Convolution Neural Networks

Jayanta Mukhopadhyay
Dept. of Computer Science and Engg.

Courtesy: K. Sairam

References

Most content are adapted from:

- Deep Learning by Ian Goodfellow, Yoshua Bengio & Aaron Courville, An MIT Press Book
- CS231n: Convolutional Neural Networks for Visual Recognition by Fei Fei Li
- CS131 Computer Vision: Foundations and Applications by Juan Carlos Niebles
- Recent Research Papers

Convolution Layer

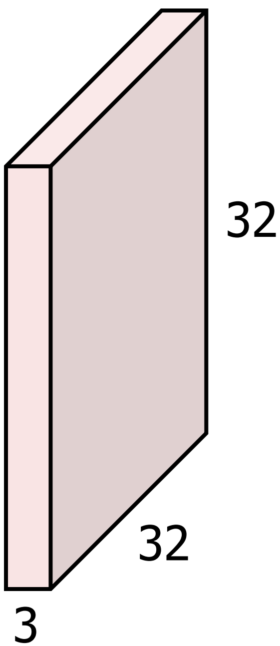
- **Locality**: objects tend to have a local spatial support
- **Translation invariance**: object appearance is independent of location
- **Weight sharing**
  - units connected to different locations have the same weights
  - equivalently, each unit is applied to all locations
  - weights of filters are invariant, the output is equivariant
- Each unit output of filter is connected to a local rectangular area in the input – Receptive Field

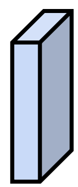
Convolution Layer

32x32x3 image

Filters always extend the full depth of the input volume

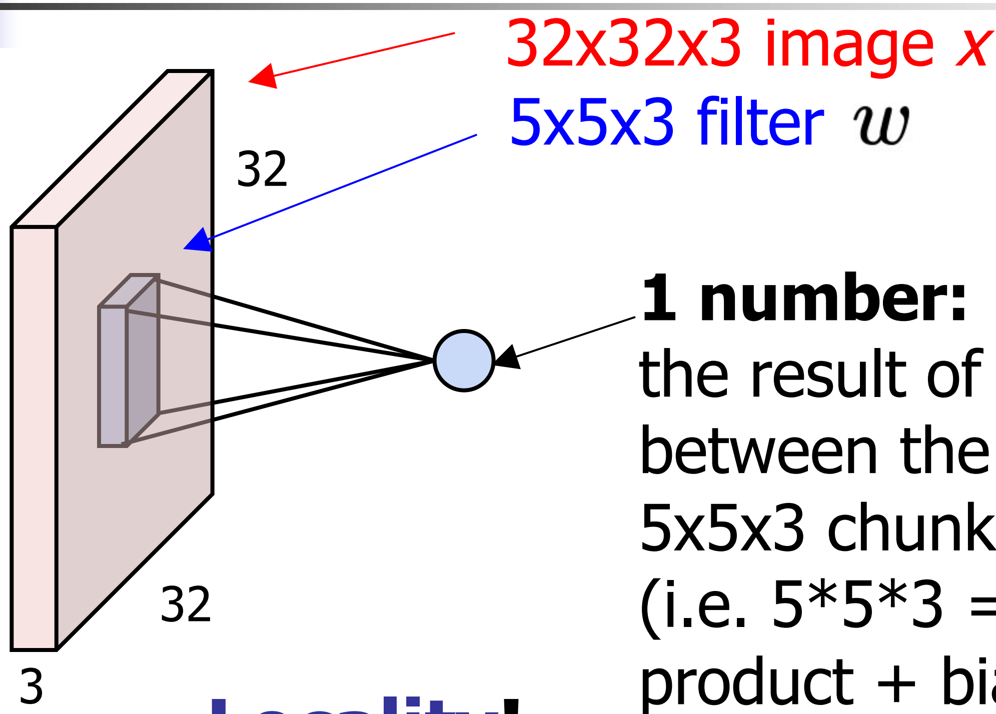
5x5x3 filter (kernel)





Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products".

Convolution Layer

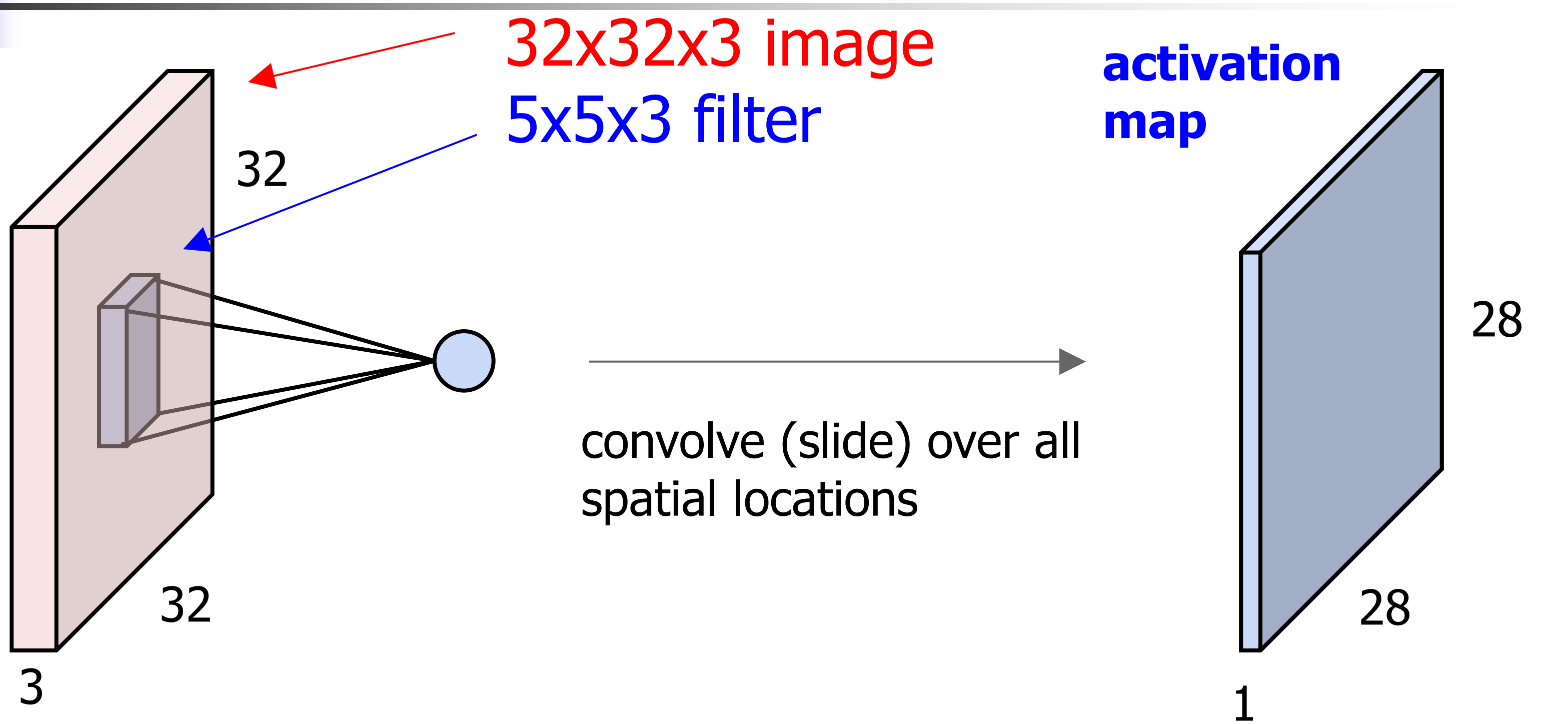


Locality!

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

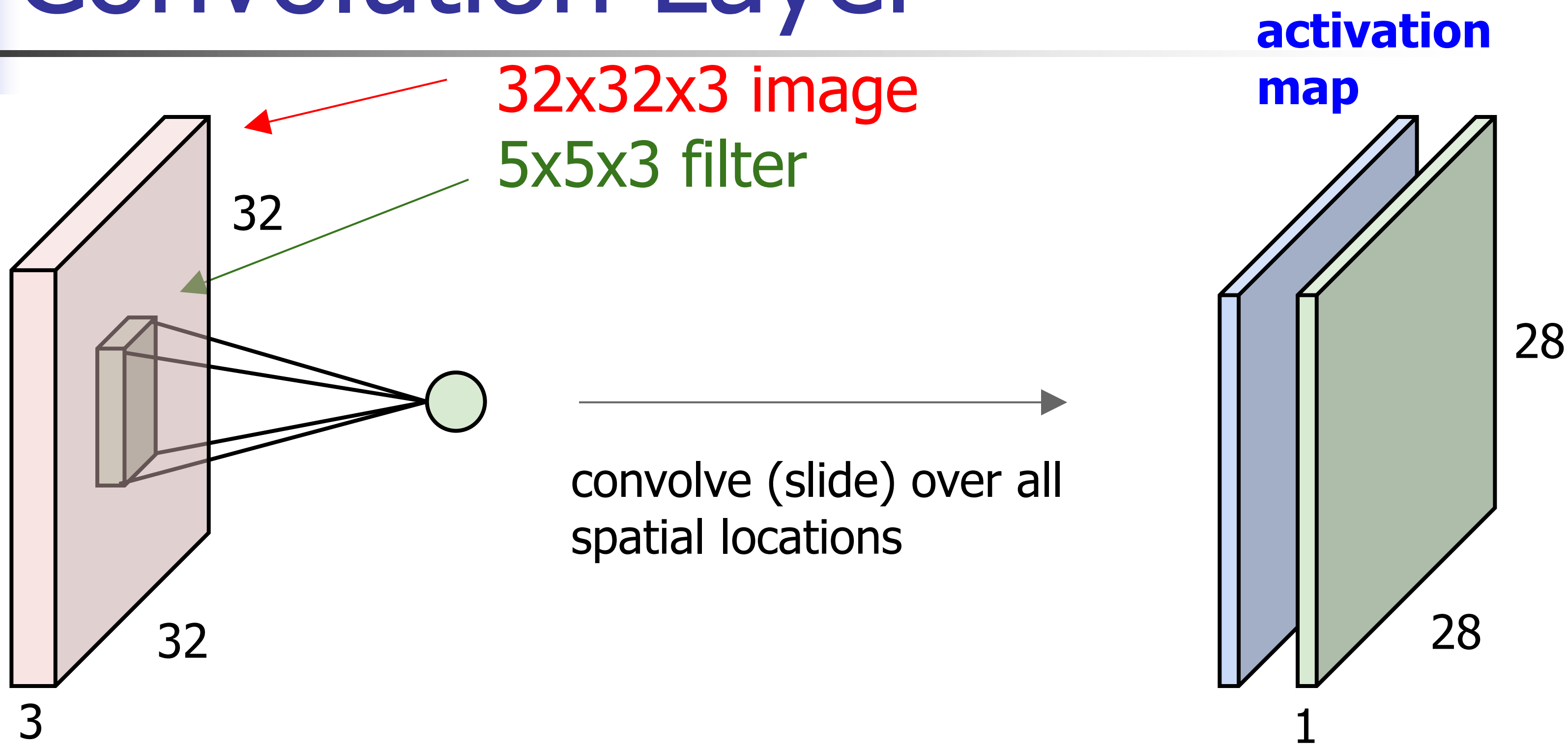
Convolution Layer



Translation Invariance!

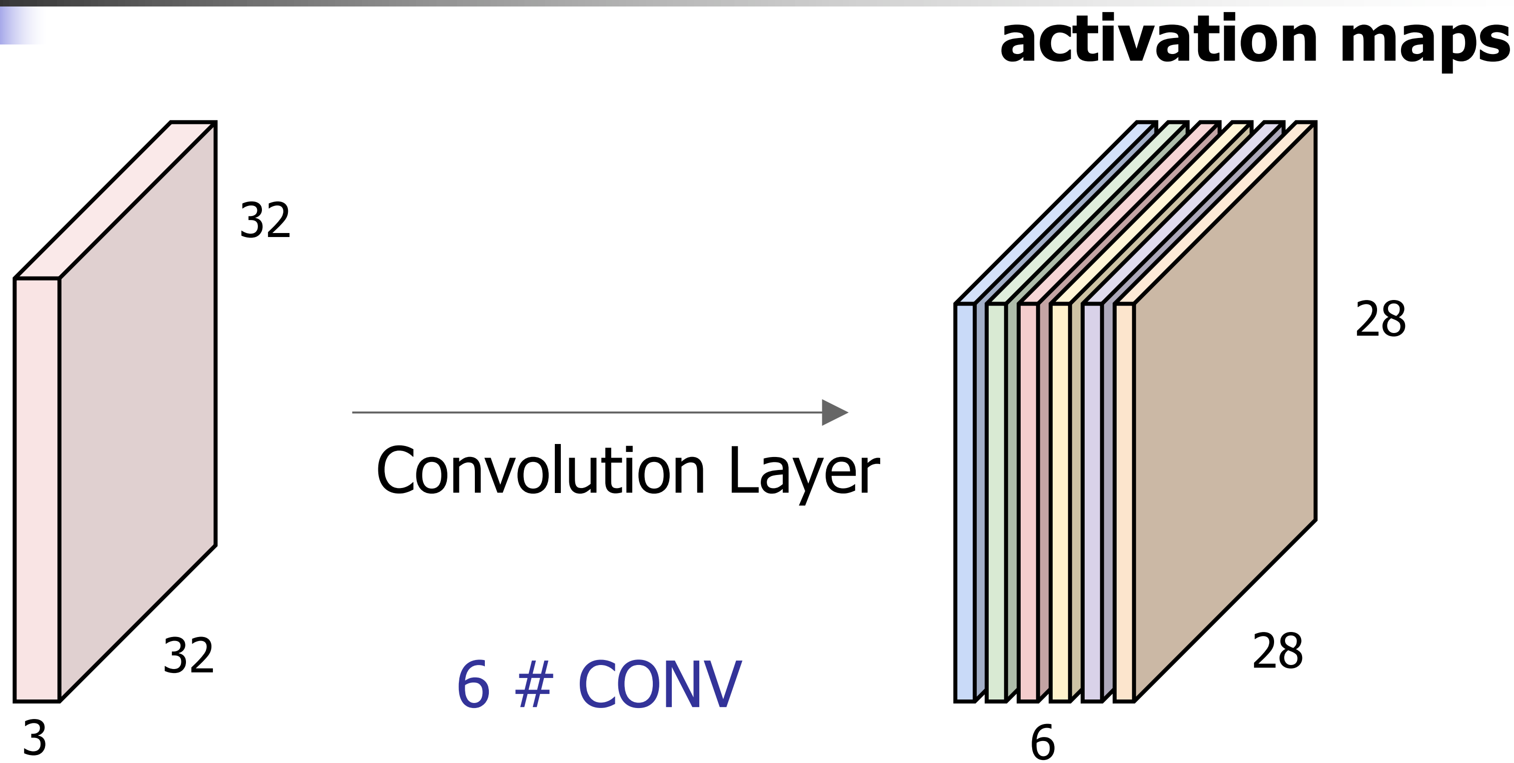
Weight sharing!

Convolution Layer



Consider a second, green filter.

Convolution Layer (CONV)



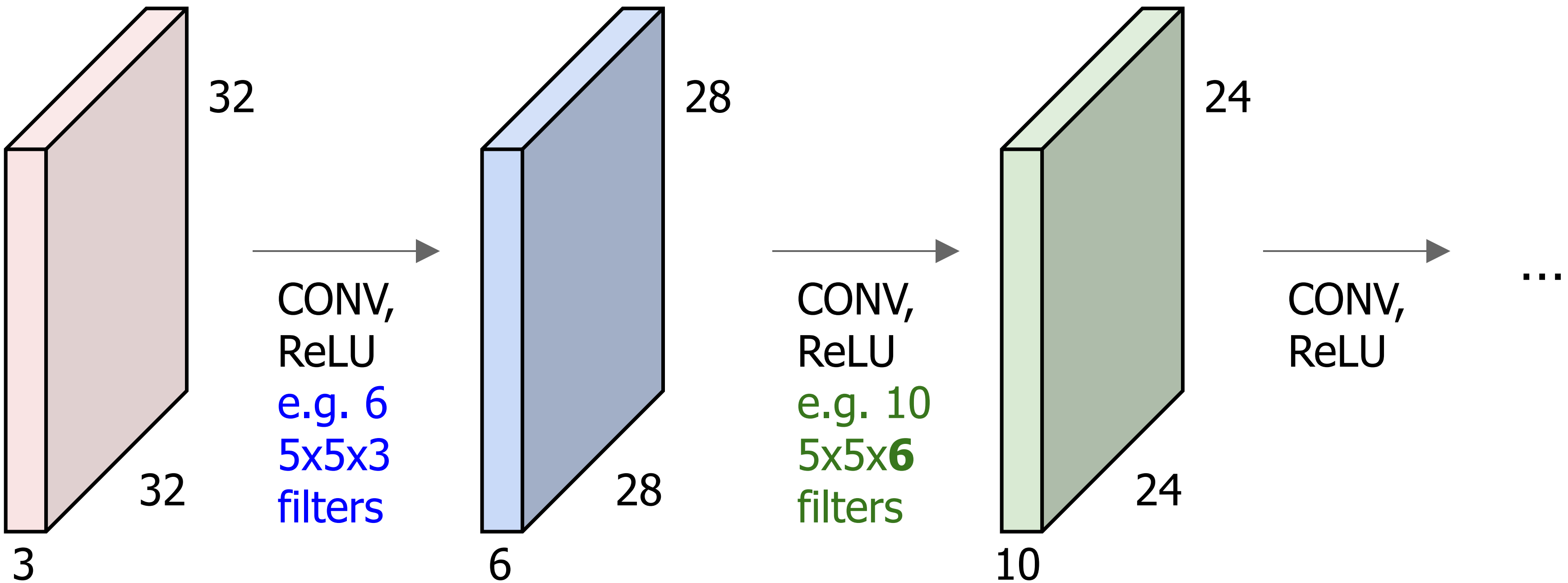
For example, if we had 6 5x5x3 filters, we'll get 6 separate activation maps:

We stack these up to get a "new image" of size 28x28x6!

Non-Linear Layer

- Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer.
- Commonly used in CNN is ReLU.

Convolutional Neural Networks (CNN)



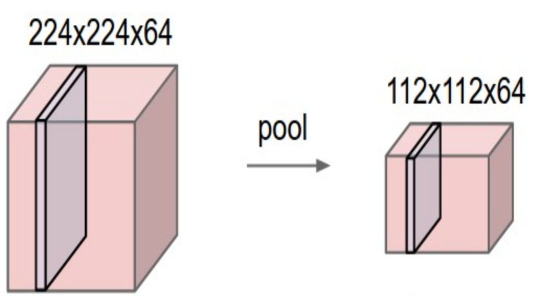
A CNN is a sequence of convolution layers and nonlinearities

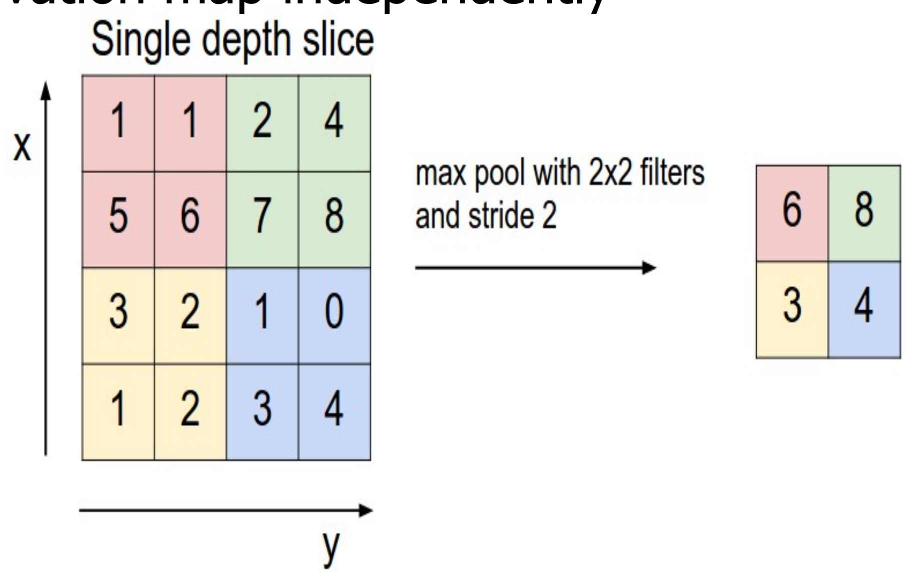
CONV

- Input Volume size W_1 x H_1 x D_1
- No. of filters K with size F_w x F_h x D_1 convolved with stride (S_w, S_h).
- Input in Zero padded by (P_w, P_h) on both sides.
- Output volume size W_2 x H_2 x D_2?
  - $W_2 = (W_1 - F_w + 2P_w)/S + 1$
  - $H_2 = (H_1 - F_h + 2P_h)/S + 1$
  - $D_2 = K$
- Parameters ?
  - $(F_w * F_h * D_1) * K$ weights + K biases
- **d-th depth slice of output is the result of convolution of d-th filter over the padded input volume with a stride, then offset by d-th bias**

Pooling Layer (POOL)

- to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value (Avg.)of the features in that region for MaxPool (AvgPool).
- operates over each activation map independently





POOL

- Input Volume size W_1 x H_1 x D_1
- Pool size F_w x F_h with stride (S_w, S_h).
- Output volume size W_2 x H_2 x D_2?
  - $W_2 = (W_1 - F_w)/S + 1$
  - $H_2 = (H_1 - F_h)/S + 1$
  - $D_2 = K$
- Parameters ?
  - 0!
- **Uncommon to use zero-padding in Pooling layers.**

Fully Connected Layer (FC)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks.
- Input volume to FC layer can also be treated as Deep Features.
- Or if the FC layer is classifier, the input to FC can also be treated as feature vector representation for the sample.

Batch Normalization

- To make Gaussian activation maps.
- Improves gradient flow through the network.
- Allows higher learning rates.
- Reduces the strong dependence on initialization.
- Acts as a form of regularization.
- Usually inserted after FC / CONV layers, and before non-linearity.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

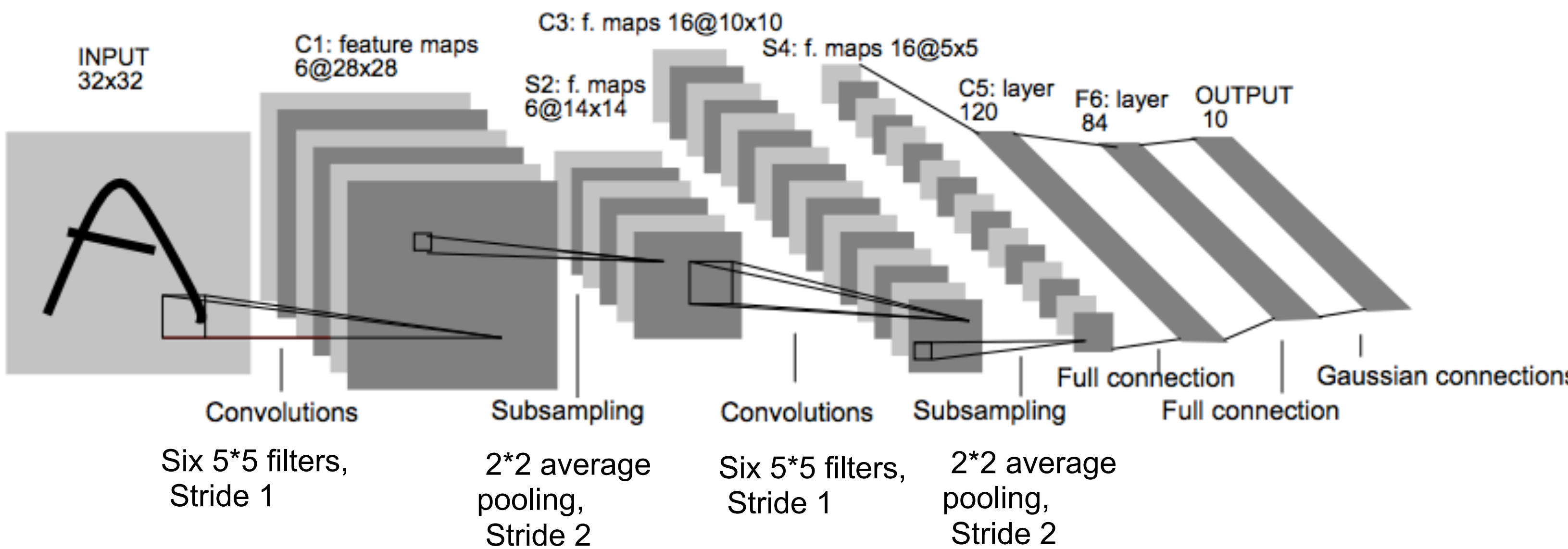
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

*Ioffe and Szegedy, 2015

CNN Architectures

LeNet

- Gradient-based learning applied to document recognition.
- Architecture: Input→CONV→POOL→CONV→POOL→FC→FC→Output
- Weights: 60k & FLOPS: 341k
- Sigmoid used for non-linearity

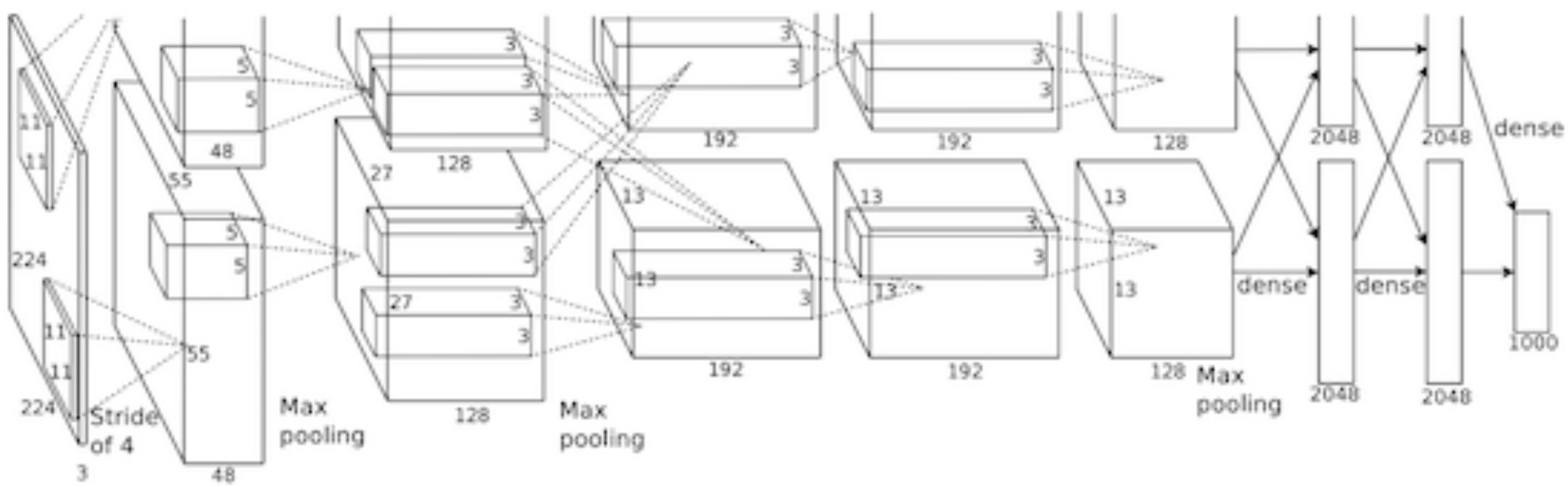


*Y. Lecun et al, Proceedings of the IEEE, 1998

AlexNet

- Uses Local Response Normalization (LRN)
- Architecture:

  Input→CONV1→MAXPOOL1→NORM1→CONV2→MAXPOOL2→NORM2 →CONV3→CONV4→CONV5→MAXPOOL3→FC6→FC7→FC8→Output
- Weights: 61M & FLOPS: 724M
- ReLU used for non-linearity



*Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks NIPS, 2012

AlexNet

Full (simplified) AlexNet architecture:

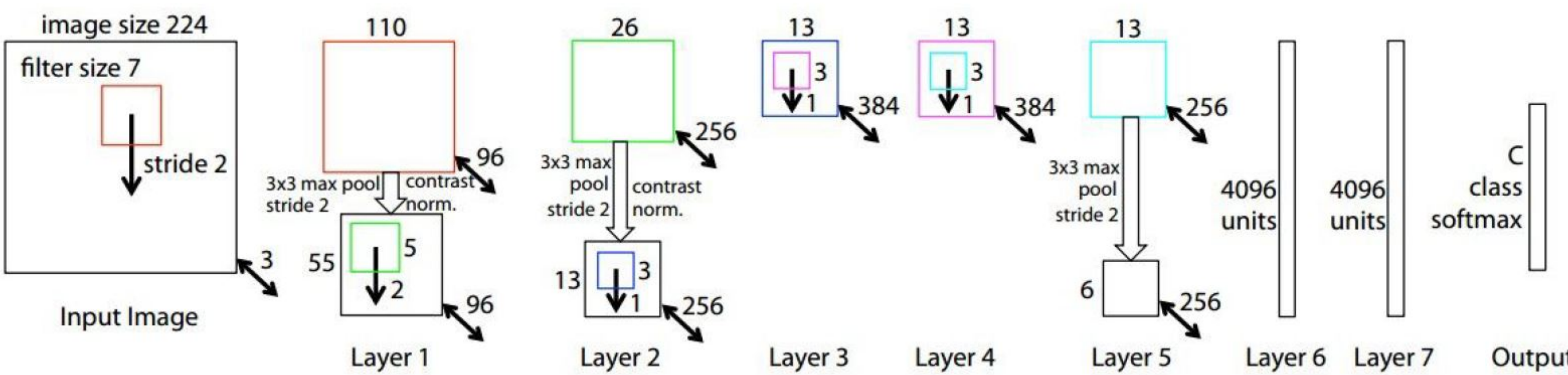
[227x227x3]	INPUT	
[55x55x96]	CONV1	: 96 11x11 filters at stride 4, pad 0
[27x27x96]	MAX POOL1	: 3x3 filters at stride 2
[27x27x96]	NORM1	: Normalization layer
[27x27x256]	CONV2	: 256 5x5 filters at stride 1, pad 2
[13x13x256]	MAX POOL2	: 3x3 filters at stride 2
[13x13x256]	NORM2	: Normalization layer
[13x13x384]	CONV3	: 384 3x3 filters at stride 1, pad 1
[13x13x384]	CONV4	: 384 3x3 filters at stride 1, pad 1
[13x13x256]	CONV5	: 256 3x3 filters at stride 1, pad 1
[6x6x256]	MAX POOL3	: 3x3 filters at stride 2
[4096]	FC6	: 4096 neurons
[4096]	FC7	: 4096 neurons
[1000]	FC8	: 1000 neurons (scores for 1000 classes)

Parameters Count: AlexNet

- Input: 227x227x3 images
- First layer (CONV1): 96 11x11 filters applied at stride 4
- Q: what is the output volume size? Hint: (227-11)/4+1 = 55
  - Output volume [55x55x96]
- Q: What is the total number of parameters in this layer?
  - Parameters: (11*11*3)*96 = 35K (Without bias)
  - Parameters: (11*11*3)*96 + 96 (With bias)
- Second layer (POOL1): 3x3 filters applied at stride 2
- Q: what is the output volume size? Hint: (55-3)/2+1 = 27
  - Output volume: 27x27x96 (Input to POOL1 is output of CONV1)
- Q: what is the number of parameters in this layer?
  - Parameters: 0

ZFNet

- AlexNet but:
- CONV1: change from (11x11 stride 4) to (7x7 stride 2)
- CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512
- ImageNet top 5 error: 16.4% → 11.7%

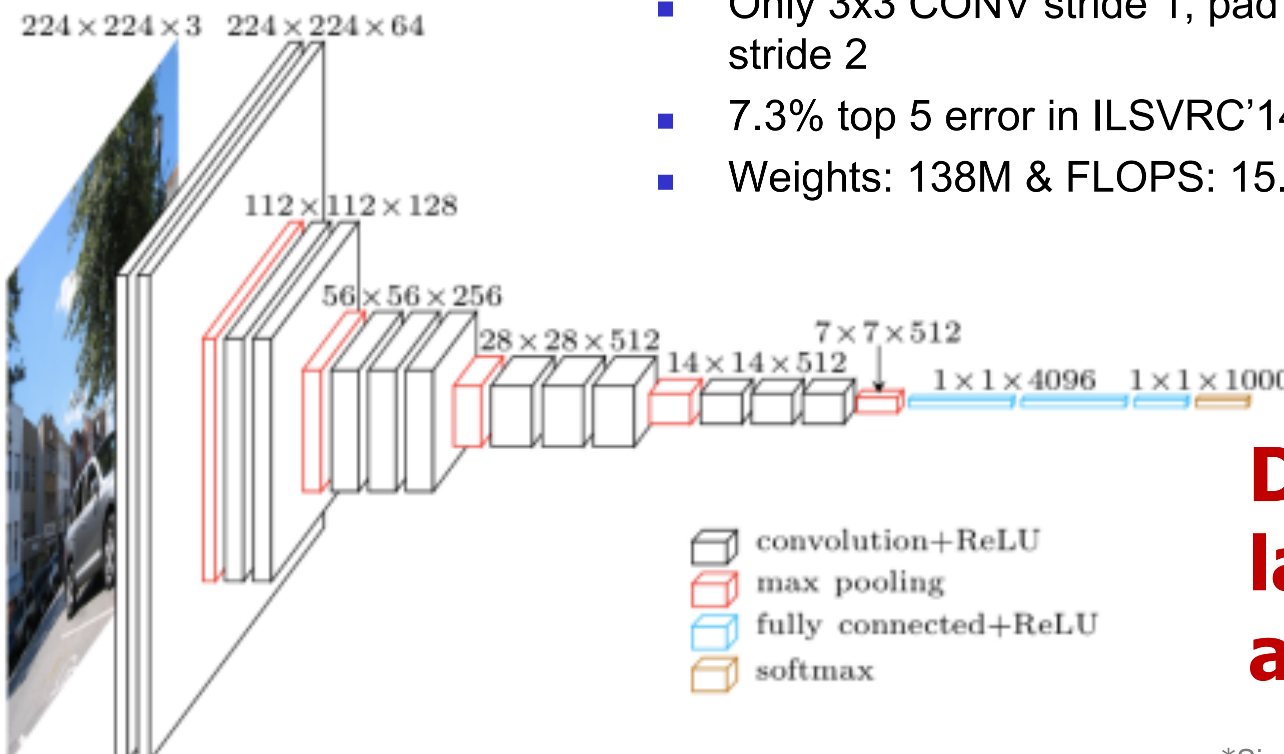


Smaller filter size, More filters in layer

* Zeller and Fergus, 2013

VGG

- Smaller filters, deeper layers
- 8 layers (AlexNet) → 13 layers (VGG13) / 16 layers (VGG16Net) / 19 layers (VGG19Net)
- Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2
- 7.3% top 5 error in ILSVRC'14
- Weights: 138M & FLOPS: 15.5G



Deeper the layer, better accuracy

*Simonyan et al., arXiv 2014, ICLR 2015

VGG

- Stack of three 3x3 conv (stride 1) layers has **same effective field** as one 7x7 conv layer
  - deeper with more non-linearities
  - Fewer parameters: How?
  - $3*(3^2\ C^2)$ vs. $(7^2\ C^2)$ for C channels per layer

VGG13 – Parameter count per layer (No bias)

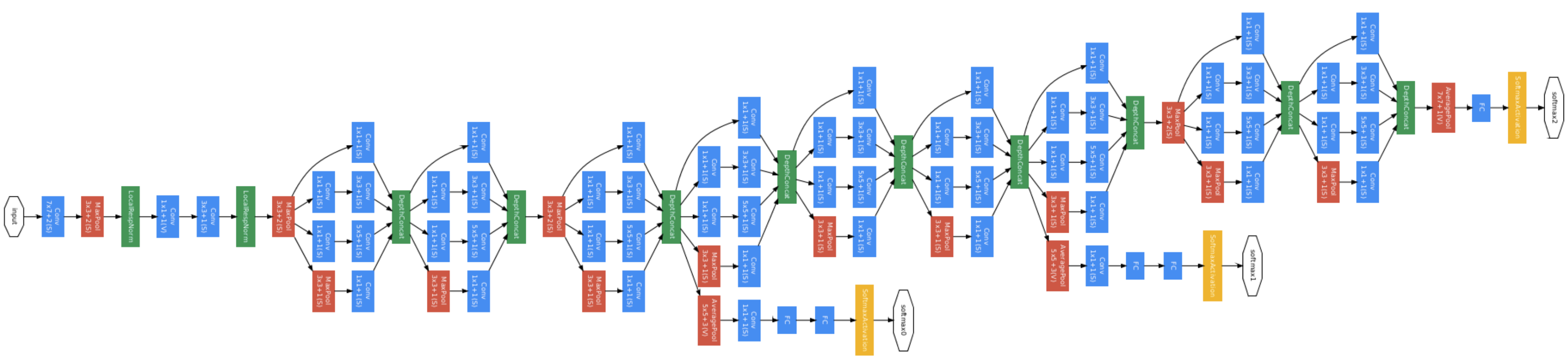
INPUT	: [224x224x3]	: 0
CONV1	: [224x224x64]	: (3*3*3)*64 = 1,728
CONV2	: [224x224x64]	: (3*3*64)*64 = 36,864
POOL1	: [112x112x64]	: 0
CONV3	: [112x112x128]	: (3*3*64)*128 = 73,728
CONV4	: [112x112x128]	: (3*3*128)*128 = 147,456
POOL2	: [56x56x128]	: 0
CONV5	: [56x56x256]	: (3*3*128)*256 = 294,912
CONV6	: [56x56x256]	: (3*3*256)*256 = 589,824
CONV7	: [56x56x256]	: (3*3*256)*256 = 589,824
POOL3	: [28x28x256]	: 0
CONV8	: [28x28x512]	: (3*3*256)*512 = 1,179,648
CONV9	: [28x28x512]	: (3*3*512)*512 = 2,359,296
CONV10	: [28x28x512]	: (3*3*512)*512 = 2,359,296
POOL4	: [14x14x512]	: 0
CONV11	: [14x14x512]	: (3*3*512)*512 = 2,359,296
CONV12	: [14x14x512]	: (3*3*512)*512 = 2,359,296
CONV13	: [14x14x512]	: (3*3*512)*512 = 2,359,296
POOL5	: [7x7x512]	: 0
FC	: [1x1x4096]	: 7*7*512*4096 = 102,760,448
FC	: [1x1x4096]	: 4096*4096 = 16,777,216
FC	: [1x1x1000]	: 4096*1000 = 4,096,000 (scores for 1000 classes)

More parameters in FC layer

GoogleNet

- CONV Layers: 21 (depth), 57 (total)
- Fully Connected Layers: 1
- Weights: 7.0M & FLOPS: 1.43G
- Architecture: (9 Inception Modules)

  INPUT→CONV1→POOL1→CONV2→CONV3→POOL2→INCEPTION1→INCEPTION2→POOL3→INCEPTION3→INCEPTION4→INCEPTION5→INCEPTION6→INCEPTION7→POOL4→INCEPTION8→INCEPTION9→POOL5→FC1→OUTPUT
- ILSVRC’14 classification winner (6.7% top 5 error)



*Szegedy et al., arXiv 2014, CVPR 2015

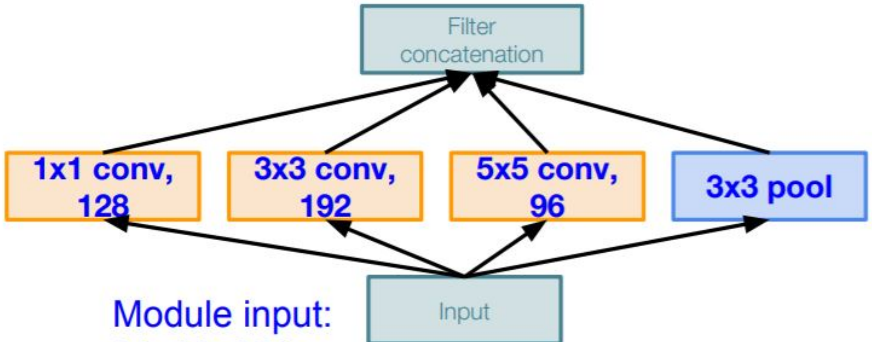
Naïve Inception Module

- Apply parallel filter operations on the input from previous layer:
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise.
- output size? (Assume zero padding to get same height and width)
- Computational Complexity in following Inception Module?

Hint: (Input H * Input W * No. of filters * Filter F * Filter F * Input C)

  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x256
  - [5x5 conv, 96] 28x28x96x5x5x256
  - Total: 854M ops

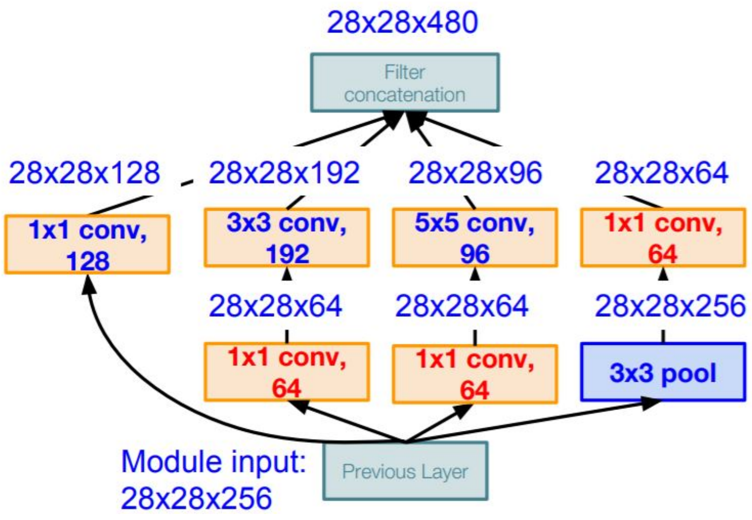
28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Inception Module with bottleneck

- Still Very expensive compute
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer which increases parameter count and computation!
- “bottleneck” layers that use 1x1 convolutions to reduce feature depth preserves spatial dimensions, reduces depth!
- Projects depth to lower dimension (combination of feature maps) adding “1x1 conv, 64 filter” bottlenecks:
- Conv Ops:
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x64
  - [5x5 conv, 96] 28x28x96x5x5x64
  - [1x1 conv, 64] 28x28x64x1x1x256
  - Total: 358M ops
- Compared to 854M ops for naive version, Bottleneck can also reduce depth after pooling layer

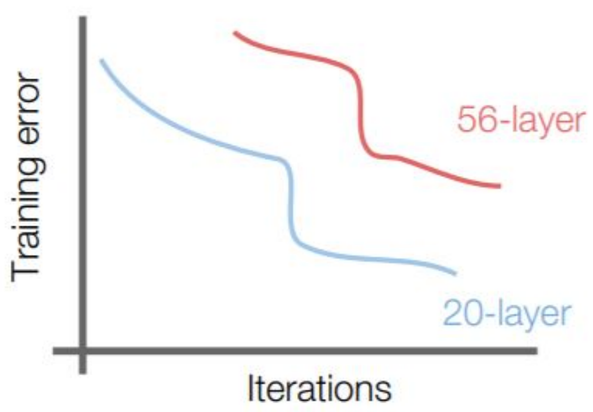


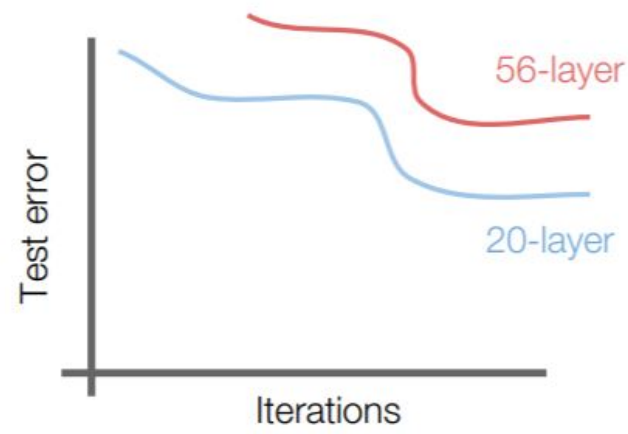
ResNet

- If we continue stacking deeper layers on a "plain" convolutional neural network, the deeper model performs worse, but it's not caused by overfitting!

→ Deeper models are harder to optimize, because of vanishing gradients.

→ The gradients die as we go deeper.

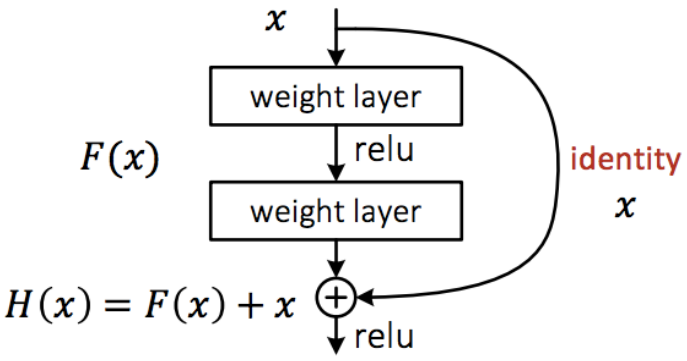




*He et al., ResNet arxiv 2015

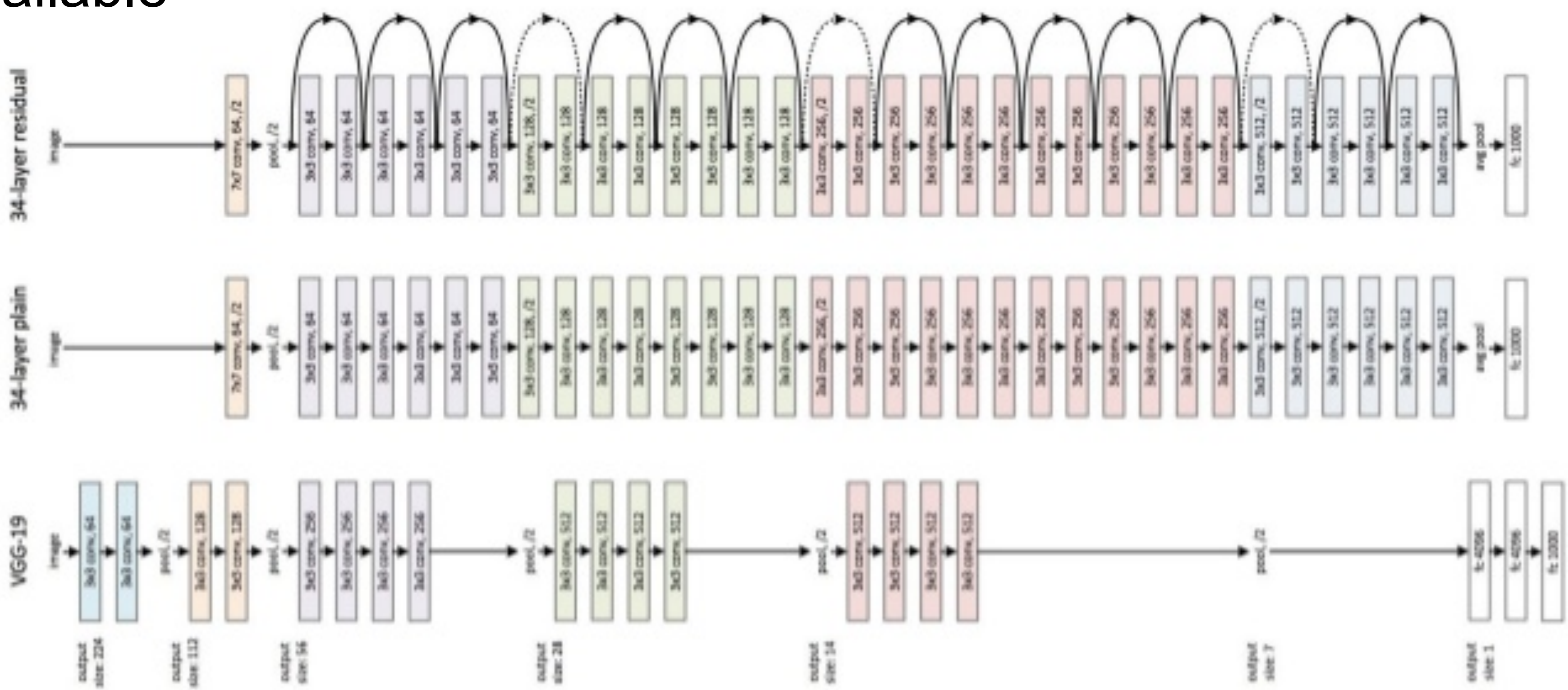
ResNet

- A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.
- Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet

Total depths of 34, 50, 101, or 152 layers architectures are also available



*https://www.slideshare.net/SomnathBanerjee17/lenet-to-resnet

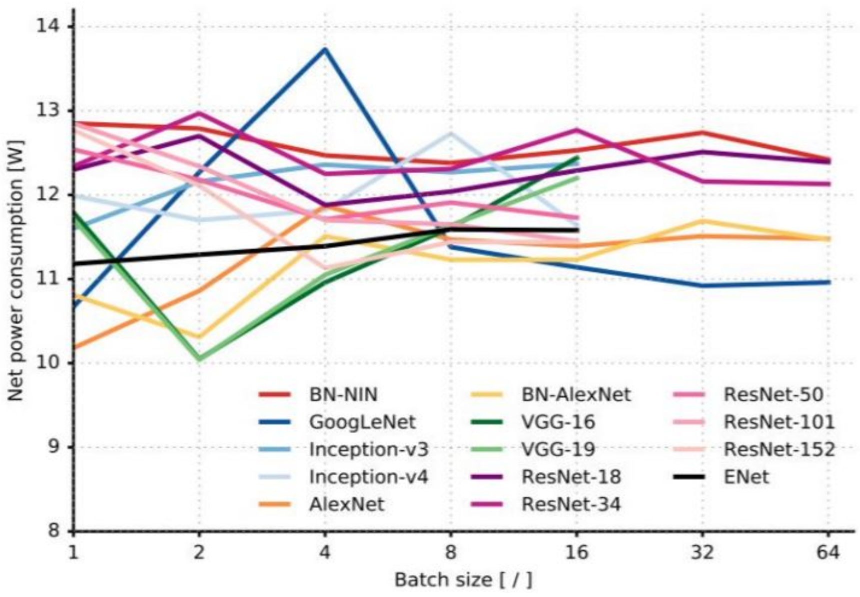
Other Networks

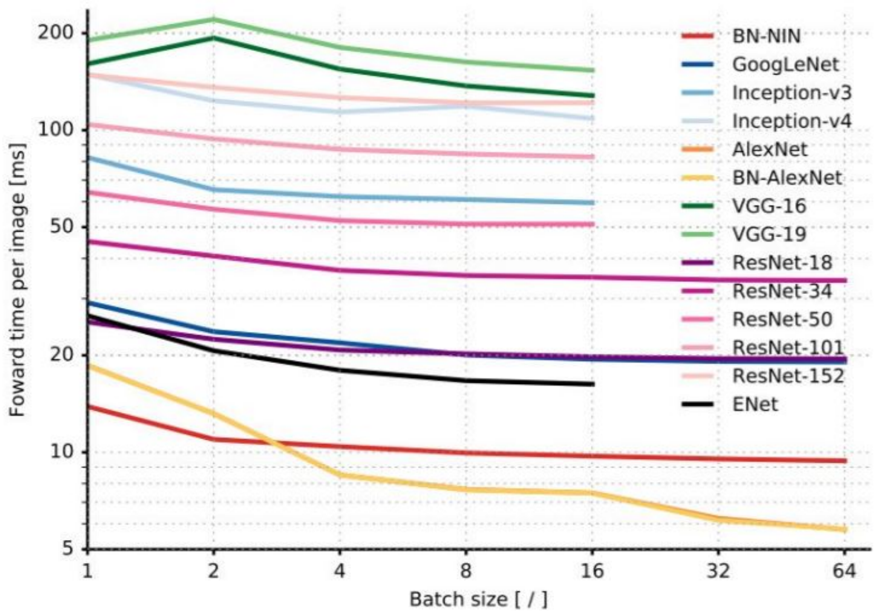
- Network in Network (NIN)
- Wide Residual Networks
- Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)
- DenseNets
- SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size
- MobileNet (Depthwise Seperable Convolutions)
- ShuffleNet (Grouped Convolutions)
- FractalNet: Ultra-Deep Neural Networks without Residuals

Comparisions

Key findings are:

1. Power consumption is independent of batch size and architecture
2. Accuracy and Inference time are in a hyperbolic relationship
3. Energy constraint is an upper bound on the maximum achievable accuracy and model complexity
4. Number of operations is a reliable estimate of the inference time.

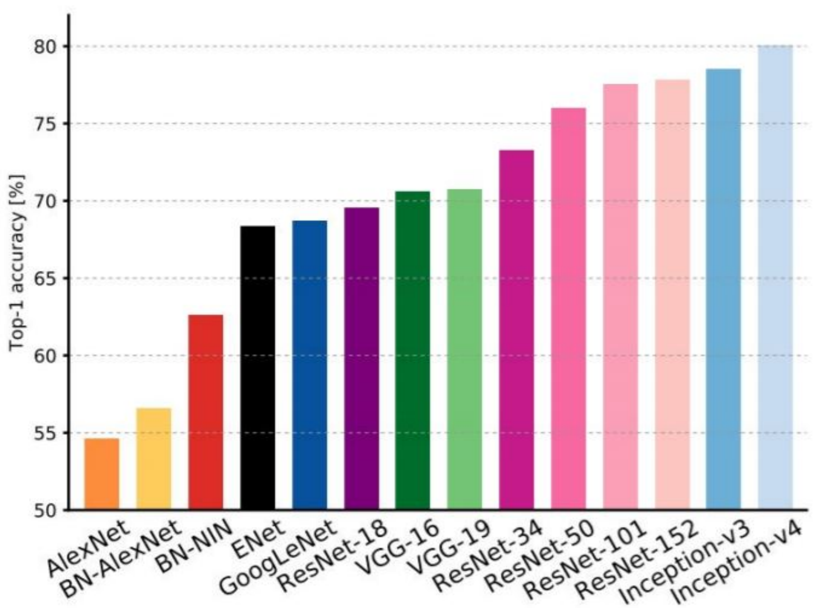


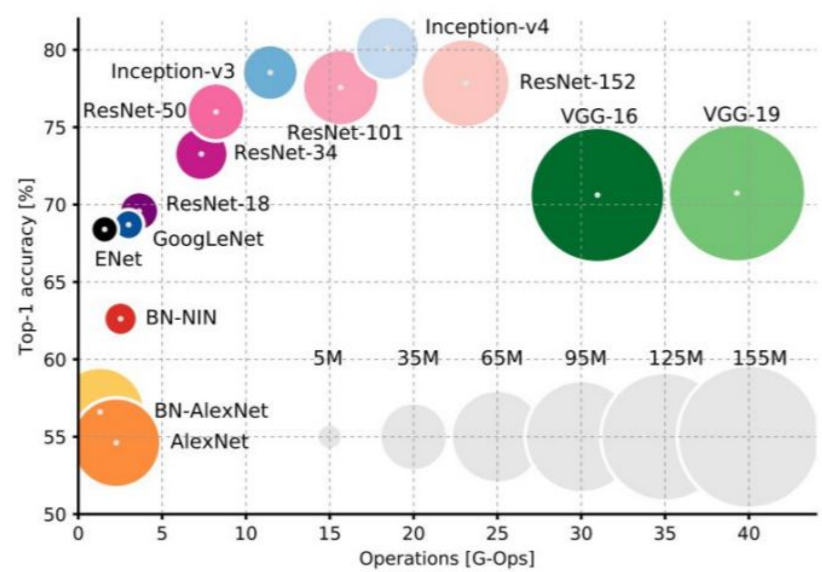


* Alfredo Canziani, An Analysis of Deep Neural Network Models for Practical Applications, 2017

Comparisons

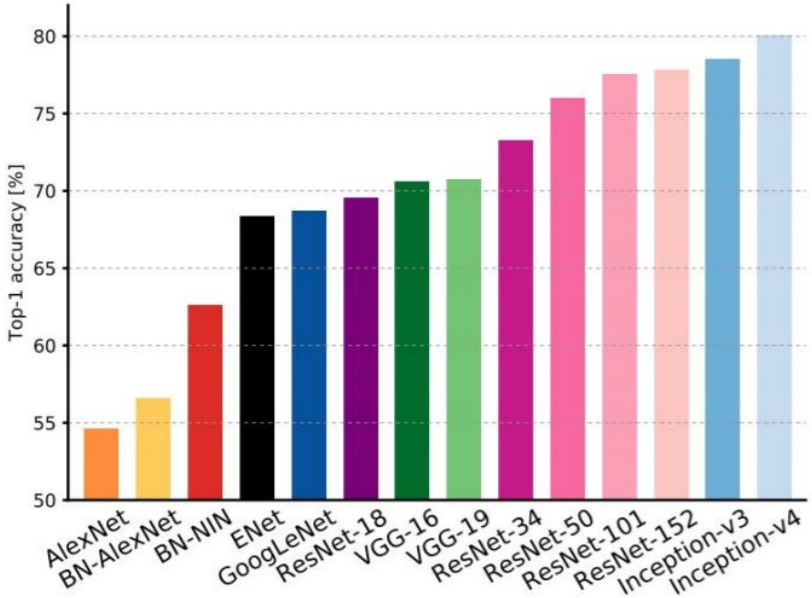
- Top1 Accuracy : Inception-v4 (Resnet + Inception)
- VGG : Highest memory, most operations
- GoogLeNet : most efficient
- AlexNet : Smaller compute, still memory heavy, lower accuracy
- ResNet : Moderate efficiency depending on model, one of the highest accuracy

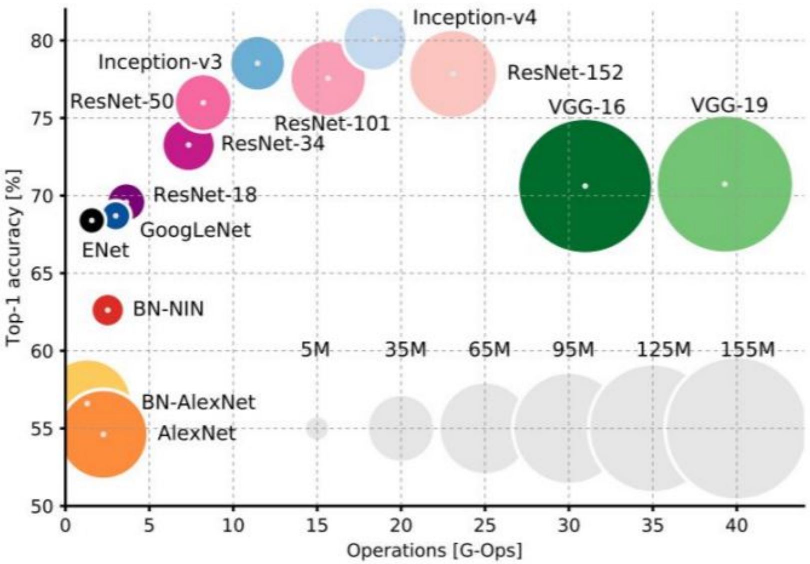




* Alfredo Canziani, An Analysis of Deep Neural Network Models for Practical Applications, 2017

Comparisons





- Top1 Accuracy: Inception-v4 (Resnet + Inception)
- VGG: Highest memory, most operations
- GoogLeNet: most efficient
- AlexNet: Smaller compute, still memory heavy, lower accuracy
- ResNet: Moderate efficiency depending on model, one of the highest accuracy

Training steps:

1. Preprocessing of training dataset.
   - Normalized data
   - Decorrelated data (Diagonal Covariance Matrix)
   - Whitening data (Identity Covariance Matrix)
   - Subtract Per-channel Mean or Mean image
2. Data augmentation.
   - Horizontal Flips
   - Random Crops on scaled input
   - Color jitter
   - Distortions
   - Transformations
3. Design the Neural Network.
4. Weight initialization (eg. Xavier initialization)
5. Train the network by update the weight parameters.

Few Training Tips

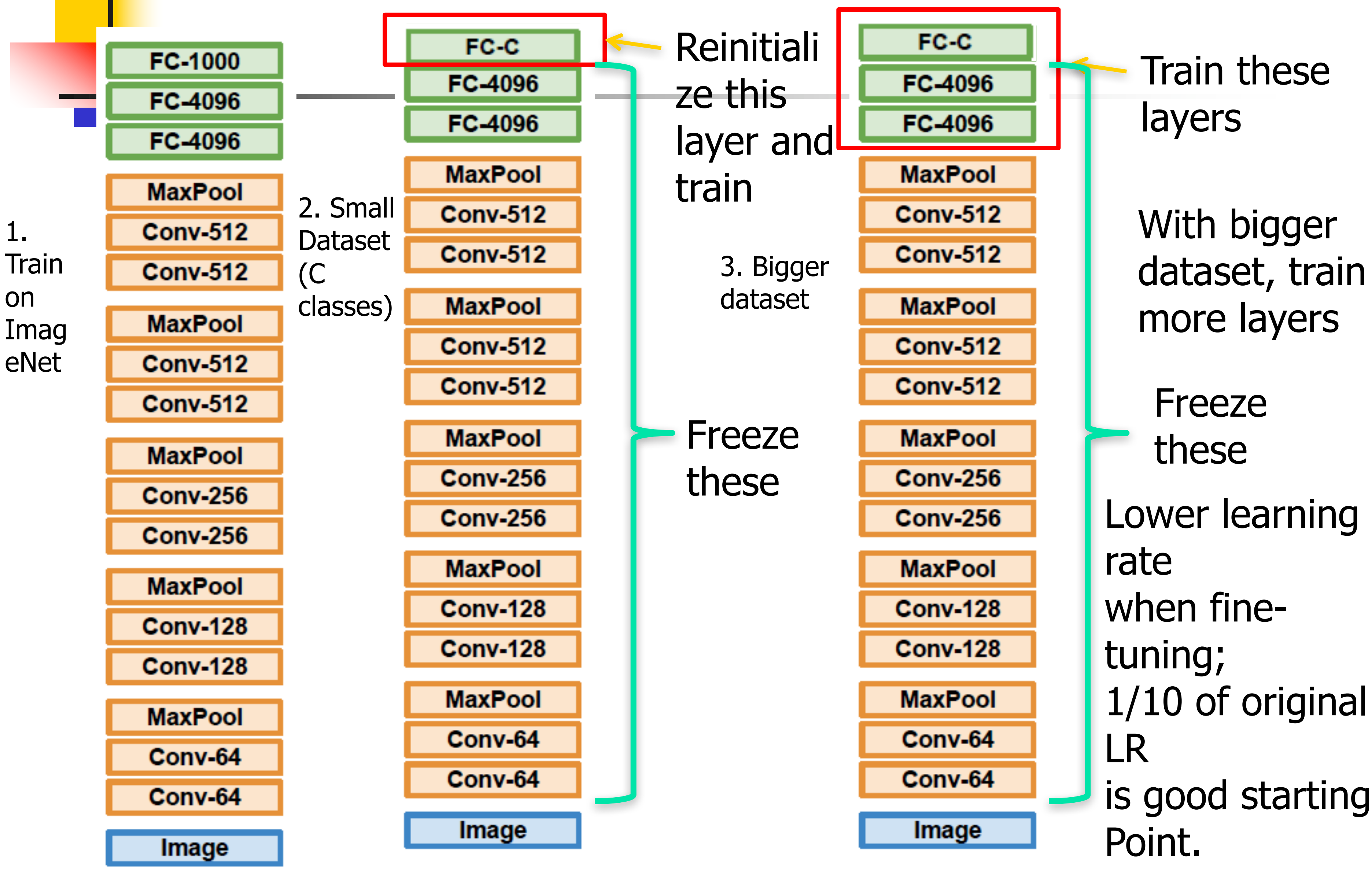
- Start with small regularization and find learning rate that makes the loss go down.
- Can overfit very small portion of the training data.
- Train first few epochs with few samples to initiate the hyper-parameters.
- If big gap between training accuracy and validation accuracy, then it is overfitting.
  - Try increase regularization.
- If no gap, then may increase model capacity.

Transfer Learning

- No need of a lot of a data if want to train CNN.
- Pre-trained models can be initialized for CNNs at the early stage of training.

*Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning



Thank you!