Deep Learning

Mayank Vatsa

Optimization Convergence

- Learning rate
  - The learning rate, α or η, indicates at which pace the weights get updated. It can be fixed or adaptively changed.
- Adaptive learning rate
  - Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution.

Gradient Descent

- Gradient descent is an iterative algorithm, that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point of that function
  - New = old + gradient * learning rate

Stochastic Gradient Descent

- Motivation
  - 10,000 data points and 10 features
  - compute the derivative with respect to each of the features
    - 10000 * 10 = 100,000 computations
    - If 1000 epochs, then
    - 100,000 * 1000 = 100000000 computations

gradient descent is slow on huge data

Stochastic Gradient Descent

- "Stochastic", in plain terms means "random"
  - sample a small number of data points
  - "mini-batch" gradient descent

Adding Momentum

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment

learning rate

weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor

weight increment, previous iteration

RMSProp

For each Parameter w^j

(*j subscript dropped for clarity*)

$$\nu_t = \rho \nu_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta \omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta \omega_t$$

η : *Initial Learning rate*

ν_t : *Exponential Average of squares of gradients*

g_t : *Gradient at time t along* ω^j

ADAM

- "... the name Adam is derived from adaptive moment estimation"

"Adam: A Method for Stochastic Optimization"

ADAM

For each Parameter w^j

(*j subscript dropped for clarity*)

$$\nu_t = \beta_1 * \nu_{t-1} + (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*

g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

ADAM

$$For\ each\ Parameter\ w^j$$

(j subscript dropped for clarity)

$$\nu_t = \beta_1 * \nu_{t-1} + (1 - \beta_1) * g_t$$

Exponential Weighted Averages for past gradients

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*

g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

ADAM

$$For\ each\ Parameter\ w^j$$

(j subscript dropped for clarity)

$$\nu_t = \beta_1 * \nu_{t-1} + (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * g_t^2$$

Exponential Weighted Averages for past squared gradients

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$

$g_t : Gradient\ at\ time\ t\ along\ \omega^j$

$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$

$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$

$\beta_1, \beta_2 : Hyperparameters$

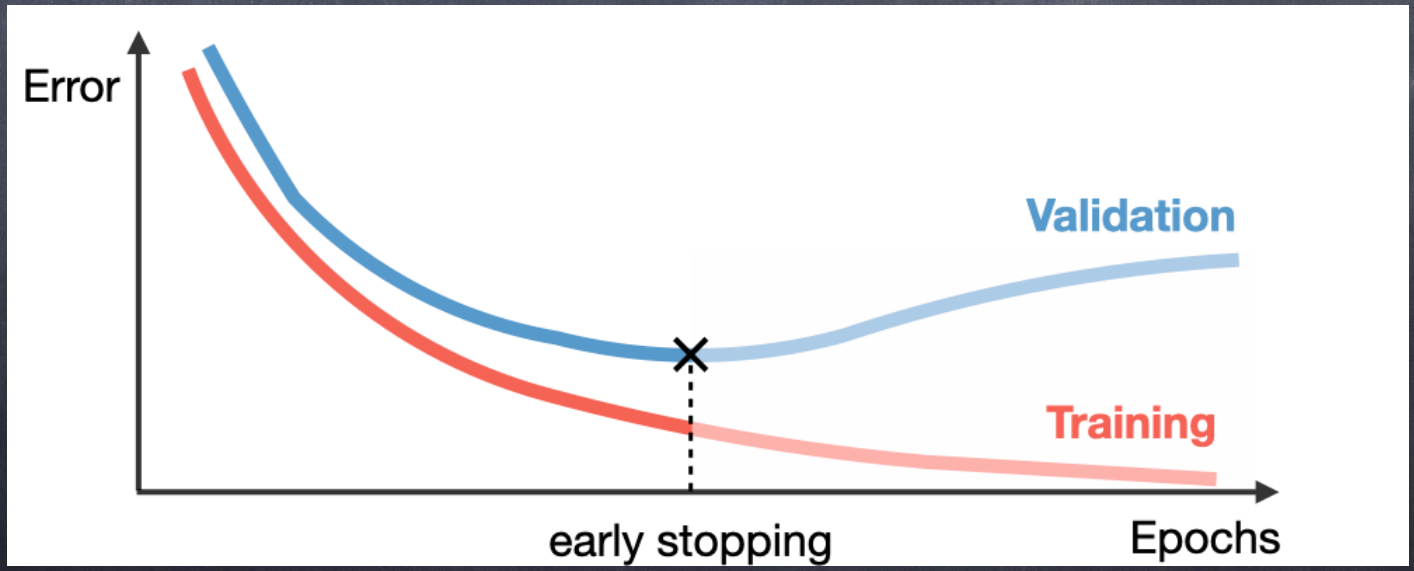
- eta. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- beta1. The exponential decay rate for the first moment estimates (e.g. 0.9).
- beta2. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
- epsilon. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

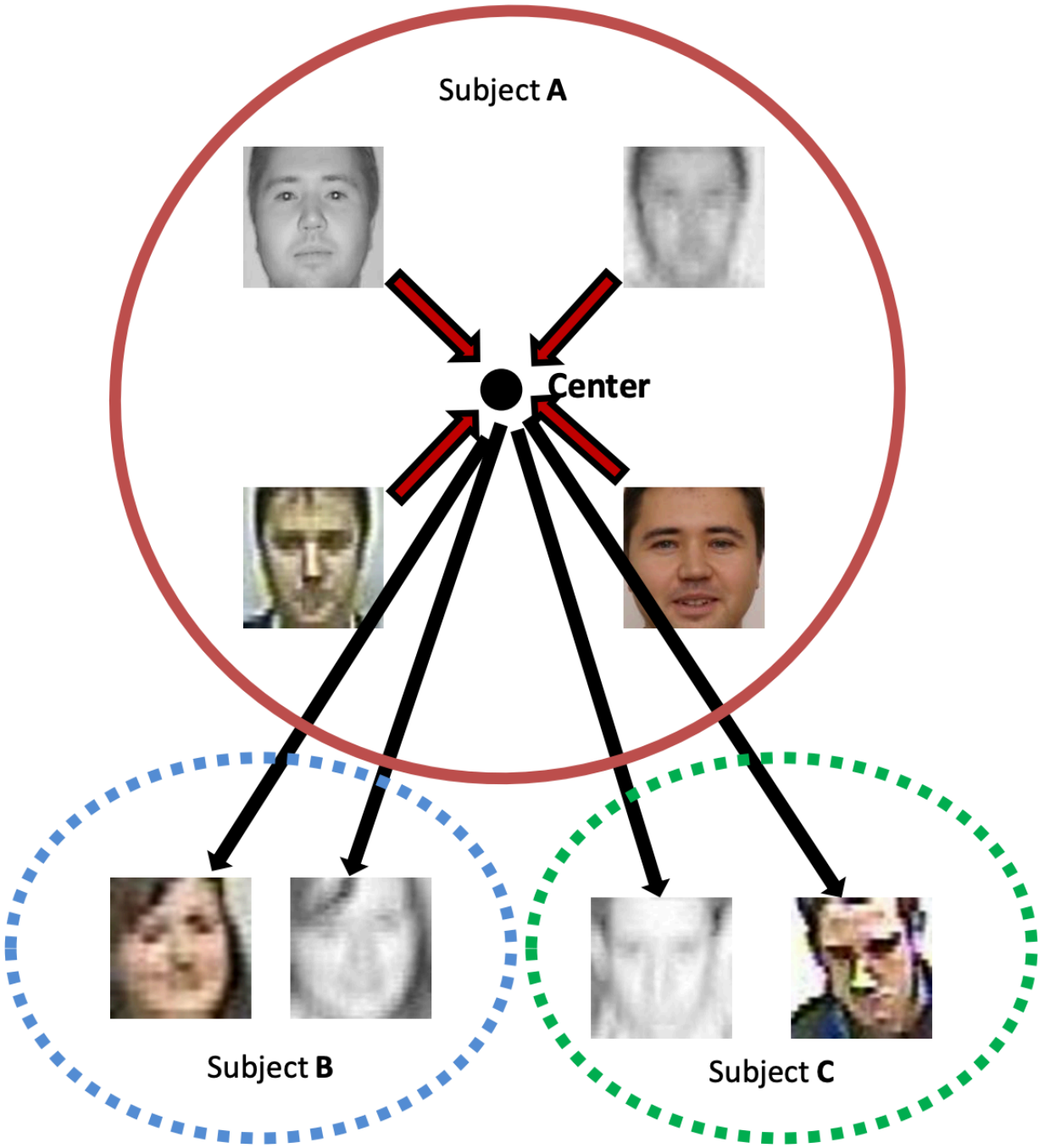
Many other optimizers

- Mostly Heuristics

Early Stopping

Early stopping This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.




Subject A
Center
Subject B
Subject C

There are many loss functions now

- Paper review assignment - write one page description on one loss function (other than discussed in the class).

ICCV, CVPR, ECCV, WACV, IEEE Transactions, IJCV, Pattern Recognition, CVIU

A big trick in the CNN World

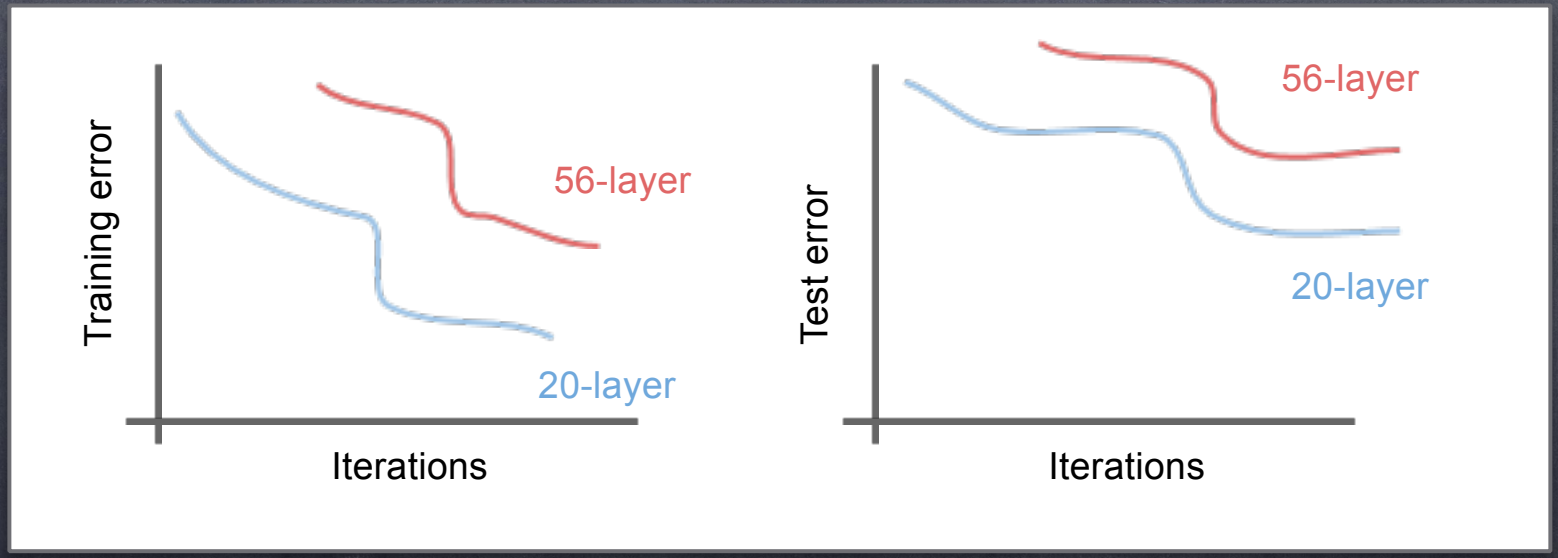
- Can you guess?

Residual Connections

- Let us understand Residual Connections and their strength

Deep CNN

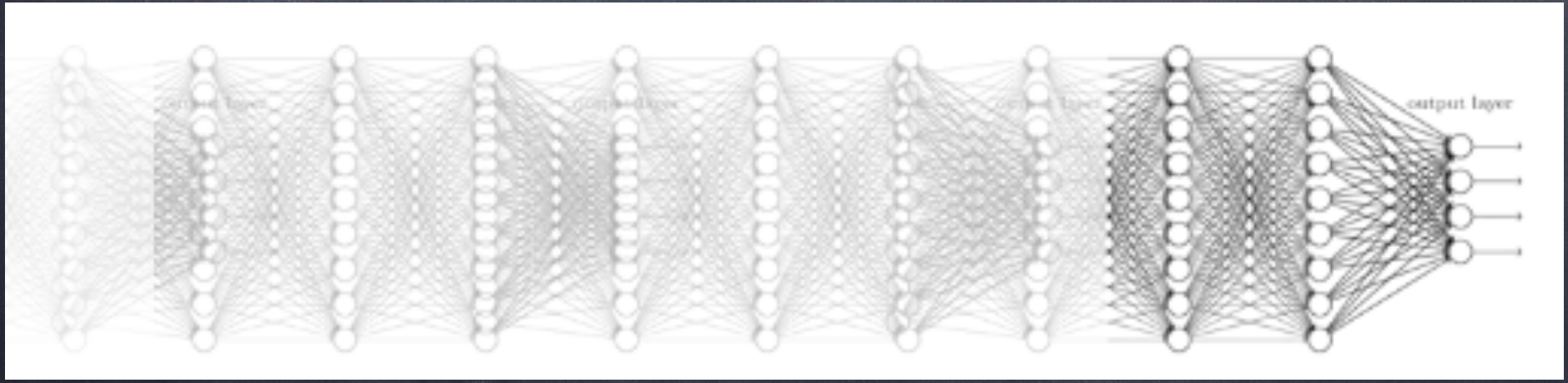
What happens when we continue stacking deeper layers on a "plain" CNN



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

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

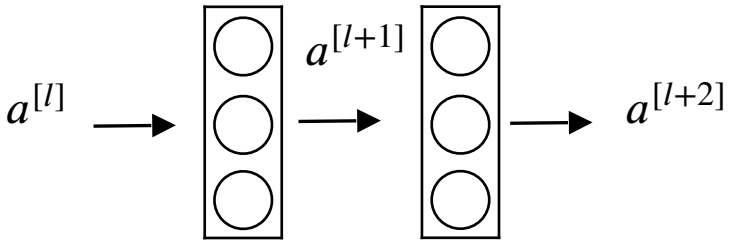
Recall Vanishing Gradients



Residual Block

- A novel concept of residual block was proposed to mitigate this problem

Understanding concept of Residual Block



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

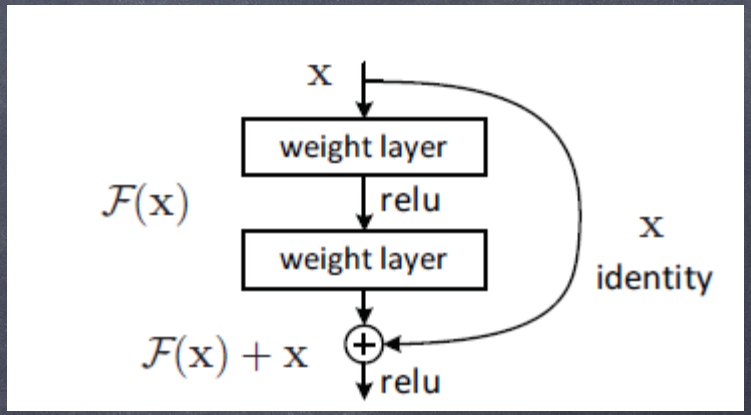
$$a^{[l+1]} = g(z^{[l+1]})$$
"relu"

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

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

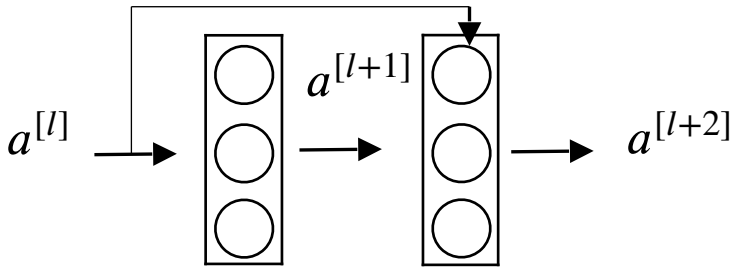
Skip Connection

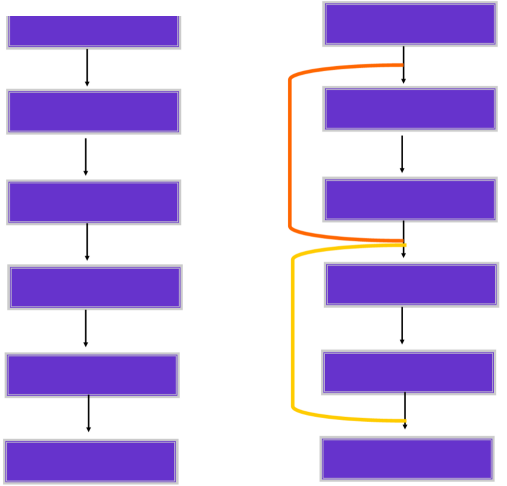
- Such connections are referred as skipped connections or shortcuts. In general similar models could skip over several layers.
- They refer to residual part of the network as a unit with input and output.
- Such residual part receives the input as an amplifier to its output - The dimensions usually are the same.



Understanding the concept

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}\left(\frac{\partial F}{\partial x} + 1\right) = \frac{\partial L}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial L}{\partial H}$$





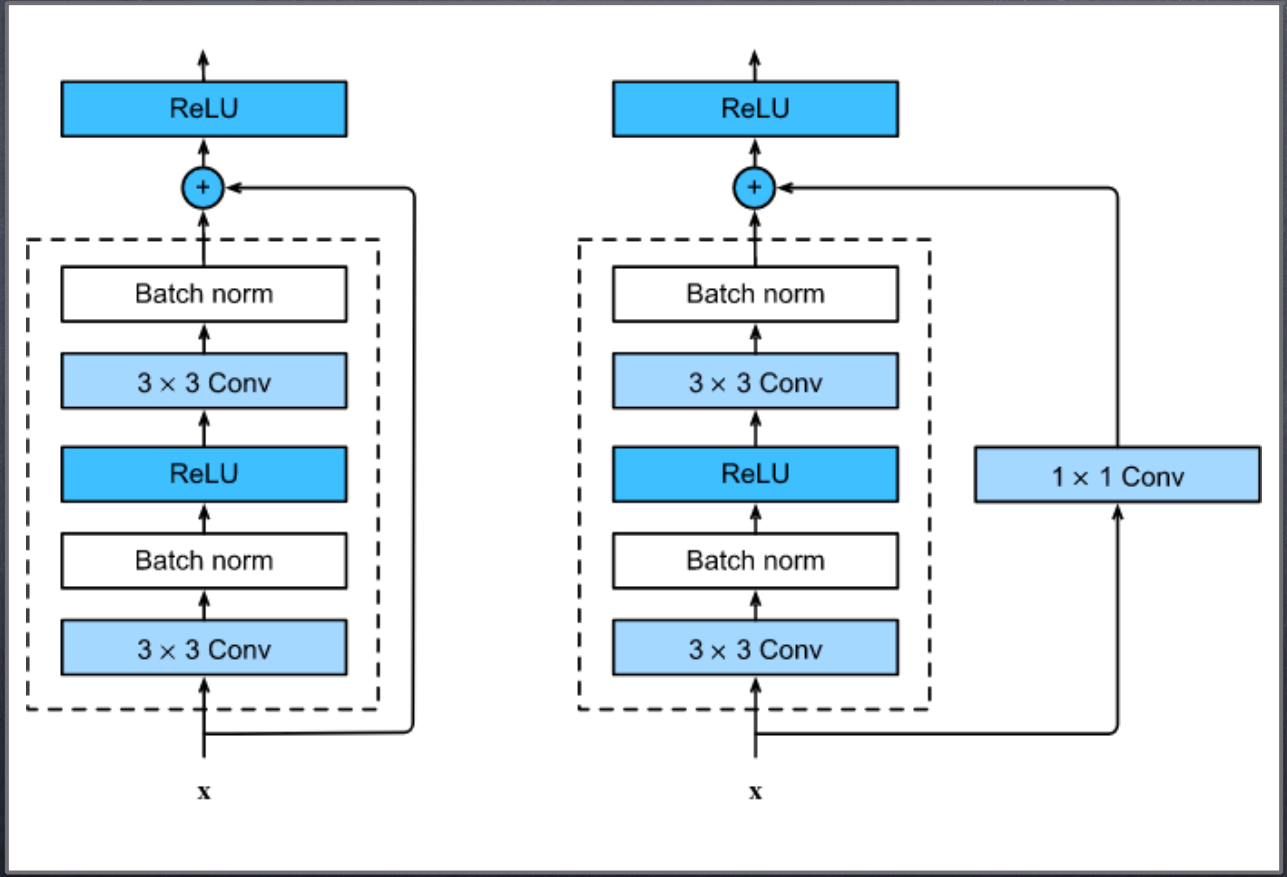
$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$
"linear"

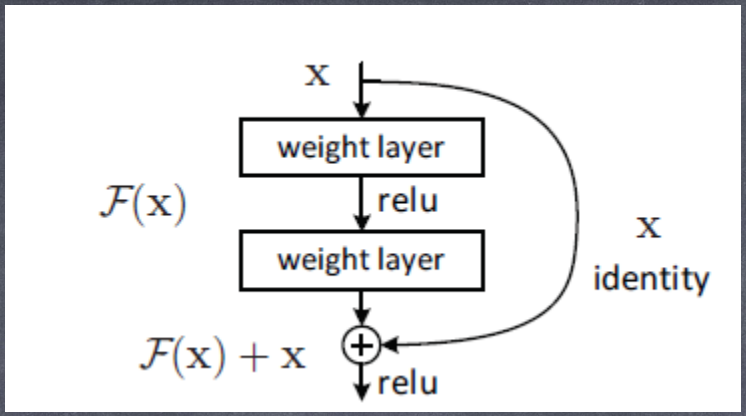
$$a^{[l+1]} = g(z^{[l+1]})$$
"relu"

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

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

Same concept is extended with Convolution

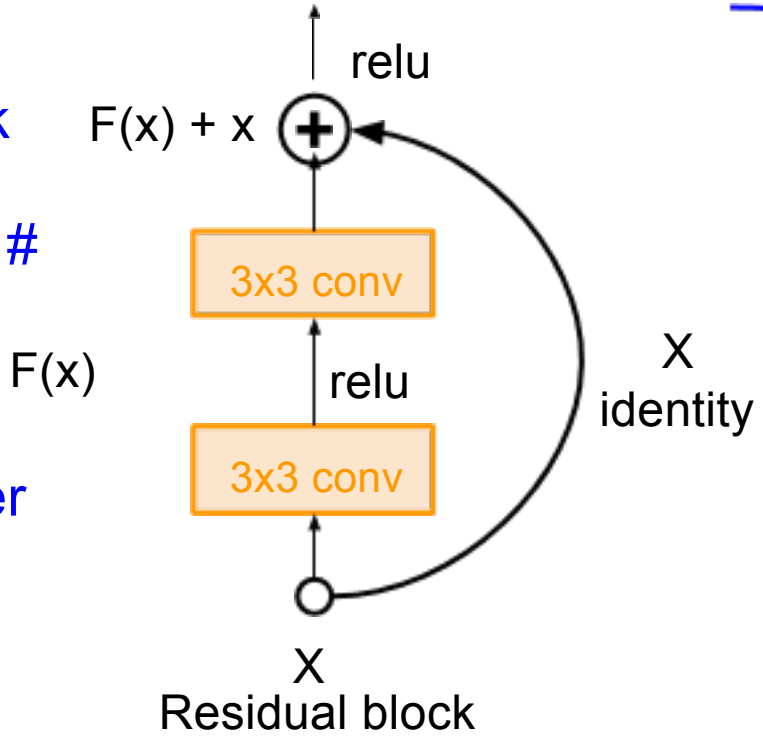


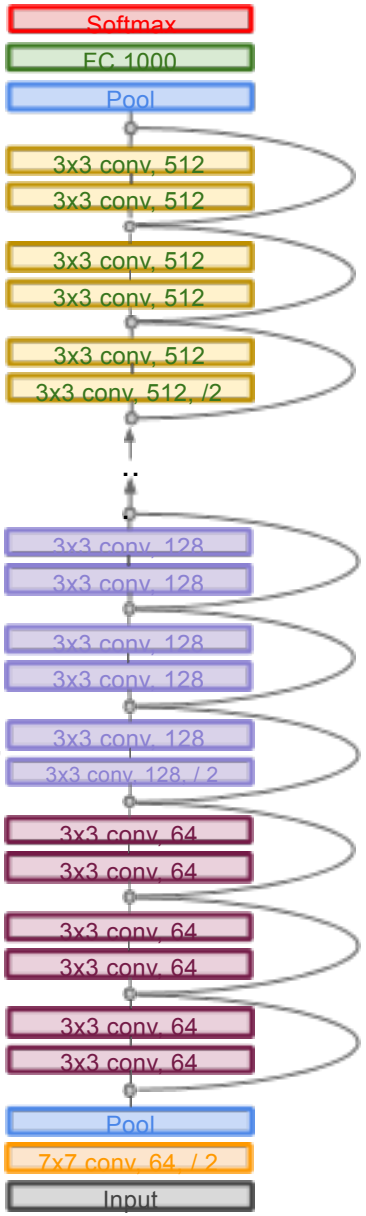


- connections are referred as skipped connections or shortcuts. In general similar models could skip over several layers.
- They refer to residual part of the network as a unit with input and output.
- Such residual part receives the input as an amplifier to its output - The dimensions usually are the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

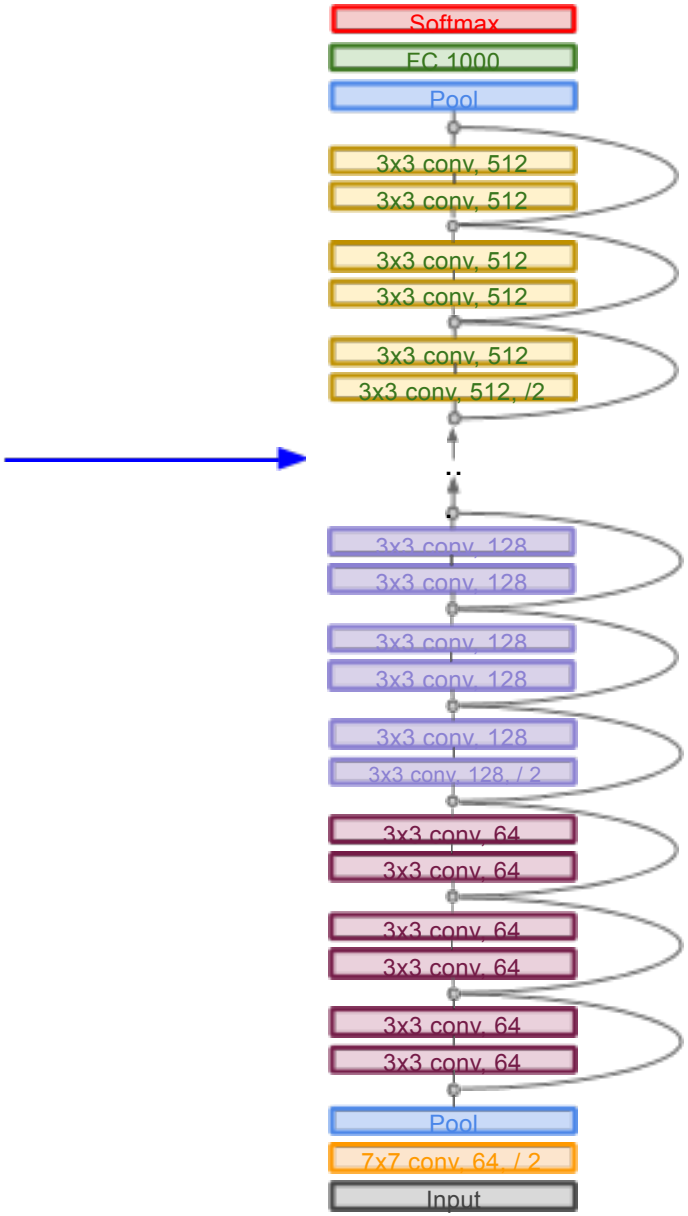
Full ResNet

- Stack residual
- Every residual block two 3x3 conv
- Periodically, double # filters and spatially using (/2 in each
- Additional conv layer the
- No FC layers at the (only FC 1000 to classes

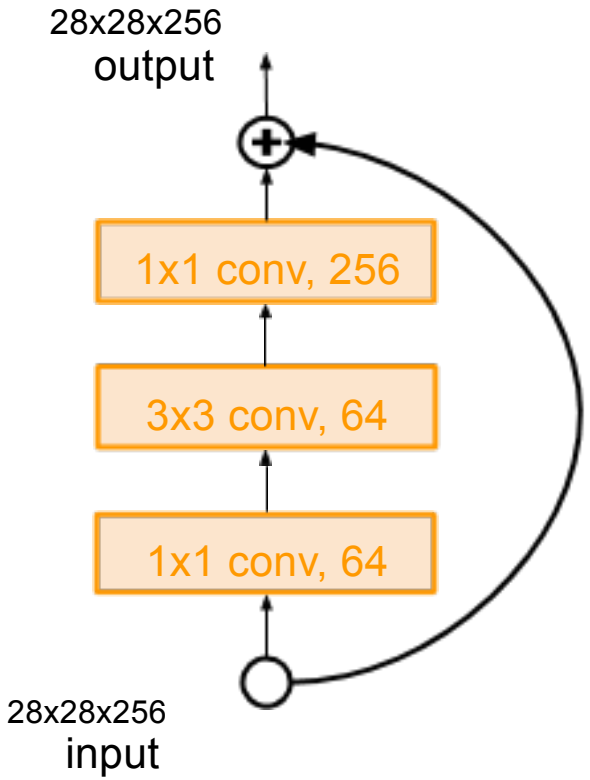




Total depths of 34, 50, 101, or 152 layers for ImageNet



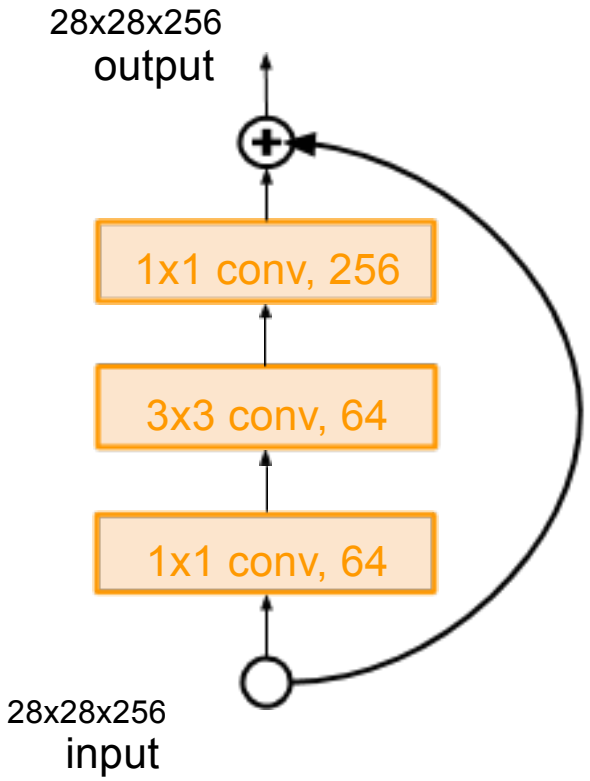
For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

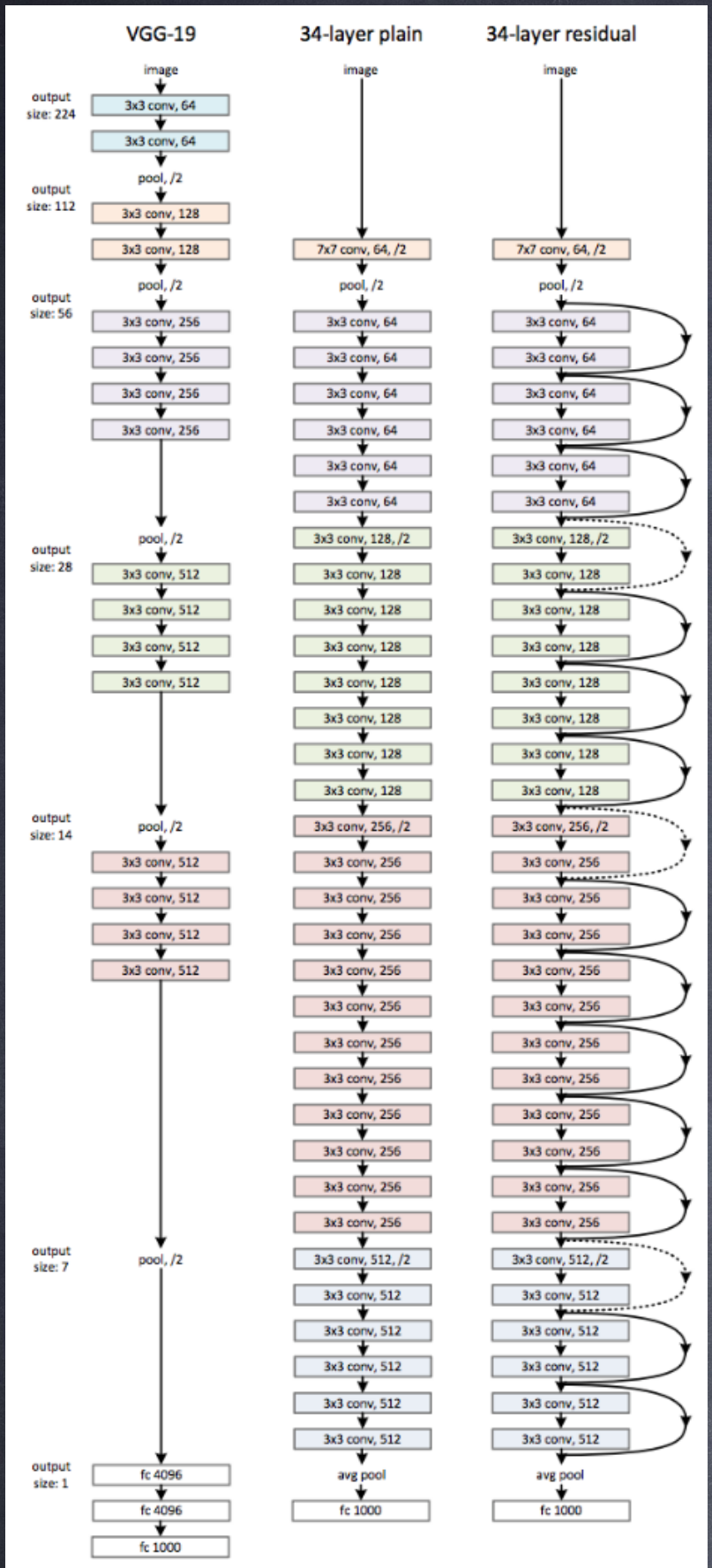


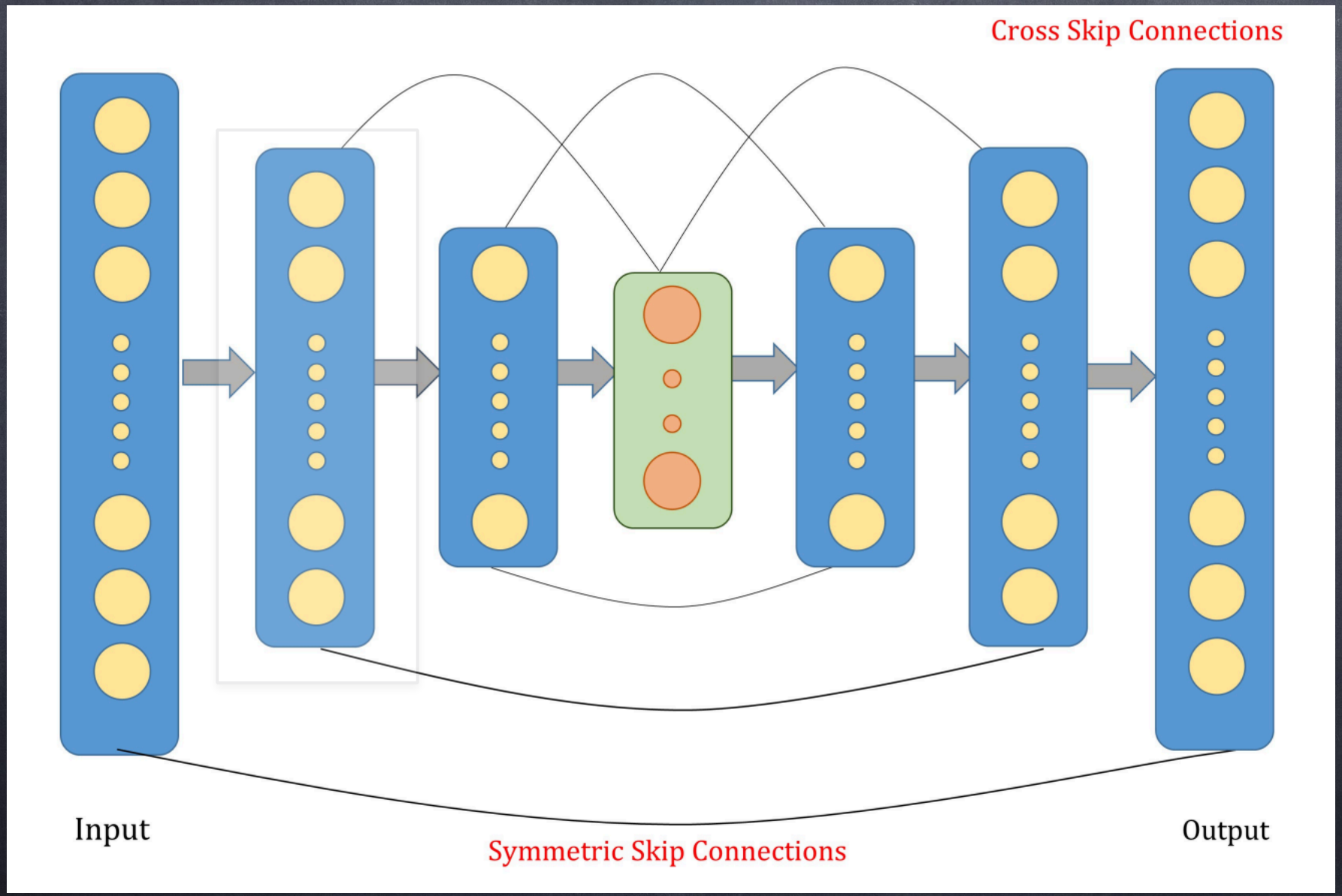
1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

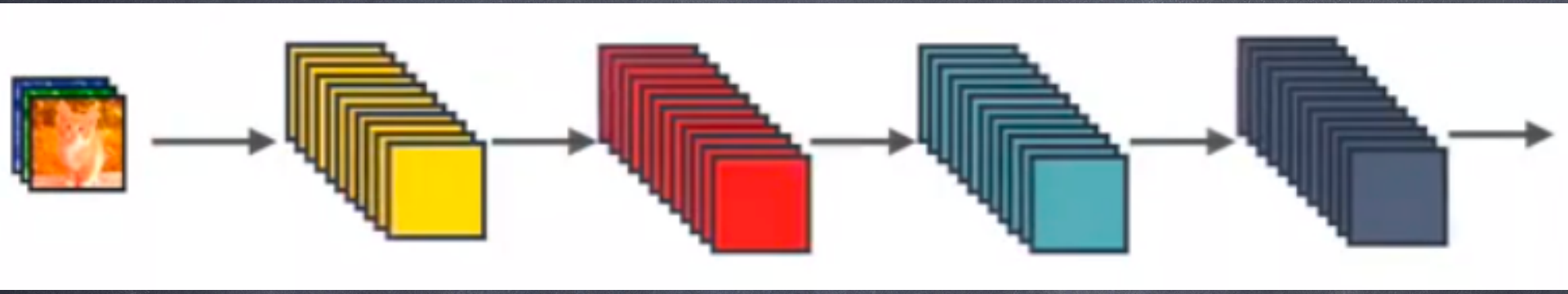
1x1 conv, 64 filters to project to 28x28x64

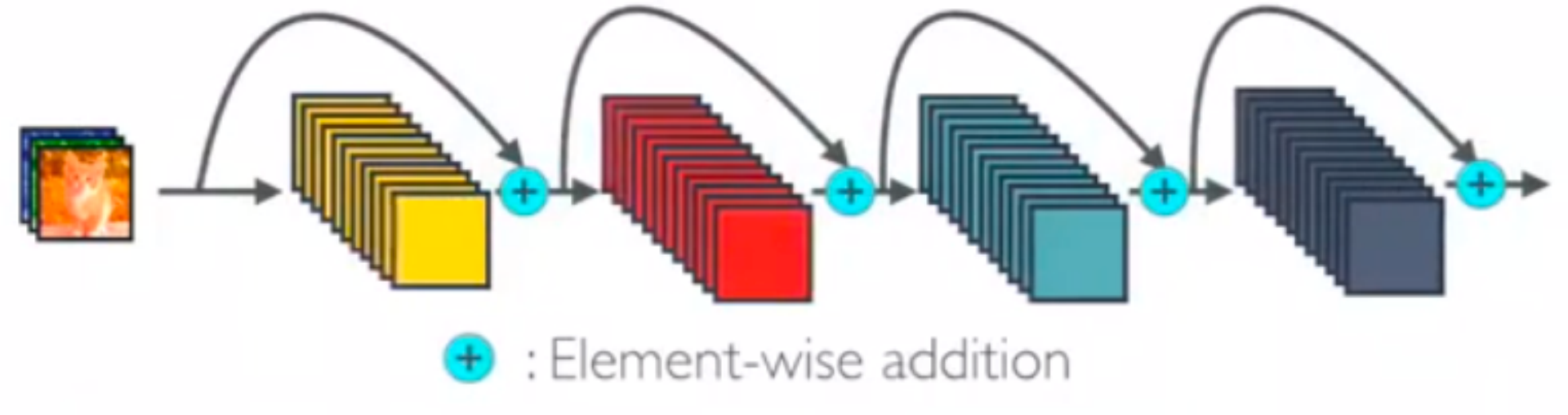



VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

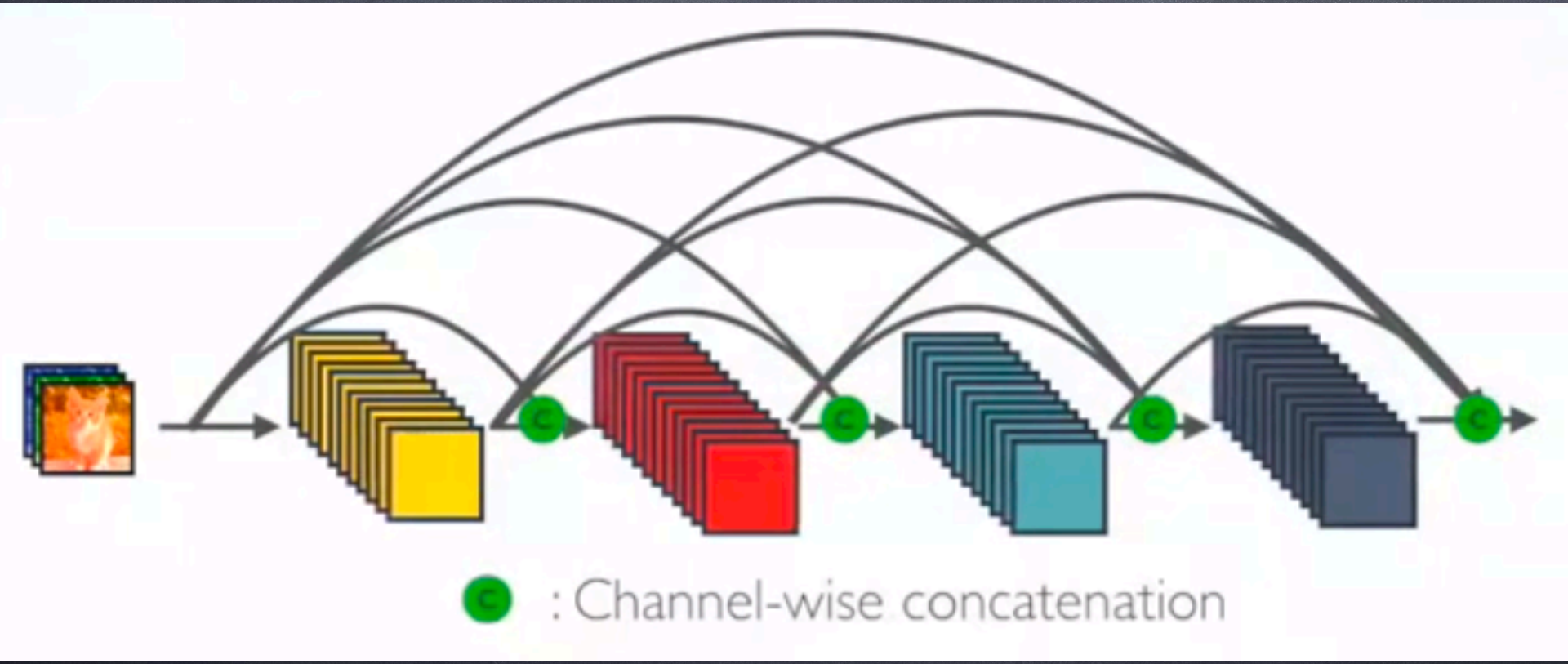

Cross Skip Connections
Input
Symmetric Skip Connections
Output

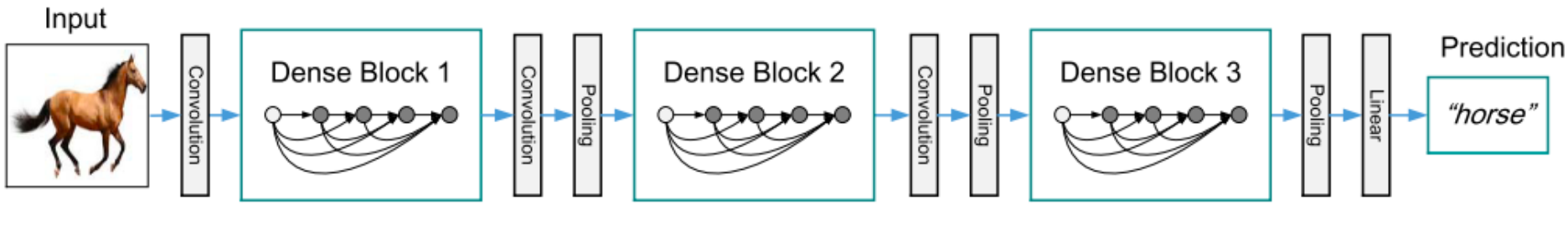
DenseNet





DenseNet




Input
Convolution
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction
“horse”

Some popular networks

- LeNet
- AlexNet
- VGG
- GoogleNet
- ResNet
- DenseNet