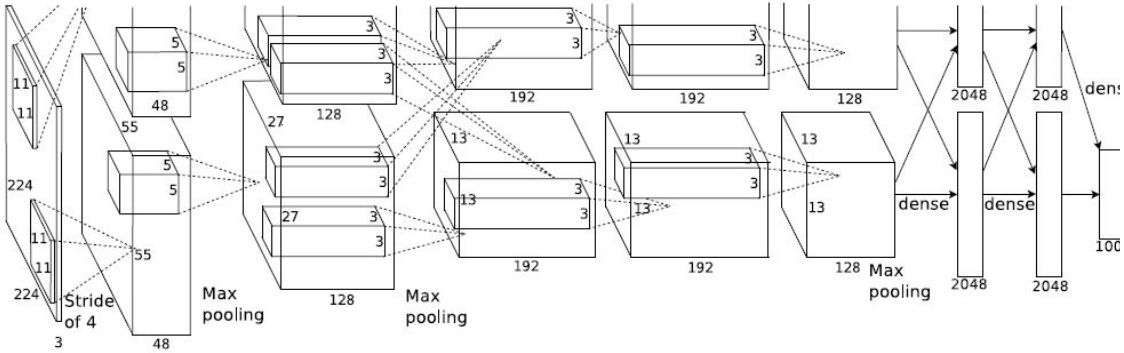
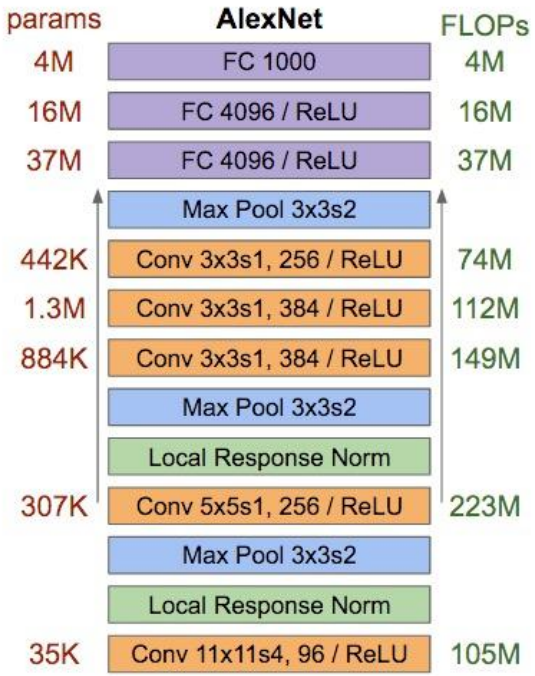
Model Compression

Girish Varma
IIIT Hyderabad
http://bit.ly/2tpy1wu

Big Huge Neural Network!

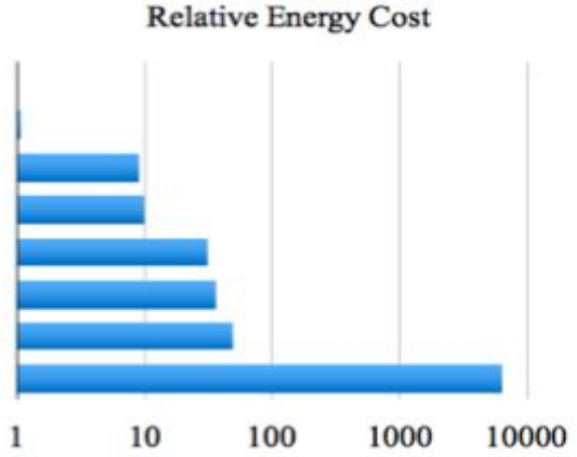


AlexNet - 60 Million Parameters = 240 MB



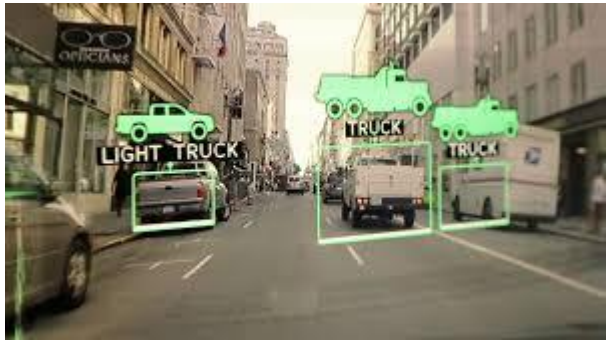
& the Humble Mobile Phone

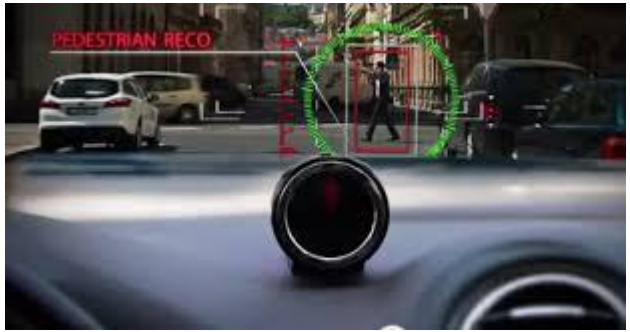
Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



But wait! What about battery life?

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 Mobile Devices

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

Model Compression

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

PRUNING

Compressing Matrices by making them Sparse

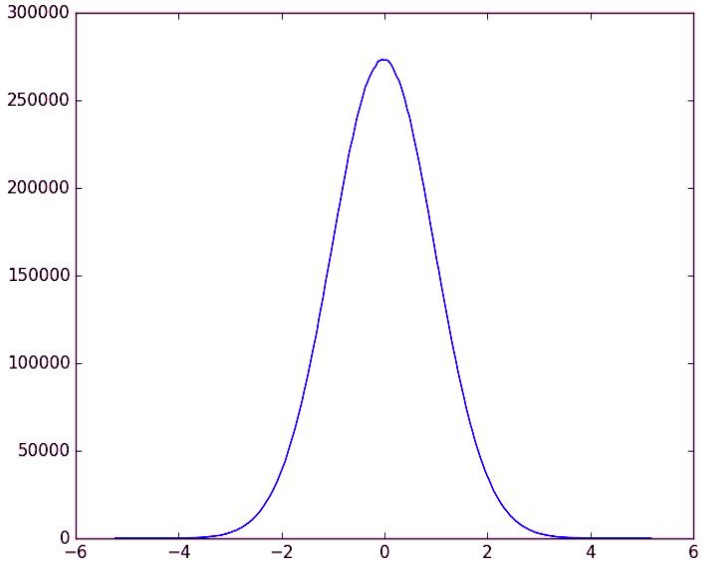
WHY PRUNING ?

Deep Neural Networks have redundant parameters.

Such parameters have a negligible value and can be ignored.

Removing them does not affect performance.

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

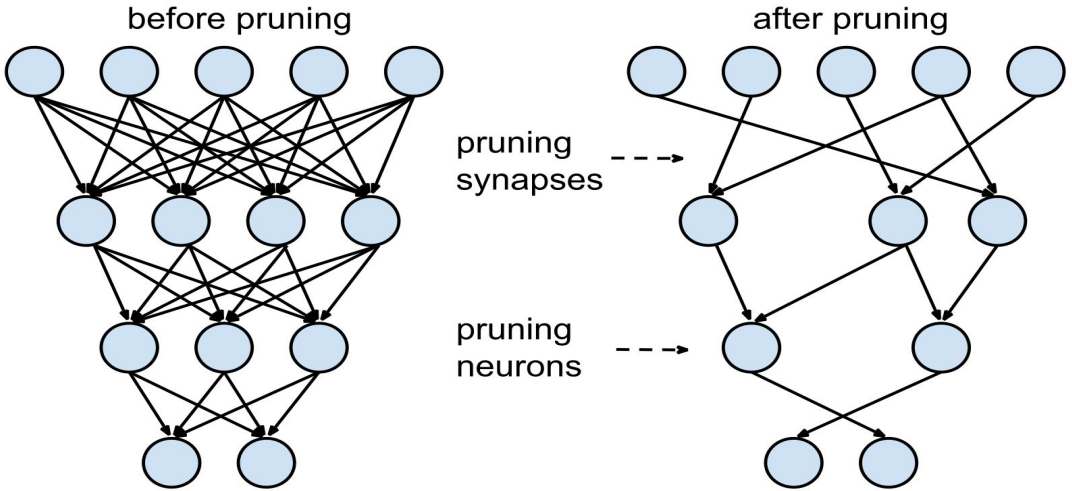
TYPES OF PRUNING

- 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)

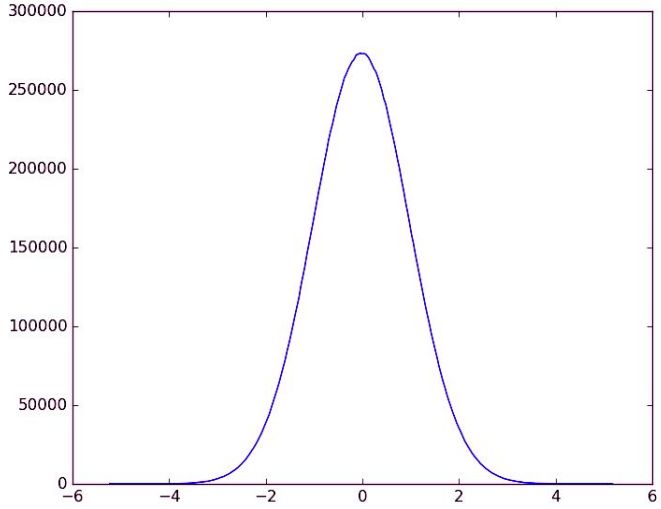


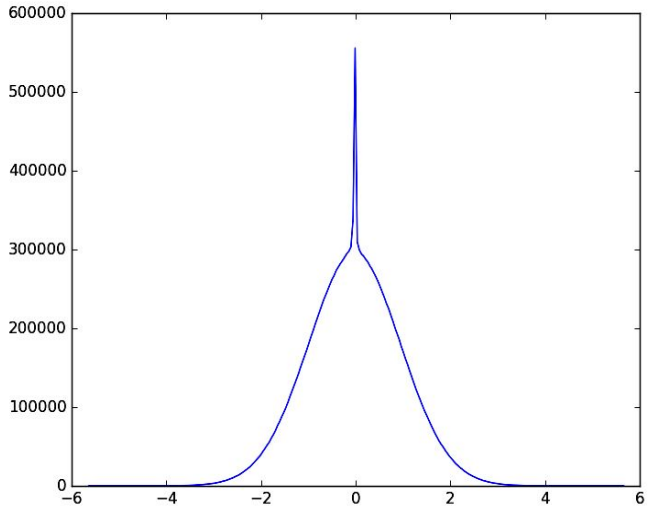
- The matrices can be made sparse. A naive method is to drop those weights which are 0 after training.
- Drop the weights below some threshold.
- Can be stored in optimized way if matrix becomes sparse.
- Sparse Matrix Multiplications are faster.

Ensuring Sparsity

Addition of L1 regulariser to ensure sparsity.

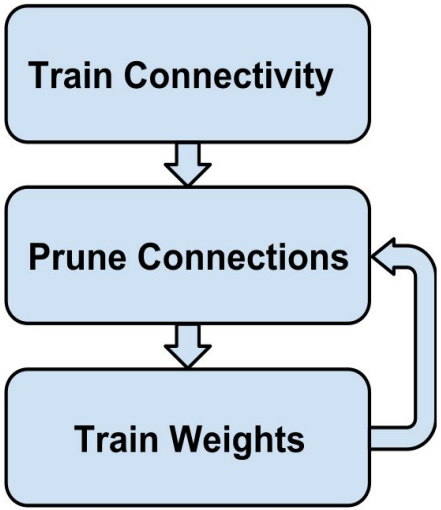
L1 better than L2 because of the slope near 0.

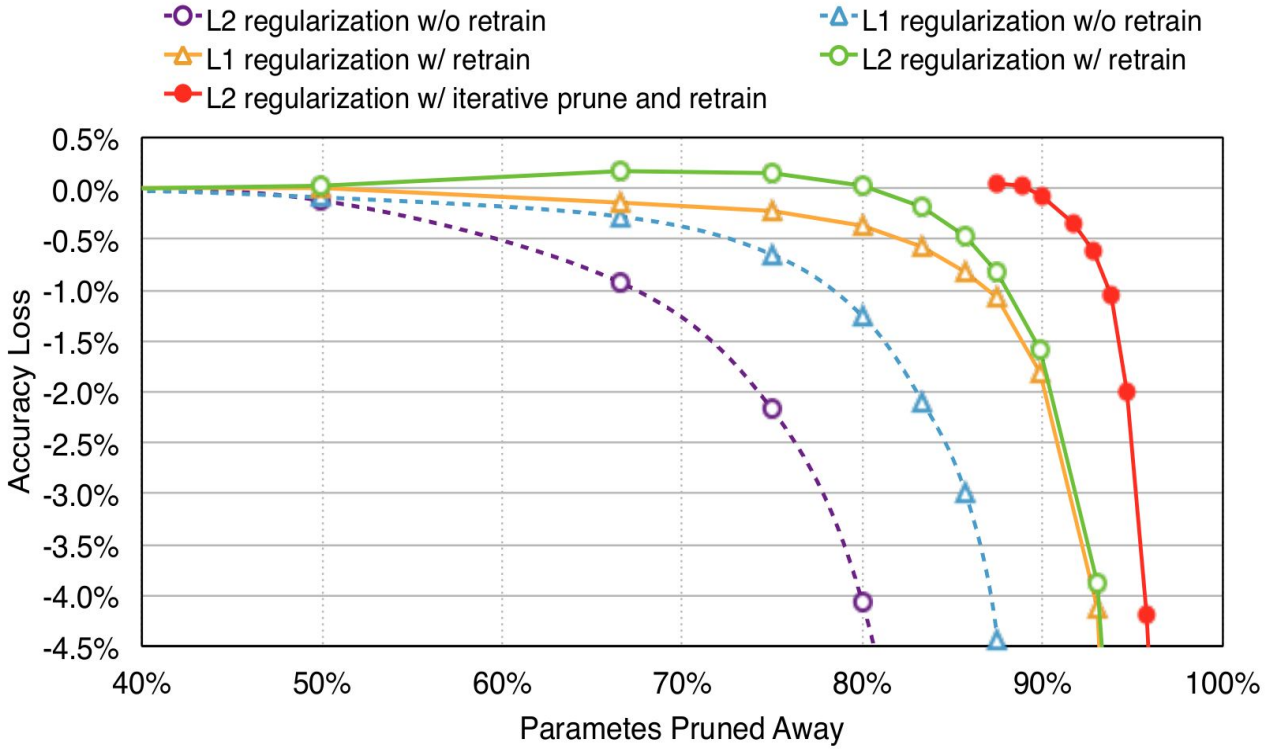




Sparsify at Training Time

Iterative pruning and retraining





Learning both Weights and Connections for Efficient Neural Networks

https://arxiv.org/pdf/1506.02626

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

Table 4: For AlexNet, pruning reduces the number of weights by $9\times$ and computation by $3\times$.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	**11%**	**30%**

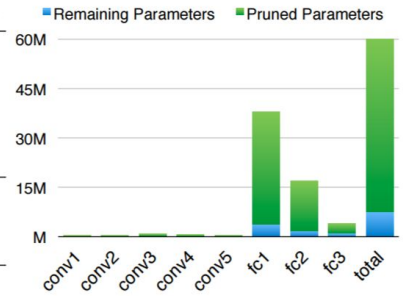


Table 5: For VGG-16, pruning reduces the number of weights by $12\times$ and computation by $5\times$.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1_1	2K	0.2B	53%	58%	58%
conv1_2	37K	3.7B	89%	22%	12%
conv2_1	74K	1.8B	80%	34%	30%
conv2_2	148K	3.7B	81%	36%	29%
conv3_1	295K	1.8B	68%	53%	43%
conv3_2	590K	3.7B	70%	24%	16%
conv3_3	590K	3.7B	64%	42%	29%
conv4_1	1M	1.8B	51%	32%	21%
conv4_2	2M	3.7B	45%	27%	14%
conv4_3	2M	3.7B	34%	34%	15%
conv5_1	2M	925M	32%	35%	12%
conv5_2	2M	925M	29%	29%	9%
conv5_3	2M	925M	19%	36%	11%
fc6	103M	206M	38%	4%	1%
fc7	17M	34M	42%	4%	2%
fc8	4M	8M	100%	23%	9%
total	138M	30.9B	64%	**7.5%**	**21%**

Sensitivity of layers to pruning

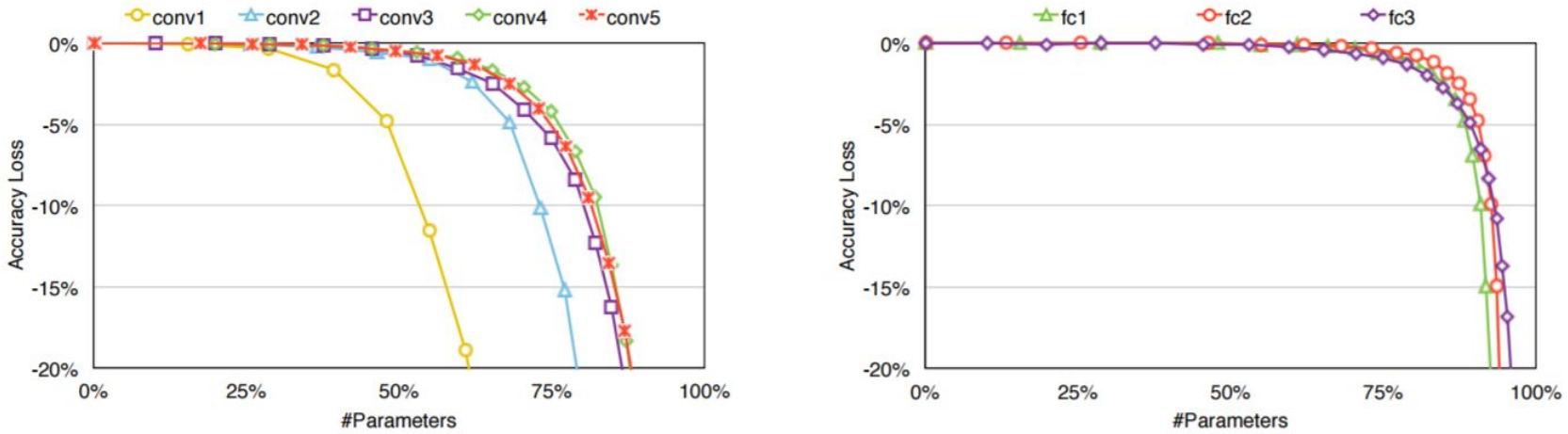
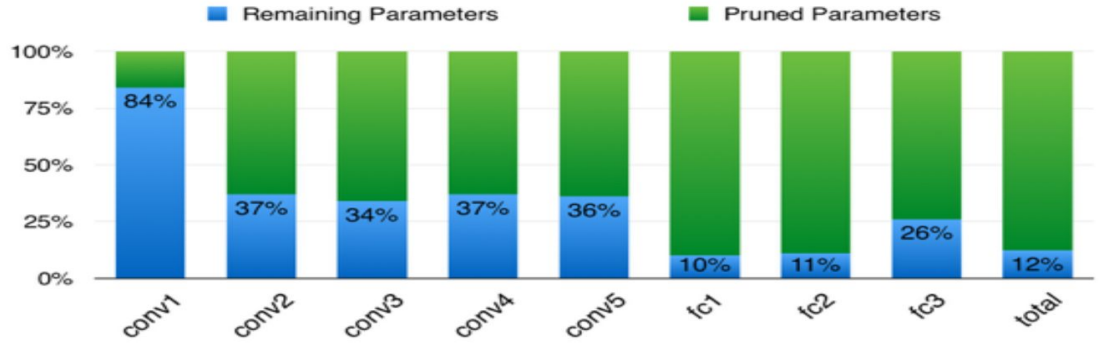
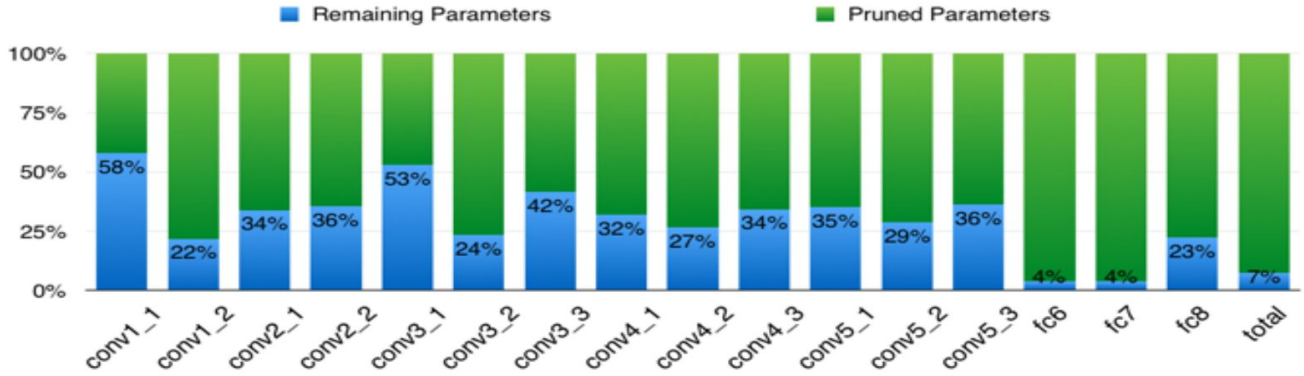


Figure 6: Pruning sensitivity for CONV layer (left) and FC layer (right) of AlexNet.

Remaining parameters in Different Layers



ALEXNET



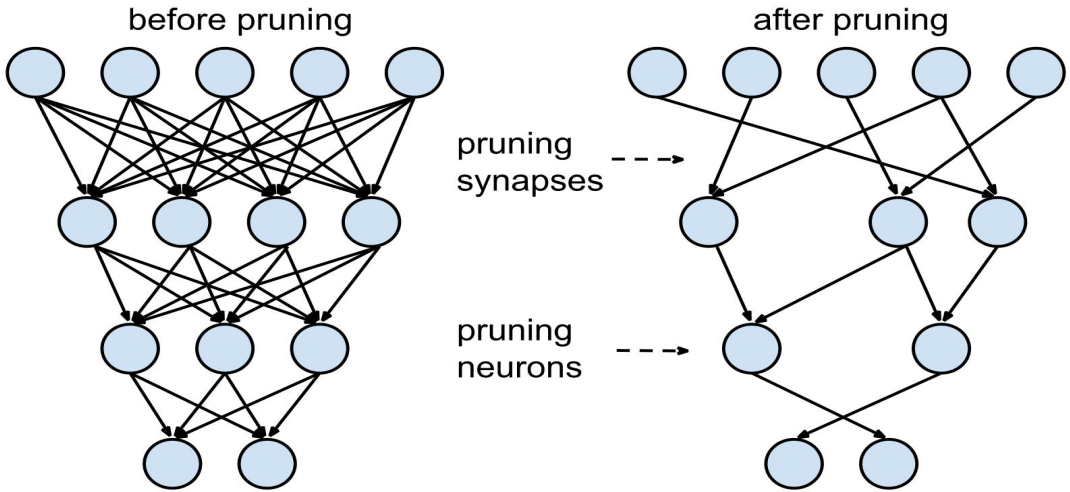
VGG16

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

Comments on Weight Pruning

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

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

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

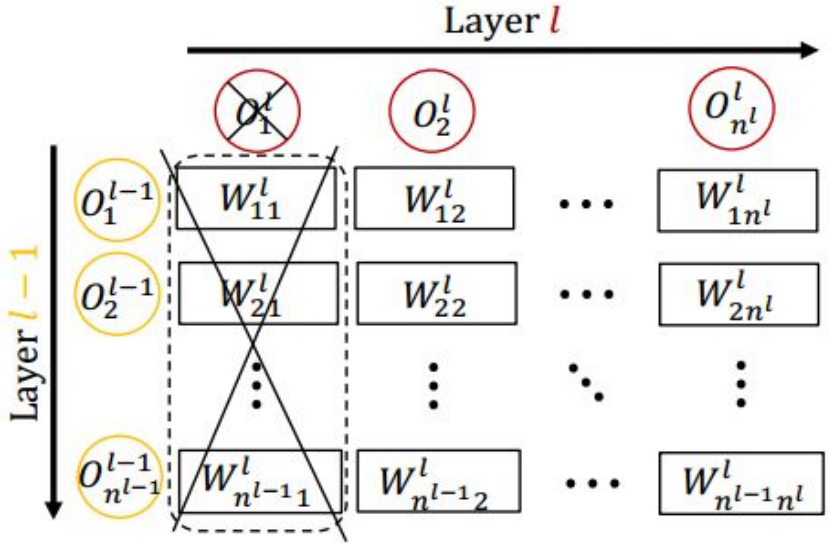
DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Dropping principles

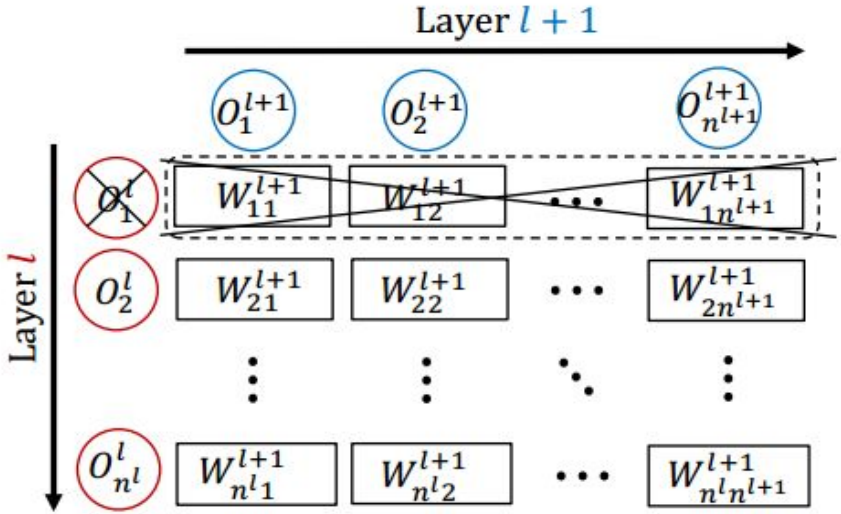
- All input connections to a neuron is forced to be 0 or as close to 0 as possible. (force li_regulariser to be small)
- All output connections of a neuron is forced to be 0 or as close to zero as possible. (force lo_regulariser to be small)
- Add regularisers to the loss function and train.
- Remove 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	$\mathbf{W}^{\mathrm{FC1}}\%$	$\mathbf{W}^{\mathrm{FC2}}\%$	$\mathbf{W}^{\mathrm{total}}\%$	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	$\mathbf{O}^{\mathrm{FC1}}\%$	$\mathbf{O}^{\mathrm{FC2}}\%$	$\mathbf{O}^{\mathrm{output}}\%$	$\mathbf{O}^{\mathrm{total}}\%$	**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

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

Neuron and Layer Pruning

- Can we learn hyperparameters by Backpropagation?
  - Hidden Layer / Filter size
  - Number of layers
- We would actually be learning the architecture
- Modifying the activation function
- ‘w’ and ‘d’ are binary variables in the equation below.

$$tsReLU(x) = \begin{cases} wx, & x \geq 0 \\ wdx, & otherwise \end{cases}$$

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

Loss Function

$$\theta, \mathbf{w}, \mathbf{d} = \underset{\theta, w_{ij}, d_i : \forall i,j}{\arg\min} \; \ell(\hat{y}(\theta, \mathbf{w}, \mathbf{d}), y) + \lambda_1 \sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ij}(1 - w_{ij}) + \lambda_2 \sum_{i=1}^{m} d_i(1 - d_i)$$

$$w'_{ij} = \begin{cases} 1, & w_{ij} \geq 0.5 \\ 0, & otherwise \end{cases}$$

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

Results

Method	Params	Accuracy (%)	Compression (%)
Reference Model (CaffeNet)	60.9M	57.41	0
Neuron Pruning ([21])	39.6M	55.60	35
SVD-quarter-F ([25])	25.6M	56.19	58
Adaptive FastFood 16 ([25])	18.7M	57.10	69
AL-conv-fc	19.6M	55.90	68
AL-fc	19.8M	54.30	68
AL-conv	47.8M	55.87	22

Table 4: Compression performance on CaffeNet.

Method	Layers Learnt	Architecture
Baseline	N/A	96 256 384 384 256 4096 4096 1000
AL-fc	fc[6,7]	96 256 384 384 256 1536 1317 1000
AL-conv	conv[1,2,3,4,5]	80 127 264 274 183 4096 4096 1000
AL-conv-fc	conv[5] - fc[6,7]	96 256 384 384 237 1761 1661 1000

Table 5: Architectures learnt by our method whose performance is given in Table 4.

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

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

Binary Quantization while Training

- Add regularizer and round at the end of training

$$\sum_i W^2{}_i(1 - W^2{}_i)$$

Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or −1 Matthieu Courbariaux, Itay Hubara , Daniel Soudry , Ran El-Yaniv, Yoshua Bengio

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

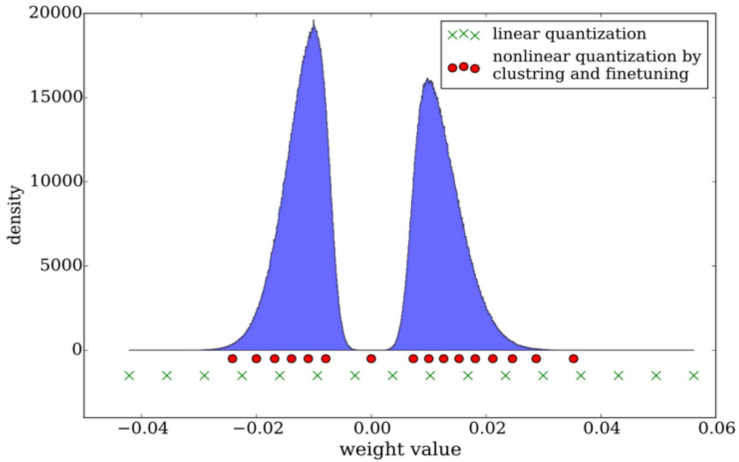
Runtime : Much faster matrix multiplication for 8 bit matrices.

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

$$\min \sum_{i}^{mn} \sum_{j}^{k} \|w_i - c_j\|_2^2,$$



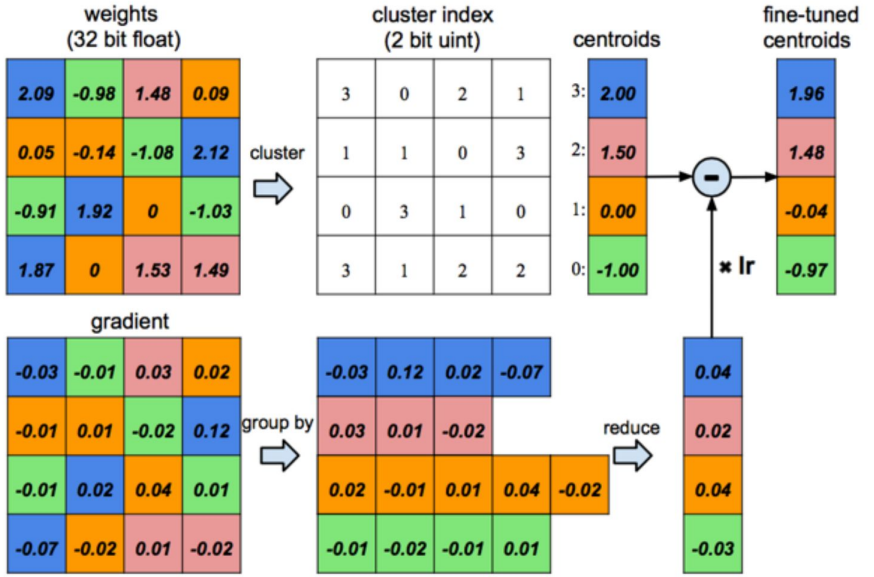
- perform k-means clustering on weights.
- Need to store mapping from integers to cluster centers. We only need log (k) bits to code the clusters which results in a compression factor rate of 32/ log (k). In this case the compression rate is 4.

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

Weight Sharing while Training

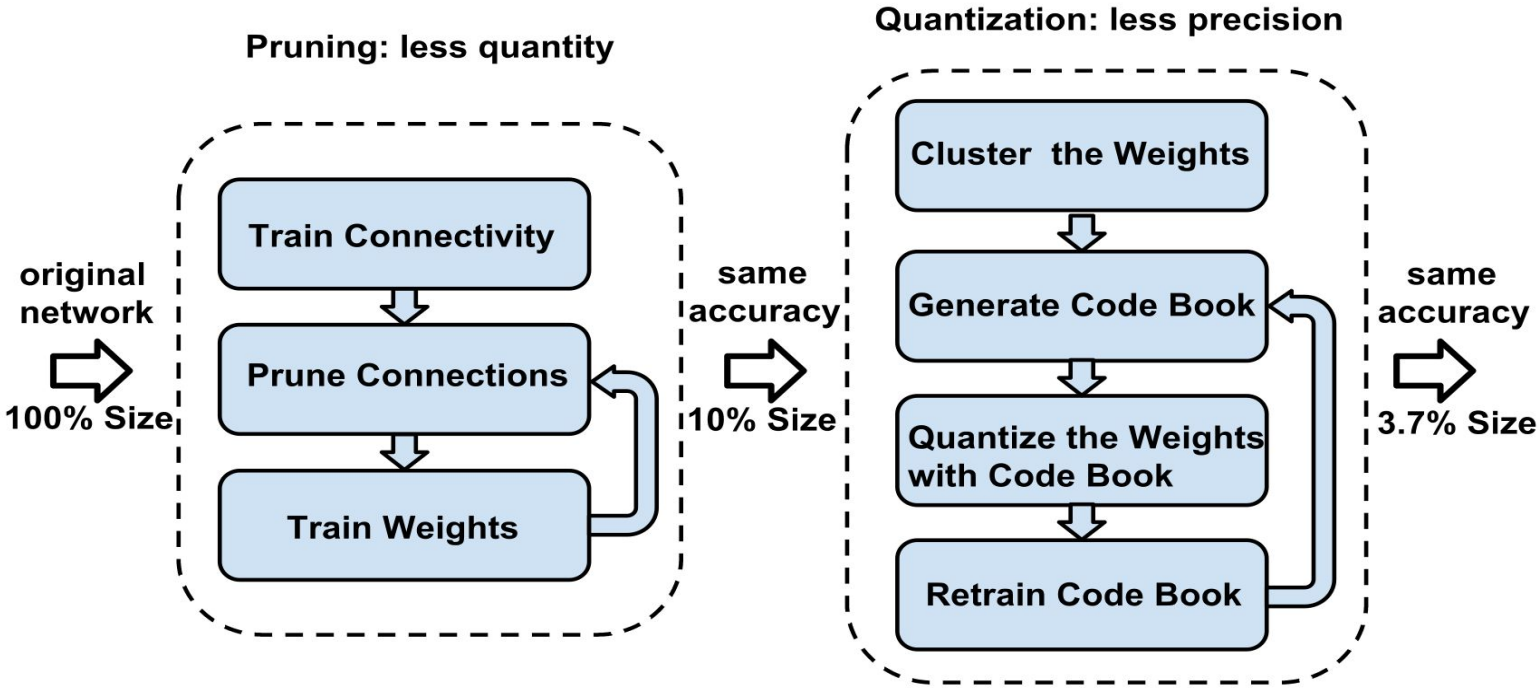
- Iterate
  - Train
  - Cluster weights
  - Make them same

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



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

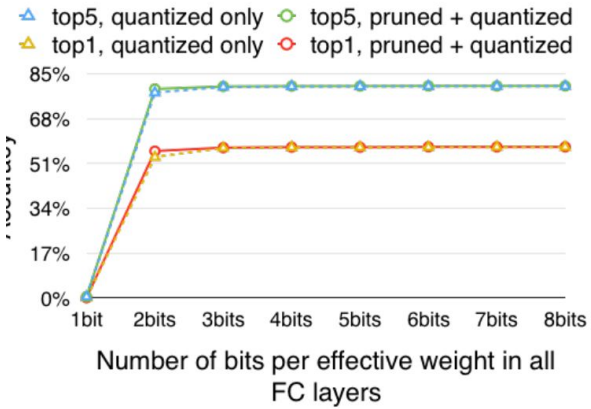
Deep Compression by Song Han

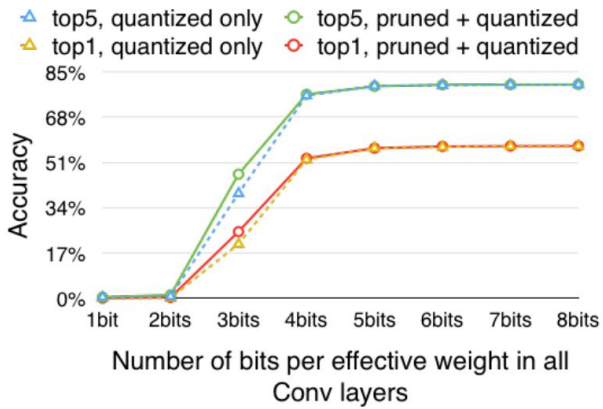


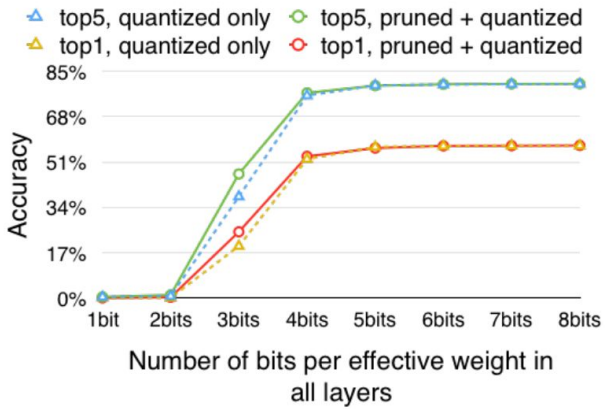
DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

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.

$$W = [W^1, W^2, \ldots, W^s], \qquad \min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2,$$

$$\hat{W} = [\hat{W}^1, \hat{W}^2, \ldots, \hat{W}^s], \quad \text{where}$$

$$\hat{\boldsymbol{w}}_j^i = \boldsymbol{c}_j^i, \quad \text{where} \quad \min_j \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2.$$

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

Residual Quantization

First quantize the vectors into k-centers.

$$\min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z - \boldsymbol{c}_j^1\|_2^2,$$

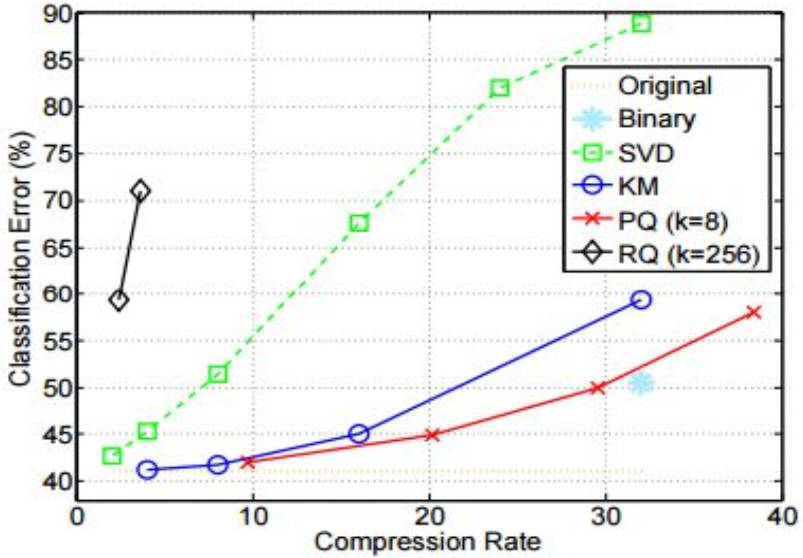
Next step is to find out the residuals for each data point(w-c) and perform k-means on the residuals

Then the resultant weight vectors are calculated as follows.

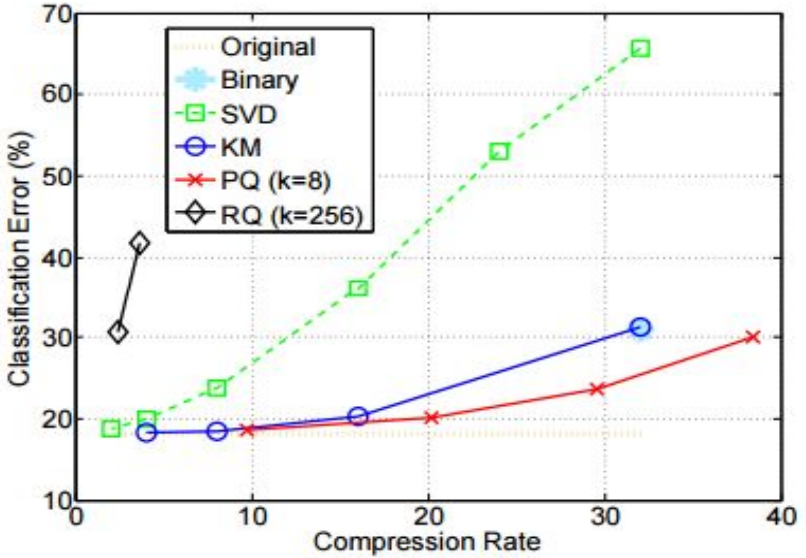
$$\hat{\boldsymbol{w}}_z = \boldsymbol{c}_j^1 + \boldsymbol{c}_j^2 + \ldots, \boldsymbol{c}_j^t,$$

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

Comparison of Quantization methods on Imagenet



(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

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

Binary weight networks

Estimating binary weights:

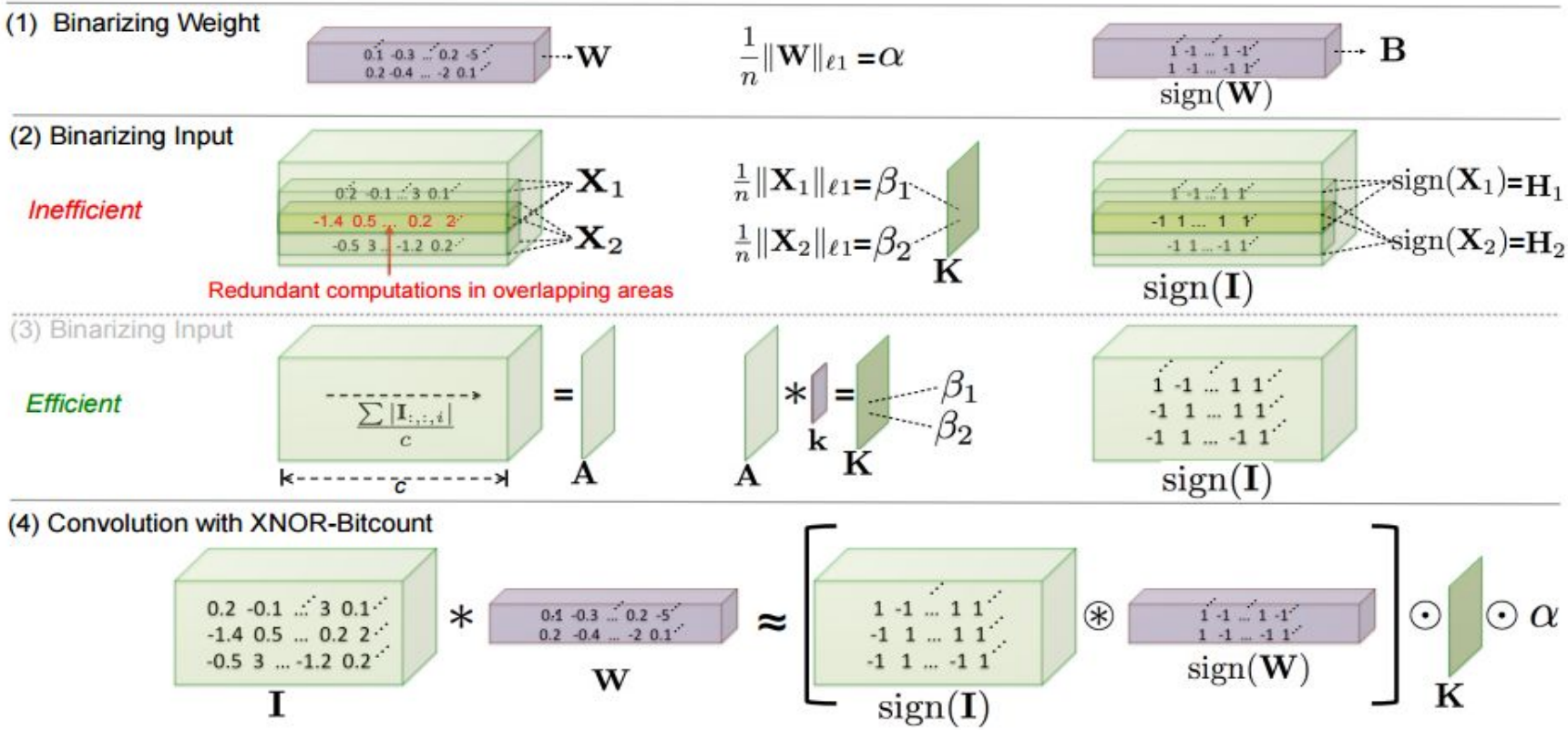
Objective function :

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$
$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

Solution : $\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$ $\alpha^* = \frac{\mathbf{W}^\intercal \operatorname{sign}(\mathbf{W})}{n} = \frac{\sum |\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1}$

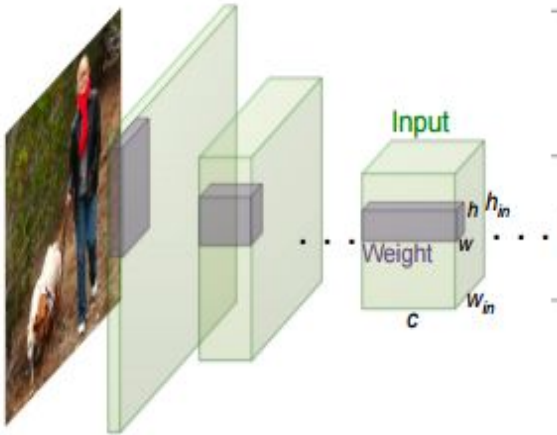
XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Approximating a convolution using binary 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

FIXED POINT REPRESENTATION

FLOATING POINT VS FIXED POINT REPRESENTATION

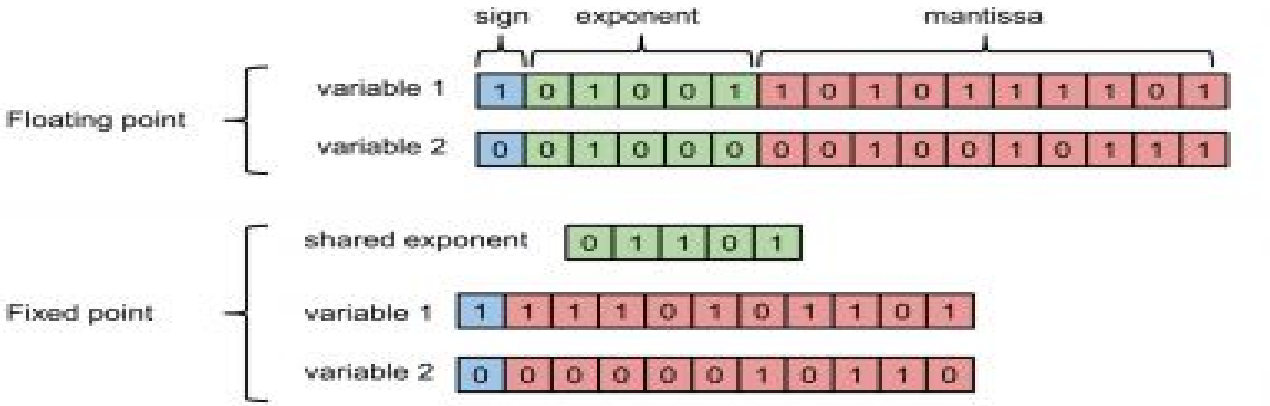


Figure 1: Comparison of the floating point and fixed point formats.

Fixed point

- ➔ Fixed point formats consist in a signed mantissa and a global scaling factor shared between all fixed point variables. It is usually ‘fixed’.
- ➔ Reducing the scaling factor reduces the range and augments the precision of the format.
- ➔ It relies on integer operations. It is hardware-wise cheaper than its floating point counterpart, as the exponent is shared and fixed.

Disadvantages of using fixed point

- When training deep neural networks :
  - Activations , gradients and parameters have very different ranges.
  - The ranges of the gradients slowly diminish during training.
  - Fixed point arithmetic is not optimised on regular hardware and specialised hardware such as FPGAs are required.

- As a result the fixed point format with its unique shared fixed exponent is ill-suited to deep learning.
- The dynamic fixed point format is a variant of the fixed point format in which there are several scaling factors instead of a single global one.

Summary

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

Compressed Architectures

Design small architectures

Compress scheme on **pre-trained model**

Vs

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

Challenges of Deep Network

Increasing the depth of the networks has two major issues:

1. More Parameters
   - Increase in the number of parameters makes the network prone to over-fitting, specially in low data regime.
2. More Computation
   - Linear increase in filters results in quadratic increase in compute

➢ We need efficient architectures which retains the capacity of the deep networks without its limitatons.

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”

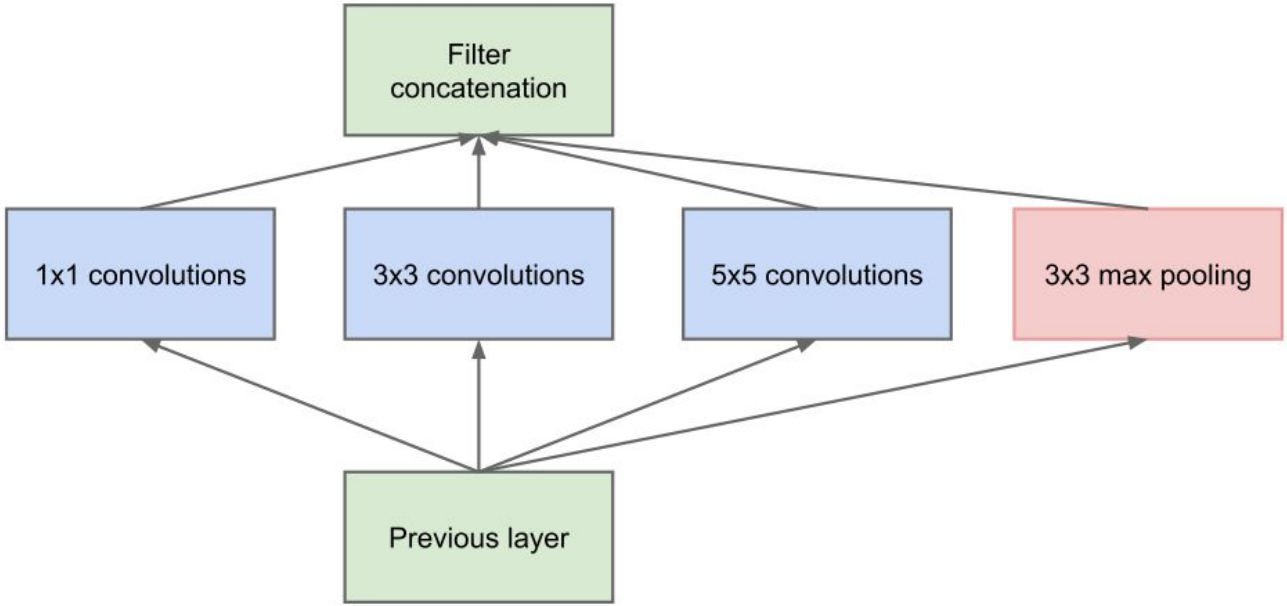
Szegedy, Christian, et al. "Going deeper with convolutions." *CVPR*, 2015.

GoogLe Net: Inception Module

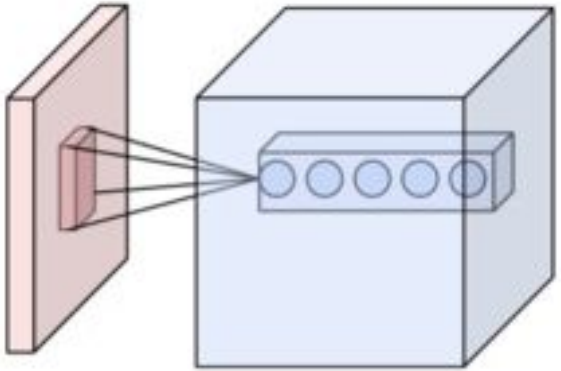
- Look at cross-channel correlations (via 1x1 convolutions) and spatial correlations (via 3x3 and 5x5 Convolutions).
- Cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly.
- To better capture spatial correlation between the pixels, it proposes to use various kernel size convolutions at each layer.
- Inspired from Network in Network, the GoogLe Net can be thought to be composed of many smaller inception networks.

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

GoogLe Net: Inception Module - Naive



Digression: Convolution Arithmetic

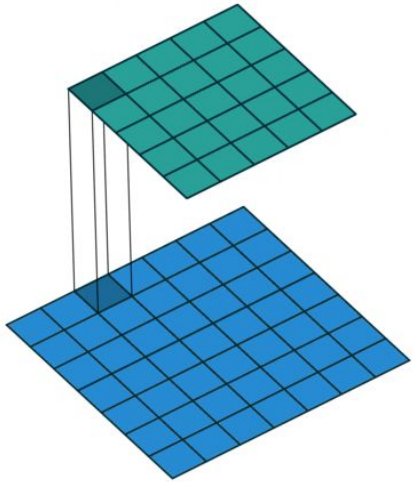


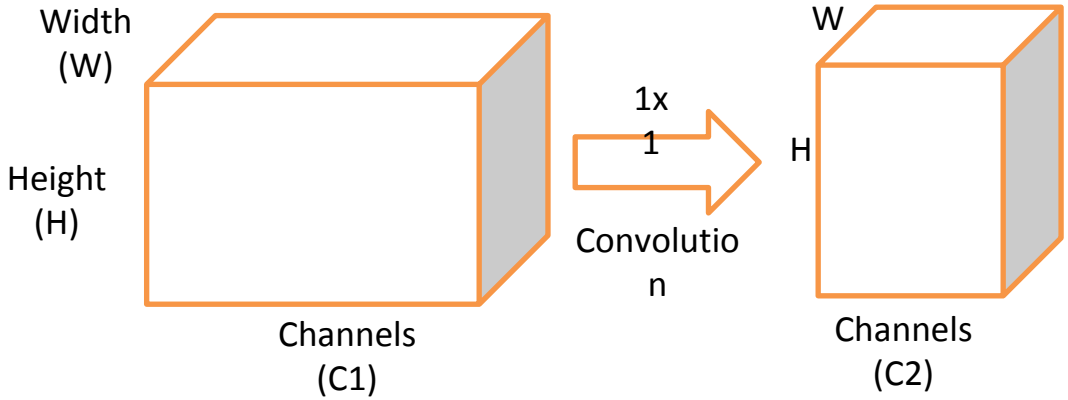
- Computes the spatial and cross channel correlation.

- The number of parameters in a layer is:

$$C_{inp} \times W \times H \times C_{out}$$

Digression: 1 × 1 Convolution

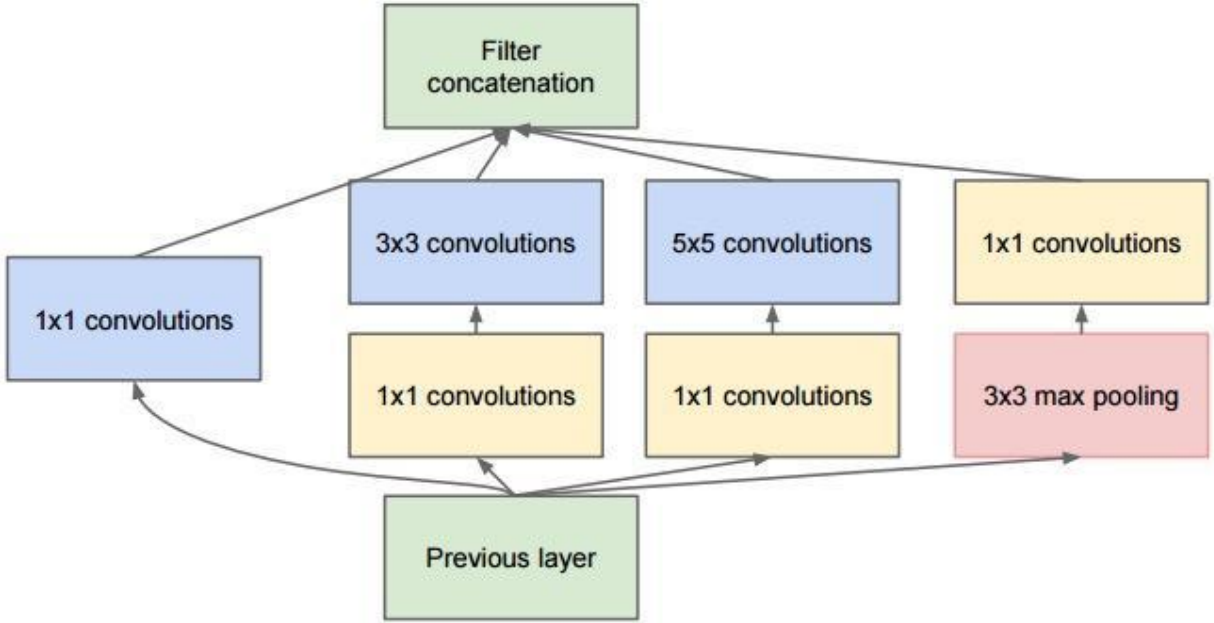




- 1 × 1 Convolution computes cross channel correlation
- Used to reduce or inflate channel dimensions

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

GoogLe Net: Inception Module



GoogLe Net: Inception Module – Analysis 1

Parameters in inception module with naïve approach

- $c_{inp} \times \big((1 \times 1) + (3 \times 3) + (5 \times 5)\big) \times c_{out}$

$$= 35 \times c_{inp} \times c_{out}$$

Parameters in inception module with dimensionality reduction

- $\big(2 \times c_{inp} \times (1 \times 1) \times c_{int}\big) + \big(c_{int} \times ((3 \times 3) + (5 \times 5)) \times c_{out}\big) + c_{inp} \times (1 \times 1) \times c_{out}$

$$= (c_{inp} \times c_{out}) + (34 \times c_{int} \times c_{out}) + (2 \times c_{inp} \times c_{int})$$

GoogLe Net: Inception Module – Analysis 2

Parameters in inception module with naïve approach

- For $c_{inp} = 512$ & $c_{out} = 1024$
- Number of parameters
  $= 35 \times 512 \times 1024$
  $= 18{,}350{,}080$
  $\sim 18.35\ million$

Parameters in inception module with dimensionality reduction

- For $c_{inp} = 512, c_{int} = 64, c_{out} = 1024$
- Number of parameters
  $= (524{,}288) + (2{,}228{,}224) + (65{,}536) = 2{,}818{,}048$
  $\sim 2.82\ million$

- Compression: $18.35 \div 2.82 = 6.51$x

GoogLe Net: Global Average Pooling 1

Problems with fully connected (FC) layers:

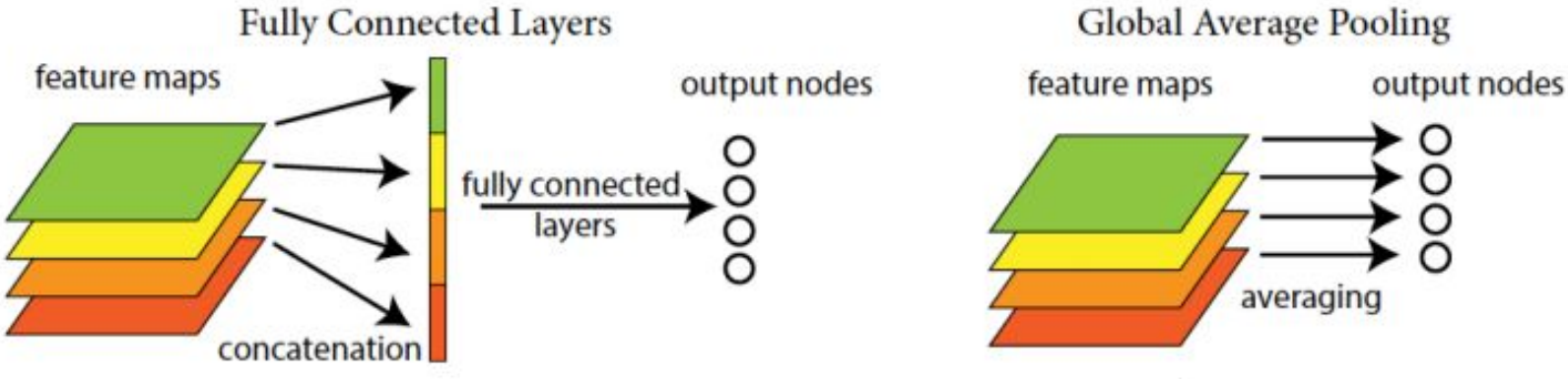
- More than 90% parameters of Alexnet and VGG are in the Fully Connected layers.
- One single particular layer in VGG contains 100 million parameters alone.
- Difficult to interpret how the category level information is passed back from the objective cost layer to the previous convolution layer.
- Prone to overfitting.
- Heavily dependent on regularization methods like dropout.

GoogLe Net: Global Average Pooling 2

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.
- Enforces correspondence between feature maps and categories.
- Stronger local modelling using the micro network.
- It is itself a structural regularizer and hence doesn’t need dropout.
- We get a uniform sized feature vector irrespective of input image size.

GoogLe Net: Global Average Pooling 3



GoogLe Net: Results and Discussion

Results:

- Achieves 93.33% top 5 accuracy on ImageNet Challenge
- 12 times lesser parameters than AlexNet.
- Computational cost “less than 2X compared to AlexNet”

Discussion:

- Key Idea 1: Reduce the dimensionality before applying expensive convolutional operation
- Key Idea 2: Replace fully connected layer with much efficient global average pooling layer

Rethinking the Inception Architecture: Inception v2

- Reduce the number of expensive convolution operations, to further reduce the number of parameters.

- Factorize the convolutions with large filter size, while keeping the receptive field same.
  - Factorize large convolution operation into smaller convolutions
  - Spatial factorization into asymmetric convolutions

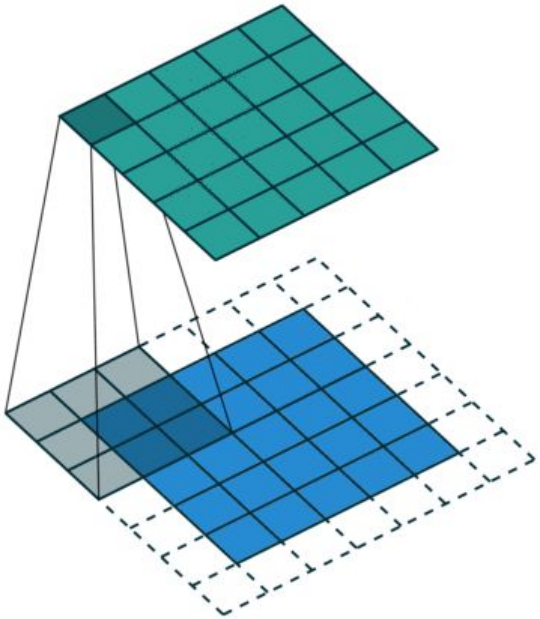
Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." *CVPR,* 2016.

Inception v2: Factorizing into Smaller Convolutions - 1

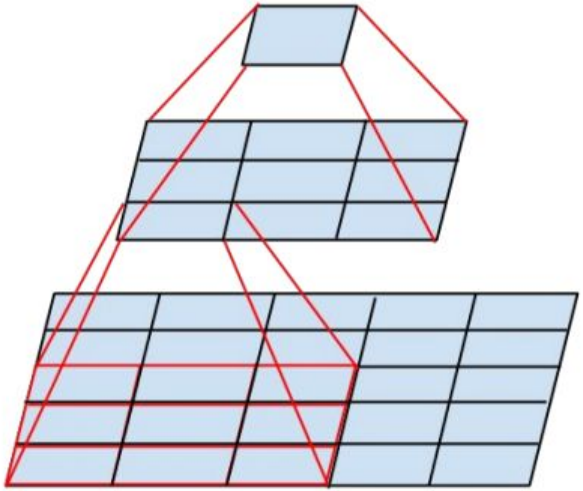
- Convolutions with large spatial filters (e.g. 5x5 or 7x7) tend to be disproportionately expensive in terms of computation.
  - A 5x5 convolution has 25/9 = 2.78x more computationally expensive than a 3x3 kernel

- Can we replace large spatial filters with multi-layered network with less parameters?

Inception v2: Factorizing into Smaller Convolutions - 2



Convolution Operation as small
fully connected sliding tile

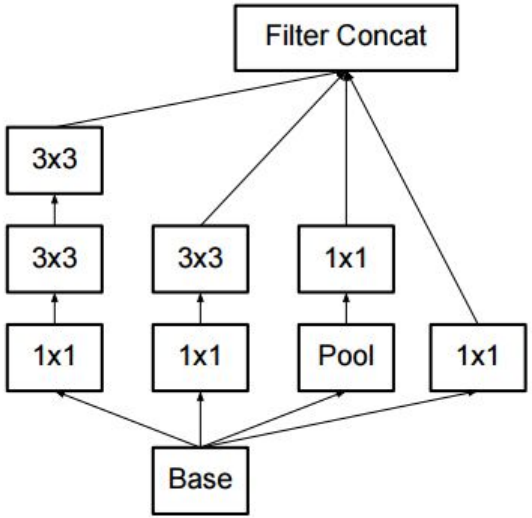


Mini Network
replacing
5x5 convolutions

Inception v2: Factorizing into Smaller Convolutions - 3

Results

- Factorization into smaller convolution results in $\frac{9+9}{25}$x reduction in computation, resulting in total gain of 28% by factorization.
- Same holds for parameter count, as each parameter is used exactly once in the computation of the activation of each unit.



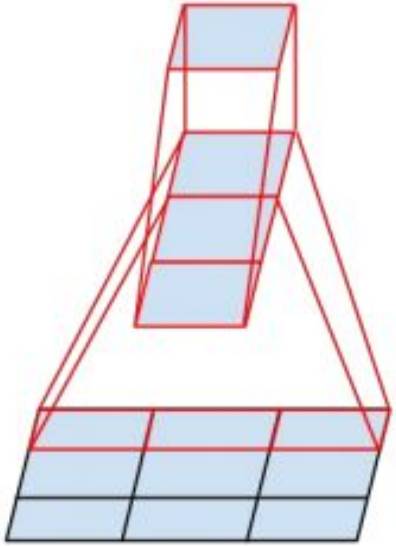
Inception module, where 5x5 convolution
is replaced by 3x3 convolutions

Inception v2: Spatial Factorizing into Asymmetric Convolutions - 1

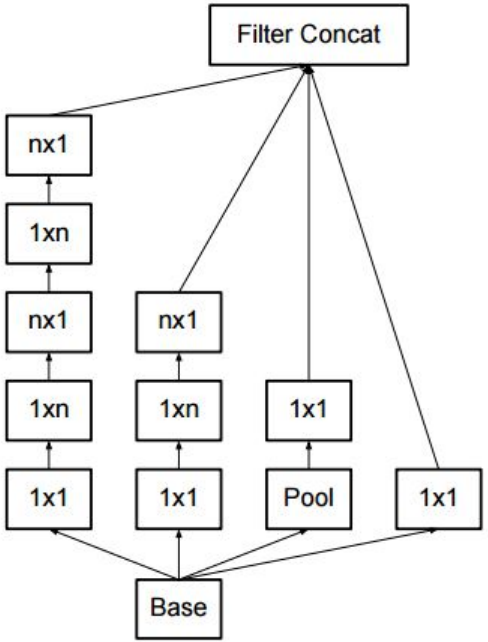
- Above result shows that is possible to convert any large convolution into a sequence of 3x3 convolutions. Can we go beyond? e.g. use 2x2 convolutions to estimate a 3x3 convolutions?

- Using asymmetric convolutions, one can do even better than 2x2, e.g. nx1.
  - For 3x3 convolution, we can factorize it into 3x1 and 1x3 asymmetric convolutions, with only 6 parameters against the original 9.
  - This is more efficient than using symmetric convolutions.

Inception v2: Spatial Factorizing into Asymmetric Convolutions - 2



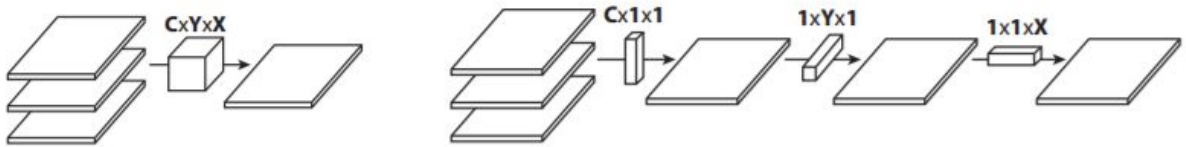
Mini Network Replacing 3x3 convolutions with 3x1 convolutions



Inception Modules after factorizing $n \times n$ convolutions

Flattened Convolutions

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



(a) 3D convolution (b) 1D convolutions over different directions

- Compression and Speedup:
  - Parameter reduction: $\mathrm{O}(XYC)$ to $\mathrm{O}(X + Y + C)$
  - Operation reduction: $O(mnCXY)$ to $O\big(mn(C + X + Y)\big)$ (where $\mathrm{W_f} \in \mathbb{R}^{m \times n}$)

Jin, Jonghoon, et al. "Flattened convolutional neural networks for feedforward acceleration." *ICLR-W,* 2015.

Inception v2: Results and Discussion

Results:

- Factorizing into smaller convolutions result in significant reductions in number of network parameters and its computation requirements.
- This architecture modification along with improved training strategy results in minor improvement (0.2%) of the network.

Discussion

- Key Idea: Factorize larger convolutions into smaller convolutions, using asymmetric convolutions.

Xception Net: Extreme Inception Net

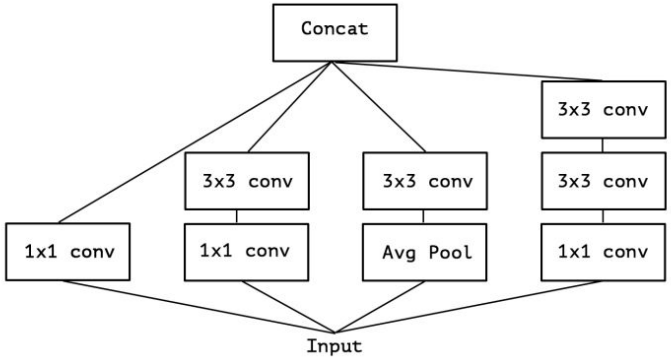
- Uses depth-wise separable convolution
  - Convolution which looks into spatial correlations across all channels independently and then uses point-wise convolutions to project to the requisite channel space leveraging inter-channel correlations

- Extends the idea of Inception module that cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly.

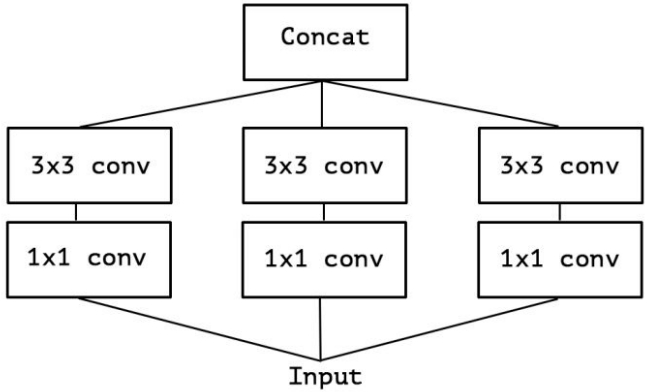
Chollet, François. "Deep Learning with Separable Convolutions." *arXiv preprint arXiv:1610.02357* (2016).

Xception Net: Depth-wise Separable Convolution

- Considering simplified version of Inception module which uses fixed size convolution (e.g. 3x3) and does not include pooling tower.

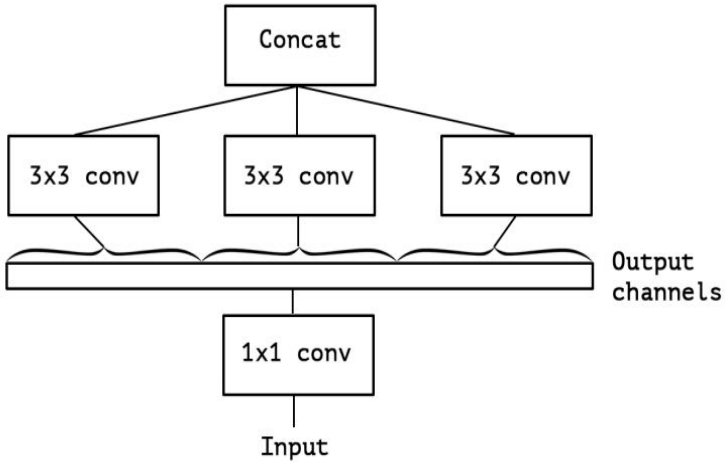


Standard Inception v3



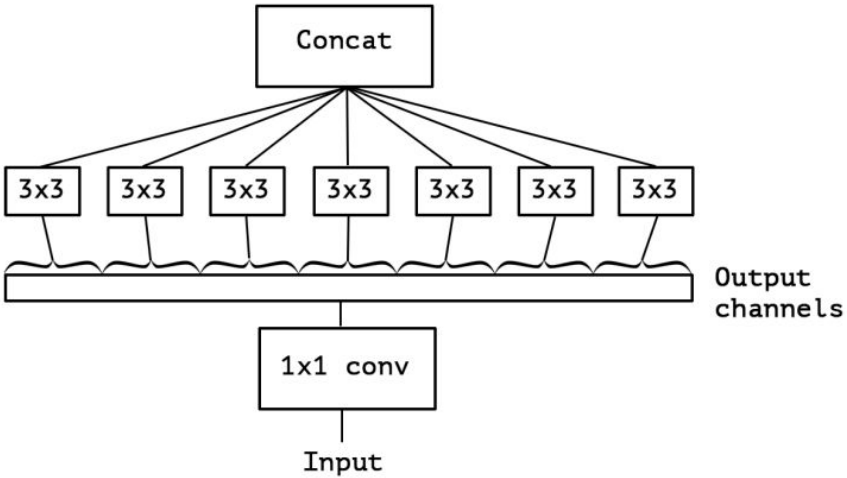
Simplified Inception v3

Xception Net: Depth-wise Separable Convolution



- This Inception module can be re-formulated as a large 1x1 convolution followed by spatial convolution that would operate on non-overlapping segments of output channels.

Xception Net: Depth-wise Separable Convolution



- The extreme version of inception, where the spatial 3x3 convolution is applied to each of the intermediate output channel.

Xception Net: Analysis

Inception Module

- Number of parameters:
  $= (c_{inp} \times c_{out}) + (27 \times c_{int} \times c_{out}) + (2 \times c_{inp} \times c_{int})$
- For $c_{inp} = 512, c_{int} = 64, c_{out} = 1024$
  $= 2{,}461{,}696$
  $\sim 2.46\ million$

Xception: Extreme Inception Module

- Number of parameters:
  $= c_{inp} \times c_{int} + (9 \times c_{int} \times c\_out)$
- For $c_{inp} = 512, c_{int} = 64, c_{out} = 1024$
  $= 512 \times 64 + 9 \times 64 \times 1024$
  $= 622{,}592$
  $\sim 0.62\ million$

- Compression: $2.46 \div 0.62 = 3.97$x

Xception Net: Results and Discussion

Results:

- Achieves ~4x compression as compared to equivalent GoogLe Net
- Slightly outperforms GoogLe Net in image classification task.
  - Achieves top-5 accuracy of 94.5% vs 94.1% accuracy by Inception v3
  - Achieves top-1 accuracy of 79.0% vs 78.2% accuracy by Inception v3

Discussion:

➢ Key Idea: Use depth wise separable convolution for inception module.

Squeeze Net

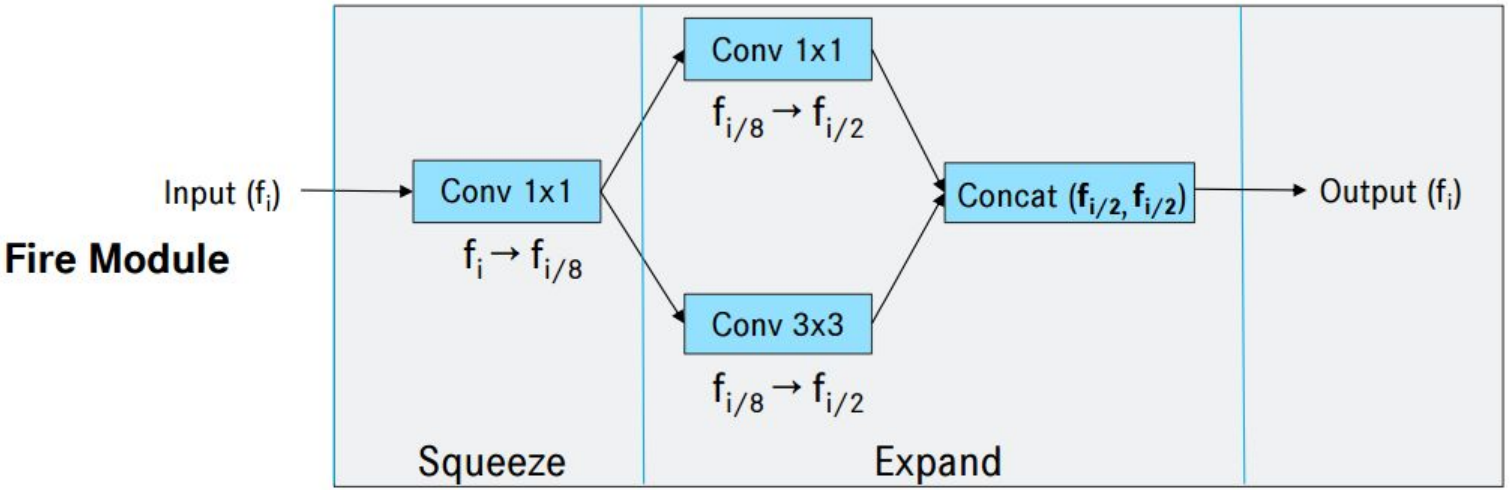
- AlexNet level accuracy with 50x fewer parameters and <0.5MB model size.

Compression Strategies:

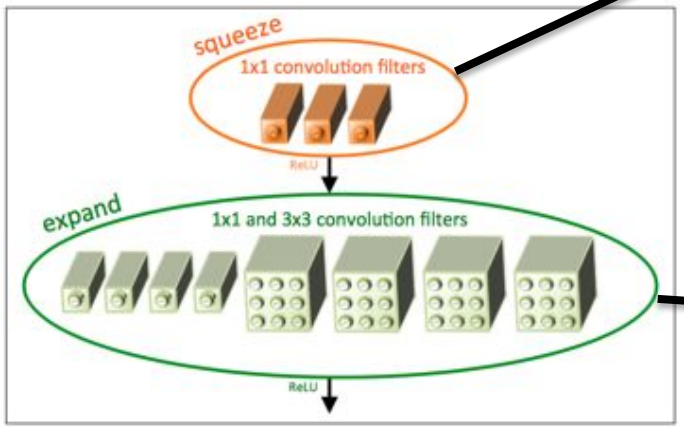
- Strategy 1. Replace 3x3 filters with 1x1 filters
- Strategy 2. Decrease the number of input channels for 3x3 filters
- Strategy 3. Downsample late in the network so that convolution layers have large activation maps

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

Squeeze Net: Fire Module - 1



Squeeze Net: Fire Module - 2

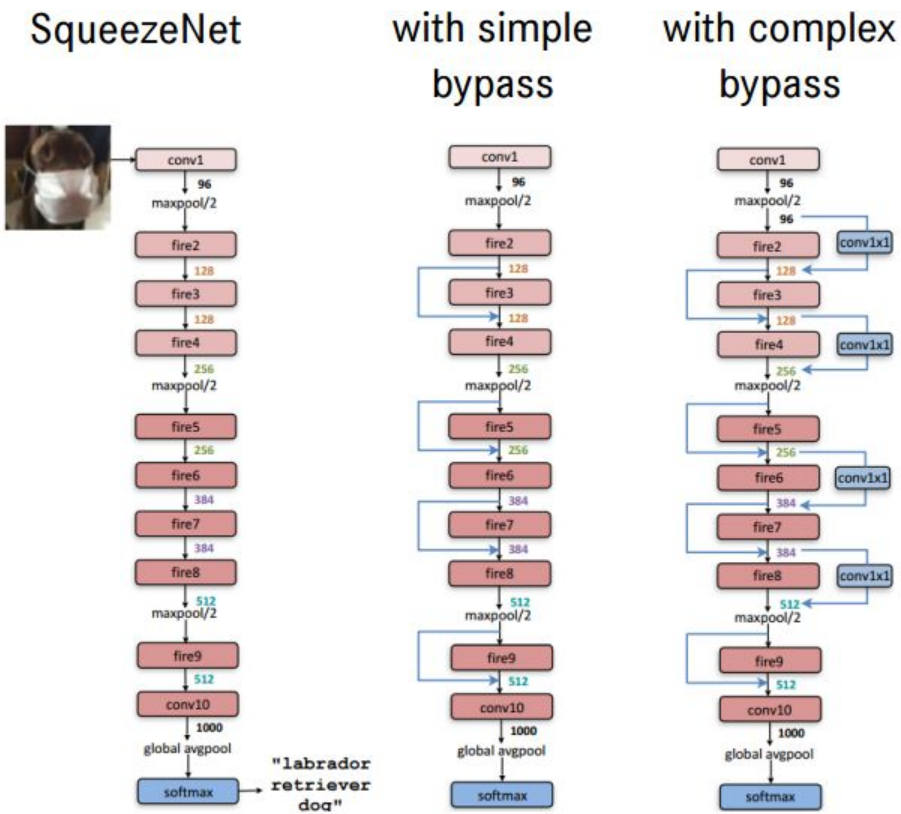


- **Strategy 2.** Decrease the number of input channels to 3x3 filters
  - Total # of parameters: (# of input channels) * (# of filters) * (# of parameters per filter)

Squeeze Layer: limits the # of input channels to 3x3 filters

- **Strategy 1.** Replace 3x3 convolutions with 1x1 convolutions

Squeeze Nets: Macro Architecture



Squeeze Nets: Results - 1

Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	**60.4%**	**82.5%**	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

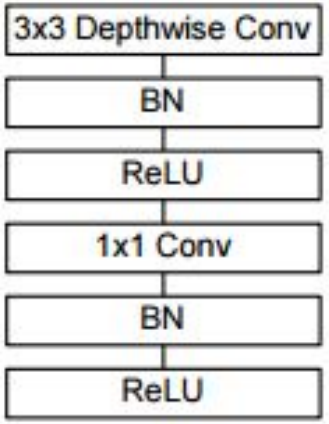
Squeeze Net: Result 2

Compression Approach	DNN Architecture	Original Model Size	Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
None (baseline)	AlexNet	240MB	240MB	1x	57.2%	80.3%
Network Pruning	AlexNet	240MB	27MB	9x	57.2%	80.3%
Deep Compression	AlexNet	240MB	6.9MB	35x	57.2%	80.3%
None	**SqueezeNet**	**4.8MB**	**4.8MB**	**50x**	57.5%	80.3%
Deep Compression	**SqueezeNet**	**4.8MB**	**0.52MB**	**461x**	57.5%	80.3%

Mobile Nets

- Mobile Nets are based on a streamlined architecture that uses depth wise convolution to build light weight neural network
- Using depth wise separable convolution only reduces the performance by 1% when compared to full convolution.
- Mobile Nets perform 4% better than AlexNet while being 45x smaller, and 9.4x less compute.
- It is also 4% better than SqueezeNet at about the same size and 22x less computation

Mobile Nets: Architecture



Micro
Architecture
of Mobile Nets

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Mobile Nets: Results and Discussion

Result

s

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Mobile Net comparison with other popular models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Smaller Mobile Nets comparison with popular model

Discussion

- Mobile Net resource per layer is given as under:

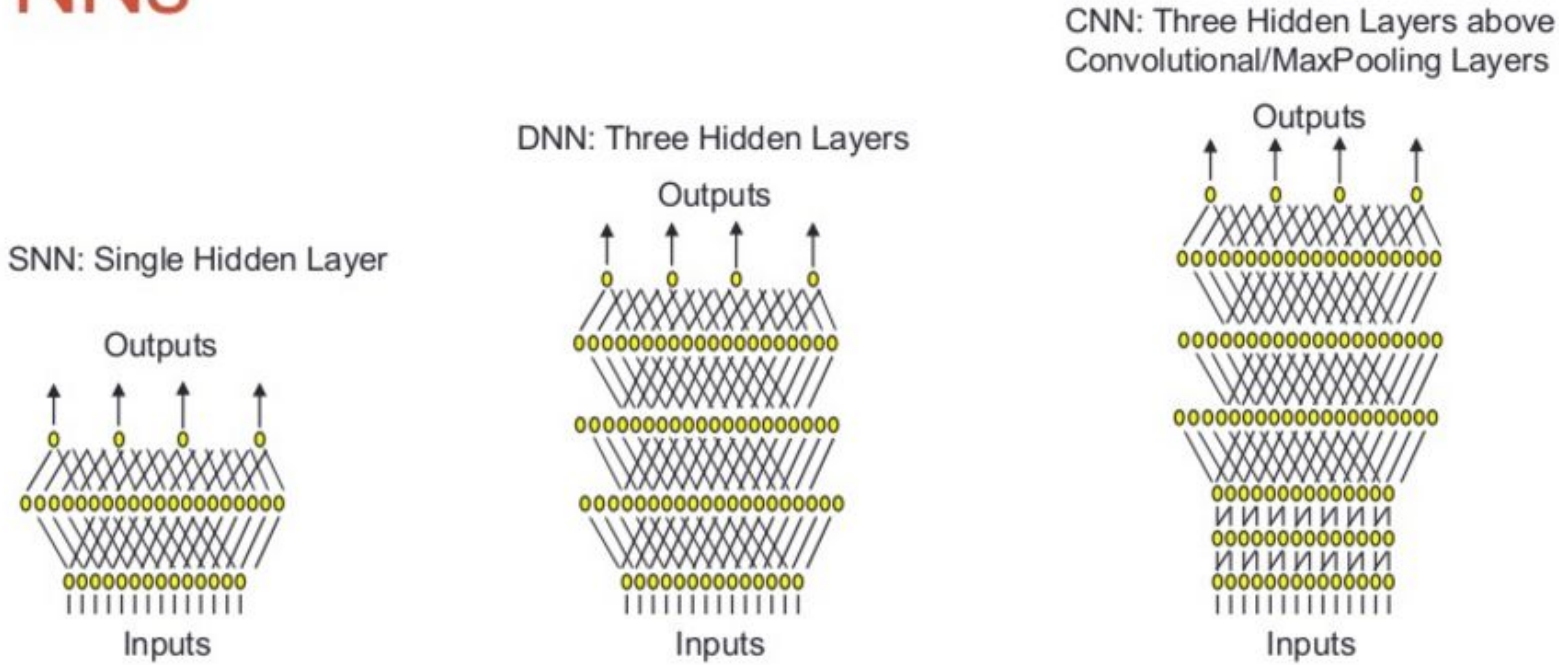
Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Summary

- ✓ Reduce the dimensions before applying spatial convolutional filter
- ✓ Use small sized convolutional filter; Reduce the large convolutional filter via asymmetric convolutions.
- ✓ Use depth separated convolution.
- ✓ Use Global Average Pooling instead of expensive Fully Connected Layer.

Student – Teacher Networks

NNs

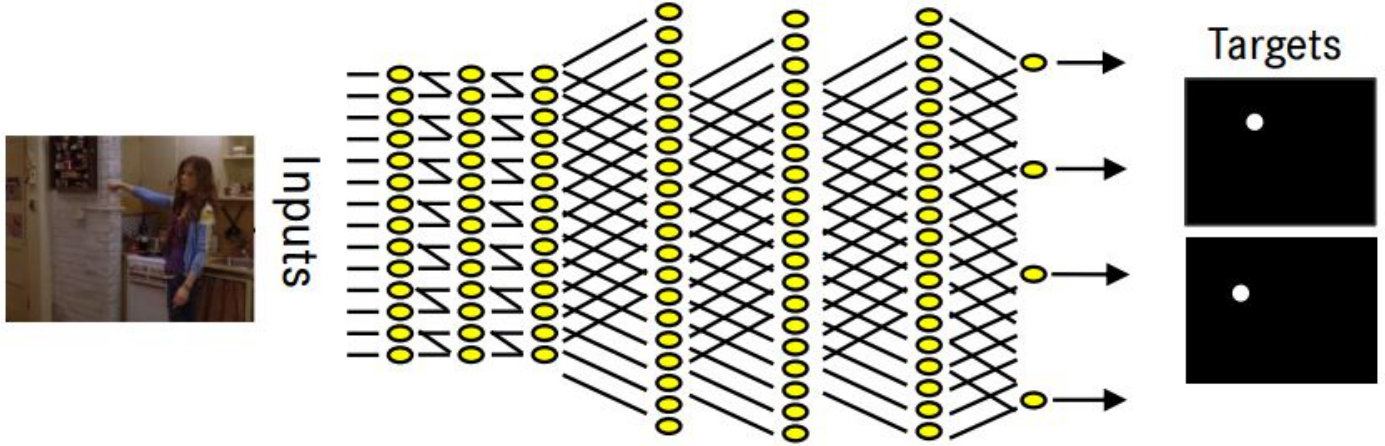


Ba, Jimmy, and Rich Caruana. "Do deep nets really need to be deep?." *NIPS,*

Student – Teacher Networks

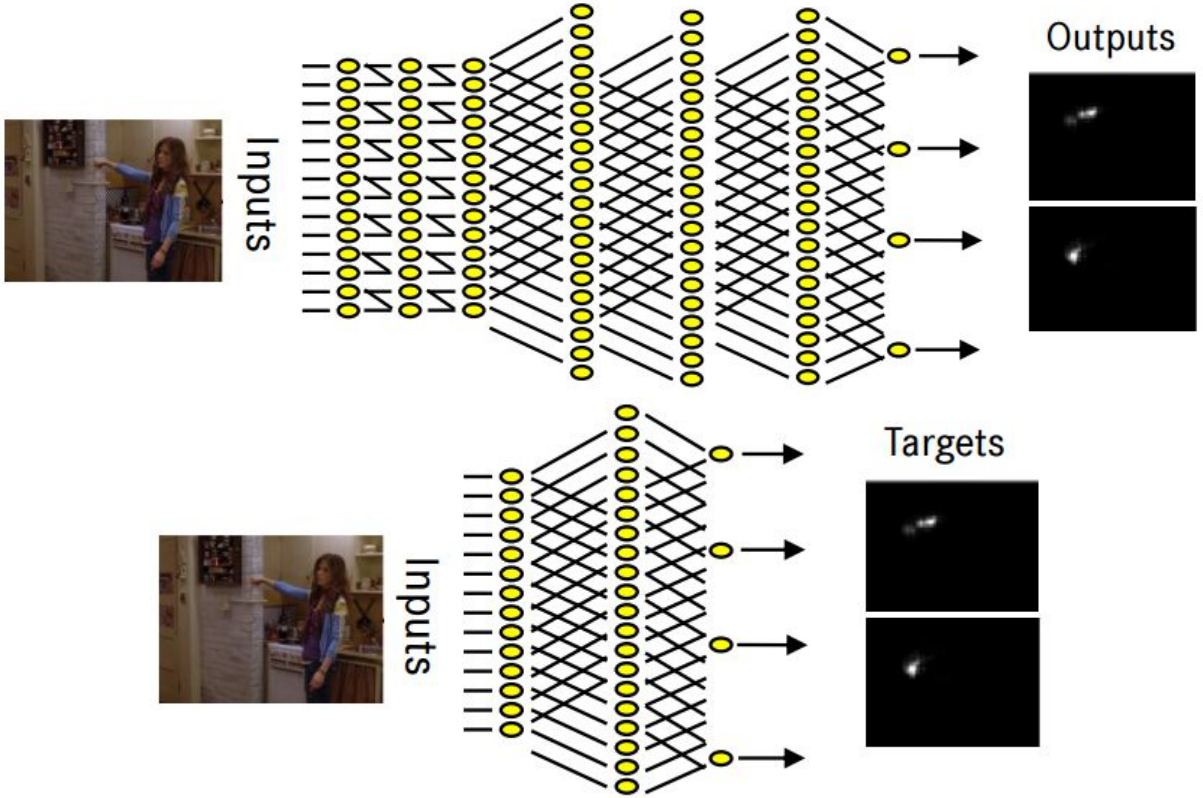
- Train a shallower network (student) with outputs of the larger network (teacher) as target.
- Deep Neural Networks (DNNs) excel over Shallow Neural Networks (SNNs). Why?
  - DNNs have more parameters.
  - DNNs can learn more complex functions?
  - Convolution gives a plus?
- Methods:
  - Do deep nets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

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
  - *Use softmax with high temperature target $q_i = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}$ and as a cost minimize cross entropy between:

$$P_T^\tau = softmax\left(\frac{a_T}{\tau}\right), P_S^\tau = softmax\left(\frac{a_S}{\tau}\right)$$
$$\mathcal{L}_{KD}(W_S) = \mathcal{H}(y_{true}.P_S) + \lambda\mathcal{H}(P_T^\tau, P_S^\tau)$$

*Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." *NIPS-W,* 2015.

Student - Teacher Net: Result and Discussion

Results:

- Softened outputs reveals the dark knowledge of the network

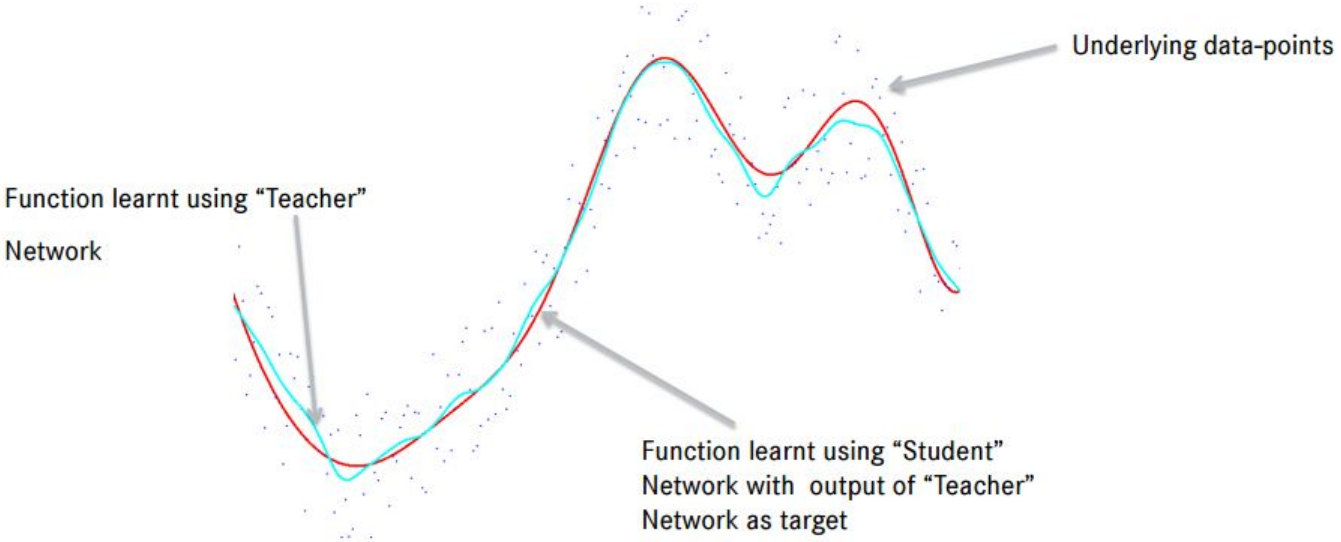
Cow	Dog	Trump	Bush	
0	1	0	0	Original Hard Targets
0.05	0.3	0.2	0.005	Softened Output

Discussion:

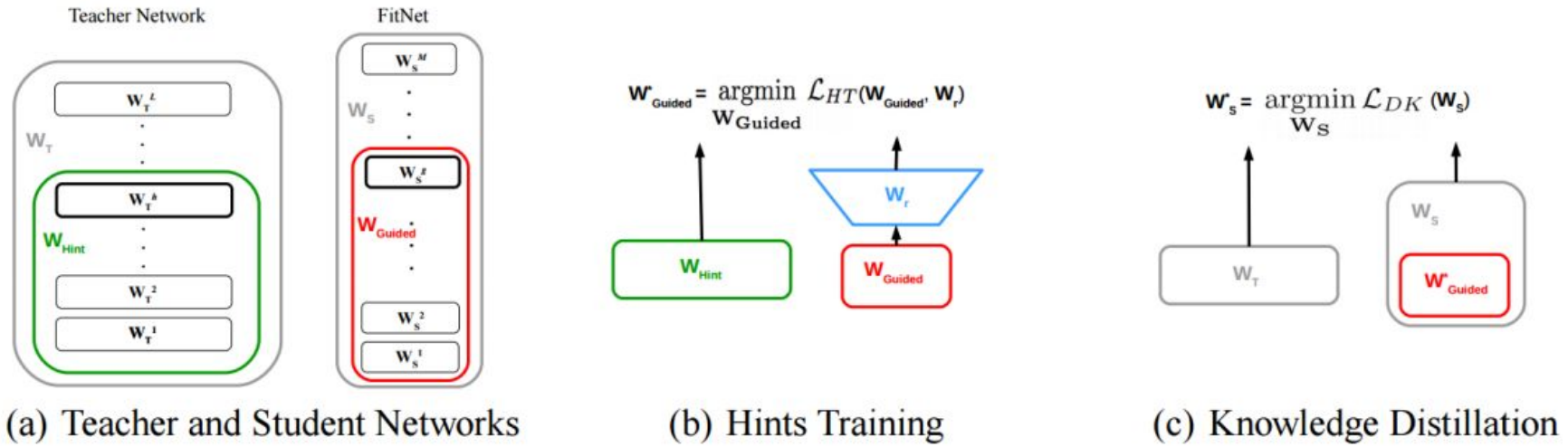
- Why MIMIC models can be more accurate than training on original lables
  - If labels have errors, tachers may eliminate them making learning easier.
  - Teacher might resolve complex regions
  - All outputs have a reason for the student while teacher may encounter

Student – Teacher Net: Discussion

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



Fit Nets: Hints for Thin Deep Nets



- Student Teacher based learning approach with hint based training.
- Also have intermediate supervision, called as hint.
- Hint is defined as the output of a teacher's hidden layer, responsible for

Fit Nets: Learning Objective

- The final learning objective can be written as:

$$P_T^{\tau} = softmax\left(\frac{a_T}{\tau}\right), P_S^{\tau} = softmax\left(\frac{a_S}{\tau}\right)$$
$$\mathcal{L}_{KD}(W_S) = \mathcal{H}(y_{true}.P_S) + \lambda\mathcal{H}(P_T^{\tau}, P_S^{\tau})$$

- The hint loss is given as:

$$\mathcal{L}_{HT}(\boldsymbol{W}_{guided}, \boldsymbol{W}_r) = \frac{1}{2}\left\|u_h(x; \boldsymbol{W}_{\boldsymbol{Hint}}) - r(v_g(x; \boldsymbol{W}_{guided}); \boldsymbol{W}_r)\right\|$$

- Here, u_h and v_g are the teacher/student deep nested functions up to their respective hint/guided layer with parameters $\boldsymbol{W}_{\boldsymbol{Hint}}$ and $\boldsymbol{W}_{guided}$; r is the regressor function on top of the guided layer with parameters $\boldsymbol{W}_r$.

Fit Nets: Results

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**91.61%**
Teacher	~9M	90.18%
Mimic single	~54M	84.6%
Mimic single	~70M	84.9%
Mimic ensemble	~70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Accuracy on CIFAR-10

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**64.96%**
Teacher	~9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Accuracy on CIFAR-100

Fast Algorithms for Convolutional Neural Networks

Andrew Lavin, Scott Gray

Uses Winograd's minimal filtering algorithms to reduce number of arithmetic ops for calculating convolutions.