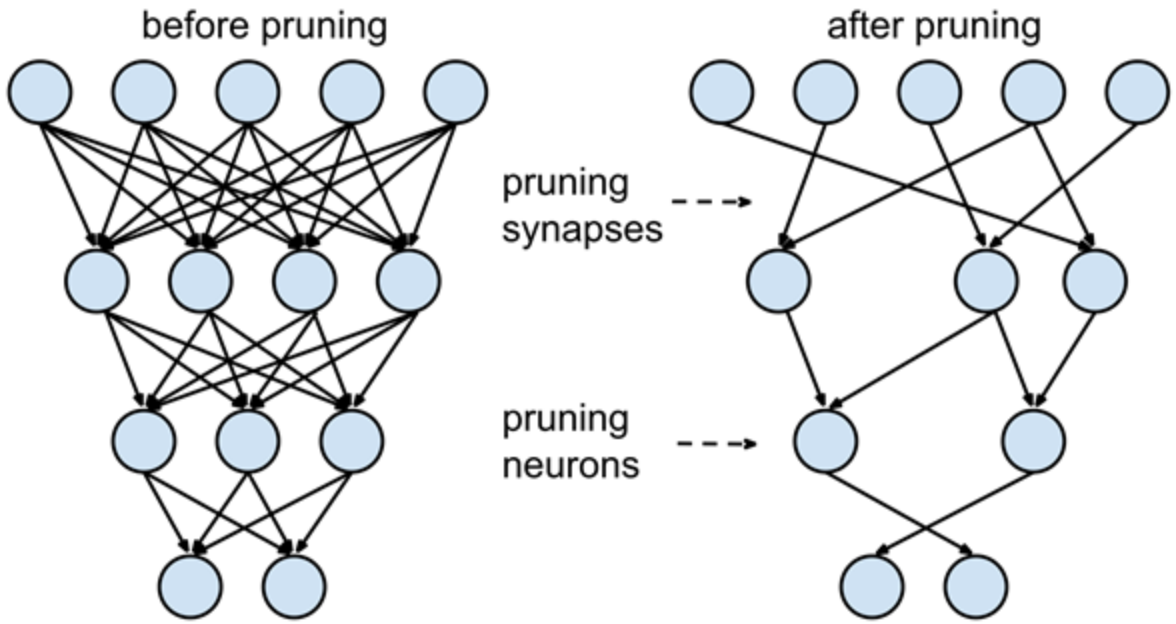
VGG16

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

Comments on Weight Pruning

- Matrices become sparse. Storage in HDD is efficient.
- Same memory in RAM is occupied by the weight matrices.
- Matrix multiplication is not faster since each 0 valued weight occupies as much space as before.
- Optimized Sparse matrix multiplication algorithms need to be coded up separately even for a basic forward pass operation.

Neuron Pruning



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

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

Dropping Neurons by Regularization

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

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

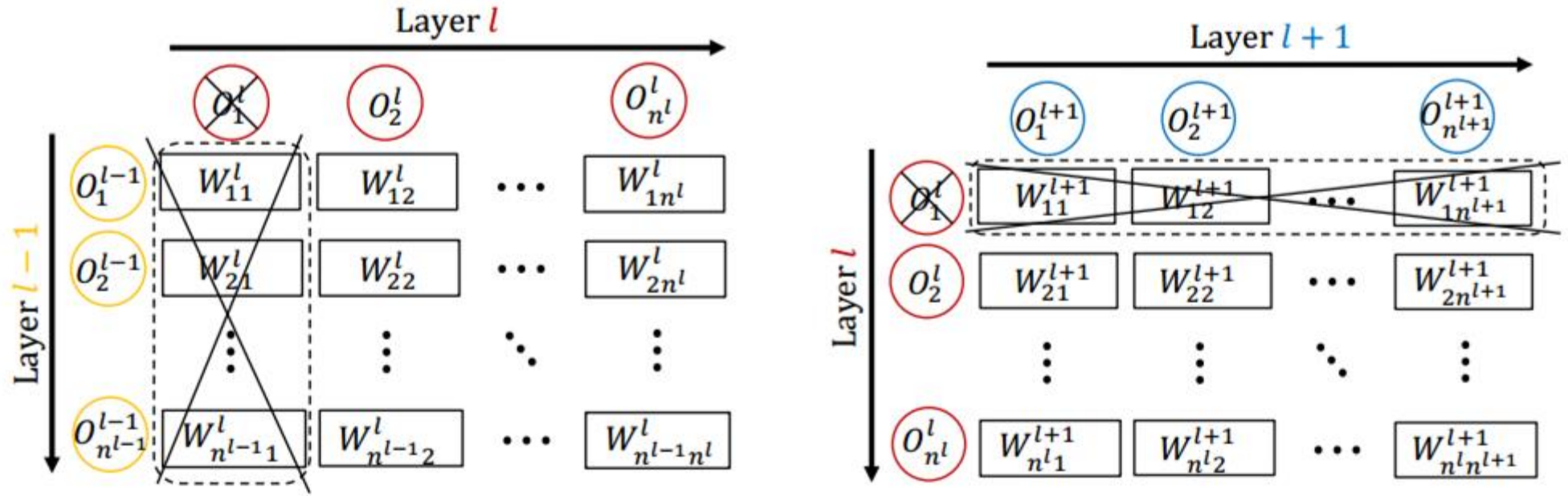
DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

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.

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

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

Results on FC Layer (MNIST)

Table 3: Summary of statistics for the fully connect layer of LeNet5 (average over 10 initialisations)

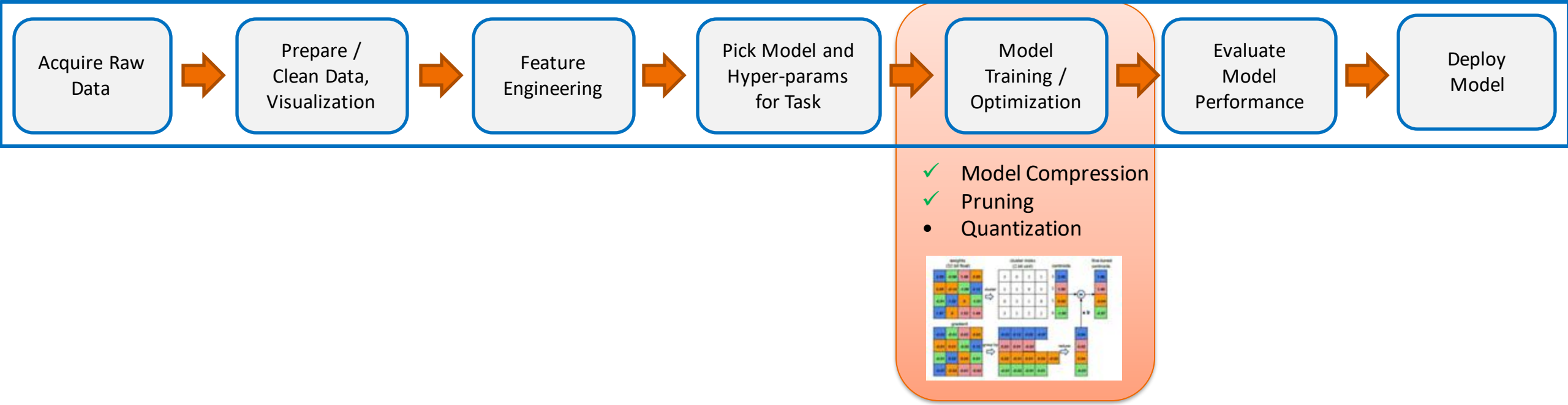
Regularisation	$\mathbf{W}^{\text{FC1}}\%$	$\mathbf{W}^{\text{FC2}}\%$	$\mathbf{W}^{\text{total}}\%$	Accuracy	Accuracy (no prune)
DO+P	55.15%	62.81%	55.17%	99.07%	99.12%
ℓ_1+DO+P	5.42%	51.66%	5.57%	99.01%	98.96%
ℓ_1+DN+P	1.44%	16.82%	1.49%	99.07%	99.14%
Regularisation	$\mathbf{O}^{\text{FC1}}\%$	$\mathbf{O}^{\text{FC2}}\%$	$\mathbf{O}^{\text{output}}\%$	$\mathbf{O}^{\text{total}}\%$	Compression Rate
DO+P	$\frac{3136}{3136} = 100\%$	$\frac{504}{512} = 98.44\%$	$\frac{10}{10} = 100\%$	$\frac{3650}{3658} = 99.78\%$	1.81
ℓ_1+DO+P	$\frac{1039}{3136} = 33.13\%$	$\frac{320}{512} = 62.5\%$	$\frac{10}{10} = 100\%$	$\frac{1369}{3658} = 37.42\%$	17.95 [4]
ℓ_1+DN+P	$\frac{907}{3136} = 28.92\%$	$\frac{110}{512} = 21.48\%$	$\frac{10}{10} = 100\%$	$\frac{1027}{3658} = 28.08\%$	67.04

- l1: l1 regularization, P: pruning, DO: Dropout, DN:DropNeuron, FC1: fully connected layer 1.

$$\texttt{l1\_regulariser} := \lambda_{\ell_1} \sum_{\ell=1}^{L} (\|\mathbf{W}^{\ell}\|_1 + \|\mathbf{b}^{\ell}\|_1)$$

- At the end of training, we prune the small-weight connections: all connections with absolute weights below a threshold (typically small, e.g. 10^−2) are removed from the network.

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



QUANTIZATION

Binary Quantization

- Size Drop : 32X $\hat{W}_{ij} = \begin{cases} 1 & \text{if } W_{ij} \geq 0, \\ -1 & \text{if } W_{ij} < 0. \end{cases}$
- 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 int's
- 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

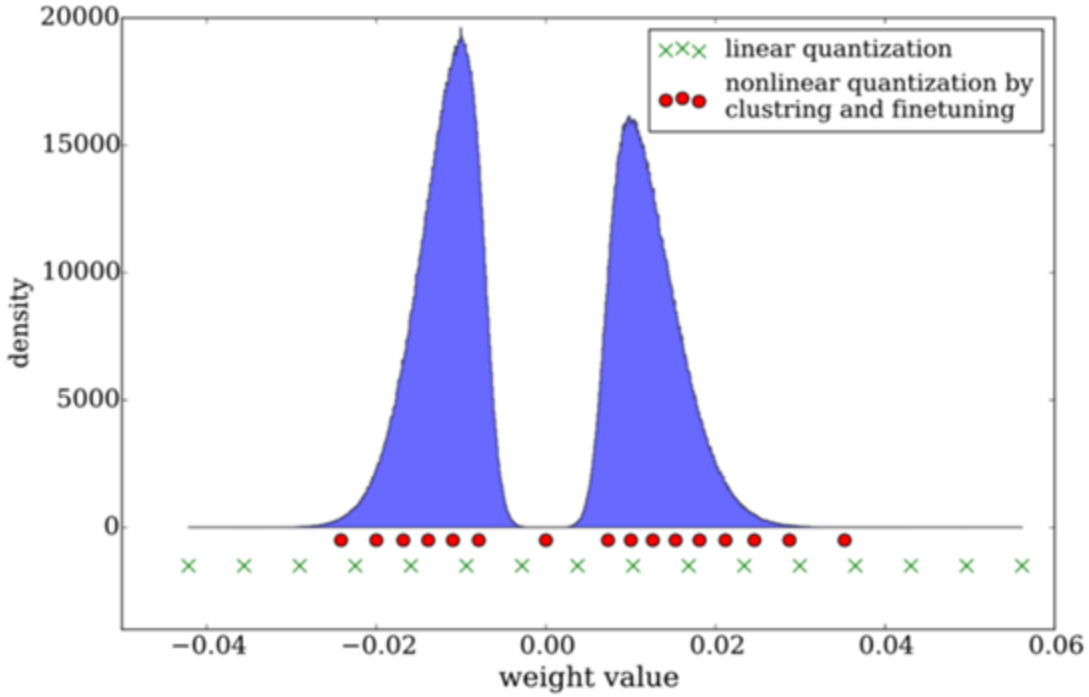
- Perform k-means clustering on weights.

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2$$

- Need to store mapping from integers to cluster centers. We only need log (k) bits to code the clusters.

- To calculate the compression rate, given k clusters, we only need log2(k) bits to encode the index. In general, for a network with n connections and each connection is represented with b bits, constraining the connections to have only k shared weights will result in a compression rate of:

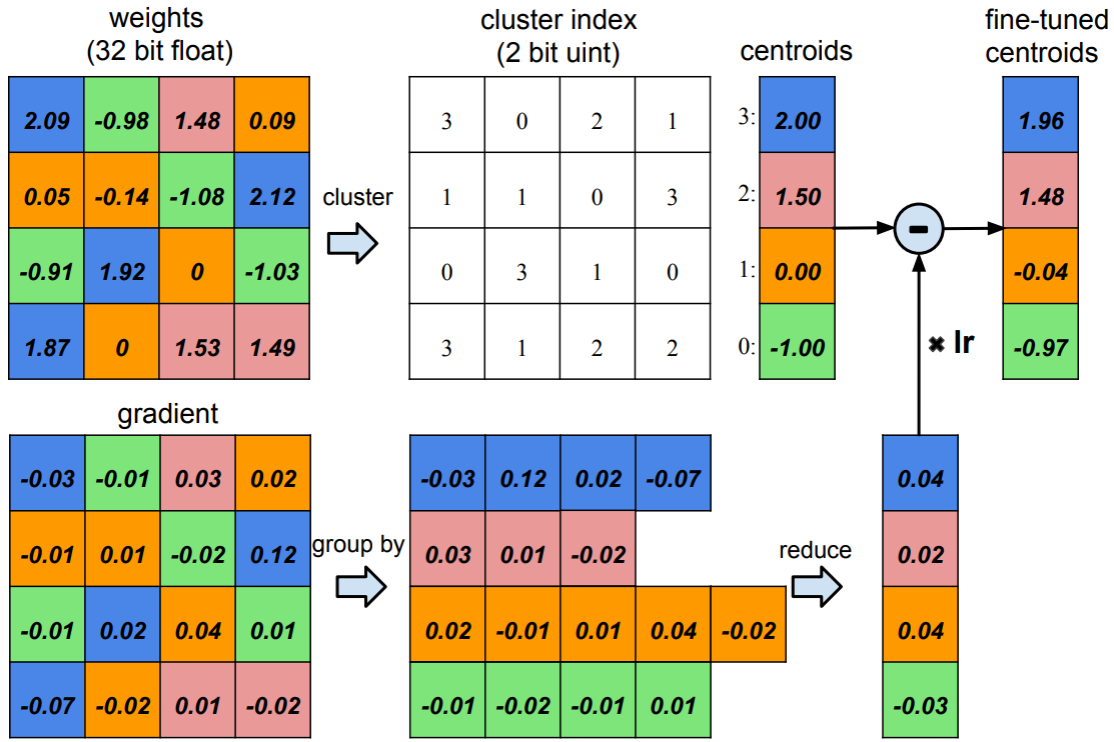
$$r = \frac{nb}{nlog_2(k) + kb}$$



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

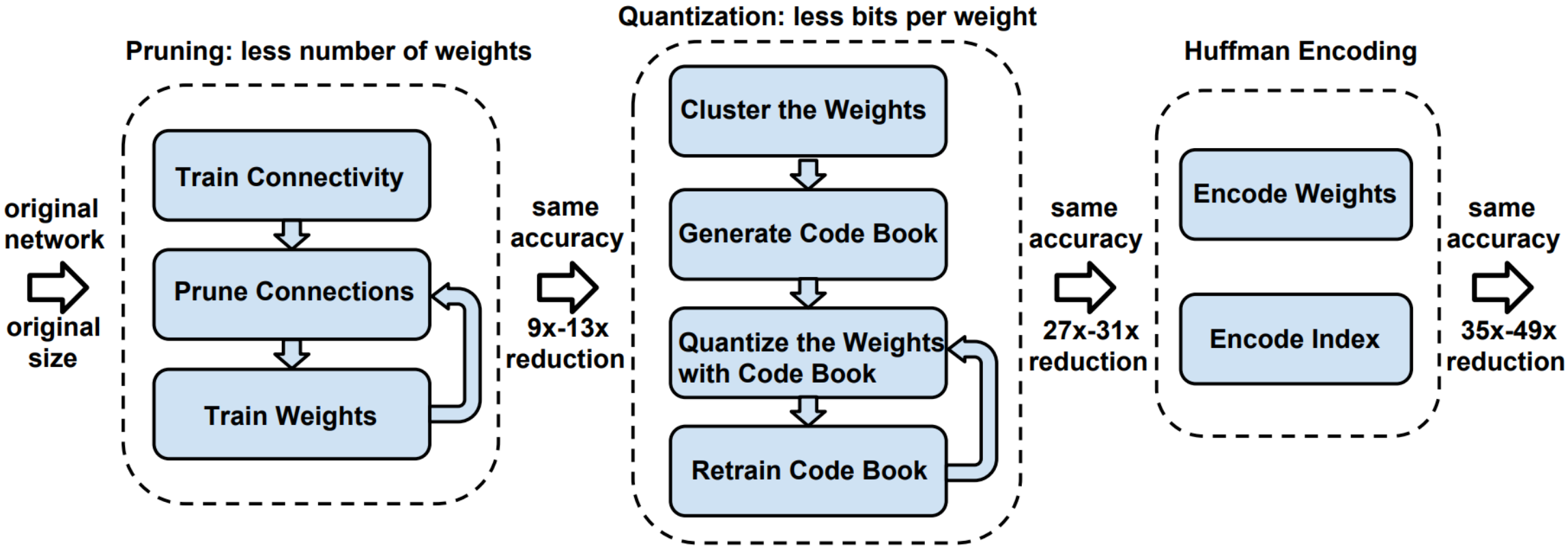
Weight Sharing while Training

- Iterate
  - Train
  - Cluster weights
  - Make them same
- Compute gradients with respect to centroids so that weight sharing is preserved during gradient update.



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

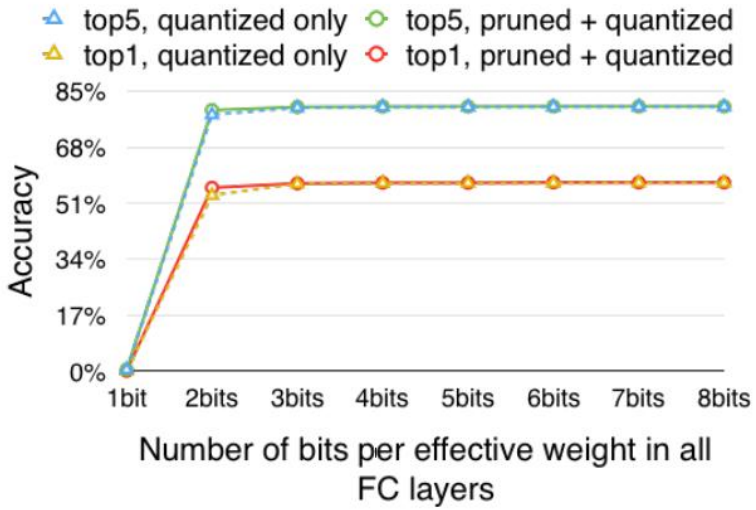
Deep Compression by Song Han

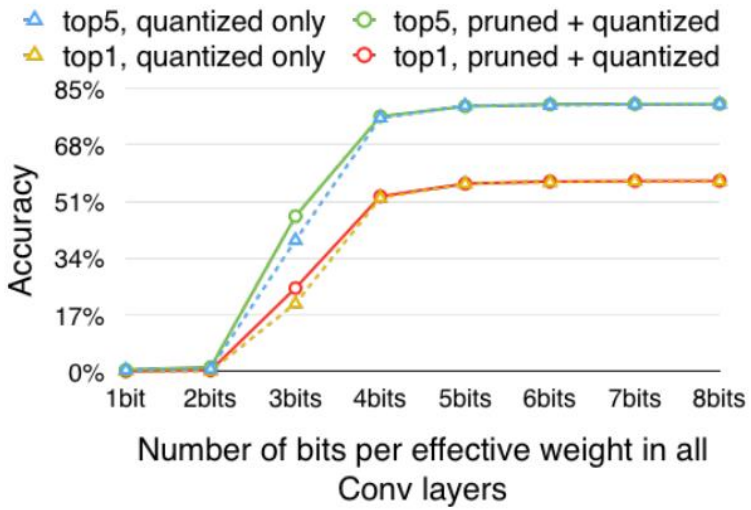


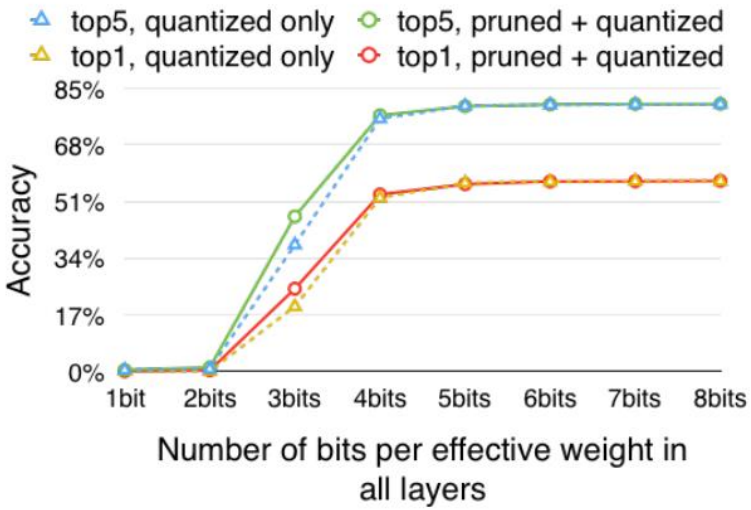
DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Deep Compression by Song Han

Pruning and Quantization works well together







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

Product Quantization

- Partition the given matrix into several submatrices and we perform k-means clustering for all of them.

$$W = [W^1, W^2, \ldots, W^s], \quad \text{where } W^i \in R^{m \times (n/s)} \text{ assuming } n \text{ is divisible by } s.$$

$$\min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2,$$

where $\boldsymbol{w}_z^i$ denotes the zth row of sub-matrix W^i, and $\boldsymbol{c}_j^i$ denotes the jth row of sub-codebook $C^i \in R^{k \times (n/s)}$. For each sub-vector $\boldsymbol{w}_z^i$, we need only to store its corresponding cluster index and the codebooks.

$$\hat{W} = [\hat{W}^1, \hat{W}^2, \ldots, \hat{W}^s], \quad \text{where } \hat{\boldsymbol{w}}_j^i = \boldsymbol{c}_j^i, \quad \text{where } \min_j \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2.$$

Note that PQ can be applied to either the x-axis or the y-axis of the matrix.

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

Residual Quantization

First quantize the vectors into k-centres. $\min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z - \boldsymbol{c}_j^1\|_2^2$

Next step is to find out the residuals for each data point (w-c) and perform k-means on the residuals. Do it recursively t times.

Then the resultant weight vectors are calculated as follows.

$$\hat{\boldsymbol{w}}_z = \boldsymbol{c}_j^1 + \boldsymbol{c}_j^2 + \ldots, \boldsymbol{c}_j^t$$

We need to store all the codebooks for each iteration, which potentially needs large a amount of memory. The compression rate is $m/(tk + \log_2(k)tn)$.

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

Matrix Factorization

We first consider matrix factorization methods, which have been widely used to speed up CNN (Denton et al., 2014) as well as for compressing parameters in linear models (Denton et al., 2014). In particular, we consider using singular-value decomposition (SVD) to factorize the parameter matrix. Given the parameter $W \in R^{m \times n}$ in one dense connected layer, we factorize it as

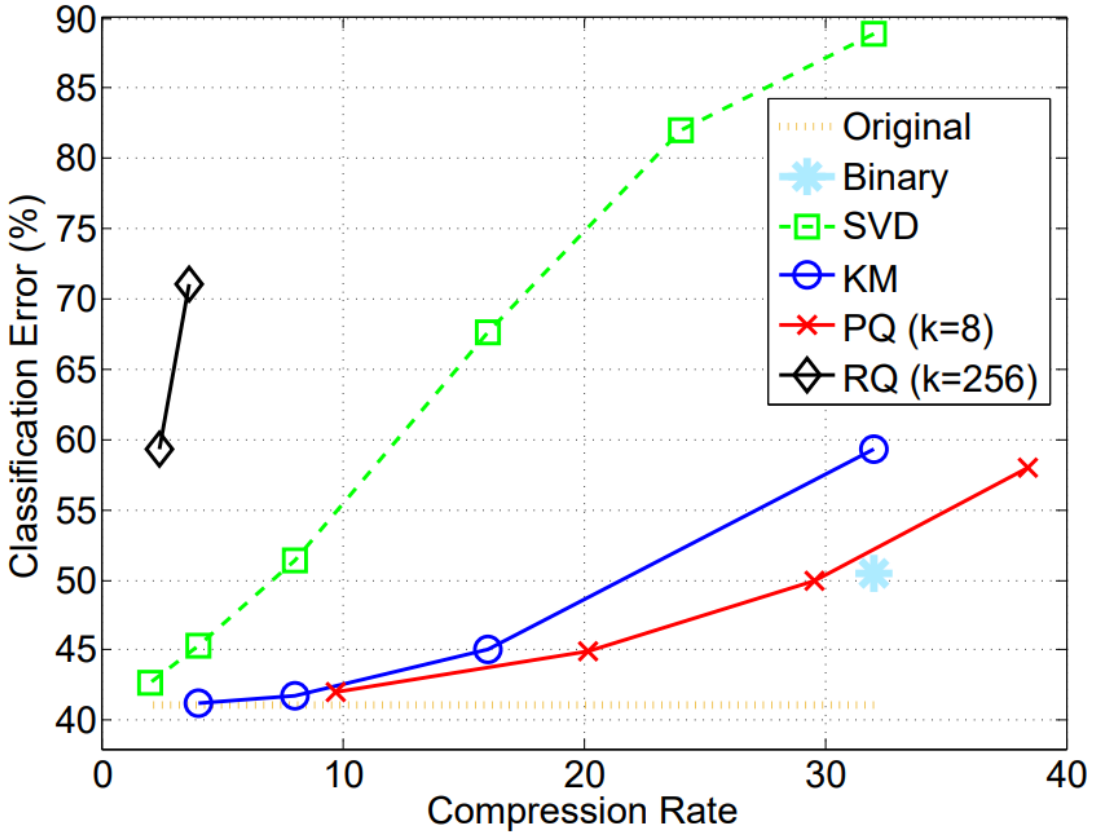
$$W = USV^T, \tag{1}$$

where $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are two dense orthogonal matrices and $S \in R^{m \times n}$ is a diagonal matrix. In order to approximate W using two much smaller matrices, we can pick the top k singular vectors in U and V with corresponding eigenvalue in S, to reconstruct W:

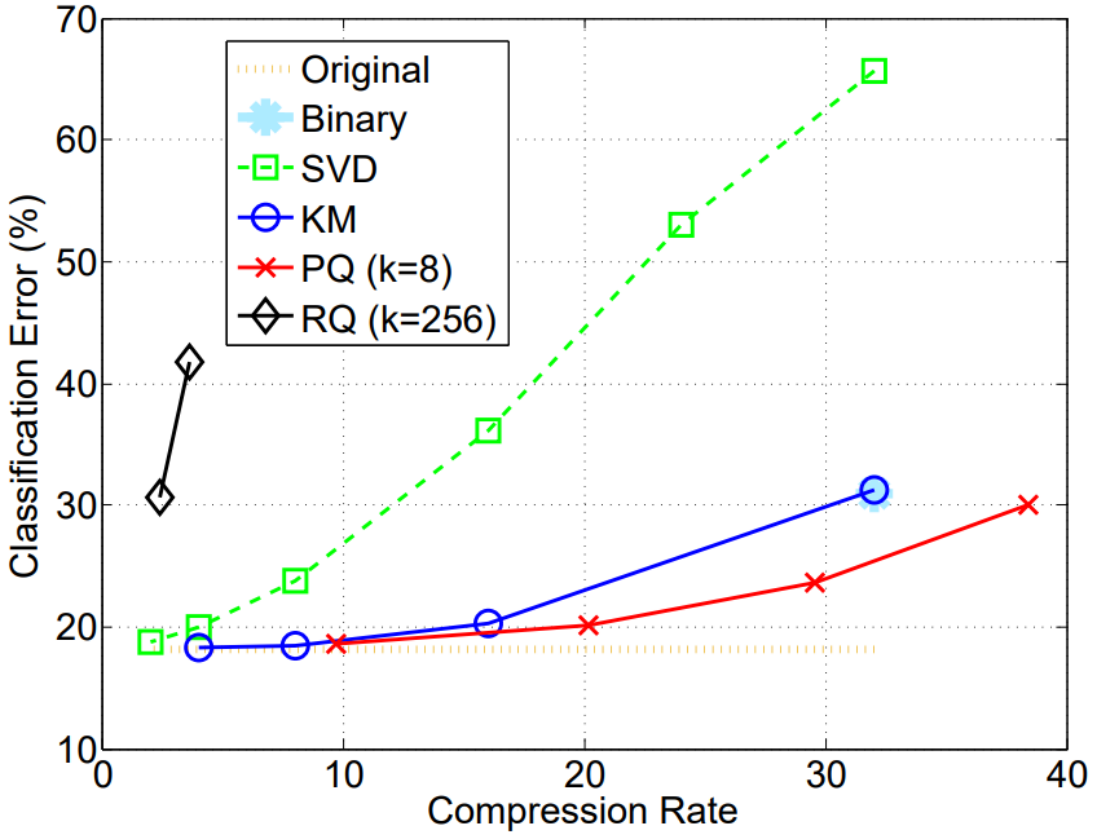
$$\hat{W} = \hat{U}\hat{S}\hat{V}^T, \tag{2}$$

where $\hat{U} \in R^{m \times k}$ and $\hat{V} \in R^{n \times k}$ are two submatrices that correspond to the leading k singular vectors in U and V. The diagonal elements in $\hat{S} \in R^{k \times k}$ correspond to the largest k singular values. The approximation of SVD is controlled by the decay along the eigenvalues in S. The SVD method is optimal in the sense of a Frobenius norm, which minimizes the MSE error between the approximated matrix $\hat{W}$ and original W. The two low-rank matrices $\hat{U}$ and $\hat{V}$, as well as the eigenvalues, must be stored. So the compression rate given m, n, and k is computed as $mn/k(m+n+1)$.

Comparison of Quantization methods on magenet



(a) Accuracy@1



(b) Accuracy@5

Figure 3: Comparison of different compression methods on ILSVRC dataset.

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

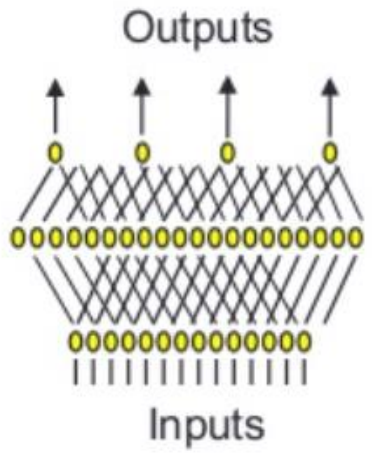
Summary

- Pruning weights and neurons
- Uniform Quantization
- Non-Uniform Quantization / Weight Sharing

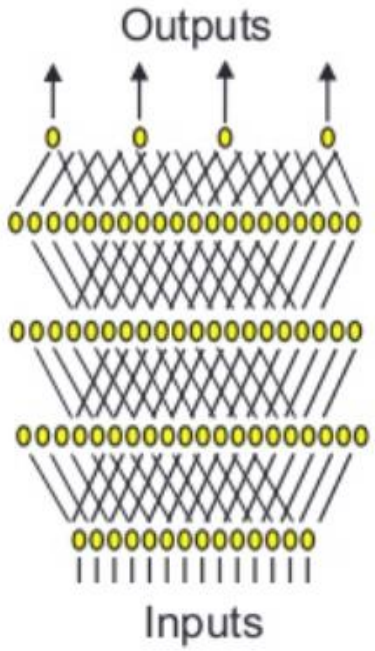
Student – Teacher Networks

NNs

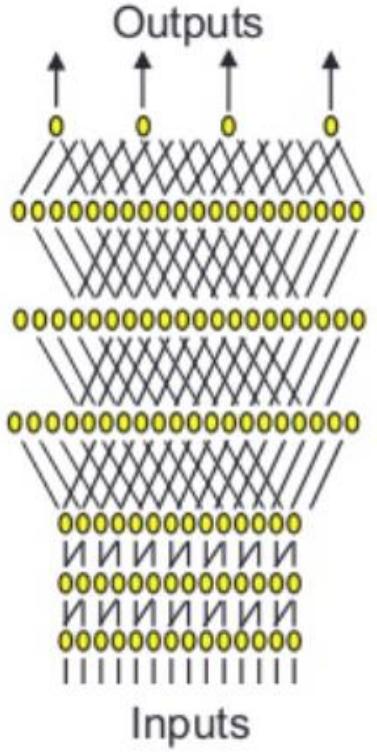
SNN: Single Hidden Layer



DNN: Three Hidden Layers



CNN: Three Hidden Layers above Convolutional/MaxPooling Layers

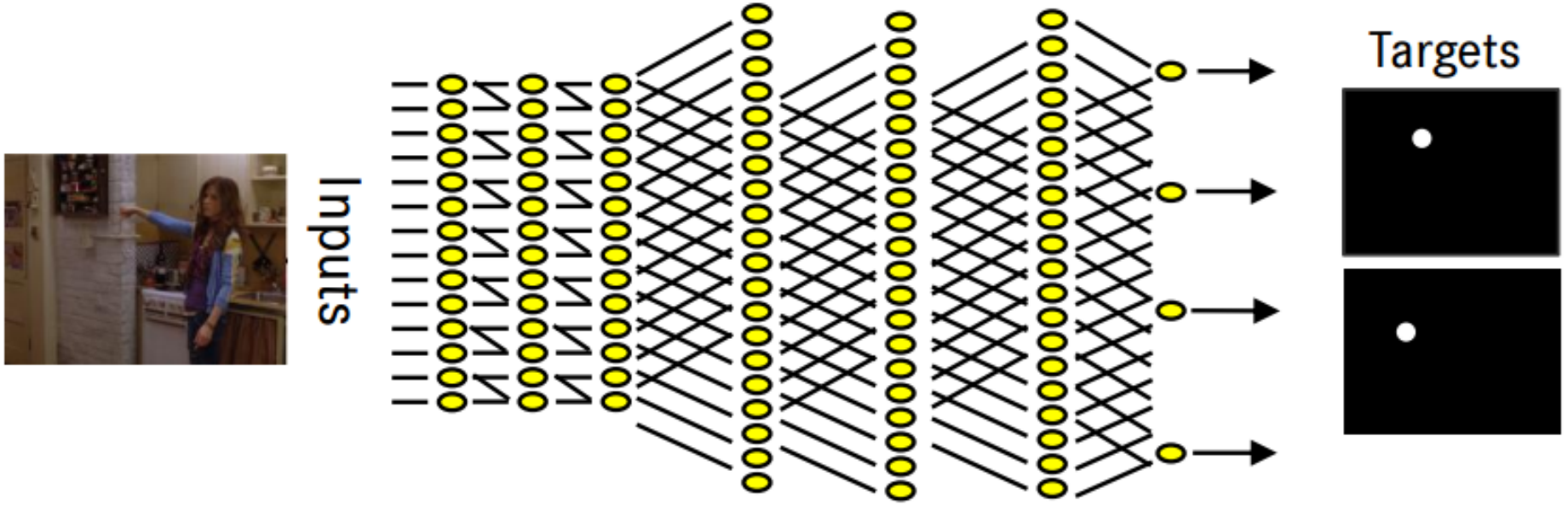


Ba, Jimmy, and Rich Caruana. "Do deep nets really need to be deep?." *NIPS,* 2014.

Student – Teacher Networks

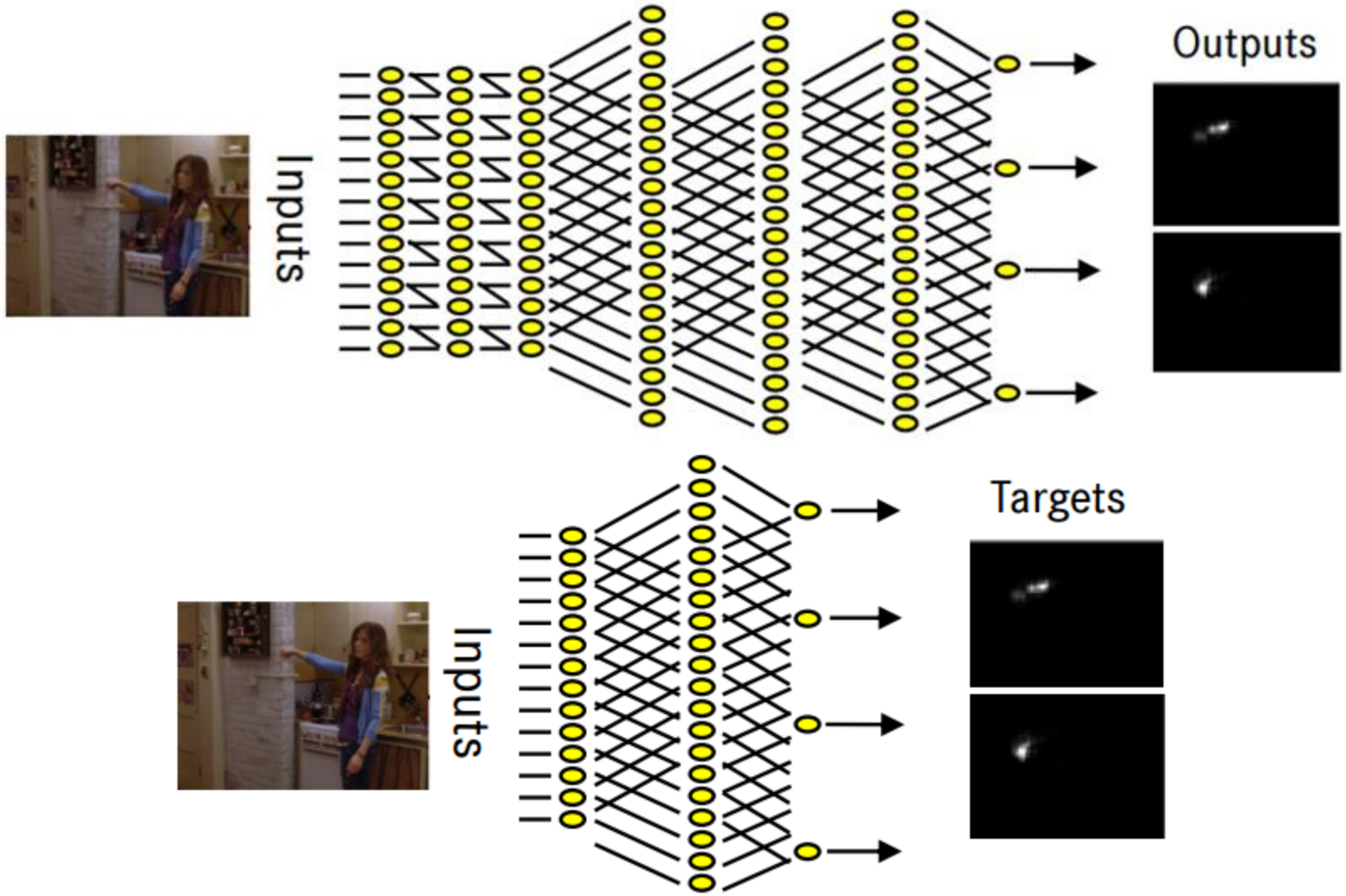
- Train a shallower network (student) with outputs of the larger network (teacher) as target.
- Deep Neural Networks (DNNs) excel over Shallow Neural Networks (SNNs). Why?
  - DNNs have more parameters.
  - DNNs can learn more complex functions
  - Convolution gives a plus
- Methods:
  - Do deep nets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

Student – Teacher Networks: Training - 3

- If network uses MSE loss (regression), simply use output of the teacher network as target for student network

- If network uses cross entropy loss (classification) with SoftMax as the last layer output:
  - Use MSE as cost for student network with the output of the teacher network before the SoftMax as target

Student – Teacher Net: Result and Discussion

Results:

- Softened outputs reveals the dark knowledge of the network

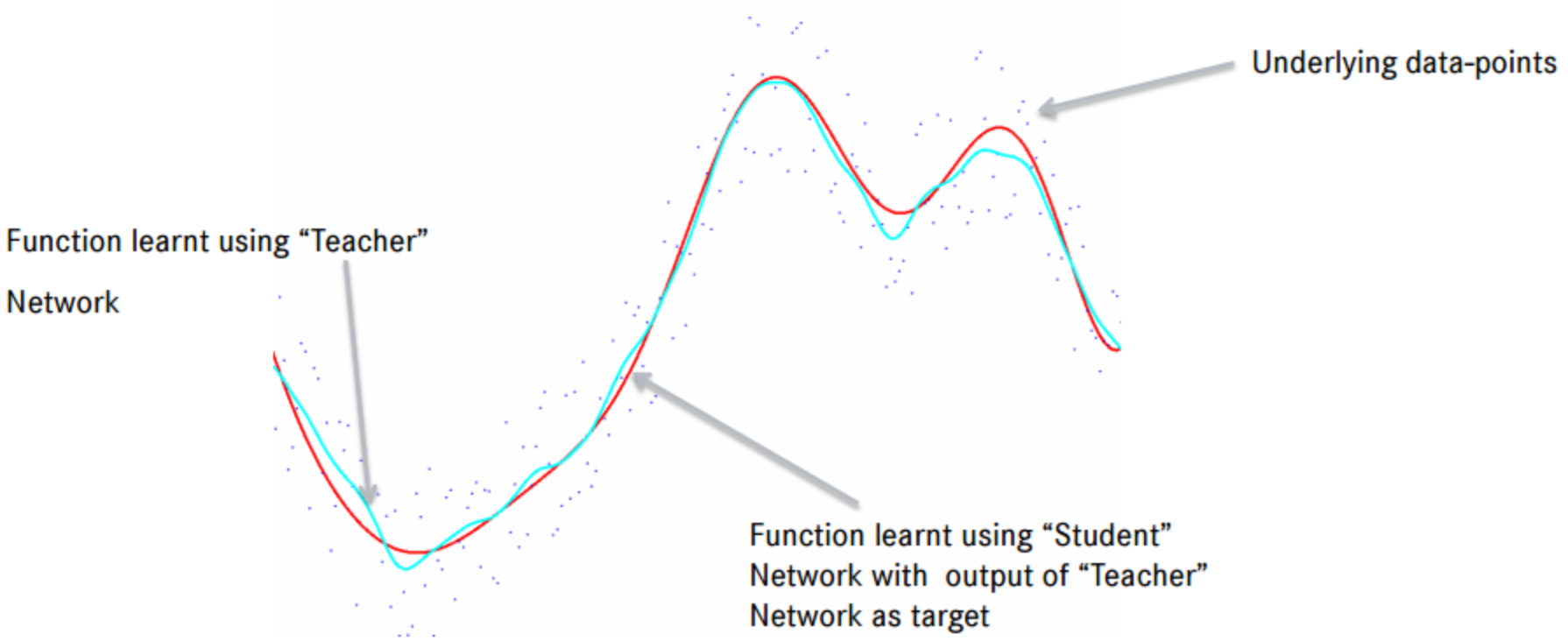
Cow	Dog	Trump	Bush	
0	1	0	0	Original Hard Targets
0.05	0.3	0.2	0.005	Softened Output

Discussion:

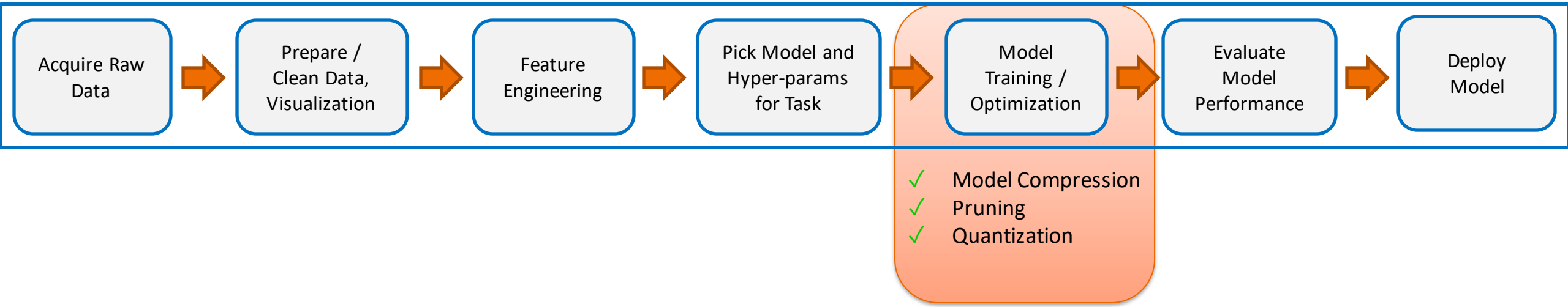
- Why MIMIC models can be more accurate than training on original labelse
  - If labels have errors, teacher may eliminate them making learning easier.
  - Teacher might resolve complex regions

Student – Teacher Net: Discussion

- Why MIMIC models can be more accurate than training on original labels



Summary


Acquire Raw Data
Prepare / Clean Data, Visualization
Feature Engineering
Pick Model and Hyper-params for Task
Model Training / Optimization
Evaluate Model Performance
Deploy Model
✓ Model Compression
✓ Pruning
✓ Quantization

Thanks!!

Questions?