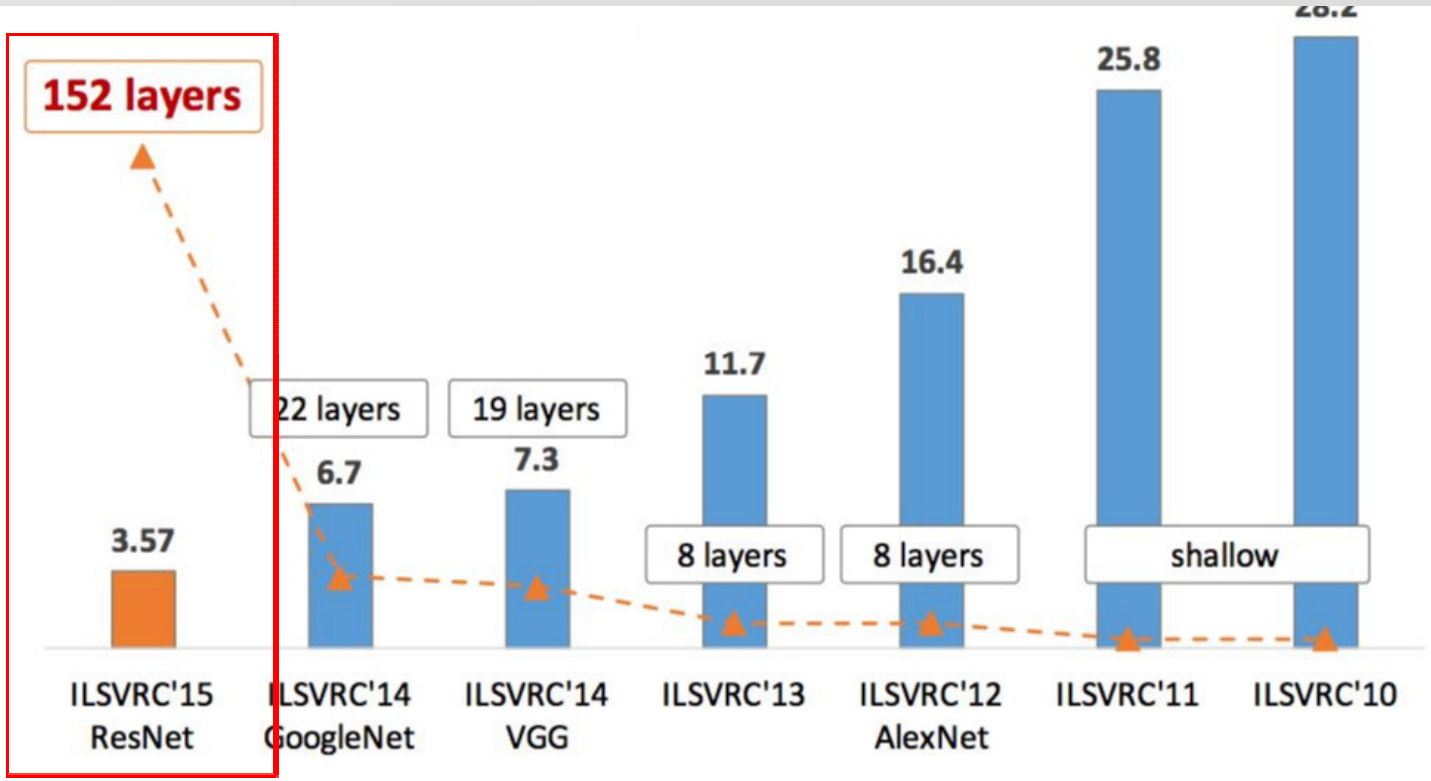
ResNet

In this Lecture

- Introducing a breakthrough neural networks architecture introduced on 2015.
- Why deep?
- What's the problem in learning deep networks?
- **ResNet** and how it allow us to gain more performance via deeper networks.
- Some results, improvements and farther works.

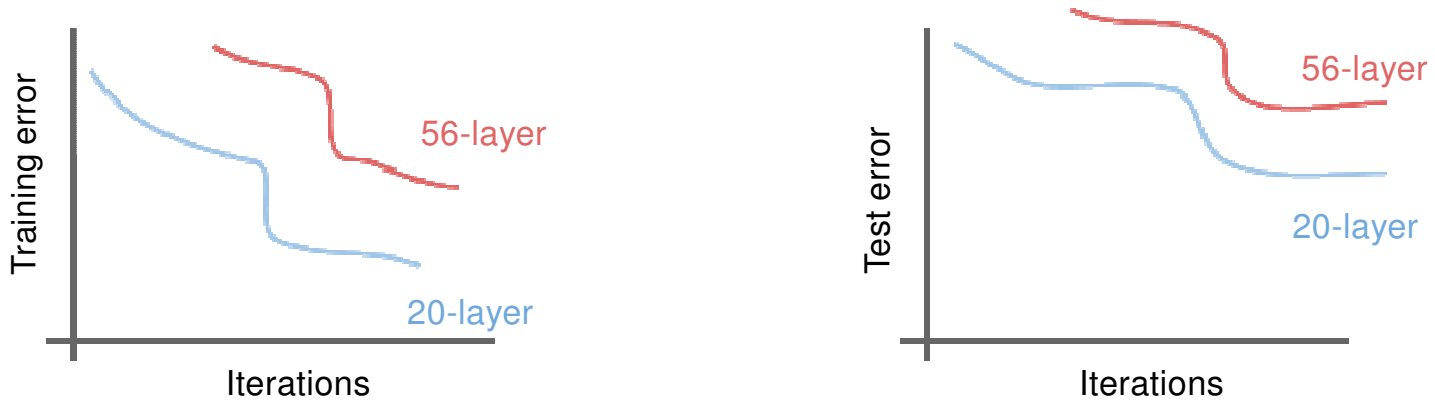
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Intro | ResNet | Technical details | Results | ResNet 1000 | Comparison

Deep vs Shallow Networks

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it’s not caused by overfitting!

Deeper models are harder to optimize

- The deeper model should be able to perform at least as well as the shallower model.
- A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

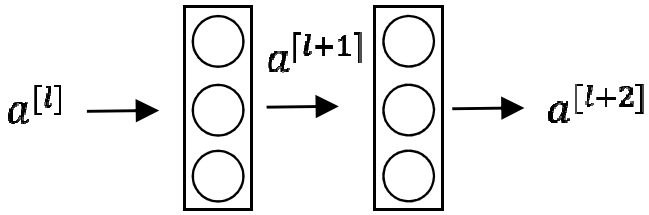
Challenges

- Deeper Neural Networks start to degrade in performance.
- Vanish/Exploding Gradient – May lead for extremely complex parameters initializations to make it work. Still might suffer from Vanish/Exploding even for the best parameters.
- Long training times – Due to too many training parameters.

ResNet

- A specialized network introduced by Microsoft.
- Connects inputs of layers into farther part of that network to allow “shortcuts”.
- Simple idea – great improvements with both performance and train time.

Plain Network



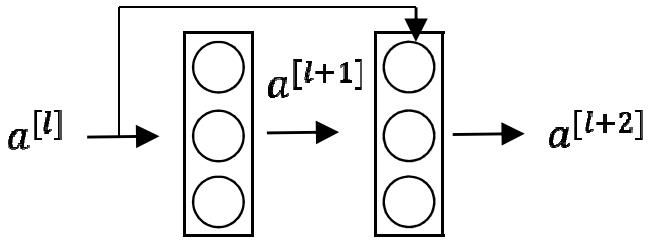
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$
"linear"

$$a^{[l+1]} = g(z^{[l+1]})$$
"relu"

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
"output"

$$a^{[l+2]} = g(z^{[l+2]})$$
"relu on output"

Residual Blocks

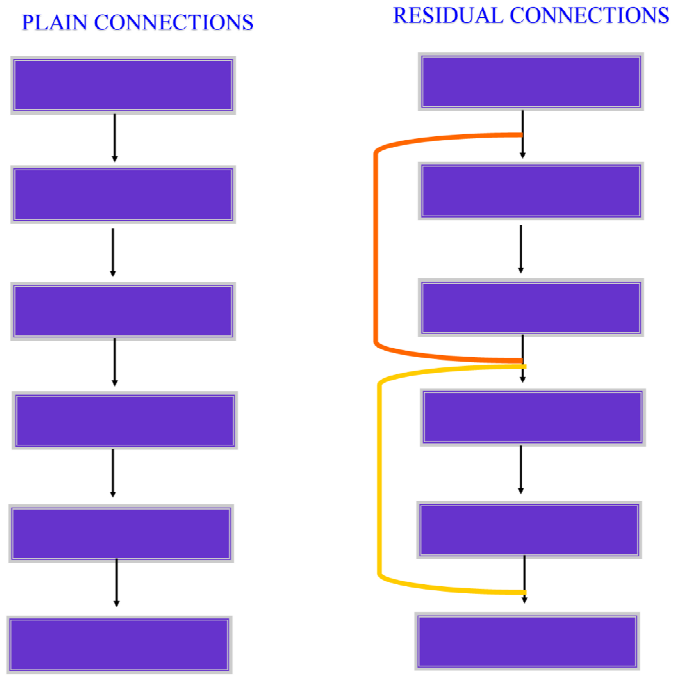


$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$
"linear"

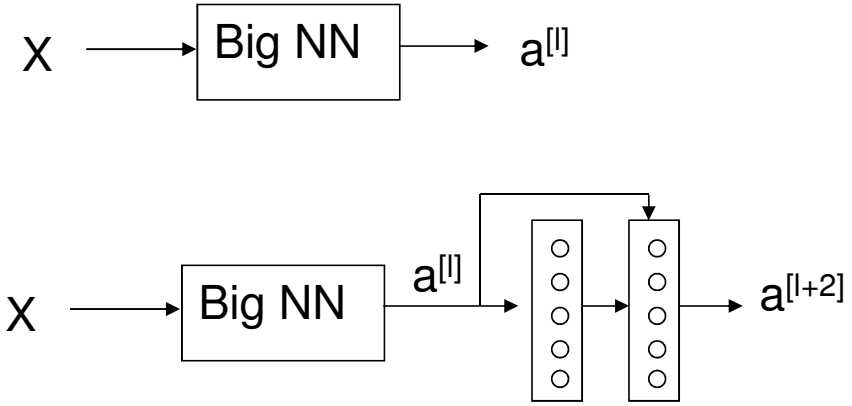
$$a^{[l+1]} = g(z^{[l+1]})$$
"relu"

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
"output"

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
"relu on output **plus input**"



Residual Blocks



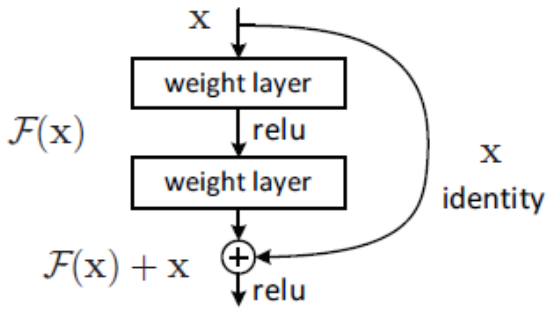
$$a^{[l+2]}=g(z^{[l+2]}+a^{[l]})$$

$$=g(w^{[l+2]}a^{[l+2]}+b^{[l+2]}+a^{[l]})=g(a^{[l]})$$

if $w^{[l+2]}=0$ and $b^{[l+2]}=0$

Identity function is easy to learn for residual block

Skip Connections “shortcuts”

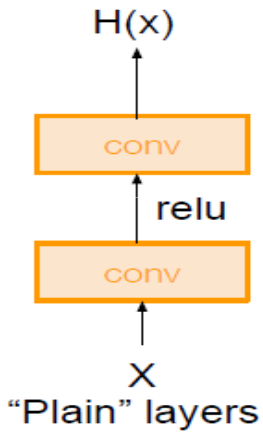


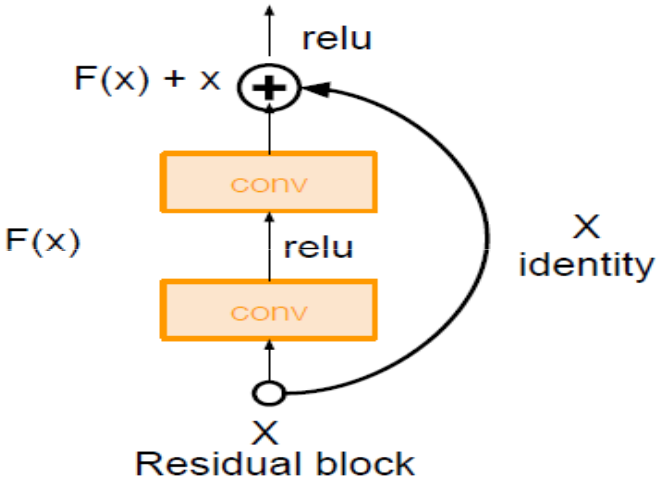
- Such connections are referred as skipped connections or shortcuts. In general similar models could skip over several layers.
- They refer to residual part of the network as a unit with input and output.
- Such residual part receives the input as an amplifier to its output – The dimensions usually are the same.
- Another option is to use a projection to the output space.
- Either way – no additional training parameters are used.

ResNet

He et. al. 2015

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping





$F(x) = w^{[l+2]} a^{[l+2]} + b^{[l+2]}$

$x = a^{[l]}$

g() is ReLU

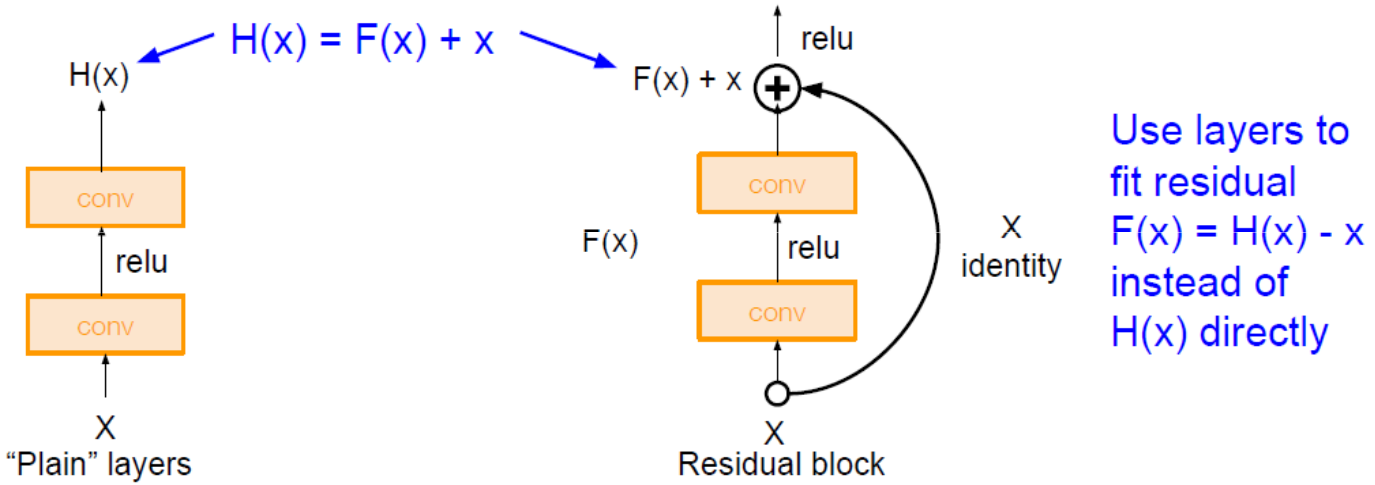
So H(x) of plain layers is replaced by new $H(x) = w^{[l+2]} a^{[l+2]} + b^{[l+2]} + a^{[l]}$

Slide Credit: Fei Li et. al.

ResNet

He et. al. 2015

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Referring the original residual function as H(x)
The residual part now fits a new function F(x)= H(x)-X
The original mapping recast into old H(x)+X
It is easier to learn residual F(x)

Slide Credit: Fei Fei Li et. al.

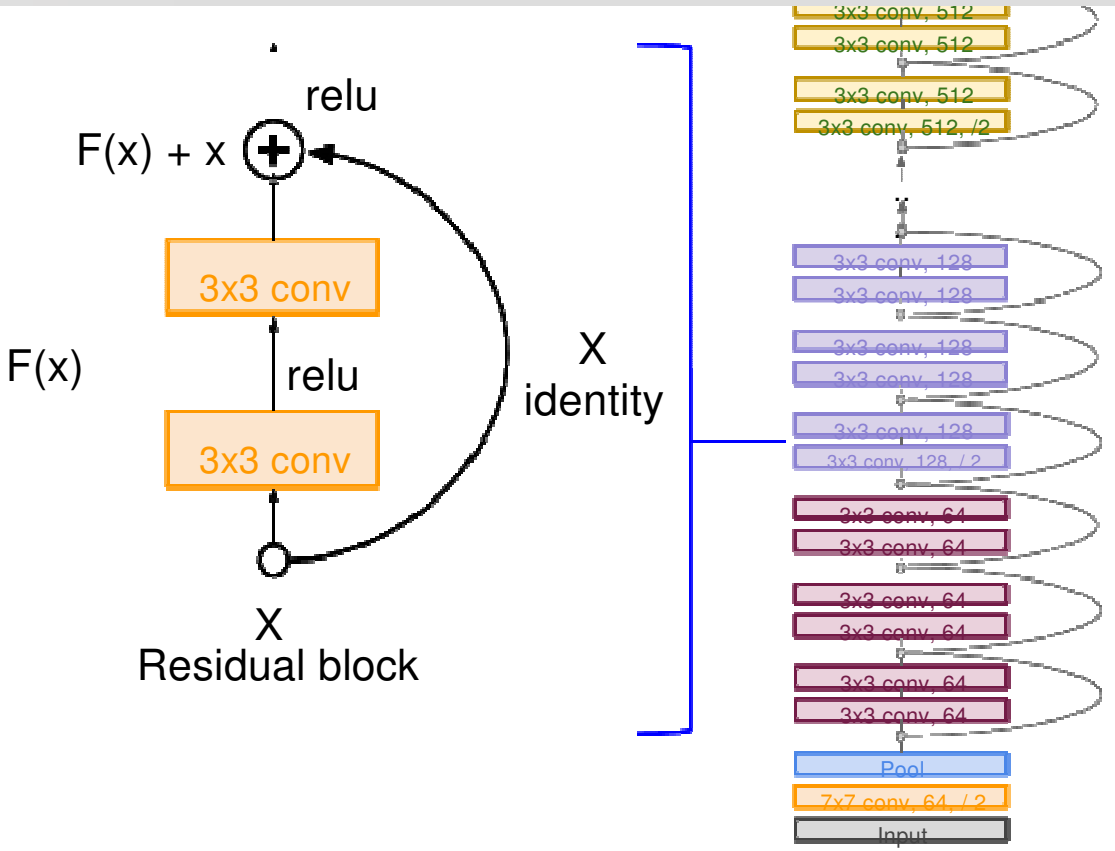
ResNet as a ConvNet

- Till now we talked about fully connected layers.
- The **ResNet** idea could easily expended into convolutional model.
- Other adaptations of this idea could be easily introduced to almost any kind of deep layered network.

ResNet Architecture

Full ResNet architecture:

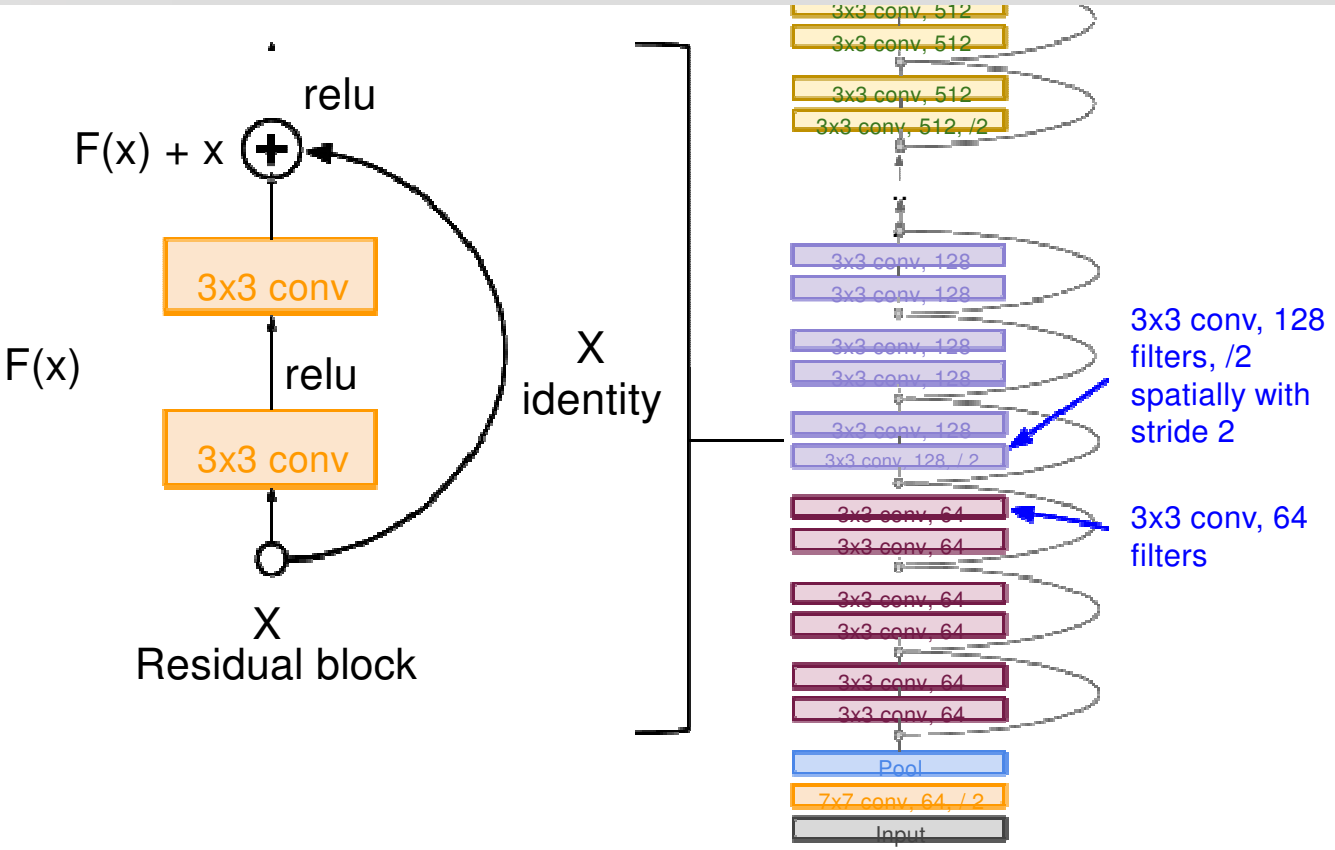
- Stack residual blocks
- Every residual block has two 3x3 conv layers



ResNet Architecture

Full ResNet architecture:

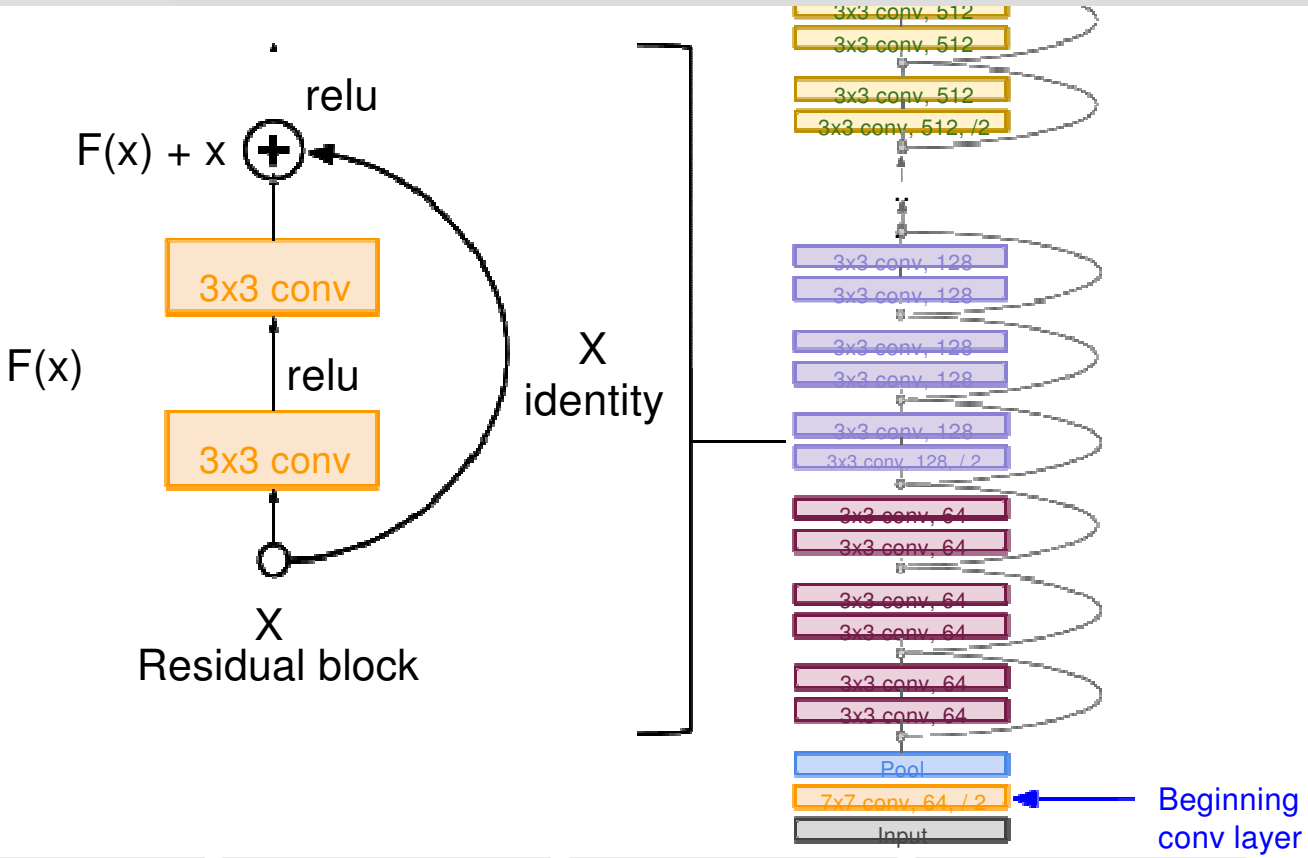
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



ResNet Architecture

Full ResNet architecture:

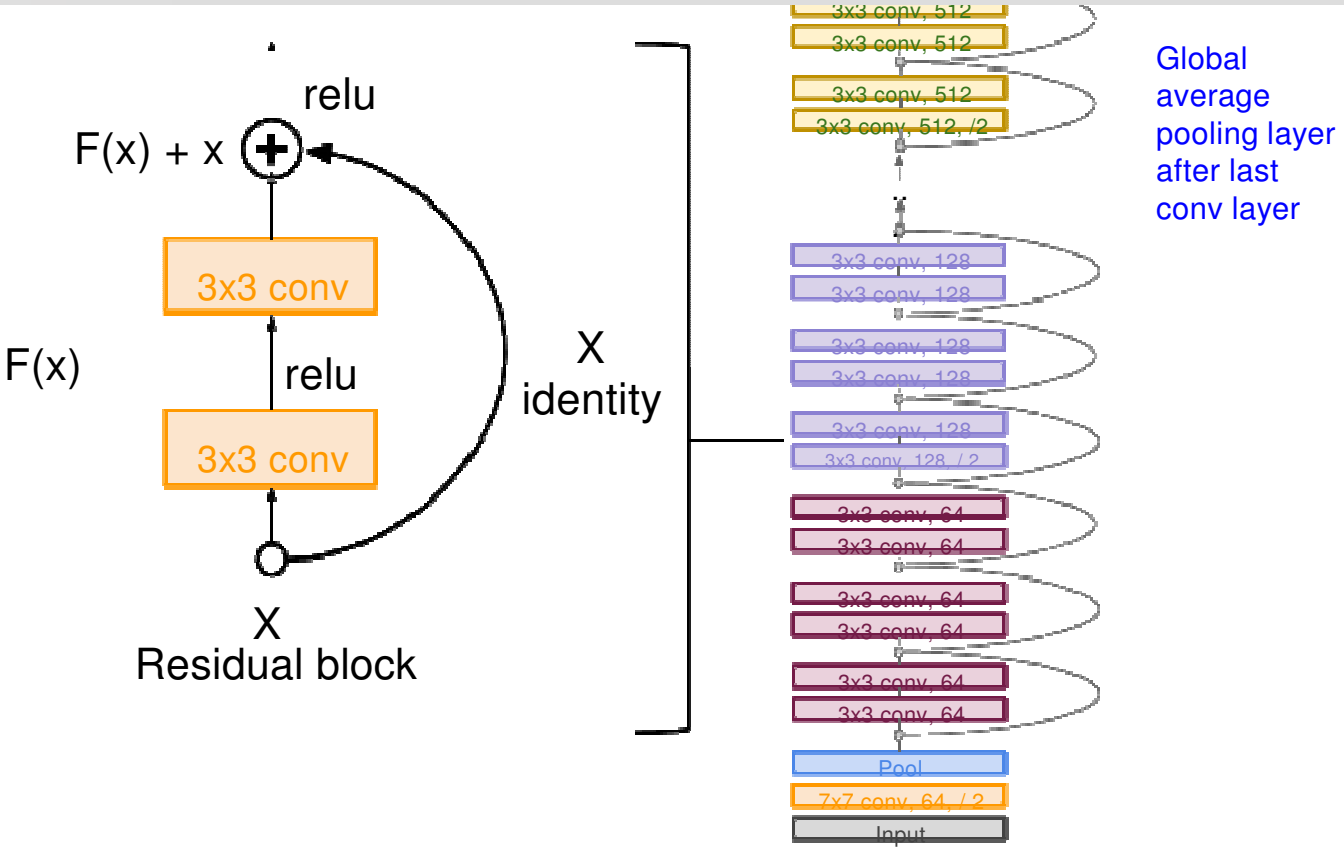
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



ResNet Architecture

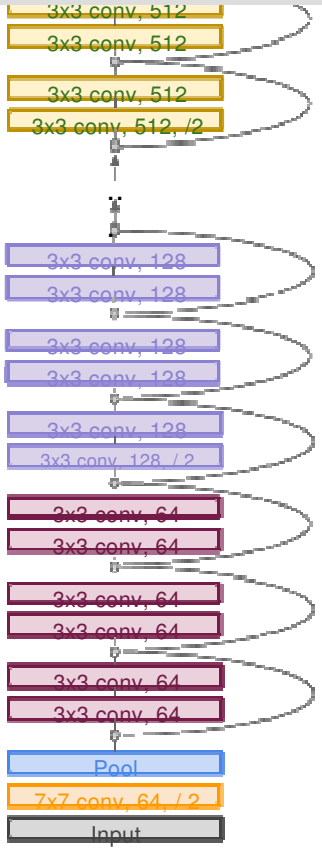
Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



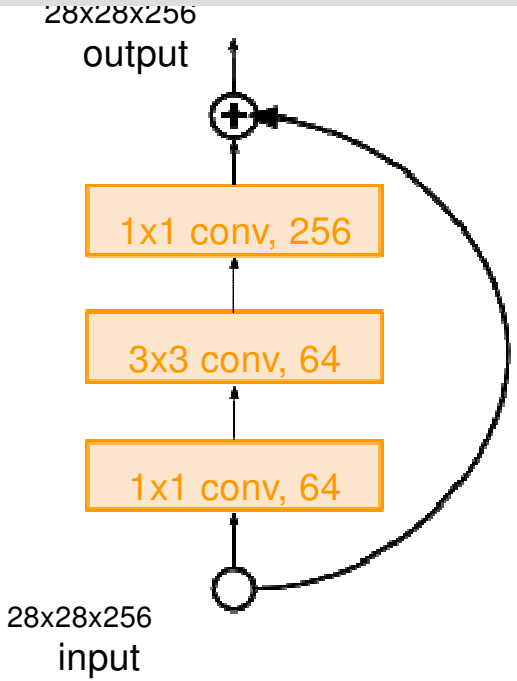
ResNet Architecture

Total depths of 34, 50, 101, or 152 layers for ImageNet



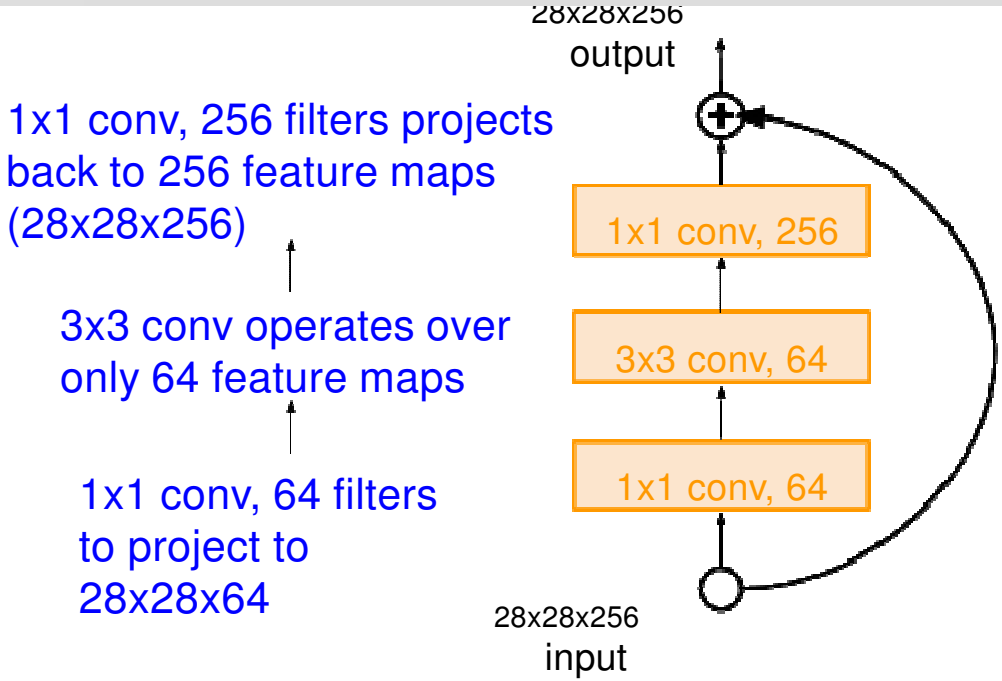
ResNet Architecture

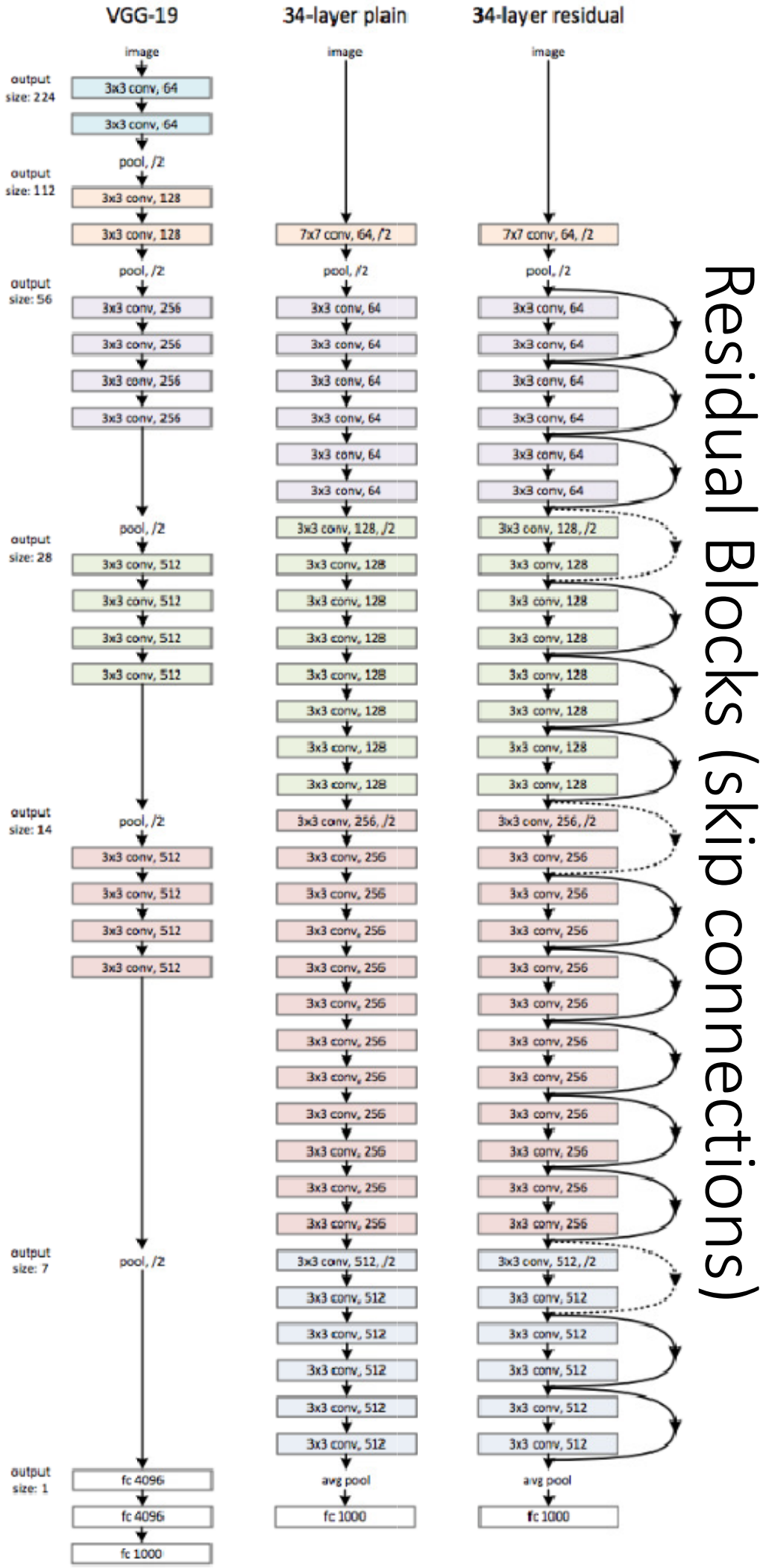
For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



ResNet Architecture

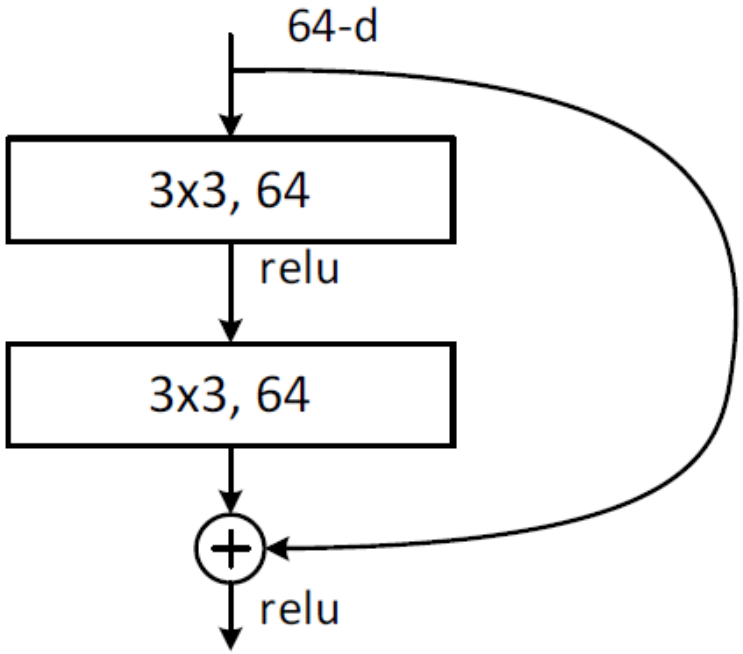
For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

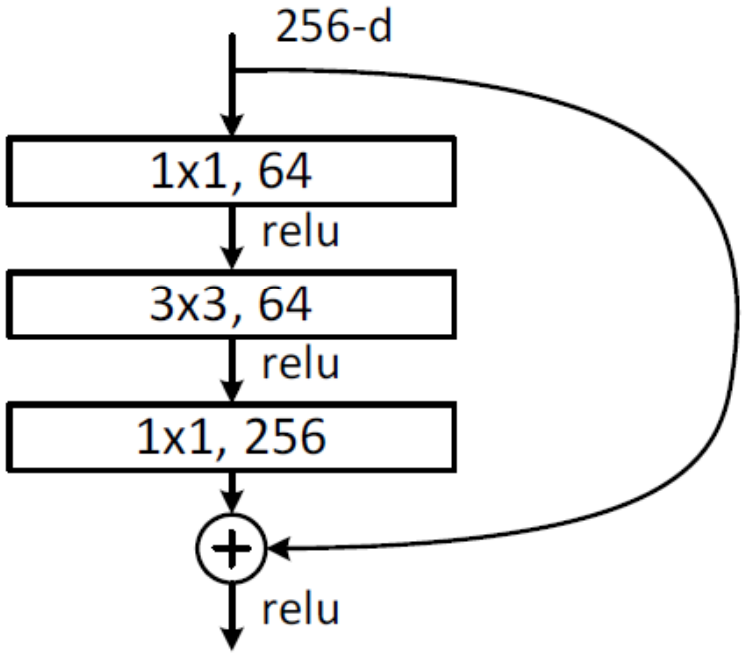




Residual Blocks (skip connections)

Deeper Bottleneck Architecture





Deeper Bottleneck Architecture (Cont.)

- Addresses high training time of very deep networks.
- Keeps the time complexity same as the two layered convolution
- Allows us to increase the number of layers
- allows the model to converge much faster.
- 152-layer ResNet has 11.3 billion FLOPS while VGG-16/19 nets has 15.3/19.6 billion FLOPS.

Why Do ResNets Work Well?

- Having a "regular" network that is very deep might actually hurt performance because of the vanishing and exploding gradients
- In most cases, ResNets will simply stop improving rather then decrease in performance
- $a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right) = g(w^{[l+1]}a^{[l+1]} + b^{[l]} + a^{[l]})$
- If the layer is not "useful", L2 regularization will bring it's parameters very close to zero, resulting in $a^{[l+2]} = g\left(a^{[l]}\right) = a^{[l]}$ (when using ReLU)

Why Do ResNets Work Well? (Cont)

- In theory ResNet is still identical to plain networks, but in practice due to the above the convergence is much faster.
- No additional training parameters introduced.
- No addition complexity introduced.

Training ResNet in practice

- Batch Normalization after every CONV layer.
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus.
- Mini-batch size 256.
- Weight decay of 1e-5.
- No dropout used.

Loss Function

- For measuring the loss of the model a combination of cross-entropy and softmax were used.
- The output of the cross-entropy was normalized using softmax function.

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

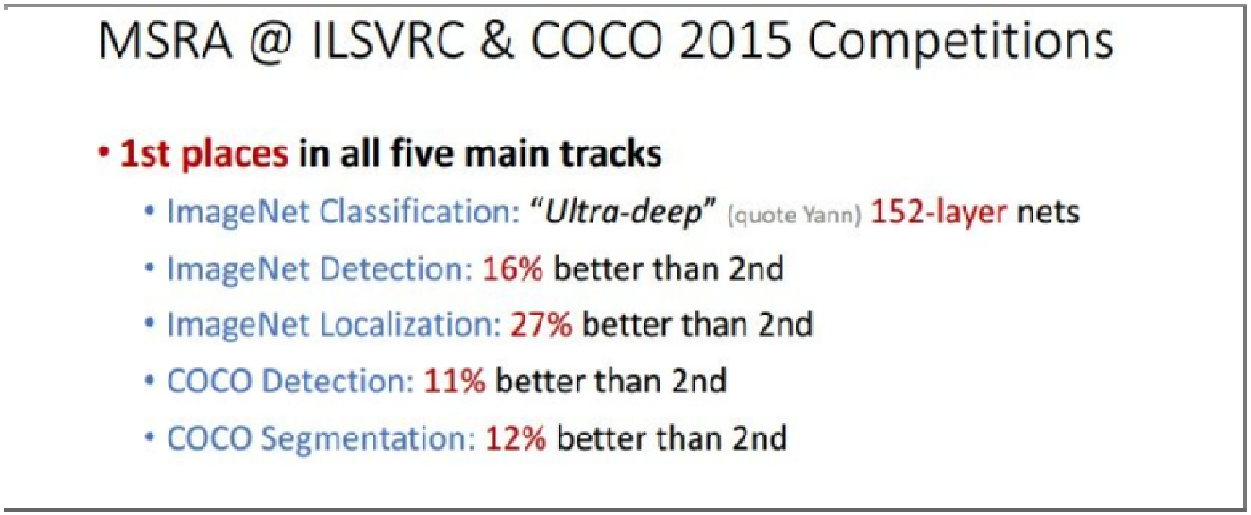
$$\sigma : \mathbb{R}^K \to \left\{ z \in \mathbb{R}^K \middle| z_i > 0, \sum_{i=1}^{K} z_i = 1 \right\}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Results

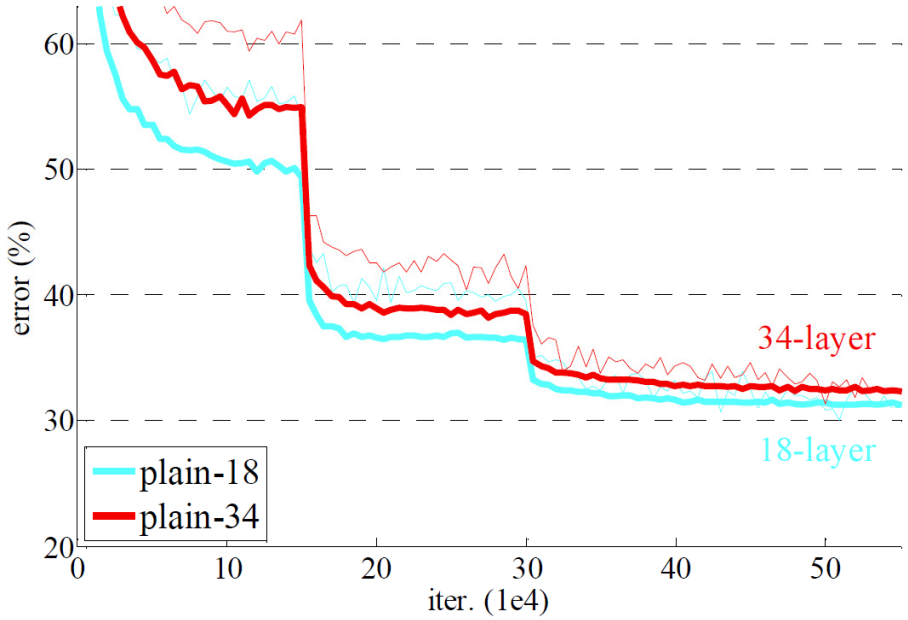
Experimental Results

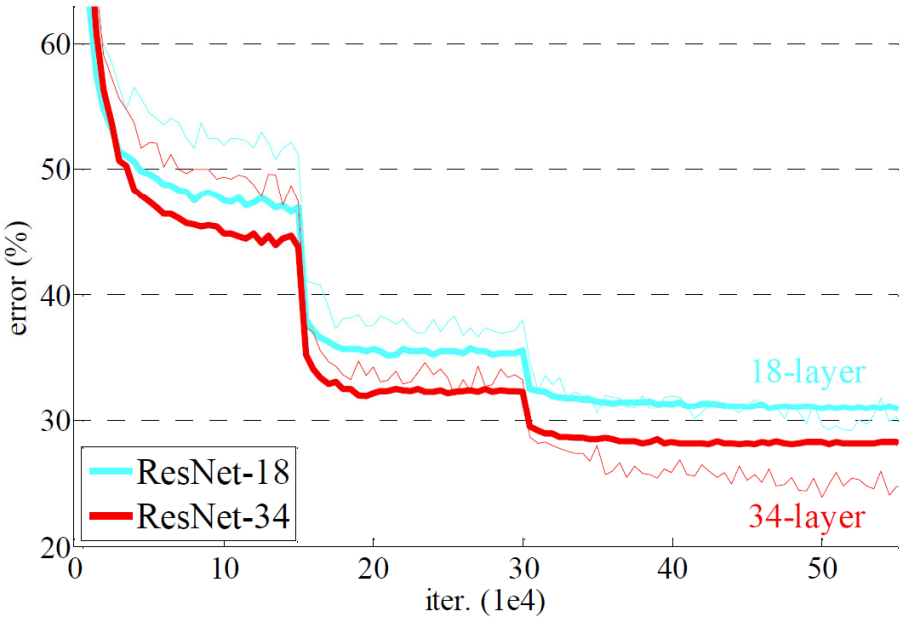
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions



ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

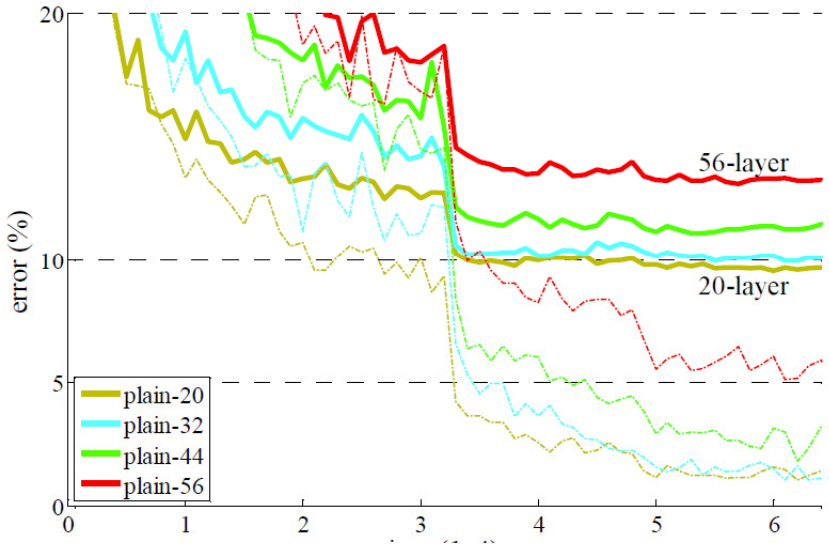
Comparing Plain to ResNet (18/34 Layers)

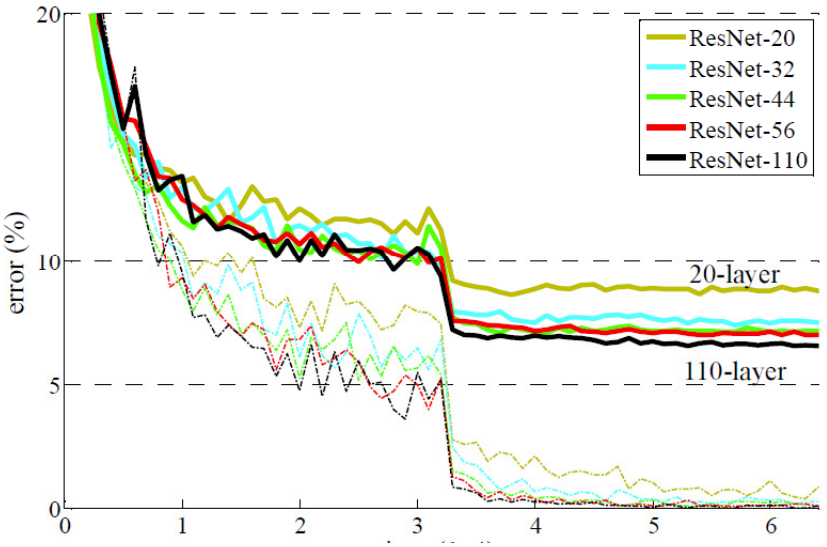




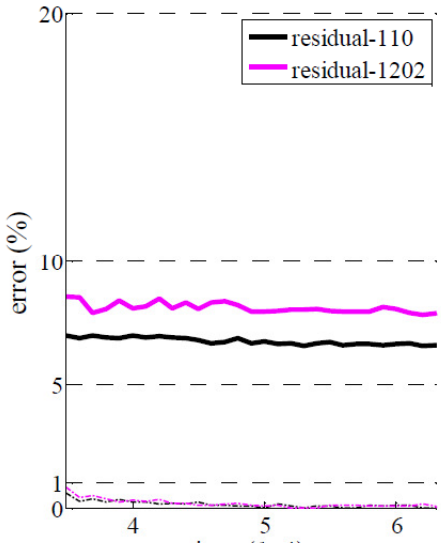
Comparing Plain to Deeper ResNet

Test Error:





Train Error:

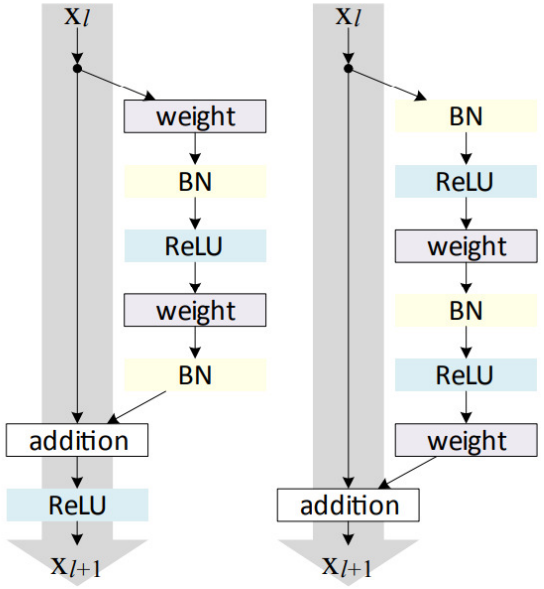


Intro | ResNet | Technical details | **Results** | ResNet 1000 | Comparison

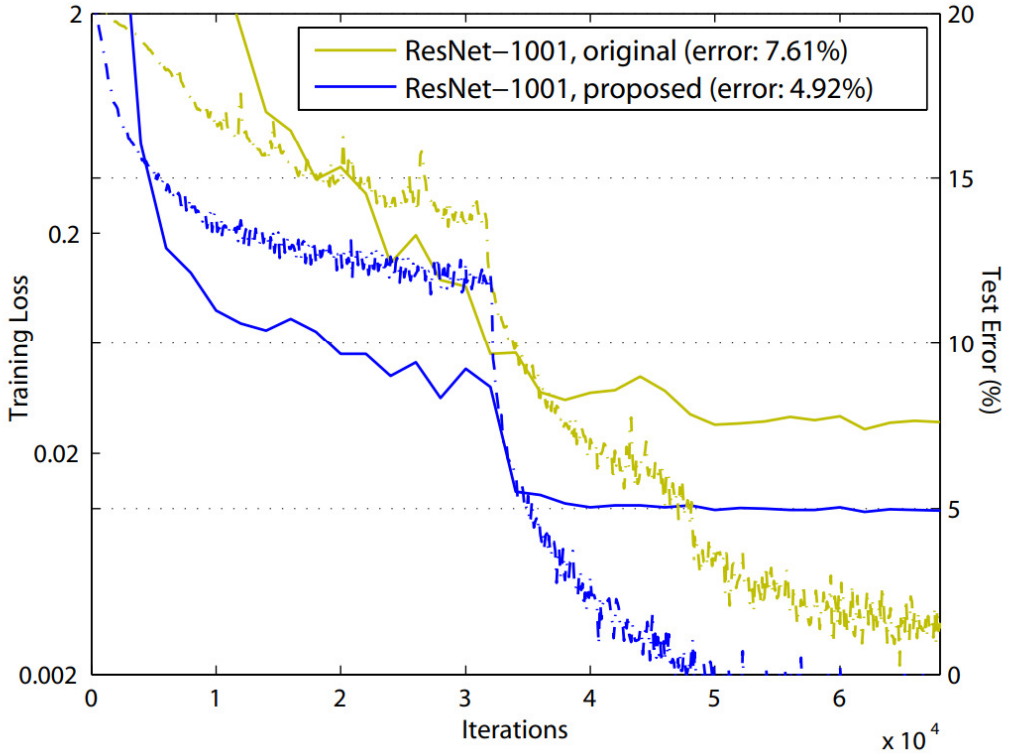
ResNet on More than 1000 Layers

- To farther improve learning of extremely deep **ResNet** “Identity Mappings in Deep Residual Networks Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun 2016” suggests to pass the input directly to the final residual layer, hence allowing the network to easily learn to pass the input as identity mapping both in forward and backward passes.

Identity Mappings in Deep Residual Networks



(a) original (b) proposed



Identity Mappings in Deep Residual Networks

- To construct identity mapping $f(y_l) = y_l$ (f is RELU, y_l the output of the residual with the identity) the new model refers to the (RELU and BN) as pre-activation of the weight layers, in contrast to the post-activation of the original model.
- Some nice properties for forward/backward passes:

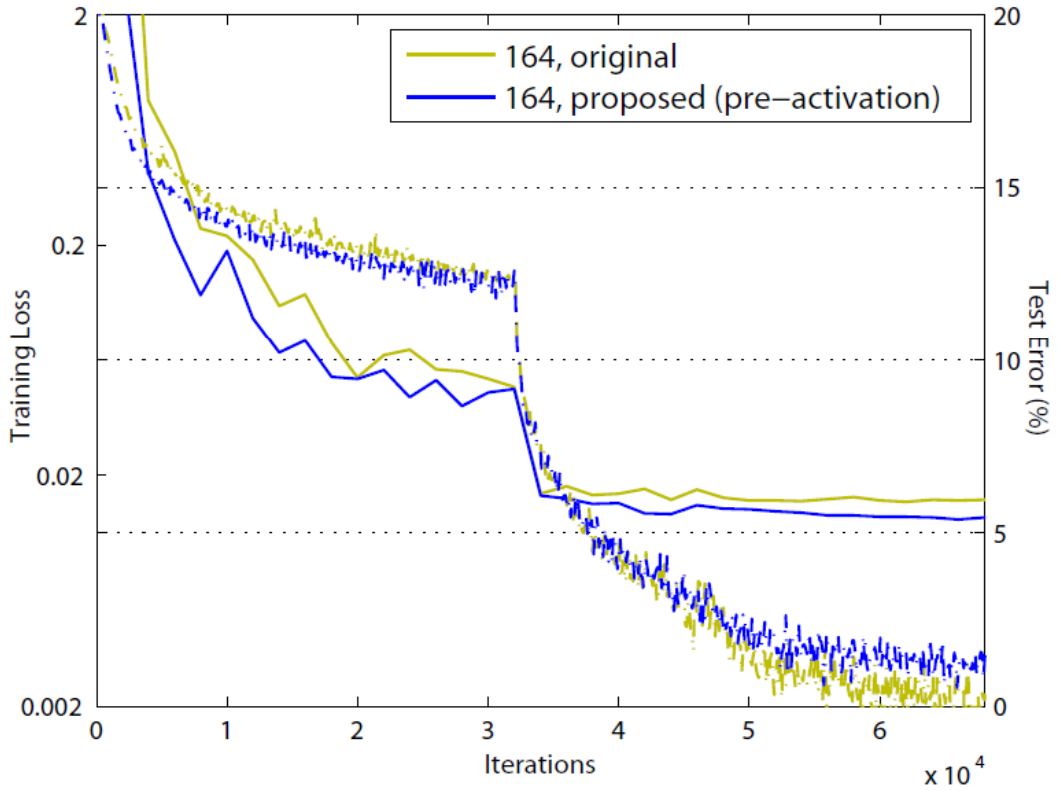
$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l),$$
$$\mathbf{x}_{l+1} = f(\mathbf{y}_l).$$

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

Improvement on CIFAR-10

- Another important improvement – using the Batch Normalization as pre-activation improves the regularization.
- This improvement leads to better performances for smaller networks as well.



Reduce Learning Time with Random Layer Drops

- Dropping layers during training, and using the full network in testing.
- Residual block are used as network's building block.
- During training, input flows through both the shortcut and the weights.
- Training: Each layer has a "survival probability" and is randomly dropped.
- Testing: all blocks are kept active.
- re-calibrated according to its survival probability during training.

To continue...