Topics:

- CNNs Continued
- Regularization & Augmentation
- Transfer Learning

CS 4644-DL / 7643-A

ZSOLT KIRA

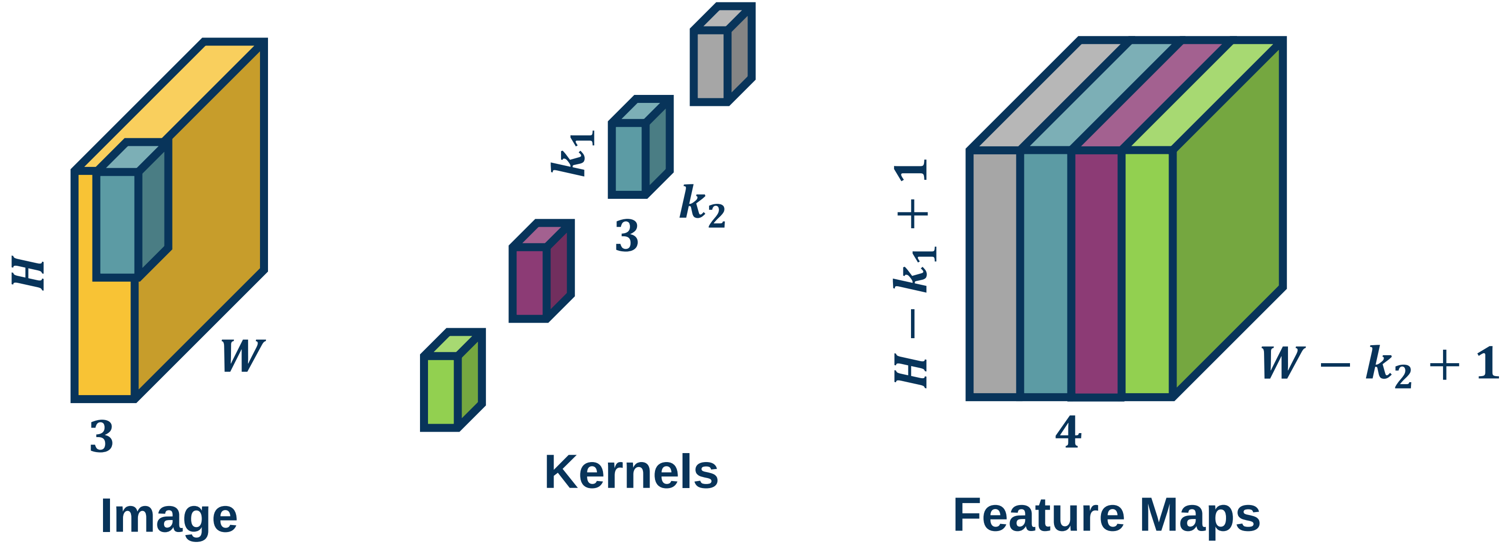
- **Assignment 2**
  - Implement convolutional neural networks
  - Resources (in addition to lectures):
    - [DL book: Convolutional Networks]
    - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
    - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
    - **HW2 Tutorial (@176), Conv backward (@181)**
    - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6) (https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0)

- **FB/Meta Office hours Friday 02/17 2pm EST!**
  - Pytorch & scalable training
  - [Module 2, Lesson 8 (M2L8), on dropbox]

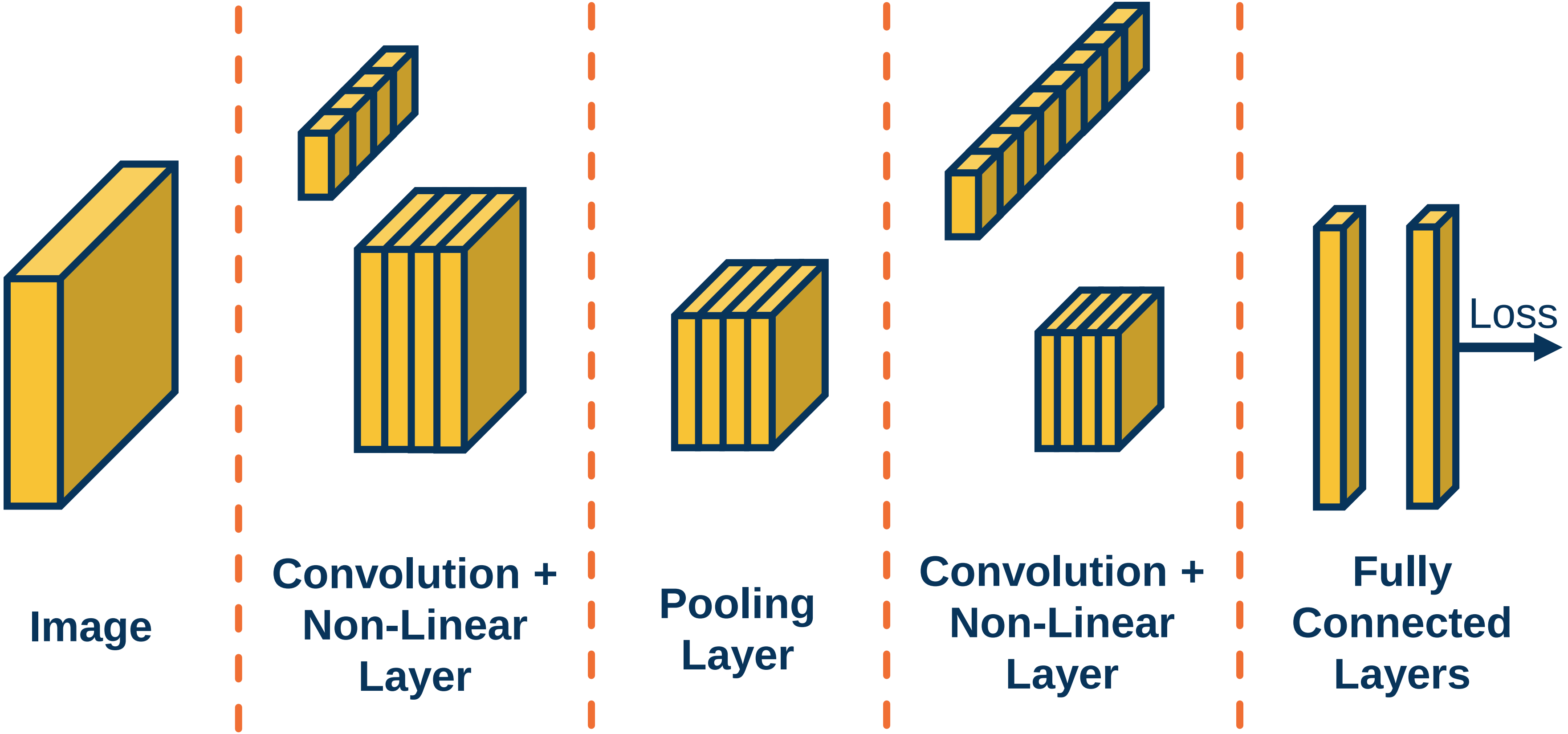
- **GPU resources**: PACE-ICE and Google Cloud announced

Number of parameters with N filters is: $N * (k_1 * k_2 * 3 + 1)$

- Example:
  $k_1 = 3, k_2 = 3,\ N = 4\ input\ channels = 3$, then $(3 * 3 * 3 + 1) * 4 = 112$

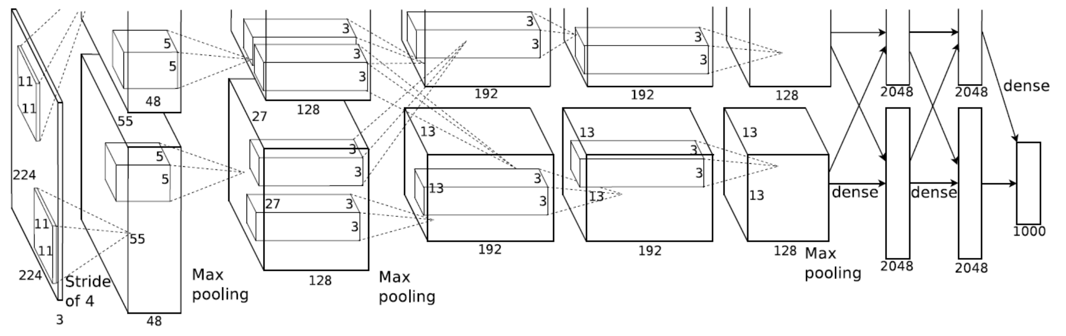


Number of Parameters



Adding a Fully Connected Layer

Full (simplified) AlexNet architecture:
[224x224x3] 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 (class scores)



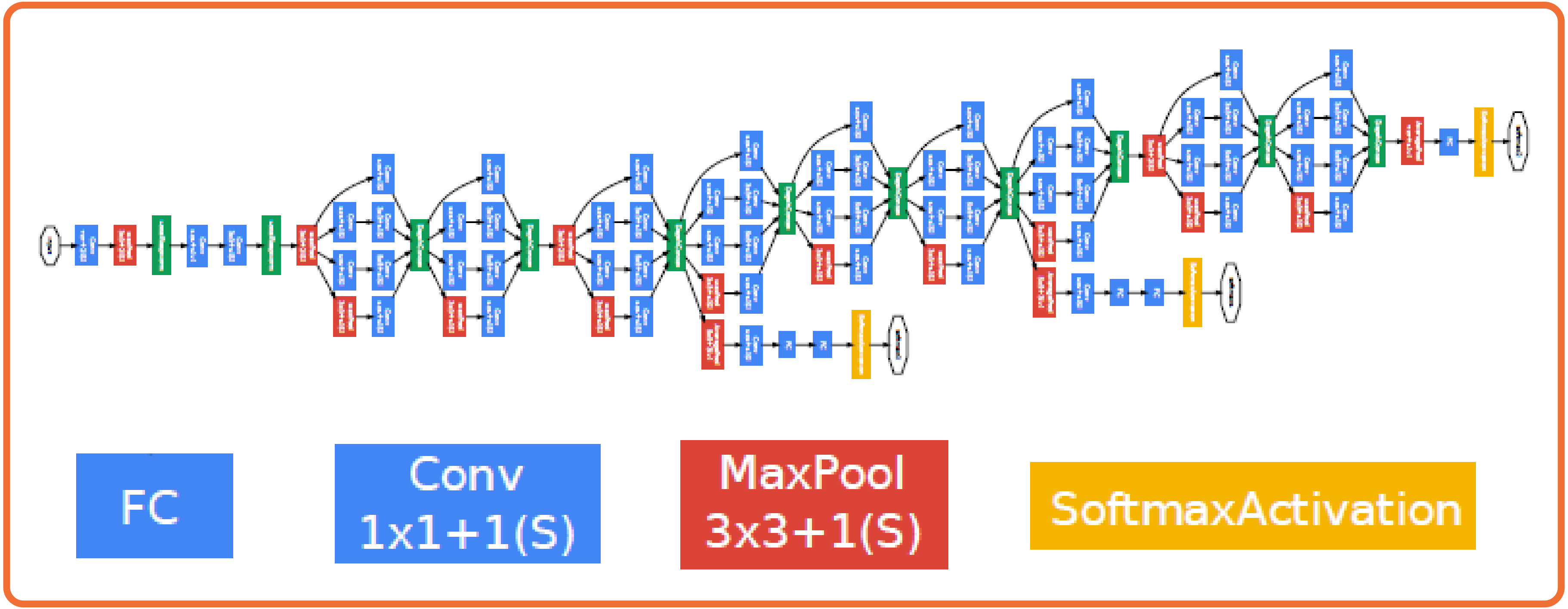
Key aspects:

- ReLU instead of sigmoid or tanh
- Specialized normalization layers
- PCA-based data augmentation
- Dropout
- Ensembling

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

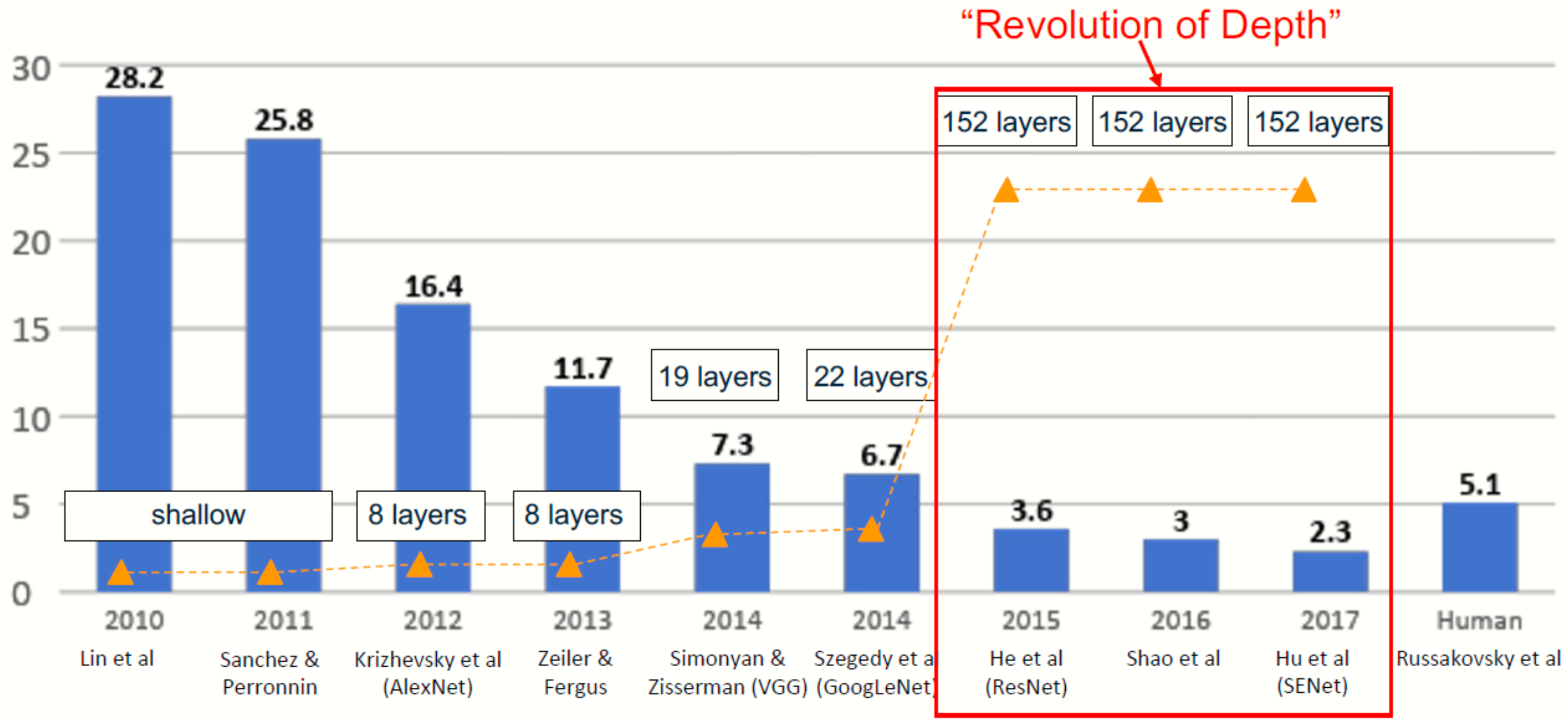
AlexNet – Layers and Key Aspects

But have become **deeper and more complex**



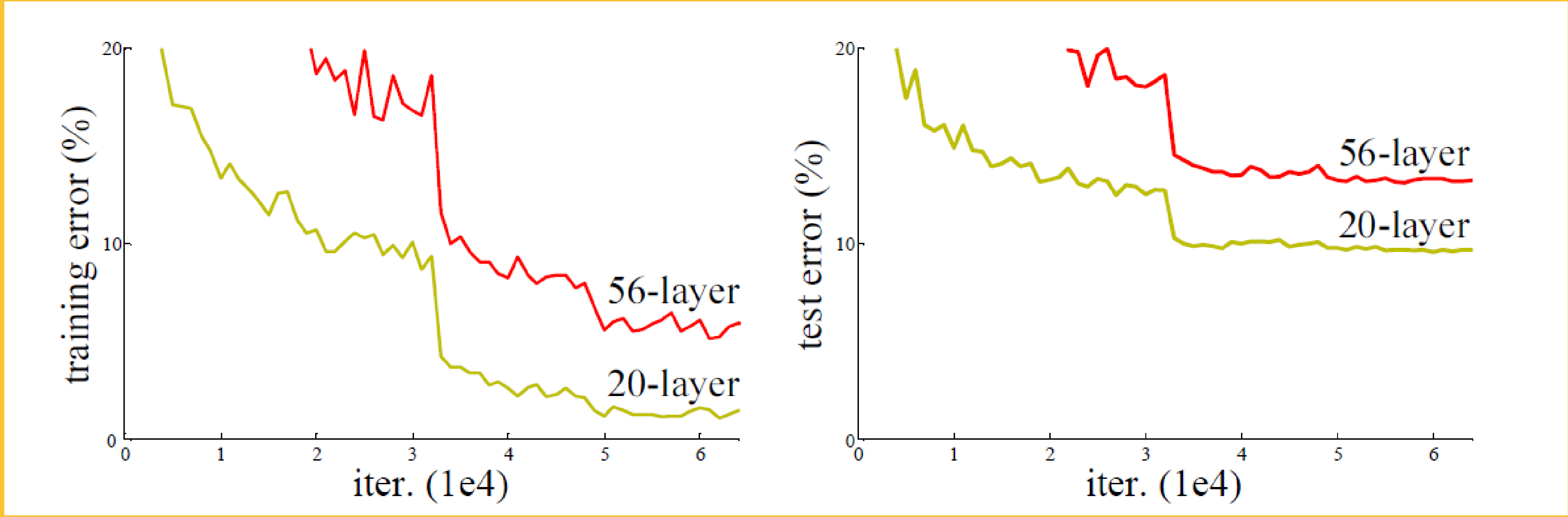
From: Szegedy et al. Going deeper with convolutions

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Revolution of Depth

The Challenge of Depth



From: He et al., Deep Residual Learning for Image Recognition

Optimizing very deep networks is challenging!

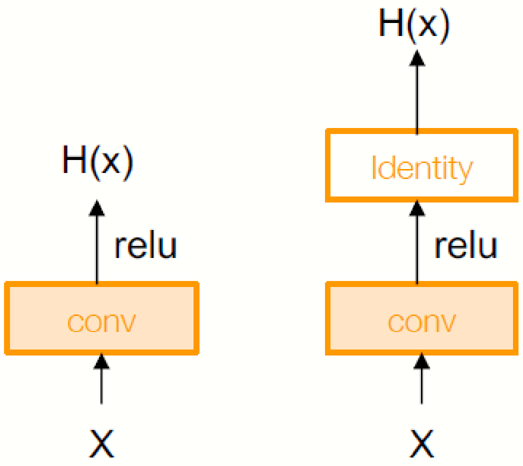
[He et al., 2015]

A deeper model can **emulate** a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

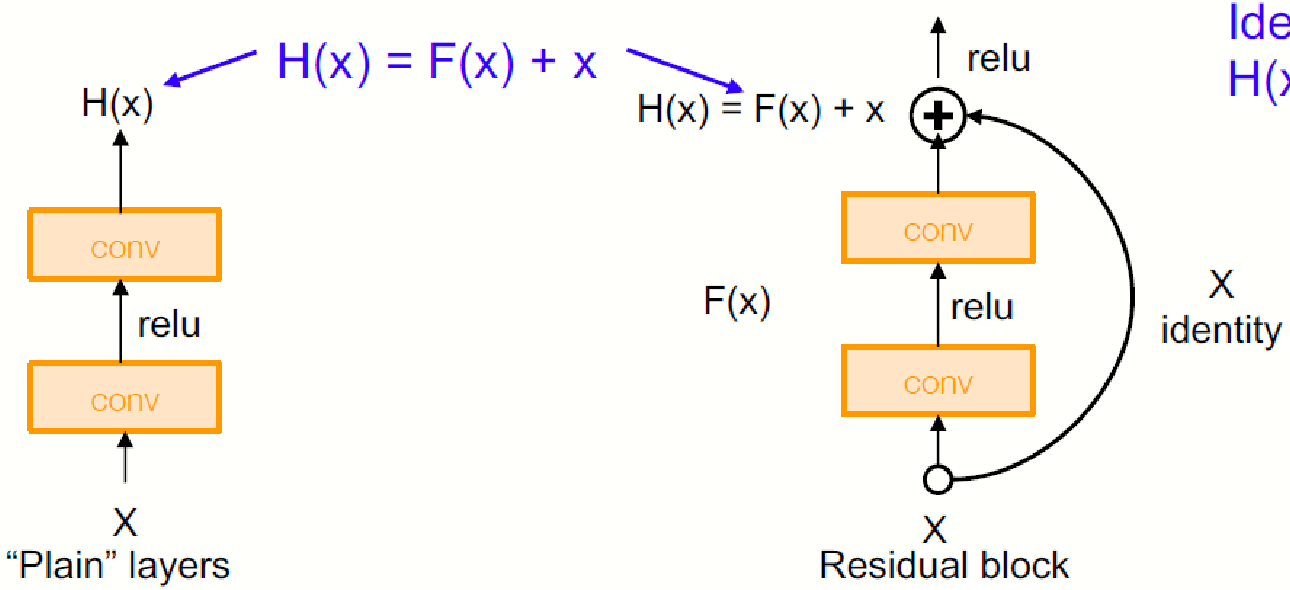
Deeper models are harder to optimize. They don't learn identity functions (no-op) to emulate shallow models

Solution: Change the network so learning identity functions (no-op) as extra layers is easy



[He et al., 2015]

Solution: Change the network so learning identity functions as extra layers is easy



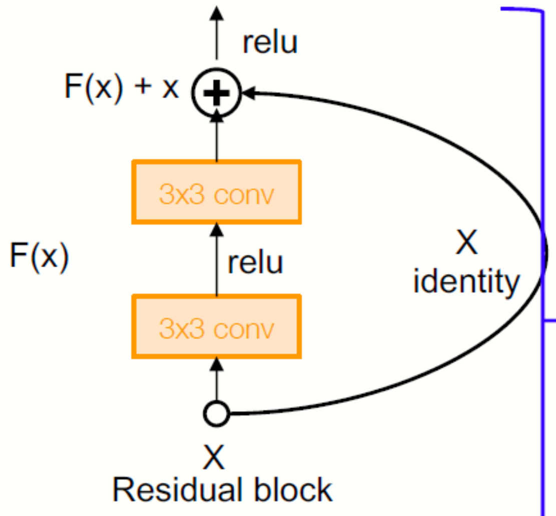
Identity mapping:
$H(x) = x$ if $F(x) = 0$

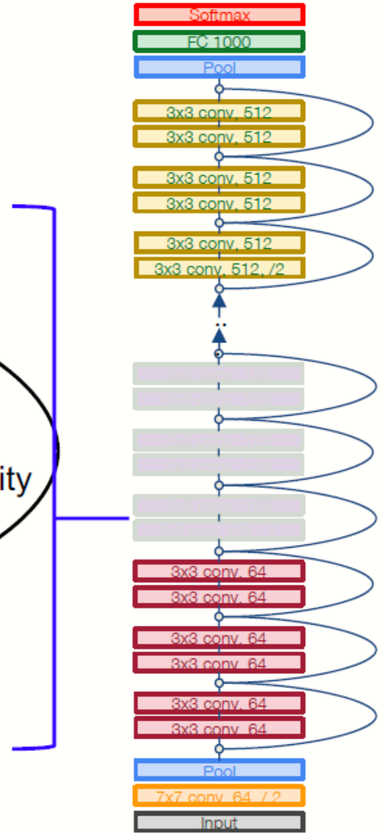
Use layers to fit **residual**
$F(x) = H(x) - x$
instead of
$H(x)$ directly

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



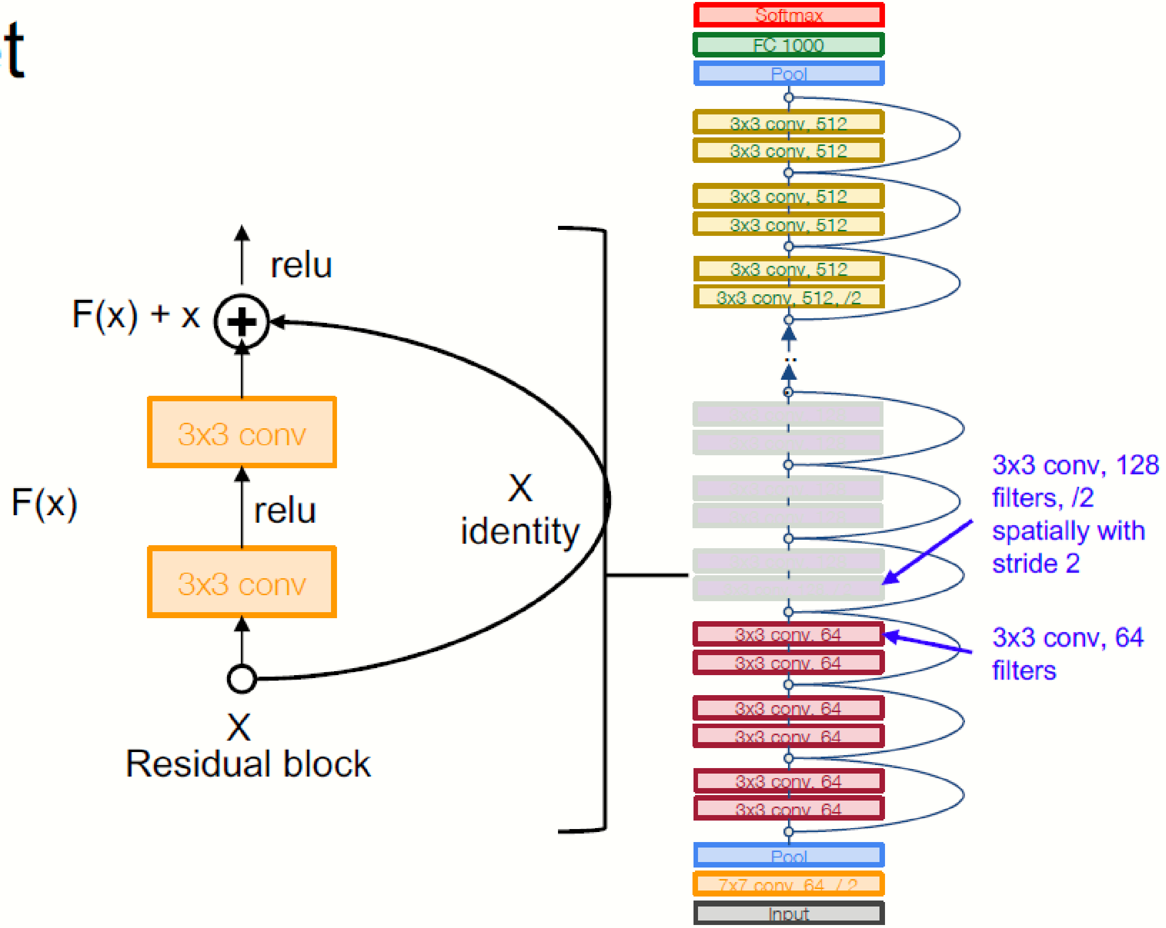


Case Study: ResNet

[He et al., 2015]

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) Reduce the activation volume by half.

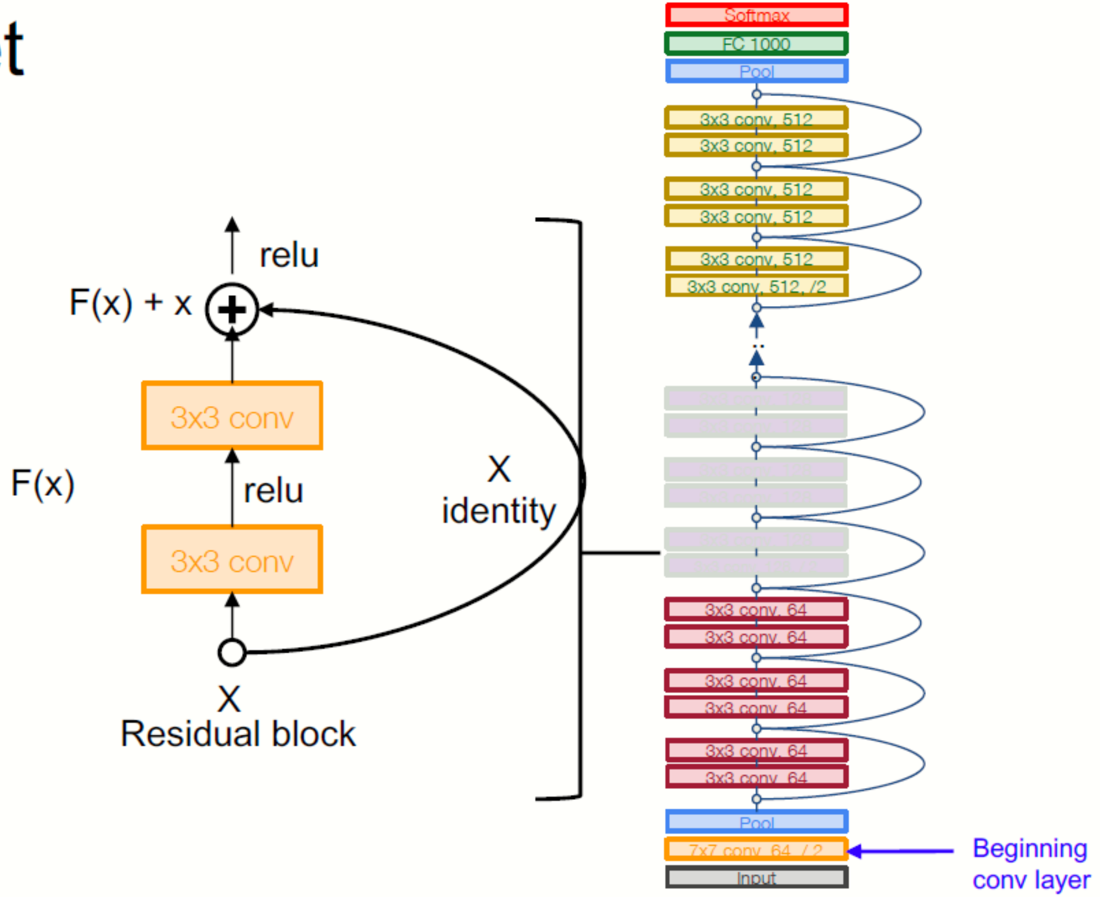


Case Study: ResNet

[He et al., 2015]

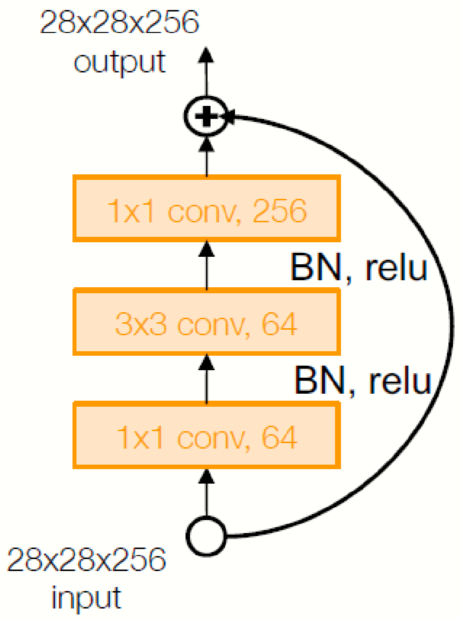
Full ResNet architecture:

- 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) Reduce the activation volume by half.
- Additional conv layer at the beginning (stem)



[He et al., 2015]

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

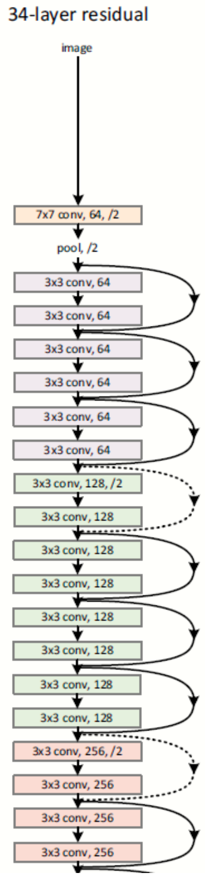


Bottleneck Layers

layer name	output size	18-layer	34-layer			
conv1	112×112					
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$			
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$			
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$			
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$			
	1×1	ave				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
>>> import torch
>>> from torchvision.models import resnet18
>>> model = resnet18()
>>> summary(model2, (3, 224, 224), device='cpu')
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 64, 112, 112]           9,408
       BatchNorm2d-2        [-1, 64, 112, 112]             128
              ReLU-3        [-1, 64, 112, 112]               0
         MaxPool2d-4          [-1, 64, 56, 56]               0
            Conv2d-5          [-1, 64, 56, 56]          36,864
       BatchNorm2d-6          [-1, 64, 56, 56]             128
              ReLU-7          [-1, 64, 56, 56]               0
            Conv2d-8          [-1, 64, 56, 56]          36,864
       BatchNorm2d-9          [-1, 64, 56, 56]             128
             ReLU-10          [-1, 64, 56, 56]               0
       BasicBlock-11          [-1, 64, 56, 56]               0
           Conv2d-12          [-1, 64, 56, 56]          36,864
      BatchNorm2d-13          [-1, 64, 56, 56]             128
             ReLU-14          [-1, 64, 56, 56]               0
           Conv2d-15          [-1, 64, 56, 56]          36,864
      BatchNorm2d-16          [-1, 64, 56, 56]             128
             ReLU-17          [-1, 64, 56, 56]               0
       BasicBlock-18          [-1, 64, 56, 56]               0
           Conv2d-19         [-1, 128, 28, 28]          73,728
      BatchNorm2d-20         [-1, 128, 28, 28]             256
             ReLU-21         [-1, 128, 28, 28]               0
           Conv2d-22         [-1, 128, 28, 28]         147,456
```

ResNet Details



When the dimensions increase (dotted line shortcuts in Fig. 3), we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

Residual Blocks and Skip Connections

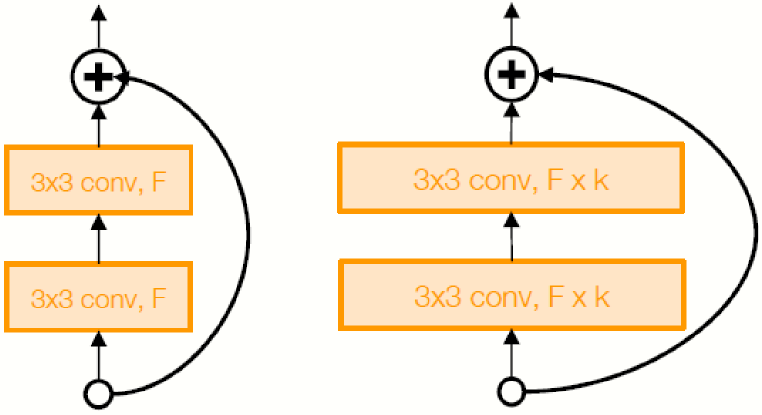
Training ResNet in practice:

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

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- Use wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)

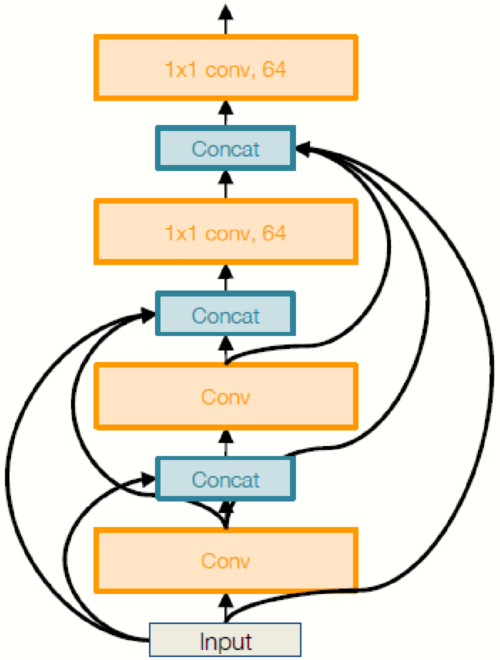


Basic residual block   Wide residual block

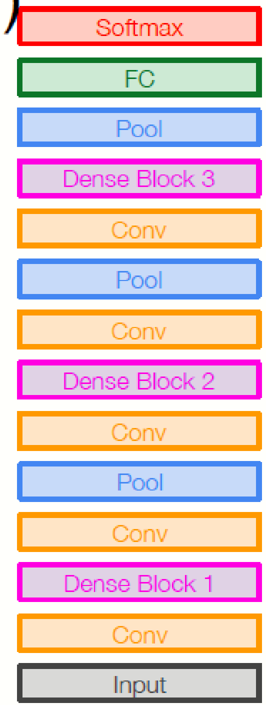
Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



Dense Block

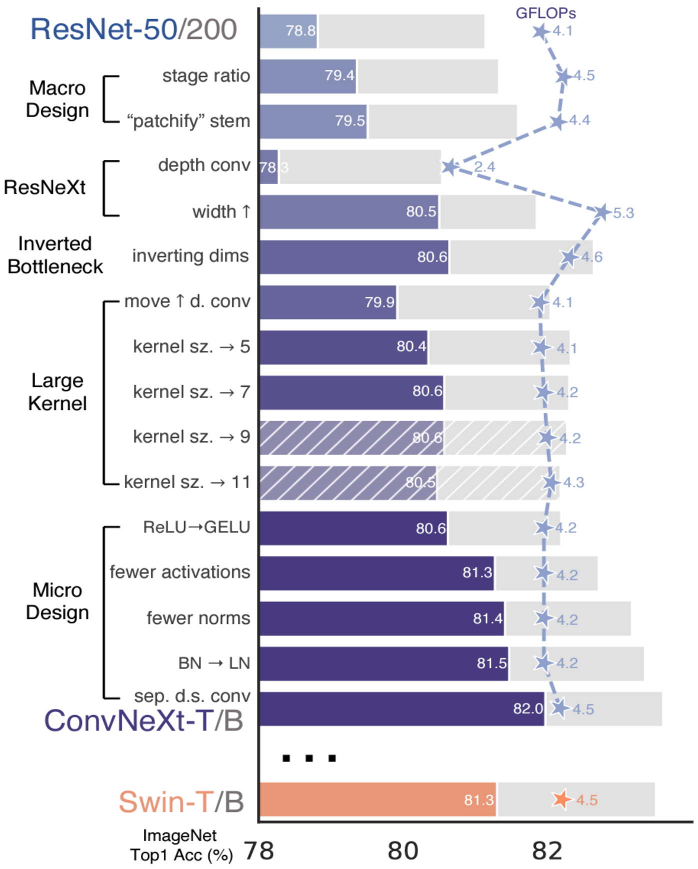


ConvNeXt (2022)

- ***To bridge the gap between the Conv Nets and Vision Transformers (ViT)***
  - ViT, Swin Transformer has been the SOTA visual model backbone
  - Is convolutional networks really not as good as transformer models?
- Investigation
  - The author start with ResNet-50 and reimplement the CNN networks with modern designs
  - The results showing that ConvNeXt achieves beat the ViT models, again.

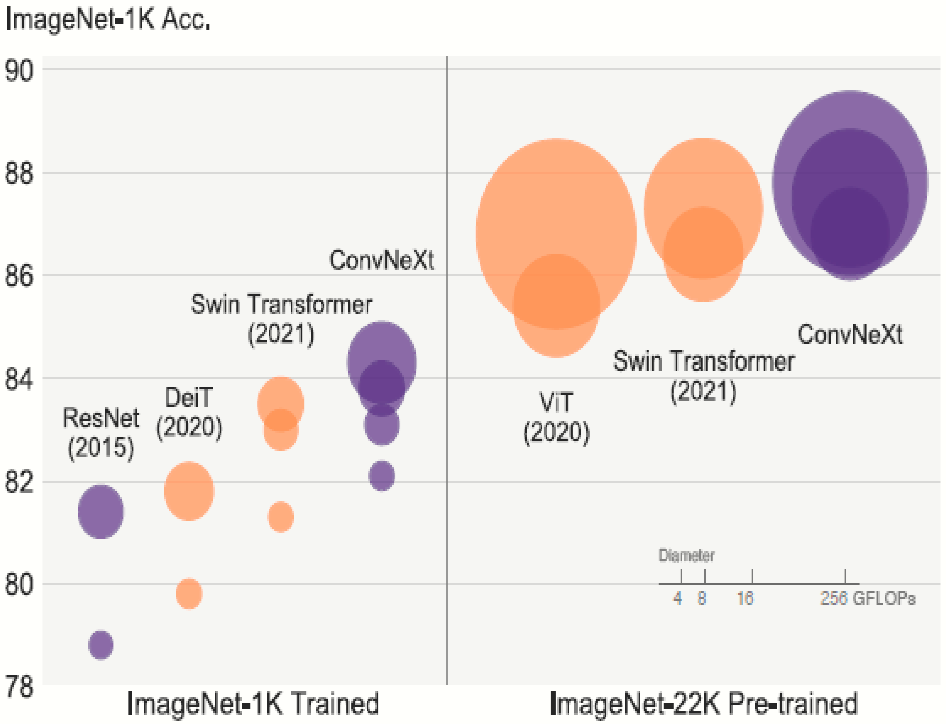
ConvNeXt (2022)

- **Modern designs added:**
- Use ResNeXt
- Apply Inverted Bottleneck
- Use larger kernel size
- Training strategy:
  - 90 epochs -> 300 epochs
  - AdamW optimizer
  - Data augmentation like Mixup, CutMix
  - Regularization Schemes like label smoothing
  - …



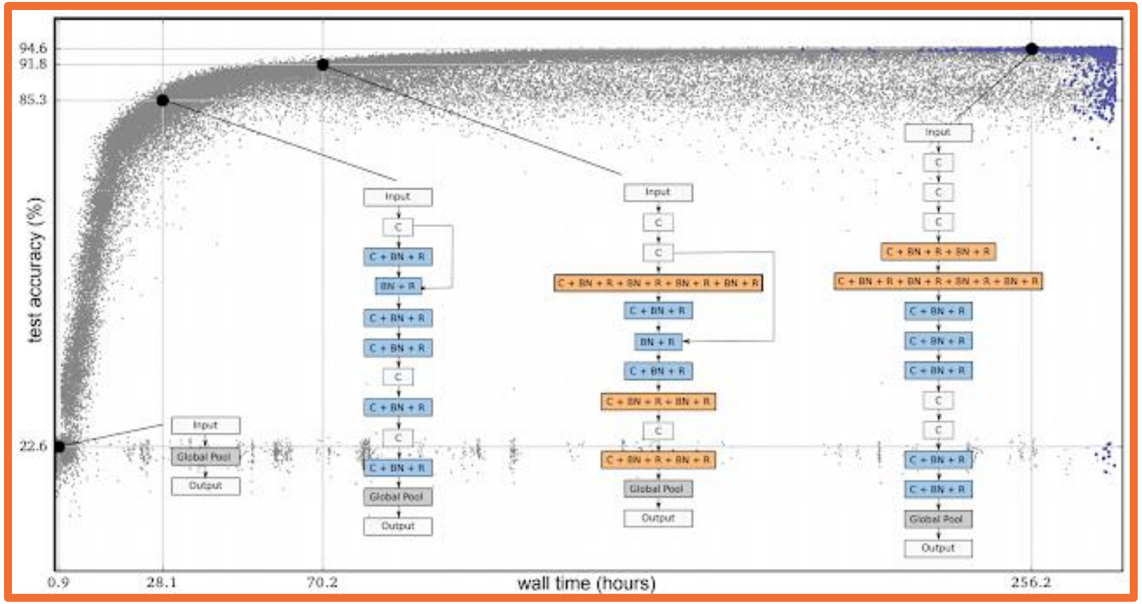
ConvNeXt (2022)

- **Modern designs added:**
- Macro Design
  - Changing stage compute ratio
  - Changing stem to “patchify”
- Micro Design
  - ReLU -> GELU
  - Fewer activation functions
  - Fewer normalization layers
  - BatchNorm -> LayerNorm
  - Separate downsampling layers



Several ways to *learn* architectures:

- Evolutionary learning and reinforcement learning
- Prune over-parameterized networks
- Learning of **repeated blocks** typical

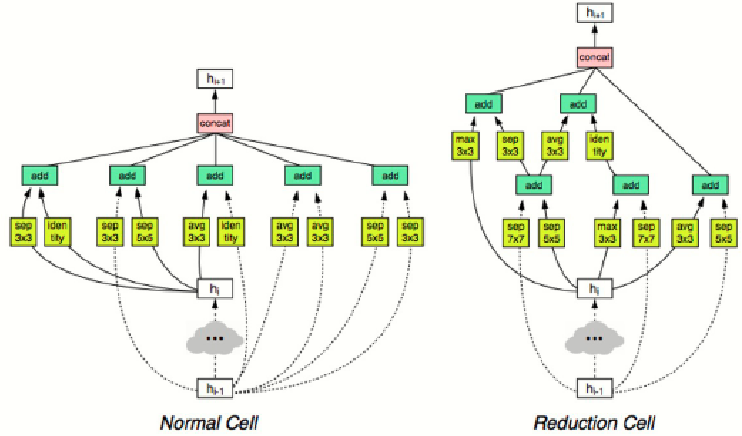


From: https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

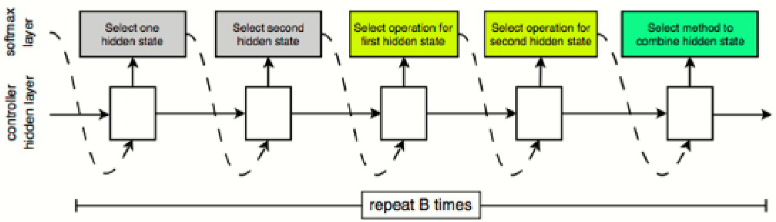
Learning Transferable Architectures for Scalable Image Recognition

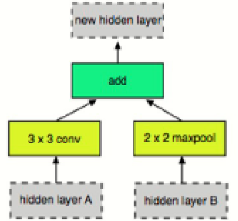
[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks ("cells") that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet

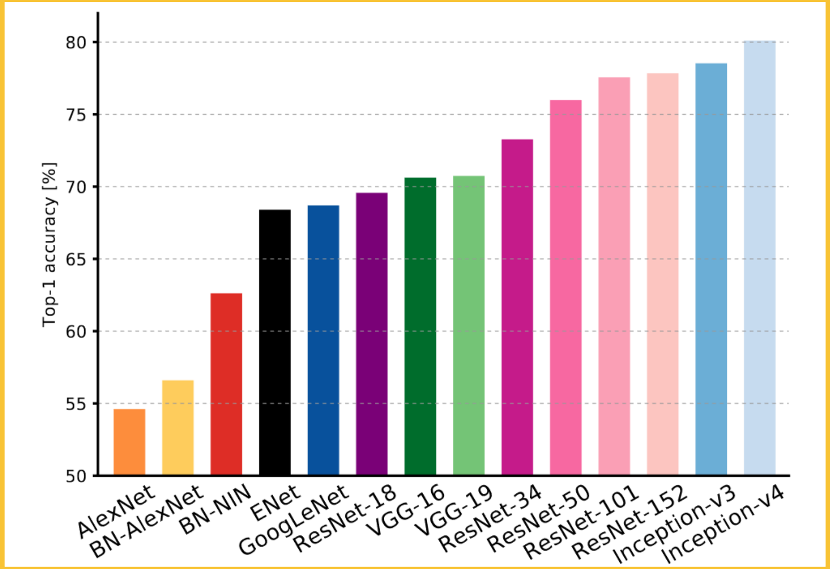


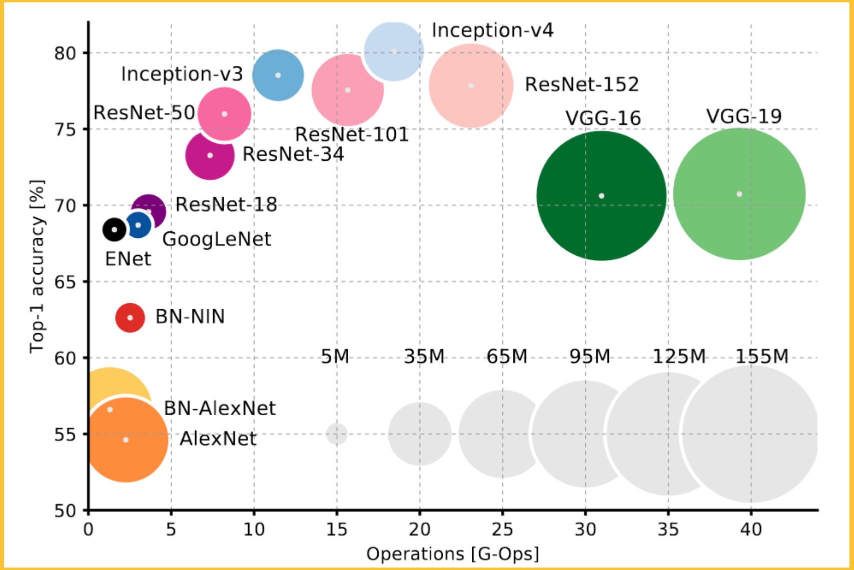
- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)





Computational Complexity



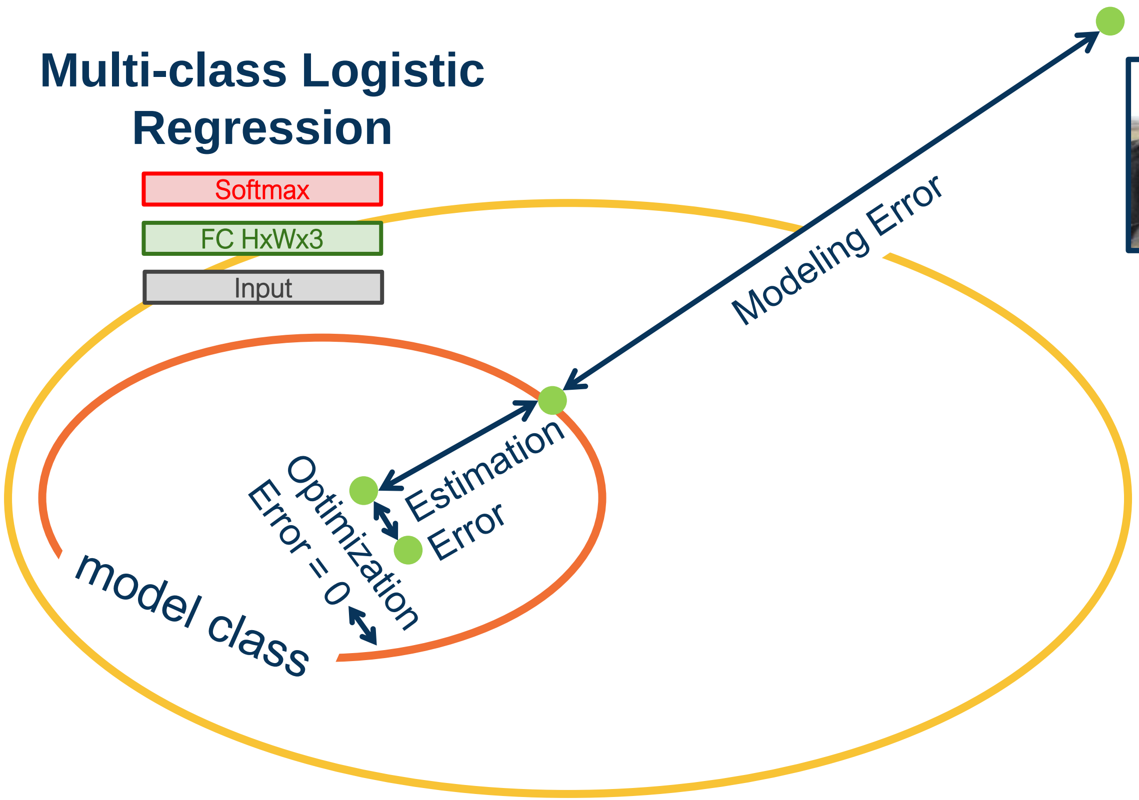


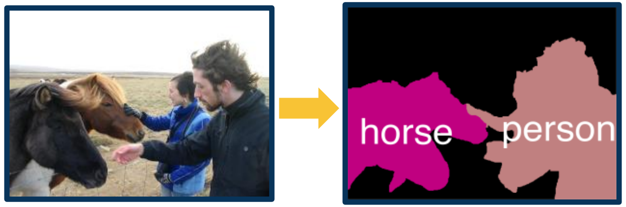
From: An Analysis Of Deep Neural Network Models For Practical Applications

Transfer Learning & Generalization

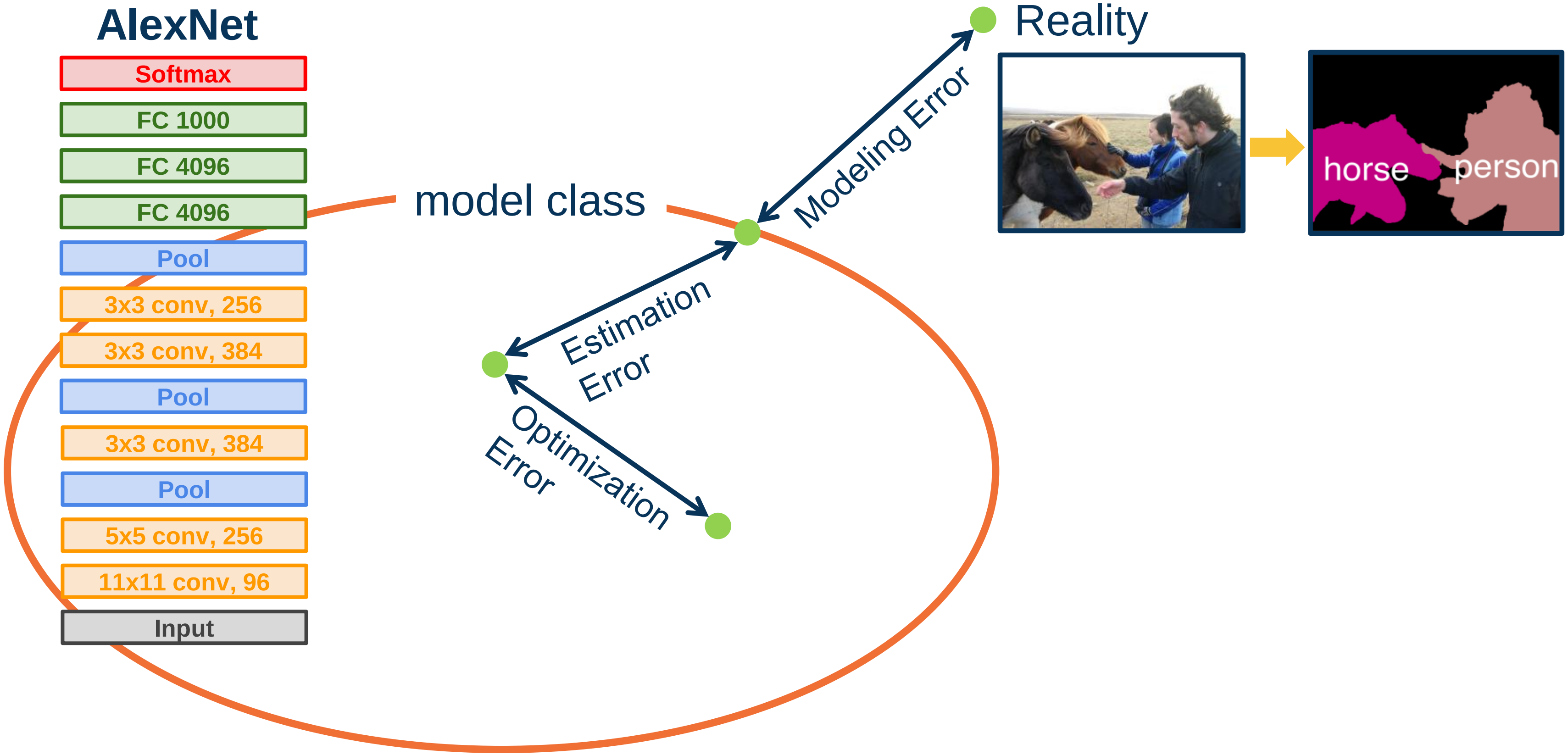
Georgia Tech

Multi-class Logistic Regression



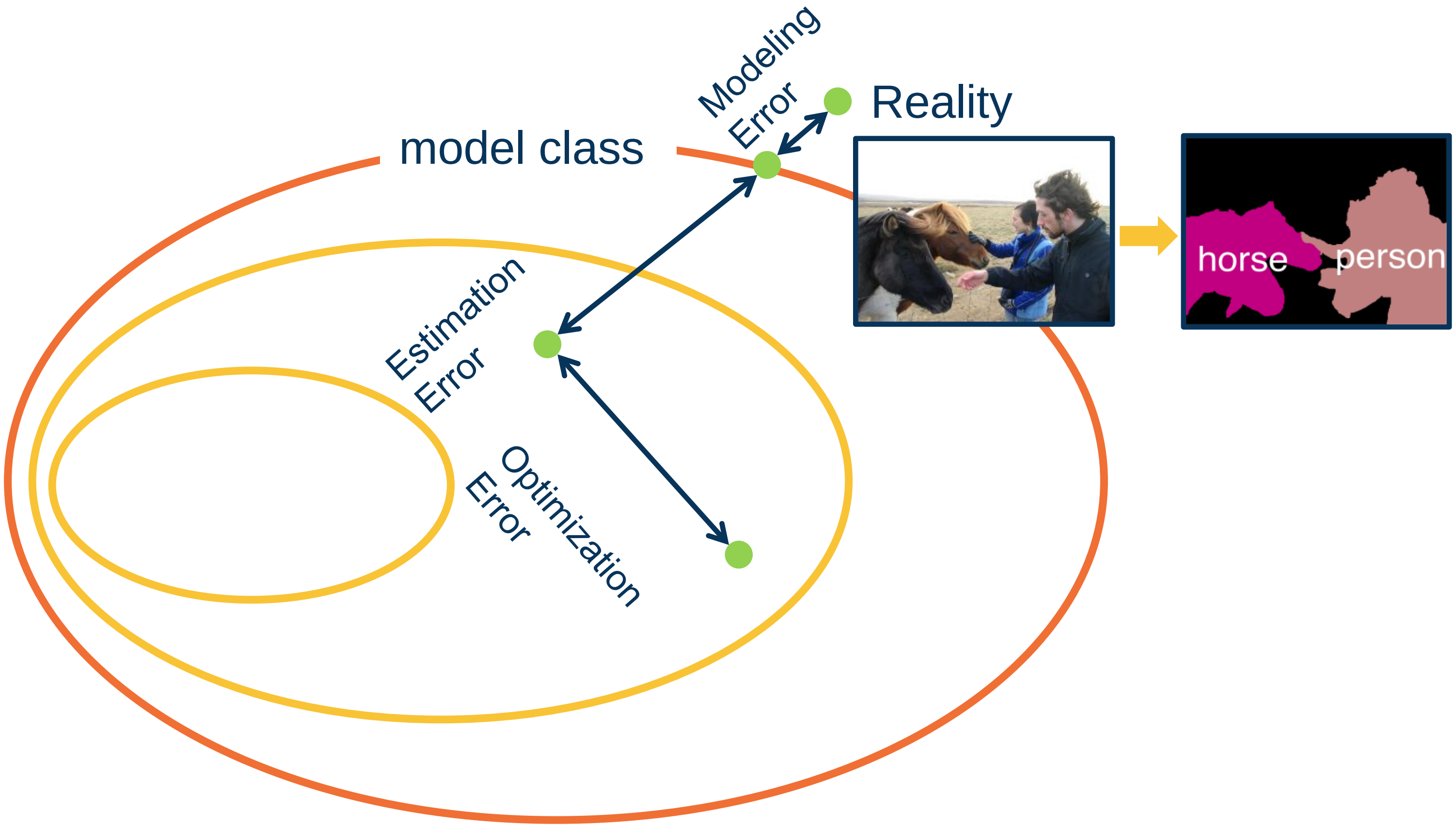


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Generalization

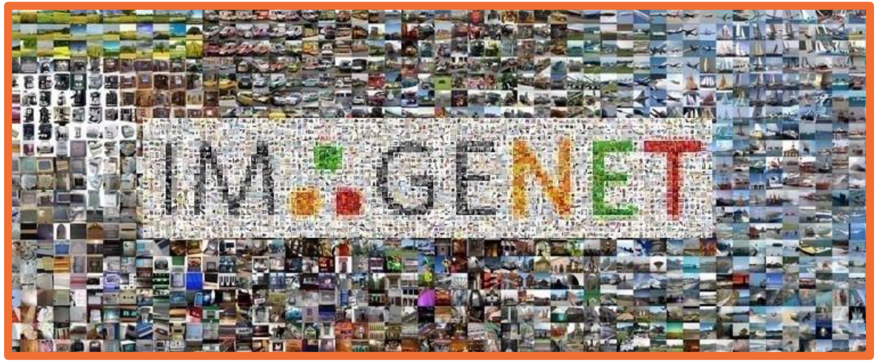


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

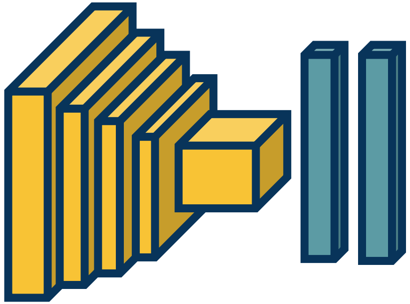
Generalization

What if we don’t have enough data?

Step 1: Train on large-scale dataset



Input Image

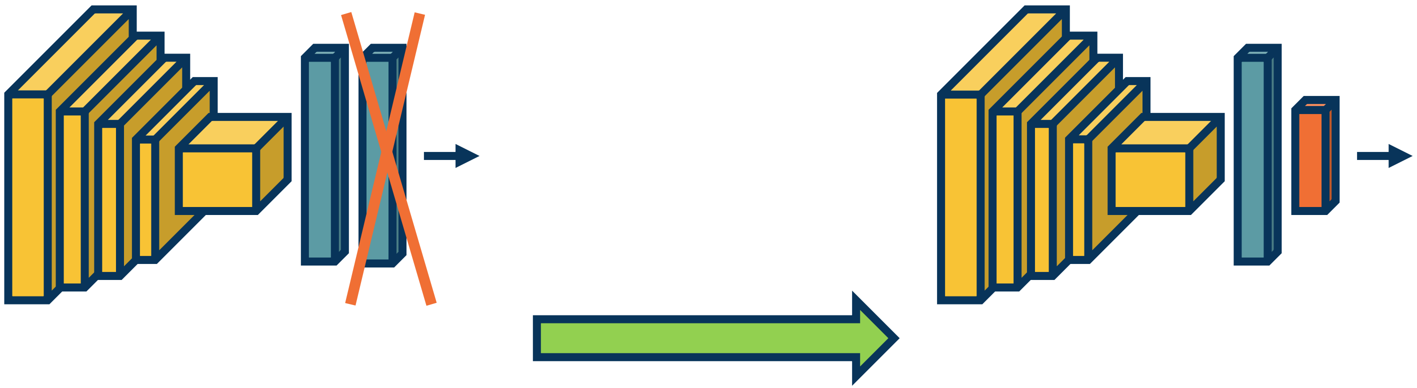


Predictions

Convolutional Neural Networks

Transfer Learning – Training on Large Dataset

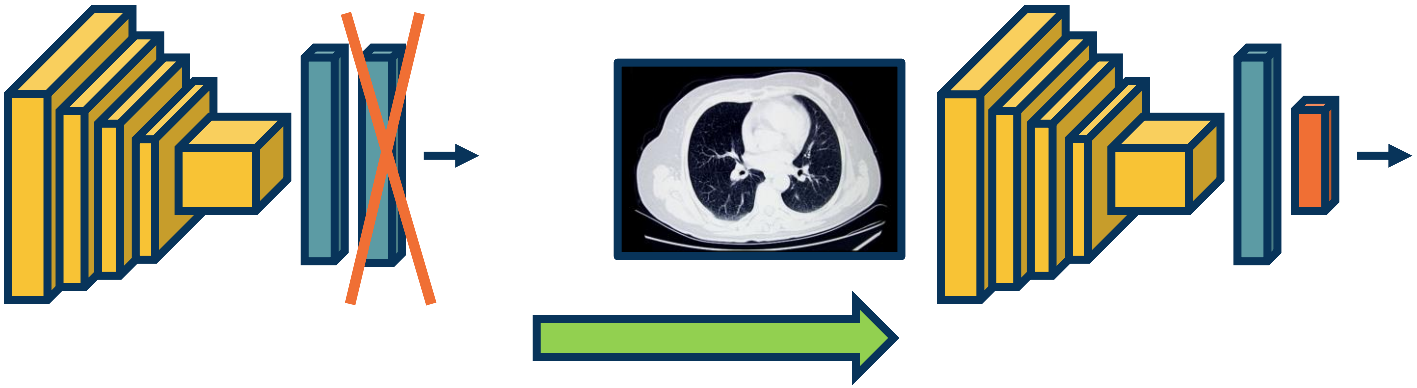
Step 2: Take your custom data and **initialize** the network with weights trained in Step 1



Replace last layer with new fully-connected for output nodes per new category

Step 3: (Continue to) train on new dataset

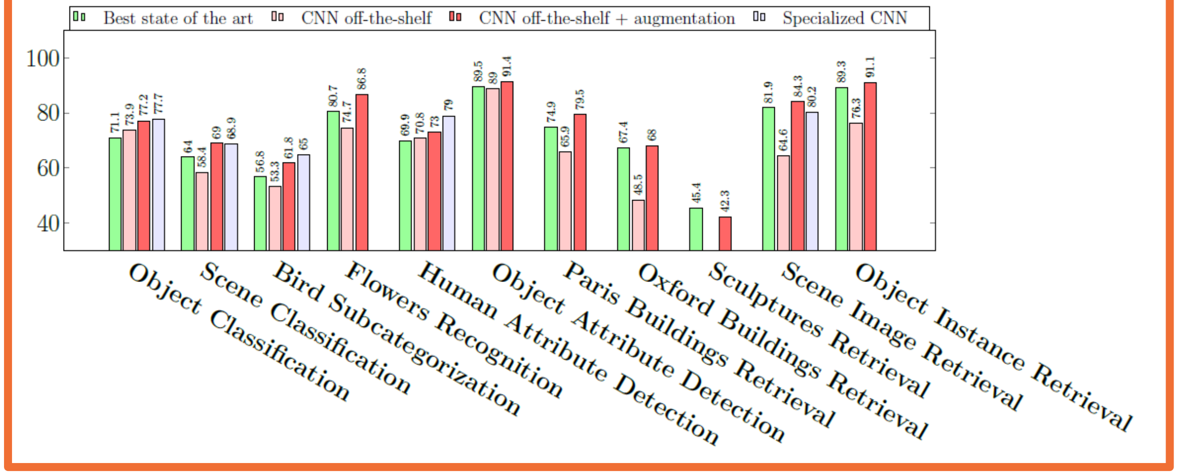
- **Finetune:** Update all parameters
- **Freeze** feature layer: Update only last layer weights (used when not enough data)



Replace last layer with new fully-connected for output nodes per new category

This works extremely well! It was surprising upon discovery.

- Features learned for 1000 object categories will work well for 1001st!
- Generalizes even across tasks (classification to object detection)



From: Razavian et al., CNN Features off-the-shelf: an Astounding Baseline for Recognition

Surprising Effectiveness of Transfer Learning

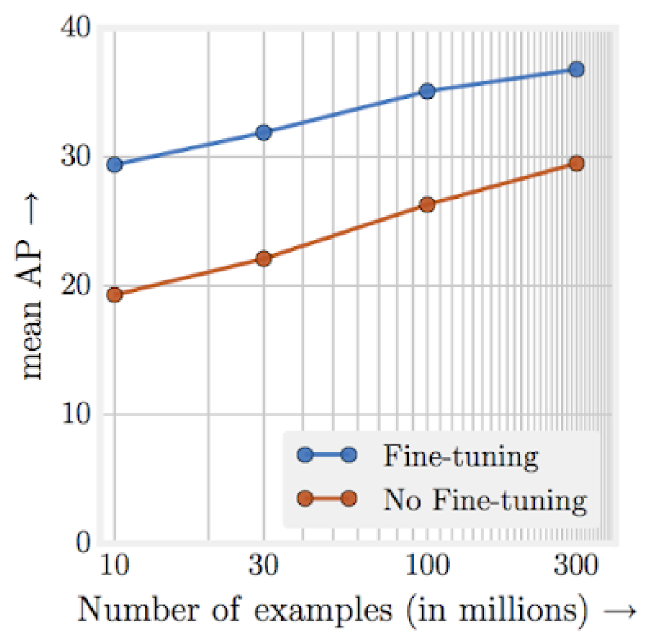
Georgia Tech

Learning with Less Labels

But it doesn't always work that well!

- If the **source** dataset you train on is very different from the **target** dataset, transfer learning is not as effective
- If you have enough data for the target domain, it just results in faster convergence
  - See He et al., "Rethinking ImageNet Pre-training"

Effectiveness of More Data



From: Revisiting the Unreasonable Effectiveness of Data https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

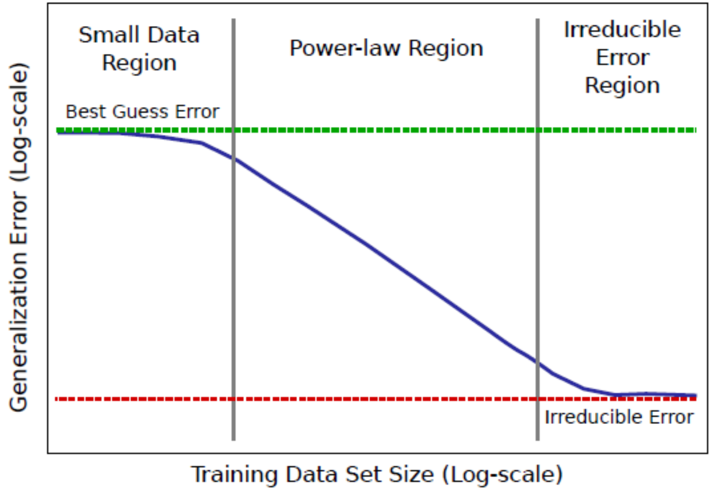


Figure 6: Sketch of power-law learning curves

From: Hestness et al., Deep Learning Scaling Is Predictable

There is a large number of different low-labeled settings in DL research

Setting	Source	Target	Shift Type
Semi-supervised	Single labeled	Single unlabeled	None
Domain Adaptation	Single labeled	Single unlabeled	Non-semantic
Domain Generalization	Multiple labeled	Unknown	Non-semantic
Cross-Task Transfer	Single labeled	Single unlabeled	Semantic
Few-Shot Learning	Single labeled	Single few-labeled	Semantic
Un/Self-Supervised	Single unlabeled	Many labeled	Both/Task

Non-Semantic Shift

Semantic Shift

Regularization

Many **standard regularization methods** still apply!

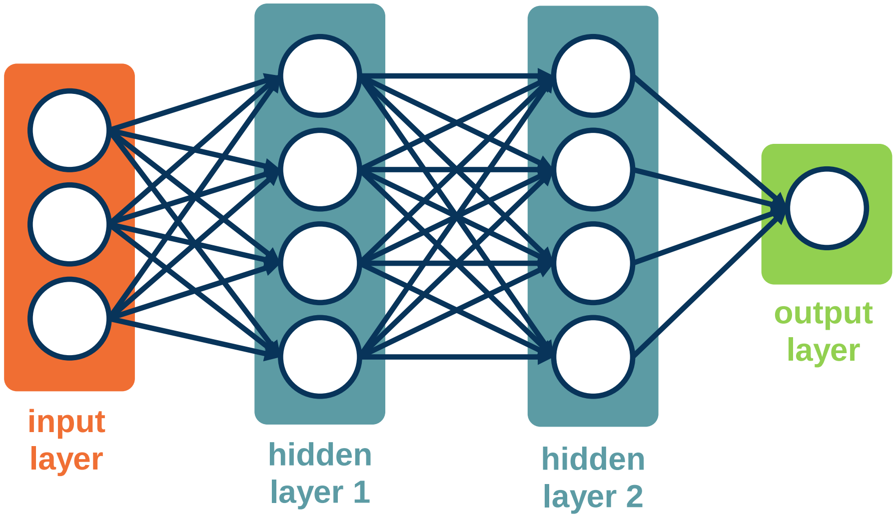
L1 Regularization

$$L = |y - Wx_i|^2 + \lambda|W|$$

where $|W|$ is element-wise

Example regularizations:

- L1/L2 on weights (encourage small values)
- L2: $L = |y - Wx_i|^2 + \lambda|W|^2$ (weight decay)
- Elastic L1/L2: $|y - Wx_i|^2 + \alpha|W|^2 + \beta|W|$

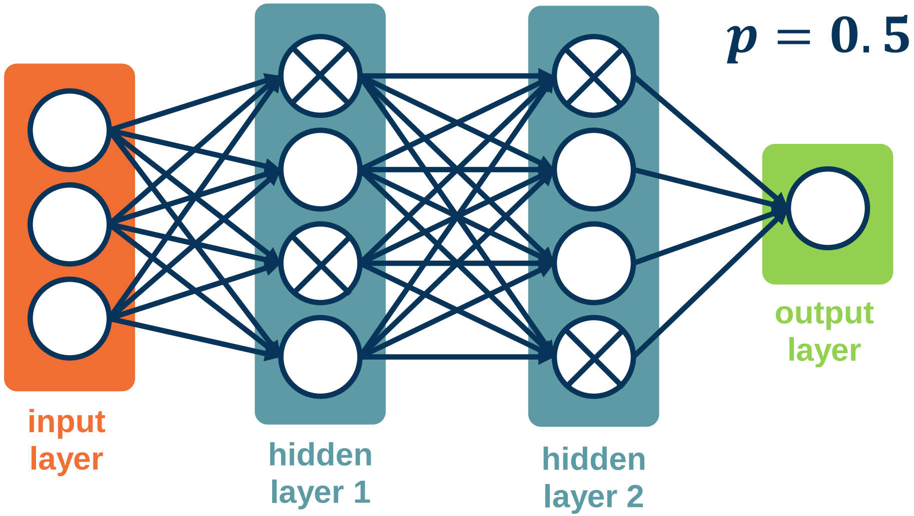


Problem: Network can learn to rely strong on a few features that work really well

- May cause **overfitting** if not representative of test data

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Preventing Co-Adapted Features



An idea: For each node, keep its output with probability p

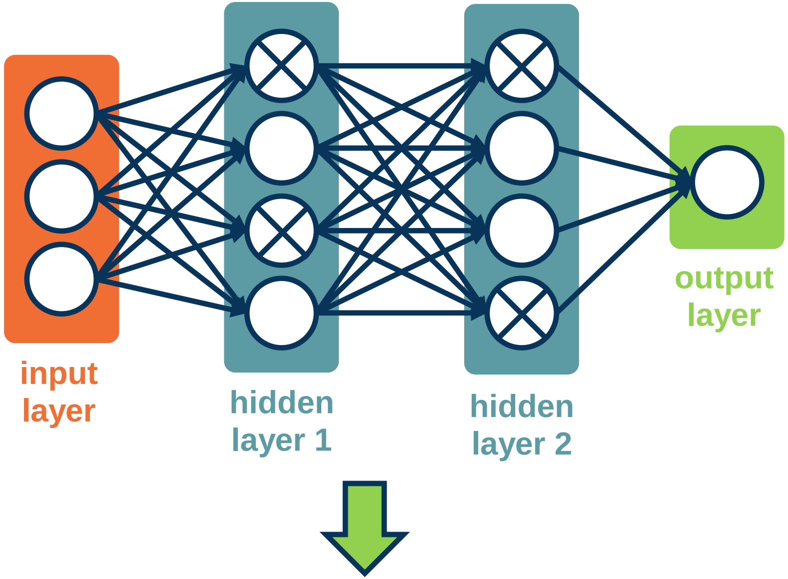
- Activations of deactivated nodes are essentially zero

Choose whether to mask out a particular node **each iteration**

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

- In practice, implement with a **mask** calculated each iteration

- During testing, no nodes are dropped

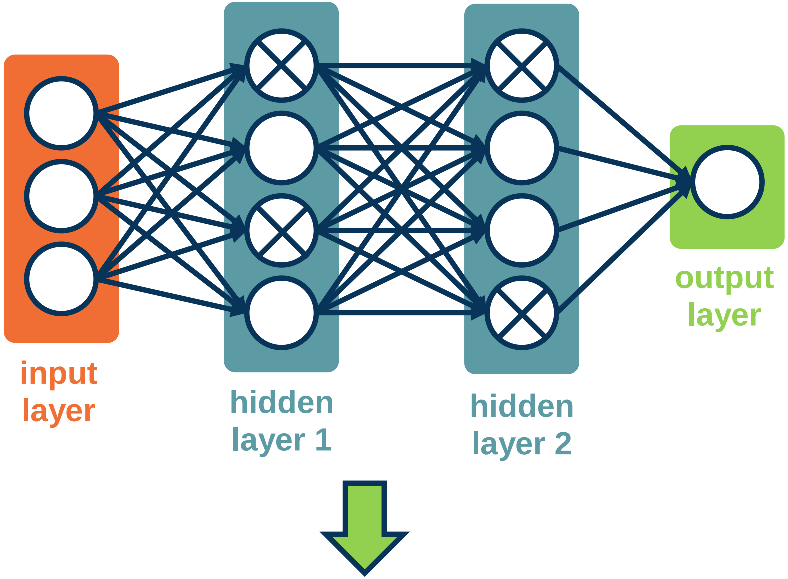


$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Dropout Implementation

- During training, each node has an expected $p * fan\_in$ nodes
- During test all nodes are activated
- **Principle:** Always try to have similar train and test-time input/output distributions!



$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Solution: During test time, scale outputs (or equivalently weights) by p

- i.e. $W_{test} = pW$

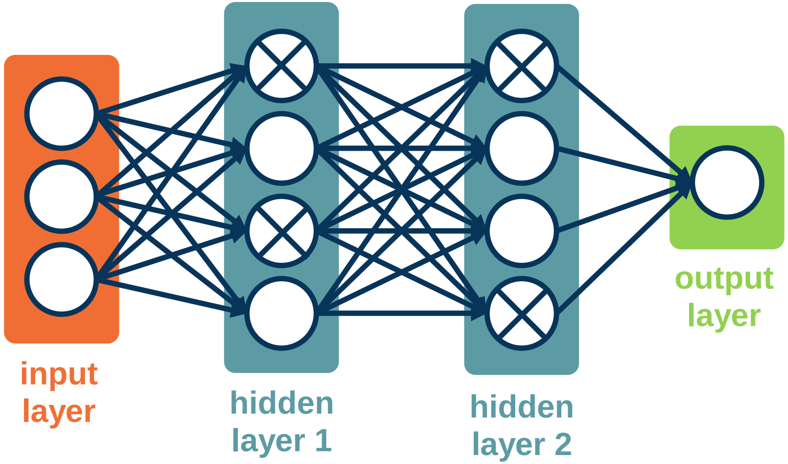
- Alternative: Scale by $\frac{1}{p}$ at train time

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Inference with Dropout

Interpretation 1: The model should not rely too heavily on particular features

- If it does, it has probability $\mathbf{1 - p}$ of losing that feature in an iteration



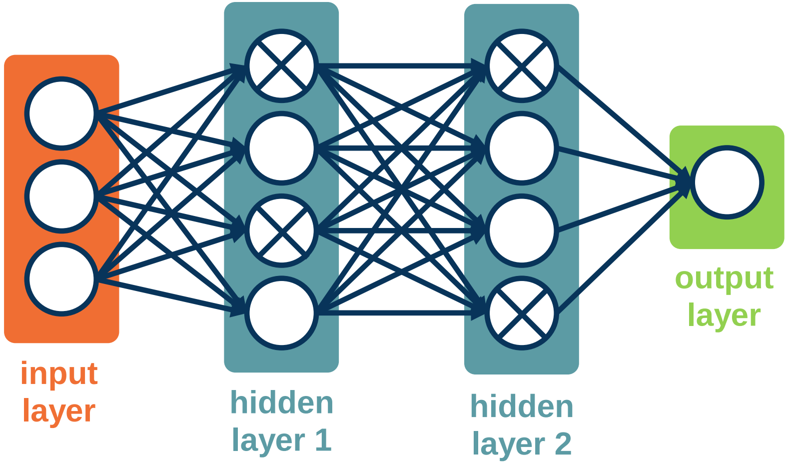
From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Interpretation 1: The model should not rely too heavily on particular features

- If it does, it has probability $1 - p$ of losing that feature in an iteration

Interpretation 2: Training 2^n networks:

- Each configuration is a network
- Most are trained with 1 or 2 mini-batches of data



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

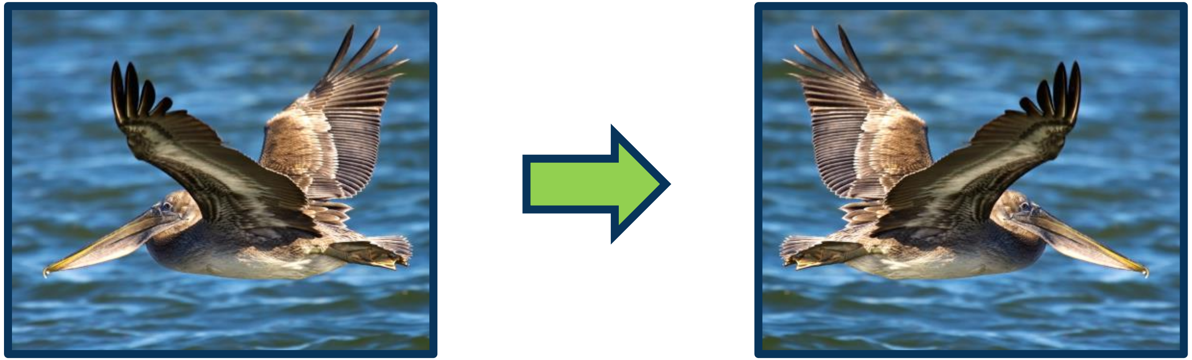
Data Augmentation

Data Augmentation: Motivation

Data augmentation – Performing a range of **transformations** to the data

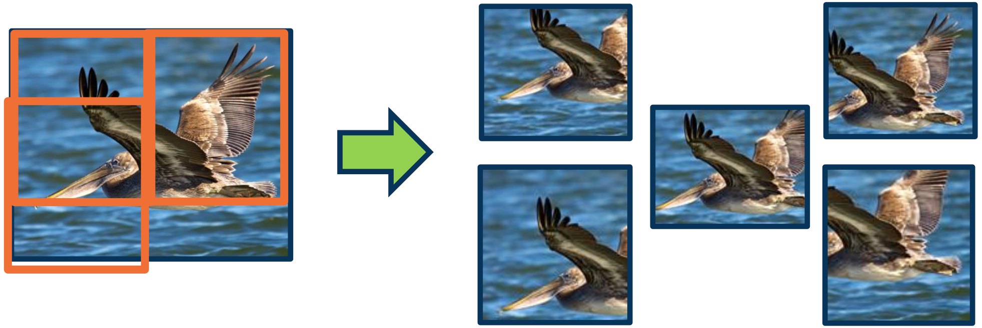
- This essentially “**increases**” your dataset
- Transformations should not change meaning of the data (or label has to be changed as well)

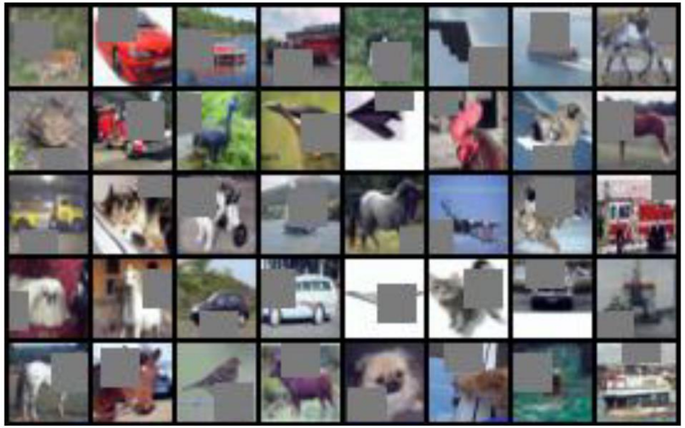
Simple example: Image Flipping



Random crop

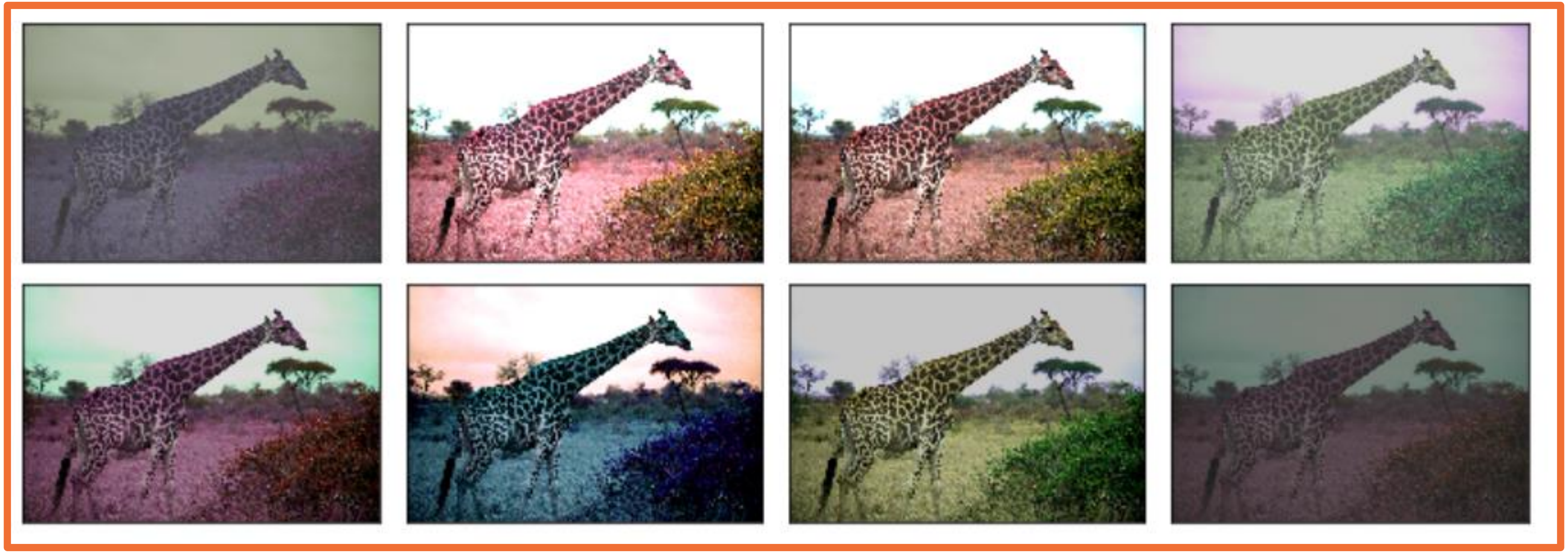
- Take different crops during training
- Can be used during inference too!





CutMix

Color Jitter



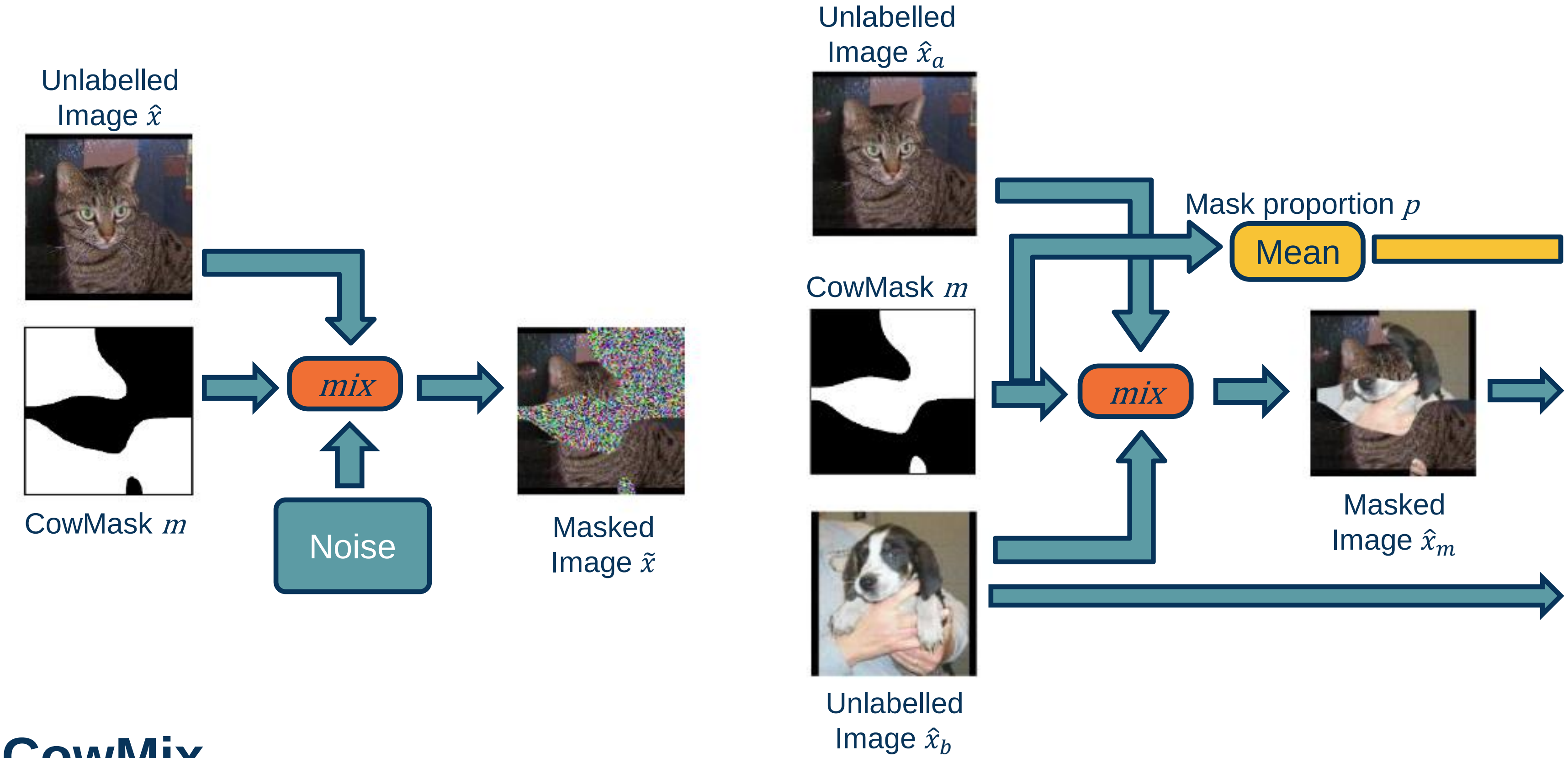
From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

We can apply **generic affine transformations:**

- **Translation**
- **Rotation**
- **Scale**
- **Shear**

We can **combine these transformations** to add even more variety!

From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html



CowMix

From French et al., "Milking CowMask for Semi-Supervised Image Classification"