

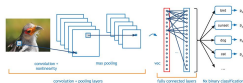
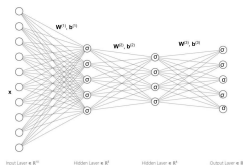
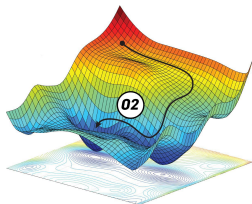


# Deep Learning for Computer Vision

Dr. Konda Reddy Mopuri  
Mehta Family School of Data Science and Artificial Intelligence  
IIT Guwahati  
Aug-Dec 2022

# So far in the class..

- Scoring function, loss function, gradient descent
- Artificial Neurons and Multi-Layered Perceptron
- CNN building blocks and a case-study



# Overview of different CNN architectures

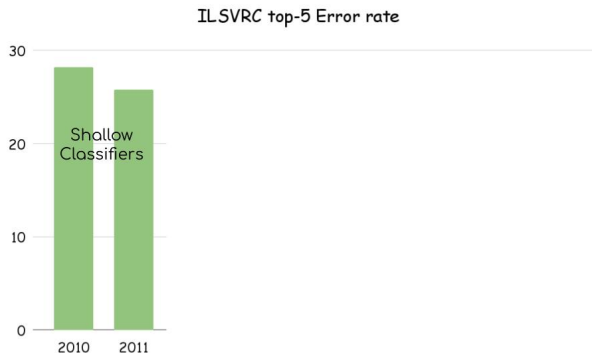


- We will ground the evolution on ILSVRC

# Overview of different CNN architectures



- We will ground the evolution on ILSVRC



# AlexNet (2012)



- ① 8-layer CNN: 5 Conv layers, 3 FC layers
- ②  $227 \times 227$  input
- ③ Max pooling, ReLU nonlinearity, LRN (not used anymore now)

# AlexNet (2012)

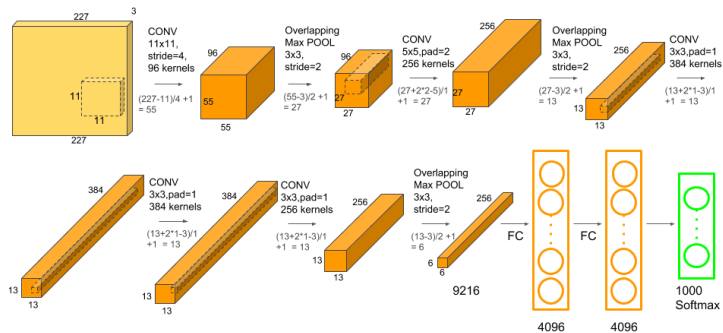


Figure credits: [neurohive.io](https://neurohive.io)

# AlexNet (2012)



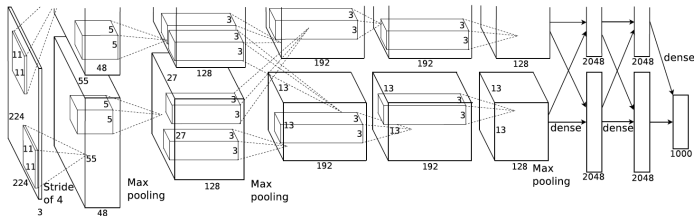
- ① Implemented on GTX 580 GPUs (2 of them; 3GB of Memory each)

---

Figure from AlexNet paper by Kryzhevsky et al.

# AlexNet (2012)

- ① Implemented on GTX 580 GPUs (2 of them; 3GB of Memory each)

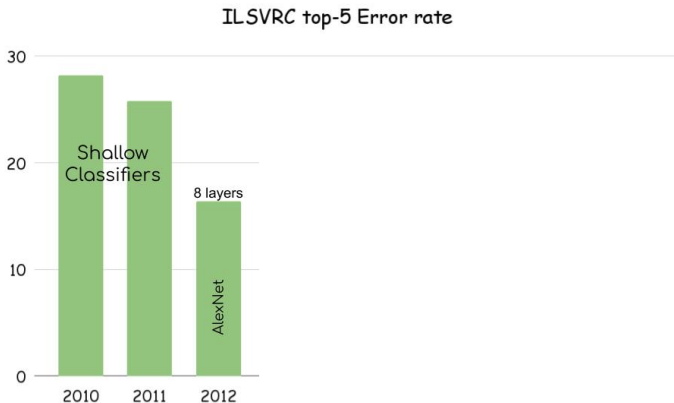


②

Figure from AlexNet paper by Kryzhevsky et al.



# AlexNet (2012)



# ZFNet (2013)



- ① A more worked-out AlexNet

# ZFNet (2013)



- ① A more worked-out AlexNet
- ② More trails on the AlexNet architecture that resulted in less error
  - $(11 \times 11 \text{ stride } 4) \rightarrow (7 \times 7 \text{ stride } 2)$
  - Conv 3, 4, and 5 (384, 384, 256)  $\rightarrow$  (512, 1024, and 512)

# ZFNet (2013)

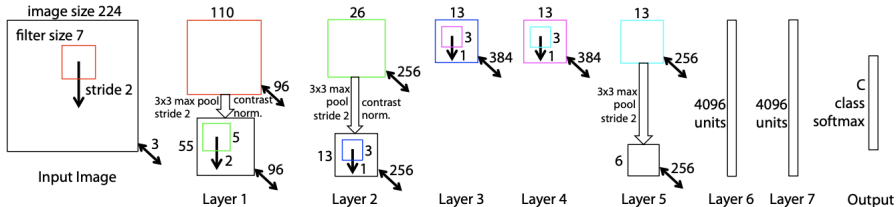
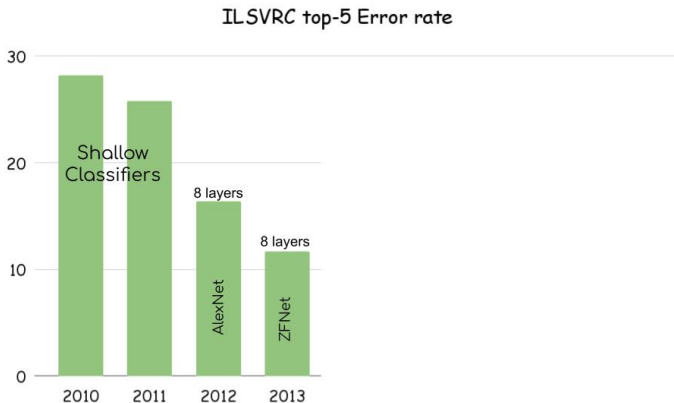


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ( $6 \cdot 6 \cdot 256 = 9216$  dimensions). The final layer is a  $C$ -way softmax function,  $C$  being the number of classes. All filters and feature maps are square in shape.

Figure from Zeiler and Fergus, ECCV 2014

# ZFNet (2013)



# VGG (2014)



- ① First architecture to have a principled design

# VGG (2014)

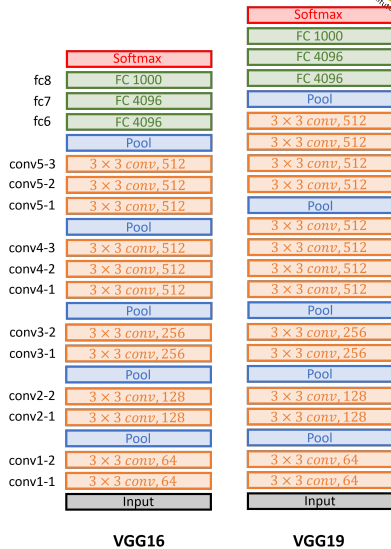


- ① First architecture to have a principled design
- ②
  - All conv:  $3 \times 3$ , stride:1, pad:1
  - All max pool:  $2 \times 2$ , stride:2
  - After pooling, double the channels

# VGG (2014)



## ① 5 Conv stages

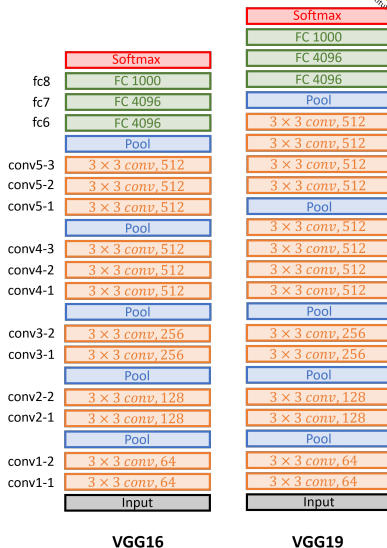




# VGG (2014)



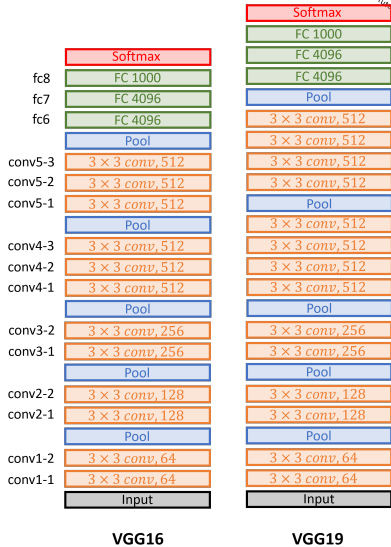
- ① 5 Conv stages
- ② (initially) Conv-Conv-Pool



# VGG (2014)



- ① 5 Conv stages
- ② (initially) Conv-Conv-Pool
- ③ (later) Conv-Conv-Conv-Pool (VGG19 has one more Conv)



# VGG (2014)



## ① Why Only $3 \times 3$ Convs?

# VGG (2014)



- ① Why Only  $3 \times 3$  Convs?
- ② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$

# VGG (2014)



- ① Why Only  $3 \times 3$  Convs?
- ② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$ 
  - Parameters:  
 $C \times C \times 5 \times 5 = 25C^2$

# VGG (2014)



- ① Why Only  $3 \times 3$  Convs?
- ② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$ 
  - Parameters:  
 $C \times C \times 5 \times 5 = 25C^2$
  - Flops:  
 $C \times H \times W \times C \times 5 \times 5 = 25C^2HW$

# VGG (2014)



① Why Only  $3 \times 3$  Convs?

② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$

- Parameters:

$$C \times C \times 5 \times 5 = 25C^2$$

- Flops:

$$C \times H \times W \times C \times 5 \times 5 = 25C^2HW$$

① **Case-2:**  $\text{Conv}(3 \times 3, C \rightarrow C)$   
and  $\text{Conv}(3 \times 3, C \rightarrow C)$

- ① Why Only  $3 \times 3$  Convs?
  - ② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$ 
    - Parameters:  
 $C \times C \times 5 \times 5 = 25C^2$
    - Flops:  
 $C \times H \times W \times C \times 5 \times 5 = 25C^2HW$
  - ① **Case-2:**  $\text{Conv}(3 \times 3, C \rightarrow C)$  and  $\text{Conv}(3 \times 3, C \rightarrow C)$ 
    - Parameters:  
 $2 \times C \times C \times 3 \times 3 = 18C^2$



① Why Only  $3 \times 3$  Convs?

② **Case-1:**  $\text{Conv}(5 \times 5, C \rightarrow C)$

- Parameters:

$$C \times C \times 5 \times 5 = 25C^2$$

- Flops:

$$C \times H \times W \times C \times 5 \times 5 = 25C^2HW$$

① **Case-2:**  $\text{Conv}(3 \times 3, C \rightarrow C)$   
and  $\text{Conv}(3 \times 3, C \rightarrow C)$

- Parameters:

$$2 \times C \times C \times 3 \times 3 = 18C^2$$

- Flops:

$$2 \times C \times H \times W \times C \times 3 \times 3 = 18C^2HW$$

# VGG (2014)



- ① Halving the spatial dimensions (max pooling) and doubling the channels  $\rightarrow$  computational cost is unchanged

# VGG (2014)



- ① Halving the spatial dimensions (max pooling) and doubling the channels  $\rightarrow$  computational cost is unchanged
- ② **Case-1:**  $C \times 2H \times 2W$ , Conv ( $3 \times 3$ ,  $C \rightarrow C'$ )

# VGG (2014)



- ① Halving the spatial dimensions (max pooling) and doubling the channels  $\rightarrow$  computational cost is unchanged
- ② **Case-1:**  $C \times 2H \times 2W$ , Conv ( $3 \times 3$ ,  $C \rightarrow C$ )
  - Memory:  $4CHW$ , parameters:  $9C^2$ , Flops:  $36HWC^2$
- ③ **Case-2:**  $2C \times H \times W$ , Conv ( $3 \times 3$ ,  $2C \rightarrow 2C$ )
  - Memory:  $2CHW$ , parameters:  $36C^2$ , Flops:  $36HWC^2$

# VGG (2014)



- ① Huge network (VGG-16) compared to AlexNet

# VGG (2014)



- ① Huge network (VGG-16) compared to AlexNet
- ② Memory: 1.9  $\rightarrow$  48.6MB (25X)

# VGG (2014)



- ① Huge network (VGG-16) compared to AlexNet
- ② Memory:  $1.9 \rightarrow 48.6\text{MB}$  (25X)
- ③ Parameters:  $61 \rightarrow 138\text{M}$  (2.3X)

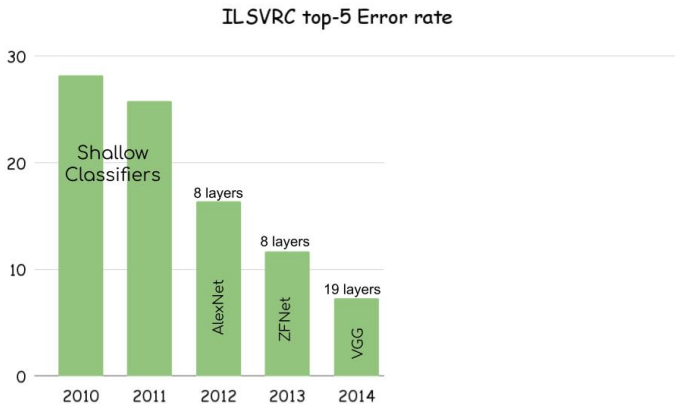
# VGG (2014)



- ① Huge network (VGG-16) compared to AlexNet
- ② Memory:  $1.9 \rightarrow 48.6\text{MB}$  (**25X**)
- ③ Parameters:  $61 \rightarrow 138\text{M}$  (**2.3X**)
- ④ Flops:  $0.7 \rightarrow 13.6\text{G Flop}$  (**19.4X**)



# VGG (2014)



# GoogLeNet (2014)



- ① Efficiency was the focus of design

---

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



- ① Efficiency was the focus of design
- ② Reduce the parameters, memory and the compute requirements (towards deployment)

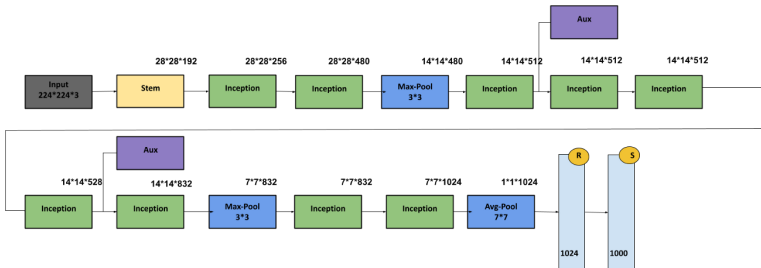
---

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



- ① Efficiency was the focus of design
- ② Reduce the parameters, memory and the compute requirements (towards deployment)



③

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



- ① Stem architecture at the early stage → aggressive down-sampling

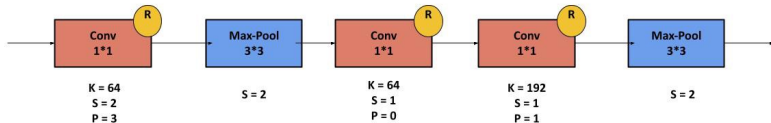
---

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



① Stem architecture at the early stage → aggressive down-sampling



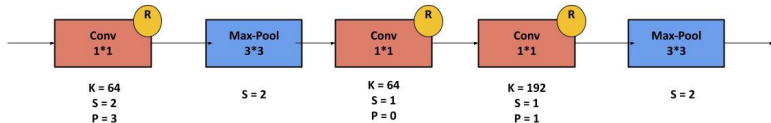
②

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



- ① Stem architecture at the early stage → aggressive down-sampling



②

- ③ From  $224 \times 224$  to  $28 \times 28$

- **GoogLeNet**: Compute - 7.5MB, parameters - 124K, and MFlops - 418
- **VGG-16**: Compute - 42.9MB (5.7X), parameters - 1.1M (8.9X), and MFlops - 7485 (17.8X)

Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLenet (2014)



- ① Inception module: unit with parallel branches

---

Figure credits: [Medium.com](#) and [Anas Brital](#)



# GoogLeNet (2014)



- 1 Inception module: unit with parallel branches
- 2 Repeated through the architecture

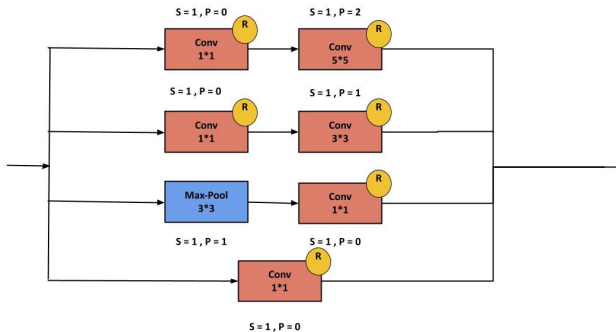


Figure credits: [Medium.com](#) and [Anas Brital](#)

# GoogLeNet (2014)



- 1 Global Average Pooling (GAP) layer

---

Alexis Cook

# GoogLeNet (2014)



- ① Global Average Pooling (GAP) layer
- ② Flattening results in huge weight matrices → GoogLeNet introduces GAP layer

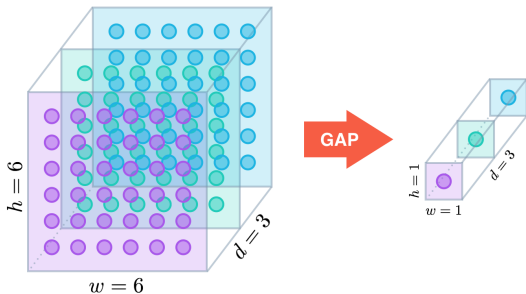
---

Alexis Cook

# GoogLeNet (2014)



- ① Global Average Pooling (GAP) layer
- ② Flattening results in huge weight matrices  $\rightarrow$  GoogLeNet introduces GAP layer
- ③ Collapses the spatial dimensions by computing the average (kernel size = spatial dimensions of the last conv layer)



# GoogLeNet (2014)



- 1 No more fully connected layers

# GoogLeNet (2014)



- ① No more fully connected layers
- ② One linear layer to predict the classification scores (feather light!)

# GoogLeNet (2014)



## ① Auxiliary classifiers

# GoogLenet (2014)



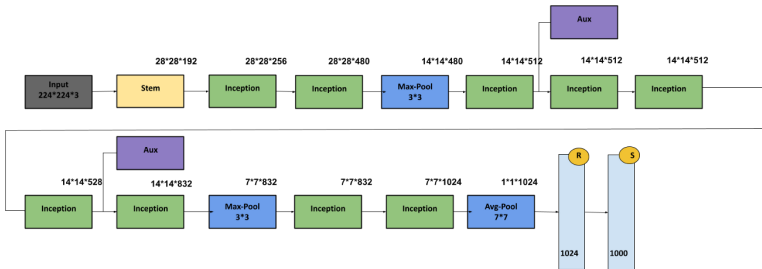
- ① Auxiliary classifiers
- ② Training using the gradients at the end of the network didn't work well (too deep, gradient propagation was not robust)



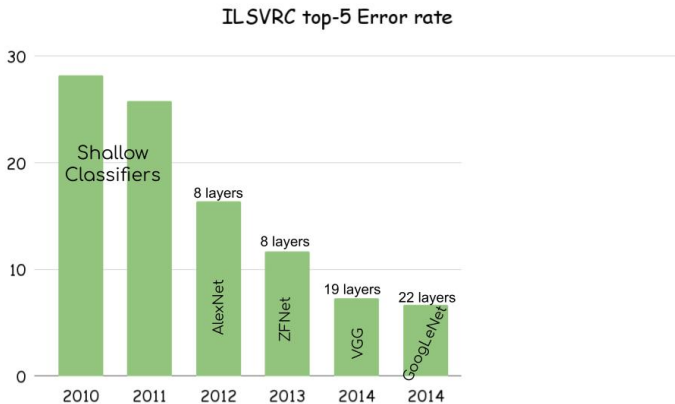
# GoogLeNet (2014)



- ① Auxiliary classifiers
- ② Training using the gradients at the end of the network didn't work well (too deep, gradient propagation was not robust)
- ③ Hack: add auxiliary classifiers at intermediate locations to receive loss/gradients



# GoogLeNet (2014)



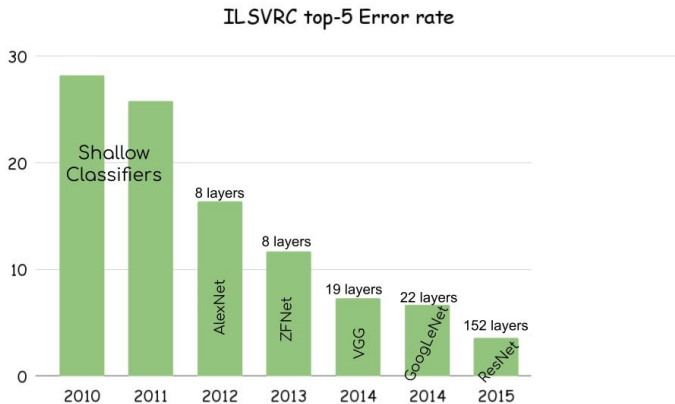
# ResNet (2015)



- ① Very important time for the DNNs
  - Batch Normalization happened
  - Depth increased by an order ( $10 \rightarrow 150+$ )
  - ILSVRC error almost halved from that of 2014

# ResNet (2015)

- ① Very important time for the DNNs
  - Batch Normalization happened
  - Depth increased by an order ( $10 \rightarrow 150+$ )
  - ILSVRC error almost halved from that of 2014



②

# Training Deeper CNNs



- ① When training the “deeper” CNNs, people observed that they were worse than shallow ones

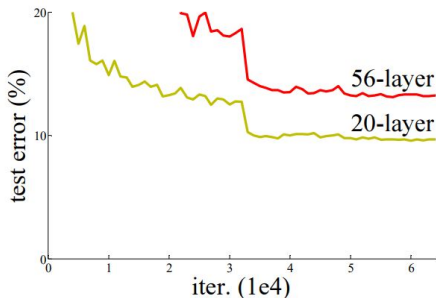
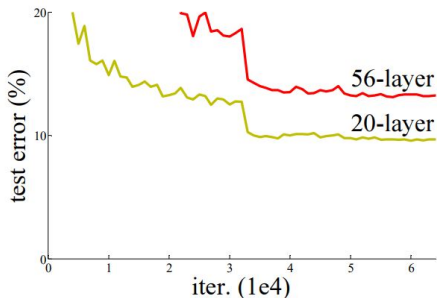


Figure Credits: He et al. 2015

# Training Deeper CNNs



- ① When training the “deeper” CNNs, people observed that they were worse than shallow ones



- ② Initial suspicion was the ‘over-fitting’!

Figure Credits: He et al. 2015

# Training Deeper CNNs



- ① Initial suspicion was the 'over-fitting'!
- ② However, it was due to the under-fitting

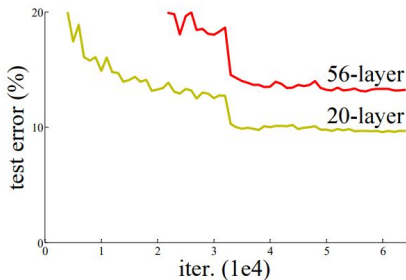
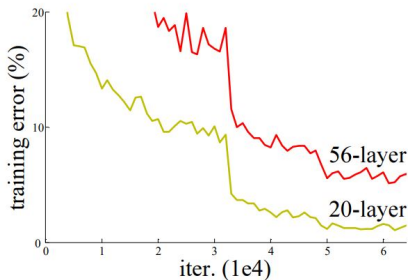


Figure Credits: He et al. 2015

# ResNet (2015)



- ① Deeper CNNs should easily emulate the shallow ones (extra layers could learn identity function)



# ResNet (2015)



- ① Deeper CNNs should easily emulate the shallow ones (extra layers could learn identity function)
- ② This is not the case → some issue in the optimization!

# ResNet (2015)



- ① Deeper CNNs should easily emulate the shallow ones (extra layers could learn identity function)
- ② This is not the case → some issue in the optimization!
- ③ Work on the architecture so that learning identity function gets easier with additional layers

# ResNet (2015)



- ① Work on the architecture so that learning identity function gets easier with additional layers

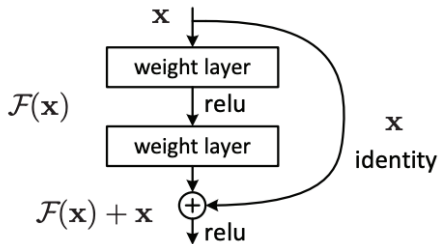
---

Yuanrui Dong

# ResNet (2015)



- ① Work on the architecture so that learning identity function gets easier with additional layers
- ② ResBlock (residual block)



---

Yuanrui Dong

# ResNet (2015)



- 1 ResBlocks help the gradient backpropagation

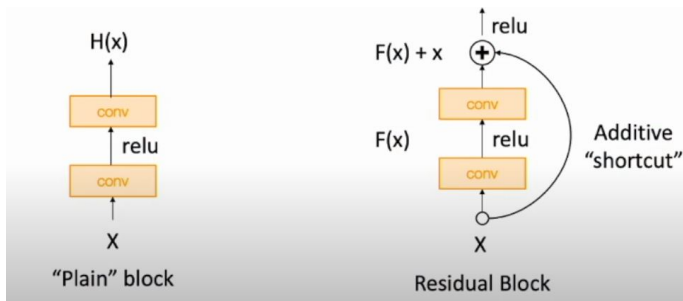


Figure Credits: Dr. Justin Johnson, U Michigan

# ResNet (2015)



- 1 ResNet is a stack of Resblocks

---

Figure credits: Dr. Justin Johnson, U Michigan

# ResNet (2015)



- ① ResNet is a stack of Resblocks
- ② Inspire from VGG and GoogLeNet

---

Figure credits: Dr. Justin Johnson, U Michigan

# ResNet (2015)



- ① ResNet is a stack of Resblocks
- ② Inspire from VGG and GoogLeNet
- ③ Simple and regular design like VGG: each resblock has two  $3 \times 3$  Conv

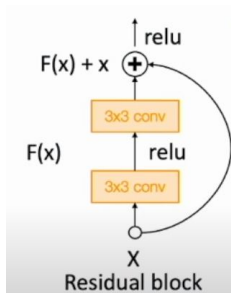


Figure credits: Dr. Justin Johnson, U Michigan



# ResNet (2015)



- ① Network has stages: first block of each stage halves the resolution and doubles the channels

# ResNet (2015)



- ① Network has stages: first block of each stage halves the resolution and doubles the channels
- ② Aggressive stem in the beginning (downsamples by 4X before the start of the resblocks)

# ResNet (2015)



- ① Network has stages: first block of each stage halves the resolution and doubles the channels
- ② Aggressive stem in the beginning (downsamples by 4X before the start of the resblocks)
- ③ Eliminates the FC layers via GAP

# ResNet (2015)

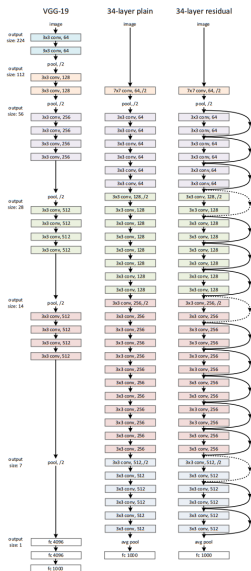


Figure credits: K. he et al., ResNets 92015)

# ResNet (2015)



## ① ResNet-18

- Stem: 1 Conv
- Stage-1 (C=64): 2 resblocks (4 Conv)
- Stage-2 (C=128): 2 resblocks (4 Conv)
- Stage-3 (C=256): 2 resblocks (4 Conv)
- Stage-4 (C=512): 2 resblocks (4 Conv)
- Linear
- Top-5 error: 10.92 and GFlop: 1.8

# ResNet (2015)



## ① ResNet-34

- Stem: 1 Conv
- Stage-1 (C=64): 3 resblocks (6 Conv)
- Stage-2 (C=128): 4 resblocks (8 Conv)
- Stage-3 (C=256): 6 resblocks (12 Conv)
- Stage-4 (C=512): 3 resblocks (6 Conv)
- Linear
- Top-5 error: 8.58 and GFlop: 3.6 (VGG: 9.6 and 13.6 respectively)

# ResNet (2015)



## 1 Bottleneck Residual block

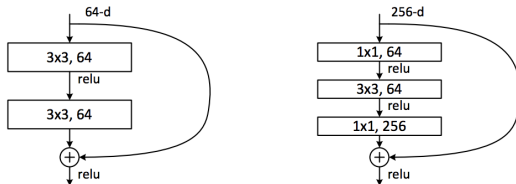


Figure Credits: [Nushaine Ferdinand](#)

# ResNet (2015)



- ① Resnet-34 becomes ResNet-50 if we replace the plain resblocks with bottleneck ones



# ResNet (2015)

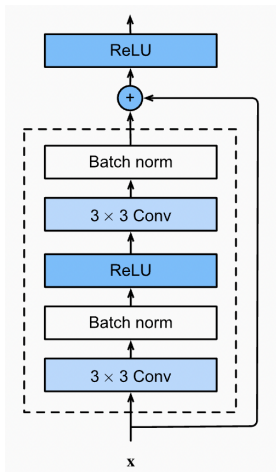


- ① Resnet-34 becomes ResNet-50 if we replace the plain resblocks with bottleneck ones
- ② More blocks at each stage result in ResNet-101 and Resnet-152 architectures

# Resnet (2015)

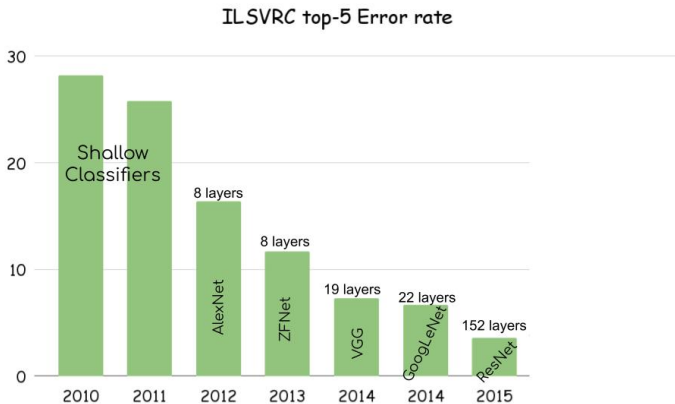


- ① Resblocks have Batch Normalization layers



Yashovardhan Shinde

# ResNet (2015)



- 1 Ensemble models, improvements to existing architectures, etc.

# Post 2015



- ① Ensemble models, improvements to existing architectures, etc.
- ② Densenets, Tiny Networks (MobileNets, ShuffleNets), etc.